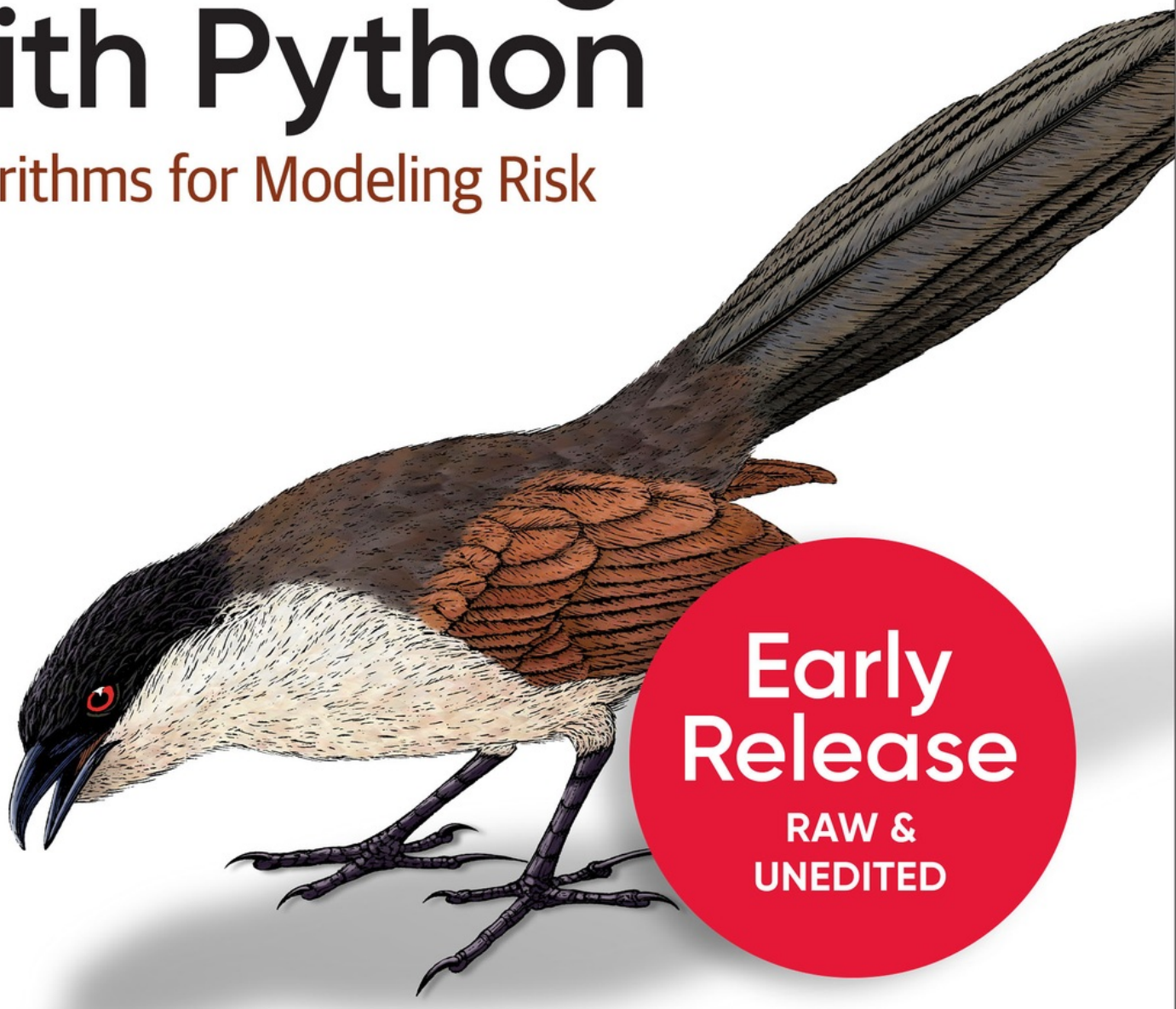
O'REILLY®

Machine Learning for Financial Risk Management with Python

Algorithms for Modeling Risk



Early Release

RAW & UNEDITED

Abdullah Karasan

Machine Learning for Financial Risk Management with Python

Algorithms for Modeling Risk

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Abdullah Karasan

Machine Learning for Financial Risk Management with Python

by Abdullah Karasan

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Michelle Smith

Development Editor: Michele Cronin

Production Editor: Daniel Elfanbaum

Interior Designer: David Futato

Cover Designer: Karen Montgomery

Illustrator: Kate Dullea

December 2021: First Edition

Revision History for the Early Release

- 2020-02-26: First Release
- 2021-05-19: Second Release
- 2021-09-10: Third Release

See http://oreilly.com/catalog/errata.csp?isbn=9781492085256 for release details.

978-1-492-08518-8

[LSI]

Preface

> AI and ML reflect the natural evolution of technology as increased computing power enables computers to sort through large data sets and crunch numbers to identify patterns and outliers.
>
> —BlackRock (2019)

Financial modeling has a long history with many successfully accomplished task but at the same time it has been fiercely critized due mainly to lack of *flexibility* and *non-inclusiveness* of these models. 2007-2008 financial crisis fueled this debate and paved the way for innovations and different approaches in the field of financial modeling.

Of course, this financial crisis is not the mere reason that precipitates the growth of AI applications in finance but also two more main drivers have spurred the adoption of AI in finance. That being said, data availability has enhanced computing power and intensified researches in 1990s.

Financial Stability Board (2017) stresses the validity this fact by stating:

> *"Many applications, or use "cases", of AI and machine learning already exist. The adoption of these use cases has been driven by both supply factors, such as technological advances and the availability of financial sector data and infrastructure, and by demand factors, such as profitability needs, competition with other firms, and the demands of financial regulation."*
>
> —FSB

As a sub-branch of financial modeling, financial risk management has been evolving with the adoption of AI in paralell with ever-growing role in financial decision making process. In his celebrated book, Bostrom (2014) denotes that there are two important revolutions in the history of mankind: *Agricultural Revolution* and *Industrial Revolution*. These two revolutions have such a profound impact that any third revolution of similar magnitude would double in size of world economy in 2 weeks. Even more strikingly, if

the third revolution accomplishes by artificial intelligence, the impact would be way more profound.

So, the expectation is sky-high about AI applications shaping financial risk management at an unprecedented scale by making use of big data and understanding the complex structure of risk processes.

With this study, I aim to fill the void about machine learning-based applications in finance so that predictive and measurement performance of financial models can be improved. As parametric models suffer from low variance-high bias issue, machine learning models, with their flexibility, can address this problem. Moreover, as a common problem in finance, changing distribution of the data always poses a thread to the realibility of the model result but machine learning models can adapt themselves to this changing pattern in a way that models fits better. So, there is a huge need and demand about applicable machine learning models in finance and what mainly distinguish this book is the inclusion of brand new machine learning-based modeling approaches in financial risk management.

In a nutshell, this book aims to shift the current landscape of financial risk management, which is heavily based on the parametric models. The main motivation for this shift is the recent experience in highly accurate financial models based on Machine Learning models. Thus, this book is intended for those who has some initial thoughts about finance and Machine Learning in the sense that I just provide brief explanations on these topics.

Consequently, the targeted audience of the book includes, but not limited to, financial risk analysts, financial engineers, risk associates, risk modelers, model validators, quant risk analysts, portfolio analyst and those who are interested in finance and data science.

In light of the background of the targeted audience, having introductory level of finance and data science knowledge would enable the highest benefit that you can get from the book. It does not, however, mean that people from different background cannot follow this book topics. Rather, readers from different background can grasp the concepts as long as they spend enough

time and refer to some other finance and data science books along with this one.

The book consists of ten chapters.

Chapter 1

Gives an introduction about the main risk management concepts. So, it covers risks, types of risks such as market, credit, operational, liquidity, risk management. After defining what the risk is, types of risks are discussed, and then risk management is explained and the issues of why it is important and how it can be used to mitigate losses is addressed. In what follows, asymmetric information, which can address the market failures are discussed. To do that, we will focus on the information asymmetry and adverse selection.

Chapter 2

Shows the time-series applications using traditional models, namely moving average model, autoregressive model, autoregressive integrated moving average model. In this part, we learn how to use an API to access financial data and how to employ it. This chapter main aims to provide us a benchmark to compare the traditional time series approach with recent development in time series modeling, which is the main focus of the next chapter.

Chapter 3

Introduces the deep learning tools for time series modeling. Recurrent Neural Network and Long Short Term Memory are two approaches by which we are able to model the data with time dimension. Besides, this chapter gives us an impression about the applicability of deep learning models to time series modeling.

Chapter 4

Focuses on the volatility prediction. Increased integration of financial markets has led to a prolonged uncertainty in financial markets, which in

turn stresses the importance of volatility. Volatility is used in measuring the degree of risk, which is one of the main engagements of the area of finance. The fourth chapter deals with the novel volatility modeling based on Support Vector Regression, Neural Network, Deep Learning, and Bayesian approach. For the sake of comparison of the performances, traditional ARCH and GARCH-type models are also employed.

Chapter 5

Employs machine learning-based models to boost estimation performance of the traditional market risk models, namely Value-at-Risk (VaR) and Expected Shortfall (ES). VaR is a quantitative approach for the potential loss of fair value due to market movements that will not be exceeded in a defined period of time and with a defined confidence level. Expected Shortfall, on the other hand, focuses on the tail of the distribution refering to big and unexpected losses. VaR model is developed using denoised covariance matrix and ES is developed incorporating liqudity dimension of the data.

Chapter 6

Tries to introduce comprehensive machine learning-based approach to estimate credit risk. Machine Learning models are applied based on the past credit information along with others. The approach starts with risk bucketing, which is suggested by Basel Accord and continue with different models bayesian estimation, Markov Chain model, support vector classification, random forest, neural network, and deep learning. In the last part of the chapter, the performance of these models will be compared.

Chapter 7

Gaussian Mixture model is used to model the liquidity, which is thought to be a neglected dimension in risk management. This model allows us to incorporate different aspects of the liquidity proxies in this chapter so

that we will be able to capture the effect of liquidity on financial risk in a more robust way.

Chapter 8

Covers the operational risk, which may result in a failure mostly due to companies internal weakness. There are several sources of operational risks but fraud risk is one of the most time-consuming and detrimental one to the companies operations. Here, in this chapter, fraud is will be our main focus and new approaches will be developed to have better-performing fraud application based on machie learning models.

Chapter 9

Introduces a brand new approach in modeling the corporate governance risk: Stock Price Crash. Many studies find an empirical link between stock price crash and corporate governance. This chapter, using Minimum Covariance Determinant model, attempts to unveil the relationship between the components of corporate governance risk and stock price crash.

Chapter 10

Makes use of synthetic data to estimate different financial risks. The aim of this chapter is to highlight the emergence of synthetic data that helps us to minimize the impact of limited historical data. So, synthetic data allows us to have large enough and high-quality data, which improves the quality of the model.

Part I. Risk Management Foundation

Chapter 1. Fundamentals of Risk Management

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

> *In 2007, no one would have thought that risk functions could have changed as much as they have in the last eight years. It is a natural temptation to expect that the next decade has to contain less change. However, we believe that the opposite will likely be true.*
>
> —Harle et al. (2016)

Risk management is a constantly evolving process. Constant evolution is inevitable due to the fact that long-standing risk management practice cannot keep pace with recent development or be a precursor of unfolding crises. Therefore, it is of importance to monitor and adopt the changes brought by structural breaks in a risk management process. Adoption to the changes implies re-defining the components and tools of risk management and this is what this book is all about.

Traditionally, in finance, empirical research in finance has a strong focus on statistical inference. Econometric has been built upon the rationale of

statistical inference. These types of models concentrate on the structure of underlying data generating process and relationship among variables. Machine learning models, however, are not assumed to define the underlying data generating process but are considered as a means to an end for the purpose of prediction (Lommers et al. 2021). Thus, machine learning models are tend to be more data-centric and prediction accuracy-oriented.

Moreover, data scarcity and unavailability has always been an issue in finance and it is not hard to guess that the econometric models cannot perform well in this case. Given the solution of machine learning models to the data unavailability via synthetic data generation, machine learning models has been on the top of the agenda in finance and financial risk management is, of course, no exception.

Before going into the detail and discuss these tools, it is worth introducing the main risk management concepts. Thus, this part of the book presents basic concepts of financial risk management, which I will refer throughout the book. These concepts include risk, types of risks, risk management, returns, and some concepts related to risk management.

Risk

Risk is always out there during the course of life but understanding and assessing risk is a bit tougher than knowing it due to its abstract nature. Risk is perceived as something hazardous and it might be in the forms of expected or unexpected. The expected risk is something that is priced but the unexpected risk can be barely accounted for, so it might be devastating.

As you can imagine there is no general consensus on the definition of risk. However, from the financial standpoint, risk refers to a likely potential loss or the level of uncertainty to which a company can expose. Differently, McNeil et al. (2015) define risk as:

> *Any event or action that may adversely affect an organization's ability to achieve its objectives and execute its strategies or, alternatively, the quantifiable likelihood of loss or less-than-expected returns.*
>
> —Harle et al. (2016)

These definitions focus on the downside of the risk implying that cost goes hand in hand with risk but it should also be noted that there is no necessarily one-to-one relationship among them. For instance, if a risk is expected, a cost incurred is relatively lower (or even ignorable) than that of unexpected risk.

Return

All financial investments are undertaken to gain profit, which is also called return. More formally, return is the gain made on an investment in a given period of time. Thus, return refers to the upside of the risk. Throughout the book, risk and return will refer to downside and upside risk, respectively.

As you can imagine, there is a trade-off between risk and return, the higher risk assumed, the greater the return realized. As it is a formidable task to come up with a optimum solution, this trade-off is one of the most controversial issues in finance. However, Markowitz (1952) proposes an intuitive and appealing solution to this long-standing issue. The way he defined risk, which was until then ambiguous, is nice and clean and led to a shift in landscape in financial research. Markowitz (1952) used standard deviation σ_{R_i} to quantify risk. This intuitive definition allows researchers to use mathematics and statistics in finance. The standard deviation can be mathematically defined as (Hull, 2012):

$$\sigma = \sqrt{\mathbb{E}(R^2) - [\mathbb{E}(R)]^2}$$

where R and $\mathbb{E}$ symbols refer to annual return and expectation, respectively. The book uses the symbol $\mathbb{E}$ numerous times as expected return represent the return of interest. This is because it is probability we are talking about in defining risk. When it comes to portfolio variance, covariance comes into the picture and the formula turns out to be:

$$\sigma_p^2 = w_a^2\sigma_a^2 + w_b^2\sigma_b^2 + 2w_a w_b \mathrm{Cov}(r_a, r_b)$$

where w denotes weight, σ^2 is variance, and *Cov* is covariance matrix.

Taking square root of the variance obtained above gives us the portfolio standard deviation:

$$\sigma_p = \sqrt{\sigma_p^2}$$

In other words, portfolio expected return is a weighted average of the individual returns and can be shown as:

$\mathbb{E}(R) = \sum_i^n w_i R_i = w_1 R_1 + w_2 R_2 \cdots + w_n R_n$

Let us explore the risk-return relationship by visualization. To do that, an hypothetical portfolio is constructed to calculate necessary statistics with Python.

```
In [1]: import statsmodels.api as sm
        import numpy as np
        import plotly.graph_objs as go
        import matplotlib.pyplot as plt
        import plotly
        import warnings
        warnings.filterwarnings('ignore')

In [2]: n_assets = 5❶
        n_simulation = 500❷

In [3]: returns = np.random.randn(n_assets, n_simulation)❸

In [4]: rand = np.random.rand(n_assets)❹
        weights = rand/sum(rand)❺


        def port_return(returns):
            rets = np.mean(returns, axis=1)
            cov = np.cov(rets.T, aweights=weights, ddof=1)
            portfolio_returns = np.dot(weights, rets.T)
            portfolio_std_dev = np.sqrt(np.dot(weights, np.dot(cov, weights)))
            return portfolio_returns, portfolio_std_dev❻

In [5]: portfolio_returns, portfolio_std_dev = port_return(returns)❼
```

```
In [6]: print(portfolio_returns)
        print(portfolio_std_dev)❽
        0.005773455631074148
        0.016994274496417057

In [7]: portfolio = np.array([port_return(np.random.randn(n_assets, i))
                              for i in range(1, 101)])❾

In [8]: best_fit = sm.OLS(portfolio[:, 1], sm.add_constant(portfolio[:, 0]))\
                  .fit().fittedvalues❿

In [9]: fig = go.Figure()
        fig.add_trace(go.Scatter(name='Risk-Return Relationship', x=portfolio[:,0],
                                 y=portfolio[:,1], mode='markers'))
        fig.add_trace(go.Scatter(name='Best Fit Line', x=portfolio[:,0],
                                 y=best_fit, mode='lines'))
        fig.update_layout(xaxis_title = 'Return', yaxis_title = 'Standard
Deviation',
                          width=900, height=470)
        plotly.offline.iplot(fig, filename="images/risk_return.png")
        fig.show()⓫
```

❶ Number of assets considered

❷ Number of simulations conducted

❸ Generating random samples from normal distribution used as returns

❹ Generating random number to calculate weights

❺ Calculating weights

❻ Function used to calculate expected portfolio return and portfolio standard deviation

❼ Calling the result of the function

❽ Printing the result of the expected portfolio return and portfolio standard deviation

9. Rerunning the function 100 times
10. To draw the best fit line, I run linear regression
11. Drawing interactive plot for visualization purposes

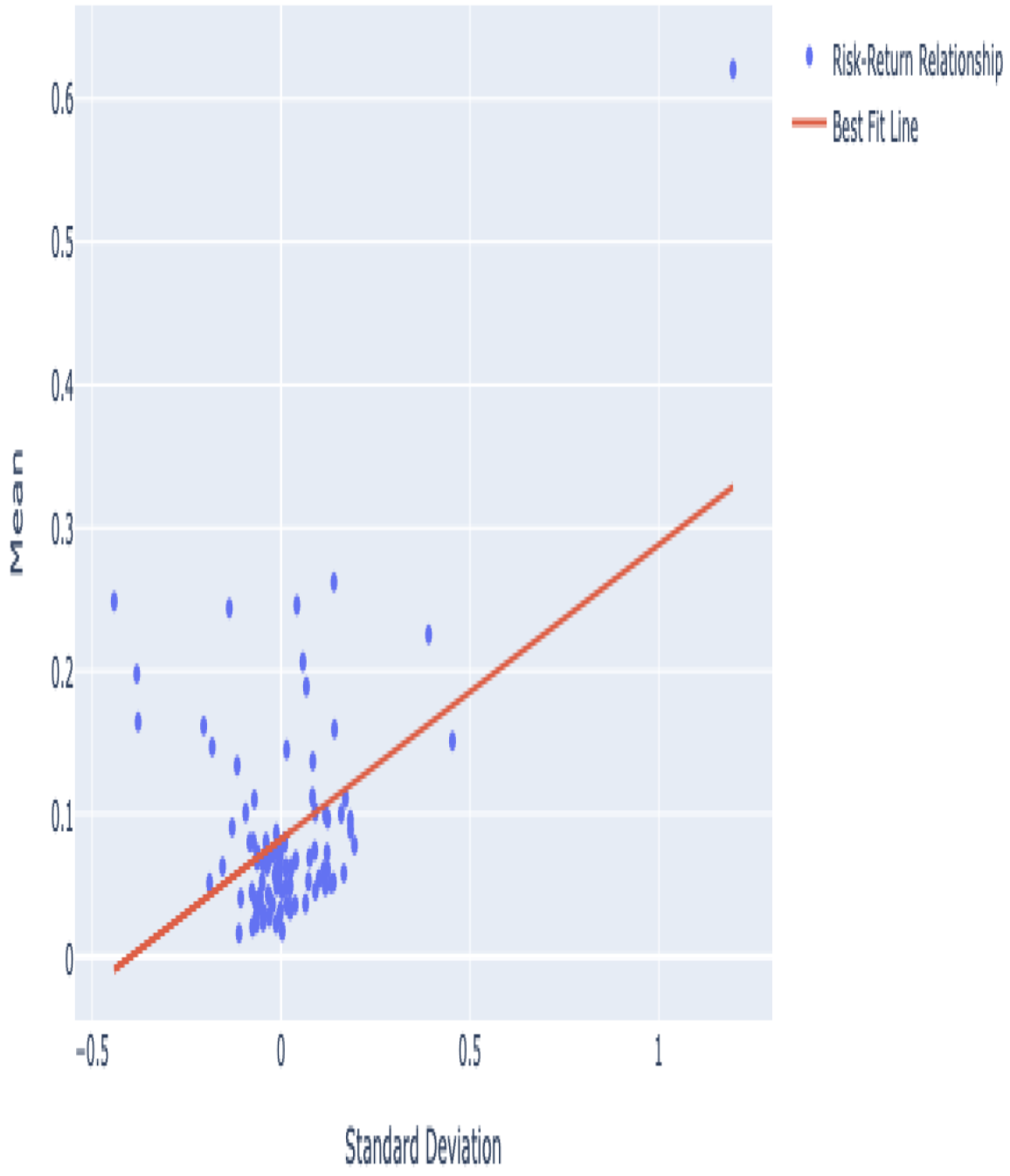


Figure 1-1. Risk-Return Relationship

Figure 1-1, generated via the above-given Python code, confirms that the risk and return goes in tandem but the magnitude of this correlation varies depending on the individual stock and the financial market conditions.

Risk Management

Financial risk management is a process to deal with the uncertainties resulting from financial markets. It involves assessing the financial risks facing an organization and developing management strategies consistent with internal priorities and policies (Horcher, 2011).

According to this definition, as every organization faces different type of risks, the way that a company deals with it is completely unique. Every company should properly assess and take necessary action against risk. It, however, does not necessarily mean that once a risk is identified, it needs to be mitigated as much as a company can do.

Risk management is, therefore, not about mitigating risk at all costs. Mitigating risk may require sacrificing return and it can be tolerable up to certain level as companies are searching for higher return as much as lowering risk. Thus, to maximize profit while lowering the risk should be delicate and well-defined task.

Managing risk is a delicate task as it comes with a cost and even though dealing with it requires specific company policies, there exists a general framework for possible risk strategies. These are:

- *Ignore*: In this strategy, companies accept all risks and their consequences and prefer to do nothing.
- *Transfer*: This strategy involves transferring the risks to a third-party by hedging or some other ways.
- *Mitigate*: Companies develop a strategy to mitigate risk partly because its harmful effect might be considered too much to bear and/or supprass the benefit attached to it.
- *Accept risk*: If companies embrace the strategy of *accepting the risk*, they properly identify risks and acknowledge the benefit of them. In other words, when assuming certain risks arising from some activities bring values to shareholder, this strategy can be picked.

Main Financial Risks

Financial companies face various risks over their business life. These risks can be divided into different categories in a way to identify and assess in an easier manner. These main financial risk types are market risk, credit risk, operational risk, and liquidity risk but again this is not an exhaustive list. However, we confine our attention to the main financial risk types throughout the book. Let's take a look at these risk categories.

Market Risk

This risk arises due to a change in factors in financial market. To be clearer, for instance, an increase in *interest rate* might badly affect a company, which has a short position.

A second example can be given about another source of market risk; *exchange rate*. An company involving involving international trade, whose commodities are priced in US dollars, are highly exposed to a change in US dollars.

As you can imagine, any change in *commodity price* might pose a threat to a company's financial sustainability. There are many fundamentals that have direct effect on commodity price, which are market players, transportation cost and so on.

Credit Risk

Credit risk is one of the most pervasive risks. Credit risk emerges when counterparty fails to honor debt. For instance, if a borrower is unable to pay its payment, credit risk is realized. The deterioration of credit quality is also a source of risk through the reduced market value of securities that an organization might own (Horcher, 2011).

Liquidity Risk

Liquidity risk has been overlooked until 2007-2008 financial crisis, which hit the financial market hard. From that point on, researches on liquidity risk have been intensified. Liquidity refers to speed and ease with which an

investor executes her transaction. This definition is also known as trading liquidity risk. The other dimension of liquidity risk is funding liquidity risk, which can be defined as the ability to raise cash or availability of credit to finance a company's operations.

If a company cannot turn its assets into cash within a short period of time, this falls under liquidity risk category and it is quite detrimental to company's financial management and reputation.

Operational Risk

Managing operational risk is beyond being a clear and foreseeable task and exploits a great deal of resources of a company due to intricate and internal nature of the risk. Now the questions are: How do financial companies do a good job for managing risk? Do they allocate necessary resources for this task? Is the importance of risk to a companies sustainability gauged properly?

As the name suggests, operational risk arises when inherent operation(s) in a company or industry poses a threat to day-to-day operations, profitability, or sustainability of that company. Operational risk includes fraudulent activities, failure to adhere regulations or internal procedures, losses due to lack of training etc.

Well, what happens if a company exposes to one or more than one of these risks in an unprepared way. Though it is not frequent, we know the answer from the historical events, company might default and run into a big financial bottleneck.

Big Financial Collapse

How important is the risk management? This question can be addressed by a book with hundreds of pages but, in fact, the rise of risk management in financial institutions speaks itself. In particular, after the global financial crisis, the risk management is characterized as "colossal failure of risk management" (Buchholtz and Wiggins,2019). In fact, the global financial crisis, emerged in 2007-2008, is just a tip of the iceberg. Numerous failures

in risk management pave the way for this breakdown in the financial system. To understand this breakdown, we need to go back to dig into past financial risk management failures. A hedge fund called Long-Term Capital Management (LTCM) presents a vivid instance of a financial collapse.

LCTM forms a team with top-notch academics and practitioners. This led to a fund inflow to the firm and began trading with 1 billion USD. In 1998, LCTM controlled over $100 billion and heavily invested in some emerging markets including Russia. So, Russian debt default deeply affected the LCTM's portfolio due to *fligh to quality* [1] and it got a severe blow, which led LCTM to bust (Allen, 2003).

Metallgesellschaft (MG) is another company that no longer exists due to bad financial risk management. MG was largely operating in gas and oil markets. Due to high exposure to the gas and oil prices, MG needed funds in the aftermath of the large drop in gas and oil prices. Closing the short position resulted in losses around 1.5 billion USD.

Amaranth Advisors (AA) is another hedge fund, which went into bankruptcy due to heavily investing in a single market and misjudging the risks arising from its investments. By 2006, AA attracted roughly $9 billion USD worth of assets under management but lost nearly half of it because of the natural gas futures and options downward move. The default of AA is attributed to low natural gas prices and misleading risk models (Chincarini, 2008).

Long story short, Stulz's paper titled "Risk management failures: What are they and when do they happen?" (2008) summarizes the main risk management failures resulting in default:

1) Mismeasurement of known risks

2) Failure to take risks into account

3) Failure in communicating the risks to top management

4) Failure in monitoring risks

5) Failure in managing risks

6) Failure to use appropriate risk metrics

Thus, the global financial crisis was not the sole event that led regulators and institutions to redesign its financial risk management. Rather, it is the drop that filled the glass and, in the aftermath of the crisis, both regulators and institutions have adopted lessons learned and improved their process. Eventually, these series of events led to a rise in financial risk management.

Information Asymmetry in Financial Risk Management

Though it is theoretically intuitive, the assumption of completely rational decision maker, main building block in modern finance theory, is too perfect to be real. This idea, therefore, attacked by behavioral economists, who believes that psychology plays a key role in decision making process.

> *Making decisions is like speaking prose—people do it all the time, knowingly or unknowingly. It is hardly surprising, then, that the topic of decision making is shared by many disciplines, from mathematics and statistics, through economics and political science, to sociology and psychology.*
>
> —Kahneman and Tversky (1984)

Information asymmetry and financial risk management goes hand in hand as cost of financing and firm valuation are deeply affected by the information asymmetry. That is, uncertainty in valuation of a firm's assets might raise the borrowing cost posing a threat to a firm's sustainability (See DeMarzo and Darrell (1995) and Froot, Scharfstein, and Stein (1993)).

Thus, roots of the failures described above lie deeper in such a way that perfect hypothetical world in which rational decision maker lives is unable to explain them. At this point, human instincts and imperfect world come into play and a mixture of disciplines provide more plausible justifications. So, it turns out *Adverse Selection* and *Moral Hazard* are two prominents categories accounting for the market failures.

Adverse Selection

It is a type of asymmetric information in which one party tries to exploit its informational advantage. Adverse selection arises when seller are better informed than buyers. This phemomenon is perfectly coined by Akerlof (1970) as “The Markets for Lemons”. Within this framework, lemons refer to low-quality.

Consider a market with lemon and high-quality cars and buyers know that it is likely to buy a lemon, which lowers then equilibrium price. However, seller is better informed if the car is lemon or high-quality. So, in this situation, benefit from exchange might disappear and no transaction takes place.

Due to the complexity and opaqueness, mortgage market in the pre-crisis era is a good example of adverse selection. More elaborately, borrowers knew more about their willingness and ability to pay than lenders. Financial risk was created through the securitizations of the loans, i.e., mortgage backed securities. From this point on, the originators of the mortgage loans knew more about the risks than the people they were selling them to investors in the mortgage backed securities.

Let us try to model adverse selection using Python. It is readily observable in insurance industry and therefore I would like to focus on this industry to model adverse selection.

Suppose that the consumer utility function is:

$$U(x) = e^{\gamma x}$$

where x is income and γ is a parameter, which takes on values between 0 and 1.

NOTE

Utility function is a tool used to represent consumer preferences for goods and service and it is concave for risk-averse individuals.

The ultimate aim of this example is to decide whether or not to buy an insurance based on consumer utility.

For the sake of practice, I assume that the income is $2 USD and cost of the accident is $1.5 USD.

Now it is time to calculate the probability of loss, π, which is exogenously given and it is uniformly distributed.

As a last step, in order to find equilibrium, I have to define supply and demand for insurance coverage. The following code block indicates how we can model the adverse selection.

```
In [10]: import matplotlib.pyplot as plt
         import numpy as np
         plt.style.use('seaborn')

In [11]: def utility(x):
             return(np.exp(x ** gamma))❶

In [12]: pi = np.random.uniform(0,1,20)
         pi = np.sort(pi)❷

In [13]: print('The highest three probability of losses are {}'.format(pi[-3:]))❸
         The highest three probability of losses are [0.73279622 0.82395421
          0.88113894]

In [14]: y = 2
         c = 1.5
         Q = 5
         D = 0.01
         gamma = 0.4

In [15]: def supply(Q):
             return(np.mean(pi[-Q:]) * c)❹

In [16]: def demand(D):
             return(np.sum(utility(y - D) > pi * utility(y - c) + (1 - pi) *
utility(y)))❺

In [17]: plt.figure()
         plt.plot([demand(i) for i in np.arange(0, 1.9, 0.02)], np.arange(0, 1.9,
0.02),
                  'r', label='insurance demand')
         plt.plot(range(1,21), [supply(j) for j in range(1,21)],
```

```
            'g', label='insurance supply')
plt.ylabel("Average Cost")
plt.xlabel("Number of People")
plt.legend()
plt.savefig('images/adverse_selection.png')
plt.show()
```

❶ Writing a function for risk-averse utility function.

❷ Generating random samples from uniform distribution.

❸ Picking the last three items.

❹ Writing a function for supply of insurance contracts.

❺ Writing a function for demand of insurance contracts.

Figure 1-2 exhibits the insurance supply and demand curve. Surprisingly, both curve are downward sloping implying that as more people demand contract and more people is added on the contract, the risk lowers that affects the price of the insurance contract.

Straight line presents the insurance supply and average cost of the contract and the line, showing step-wise downward slope, denotes the demand for insurance contracts. As we start analysis with the risky customers as you can add more and more people to the contract the level of riskiness diminishes in parallel with the average cost.

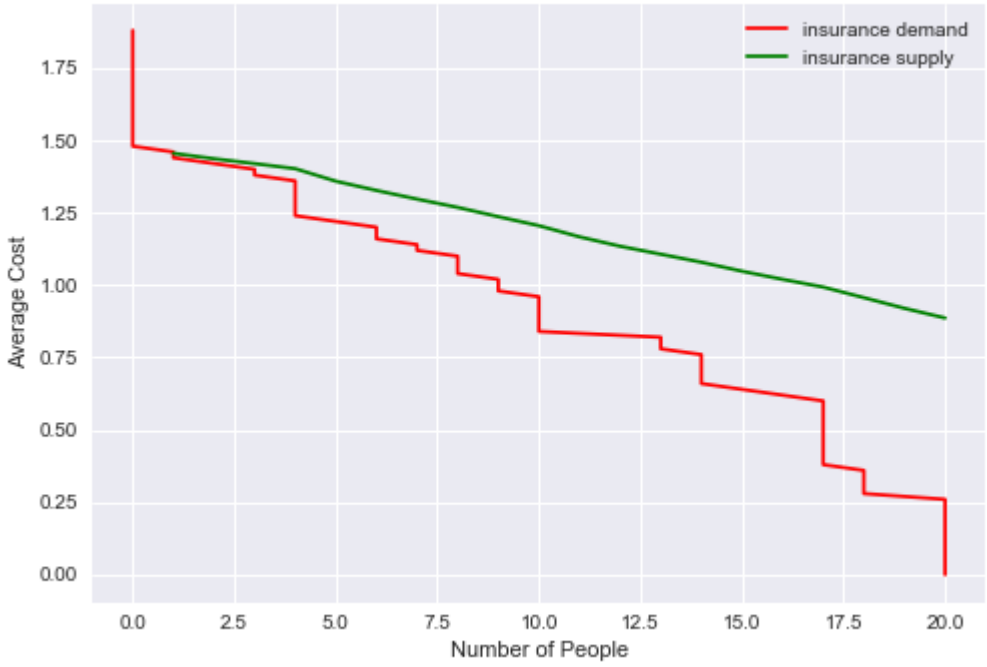


Figure 1-2. Adverse Selection

Moral Hazard

Market failures also result from asymmetric information. In moral hazard situation, one party of the contract assumes more risk than the other party. Formally, moral hazard may be defined as a situation in which more informed party takes advantages of the private information at his disposal to the detriment of others.

For a better understanding of moral hazard, a simple example can be given from the credit market: Suppose that entity A demands credit for use in financing the project, which is considered as feasible to finance. Moral hazard arises if entity A utilizes the loan for the payment of credit debt to bank C, without prior notice to the lender bank. While allocating credit, the moral hazard situation that banks may encounter arises as a result of asymmetric information, decreases banks' lending appetites and appears as one of the reasons why banks put so much labor on credit allocation process.

Some argue that rescue operation undertaken by FED for LCTM can be considered as moral hazard in a way that FED enters into contracts on bad faith.

Conclusion

This chapter presents the main concepts of financial risk management with a view to make sure that we are all on the same page. This refresher helps us a lot throughout this book because we use these terms and concepts.

In addition to that, a behavioral approach, attacking the rationale of a finance agent, is discussed in the last part of the first chapter so that we have more encompassing tools to account for the sources of financial risk.

In the next chapter, we will discuss the time series approach, which is one of the main pillars of the financial analysis in the sense that most of the financial data has a time dimension requiring special attention and technique to deal with.

Further Resources

Articles cited in this chapter:

- Akerlof, George A. “The market for “lemons”: Quality uncertainty and the market mechanism.” In Uncertainty in economics, pp. 235-251. Academic Press, 1978.
- Buchholtz, Alec, and Rosalind Z. Wiggins. “Lessons Learned: Thomas C. Baxter, Jr., Esq.” Journal of Financial Crises 1, no. 1 (2019): 202-204.
- Chincarini, Ludwig. “A case study on risk management: lessons from the collapse of amaranth advisors llc.” Journal of Applied Finance 18, no. 1 (2008): 152-74.

- DeMarzo, Peter M., and Darrell Duffie. “Corporate incentives for hedging and hedge accounting.” The review of financial studies 8, no. 3 (1995): 743-771.
- Froot, Kenneth A., David S. Scharfstein, and Jeremy C. Stein. “Risk management: Coordinating corporate investment and financing policies.” The Journal of Finance 48, no. 5 (1993): 1629-1658.
- Lommers, Kristof, Ouns El Harzli, and Jack Kim. “Confronting Machine Learning With Financial Research.” Available at SSRN 3788349 (2021).
- Stulz, René M. “Risk management failures: What are they and when do they happen?” Journal of Applied Corporate Finance 20, no. 4 (2008): 39-48.

Books cited in this chapter:

- Horcher, Karen A. Essentials of financial risk management. Vol. 32. John Wiley & Sons, 2011.
- Hull, John. Risk management and financial institutions,+ Web Site. Vol. 733. John Wiley & Sons, 2012.
- Kahneman, D., and A. Tversky. “Choices, Values, and Frames. American Psychological Association.” (1984): 341-350.
- Harle, P., Havas, A., & Samandari, H. (2016). The future of bank risk management. McKinsey Global Institute.
- McNeil, Alexander J., Rüdiger Frey, & Paul Embrechts. Quantitative risk management: concepts, techniques and tools-revised edition. Princeton university press, 2015.

---

1 Flight to quality term refers to an herd behavior by which investors stay away from risky assets such as stocks and takes long position in safer assets such as government-issued bond.

Chapter 2. Introduction to Time Series Modeling

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

Market behavior is examined using large amounts of past data, such as high-frequency bid-ask quotes of currencies or stock prices. It is the abundance of data that makes possible the empirical study of the market. Although it is not possible to run controlled experiments, it is possible to extensively test on historical data.

— Henri Poincare

Some models account better for some phenomena, certain approaches capture the characteristics of an event in a solid way. Time series modeling is a vivid example for this because vast majority of financial data have time dimension, which makes time series applications a necessary tool for finance. In simple terms, the ordering of the data and correlation among them is of importance.

In this chapter of the book, classical time series models will be discussed and the performance of these models will be compared. On top of that, the deep learning based time series analysis will be introduced in the [Link to Come], which has entirely different approach in terms of data preparation and model structure. The classical models to be visited include moving average model (MA), autoregressive model (AR), and finally autoregressive integrated moving average

model (ARIMA). What is common across these models are the information carried by the historical observations. If these historical observations are obtained from error terms, it is referred to as moving average, if these observations come out of time series itself, it turns out to be autoregressive. The other model, namely ARIMA, is an extension of these models.

Here is a formal definition of time series:

> *A time series is a set of observations X_t, each one being recorded at a specific time t. A discrete-time time series (the type to which this book is primarily devoted) is one in which the set T_0 of times at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed time intervals. Continuous time series are obtained when observations are recorded continuously over some time interval.*
>
> —Brockwell and Davis (2016)

Let us observe what data with time dimension looks like. Figure 2-1 exhibits the oil prices for the period of 1980-2020 and the following Python code shows us a way of producing this plot.

```
In [1]: import quandl
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
        plt.style.use('seaborn')

In [2]: oil = quandl.get("NSE/OIL", authtoken="vEjGTysiCFBuN-z5bjGP",
                         start_date="1980-01-01",
                         end_date="2020-01-01")❶

In [3]: plt.figure(figsize=(10, 6))
        plt.plot(oil.Close)
        plt.ylabel('$')
        plt.xlabel('Date')
        plt.savefig('images/Oil_Price.png')
        plt.show()
```

❶ Extracting data from *Quandl* database

Figure 2-1. Oil Price between 1980-2020

> **NOTE**
>
> An API, or Application Programming Interface, is a tool designed for retrieving data using code. We will make use of different APIs throughout the book. In the preceding practice, *Quandl* API is used.
>
> Quandl API allows us to access financial, economic, and alternative data from the *Quandl* website. In order to have your Qnadl API,please visit quanld website first and get your own API key following the necessary steps.

As can be understood from the definition provided above, time series models can be applicable to diverse areas such as:

- Health Care
- Finance
- Economics
- Network Analysis

- Astronomy
- Weather

The superiority of time series approach comes from the idea that correlation of observations in time explain better the current value. Having data with correlated structure in time implies a violation the famous identically and independently distribution, IID assumption for short, which is at the heart of many models.

THE DEFINITION OF IID

IID assumption enables us to model joint probability of data as the product of probability of observations. The process X_t is said to be an IID with mean 0 and variance σ^2:

$$X_t \sim IID(0, \sigma^2)$$

So, due to the correlation in time, the dynamics of contemporaneous stock price can be better understood by its own historical values. How can we comprehend the dynamics of the data? This is a question that we can address by elaborating the components of time series.

Time Series Component

Time series has four components, namely trend, seasonality, cyclicality, and residual. In python, we can easily visualize the components of a time series by *seasonal_decompose* function:

```
In [4]: import yfinance as yf
        import numpy as np
        import pandas as pd
        import datetime
        import statsmodels.api as sm
        from statsmodels.tsa.stattools import adfuller
        from statsmodels.tsa.seasonal import seasonal_decompose

In [5]: ticker = '^GSPC'❶
        start = datetime.datetime(2015, 1, 1)❷
        end = datetime.datetime(2021, 1, 1)❷
        SP_prices = yf.download(ticker, start=start, end=end, interval='1mo').Close❸
```

```
[**********100%**********]  1 of 1 completed
```

```
In [6]: seasonal_decompose(SP_prices, period=12).plot()
        plt.savefig('images/decomposition.png')
        plt.show()
```

❶ Denoting ticker of S&P-500

❷ Identifying the start and end dates

❸ Accessing the closing price of S&P-500

In the top panel of the Figure 2-2, we see the plot of raw data and in the second panel trend can be observed showing upward movement. In the third panel, seasonality is exhibited and finally residual is presented exhibiting an erratic fluctuations. You might wonder where the cyclicality component is, noise and cyclical component are put together under residual component.

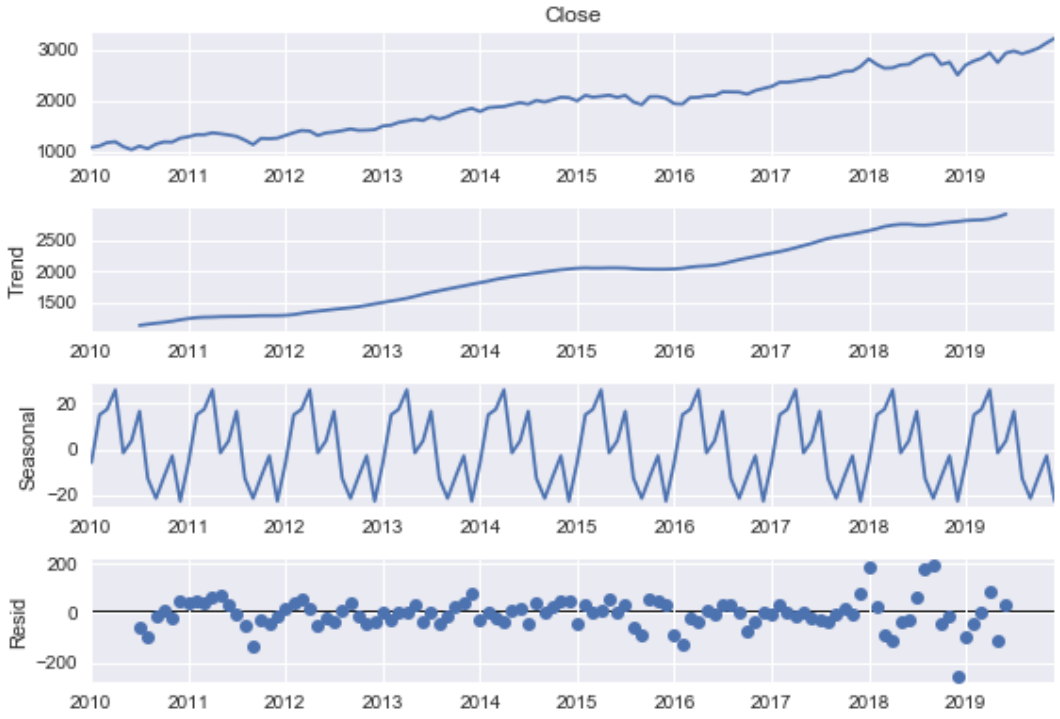


Figure 2-2. Time Series Decomposition of S&P-500

Becoming familiar with time series components is important for further analysis so that we are able to understand characteristic of the data and propose a suitable

model. Let's start with *trend* component.

Trend

Trend indicates a general tendency of an increase or decrease during a given time period. Generally speaking, trend is present when the starting and ending points are different or having upward/downward slope in a time series.

```
In [7]: plt.figure(figsize=(10, 6))
        plt.plot(SP_prices) ❶
        plt.title('S&P-500 Prices')
        plt.ylabel('$')
        plt.xlabel('Date')
        plt.savefig('images/SP_price.png')
        plt.show()
```

Aside from the period in which S&P-500 index price plunges, we see a clear upward trend in the Figure 2-3 between 2010 and 2020.

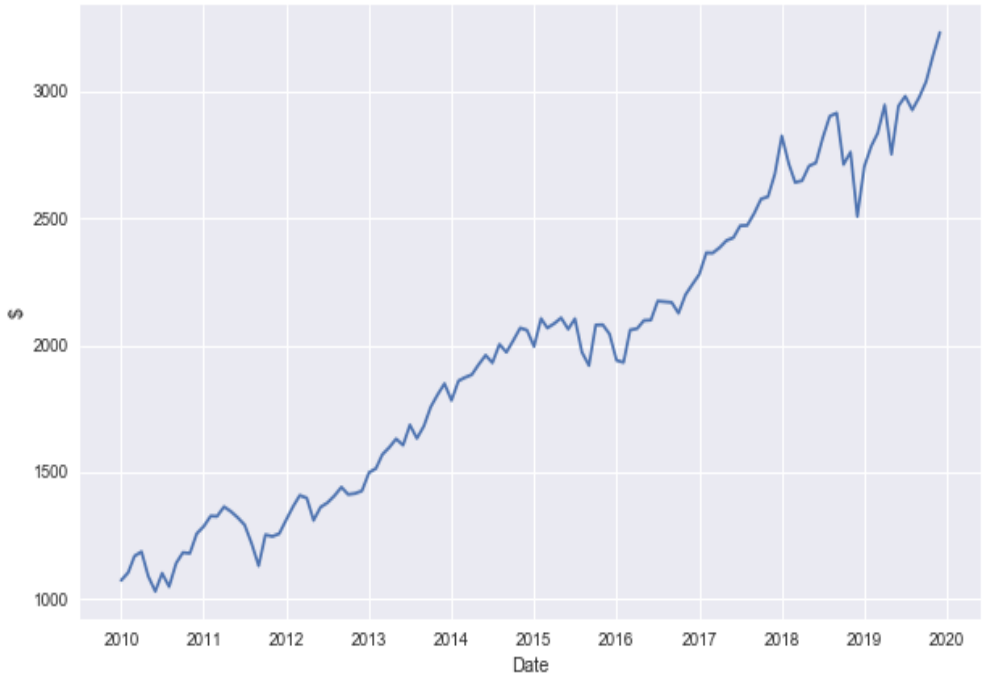


Figure 2-3. S&P-500 Price

Plotting line plot is not the mere option used for understanding trend. Rather we have some other strong tool for this task. So, at this point, it is worthwhile to talk about the two important statistical concepts:

- Autocorrelation Function
- Partial Autorcorrelation Function

Autocorrelation function, known as ACF, is a statistical tool to analyze the relationship between current value of a time series and its lagged values. Graphing ACF enables us to readily observe the serial dependence in a time series.

$$\hat{\rho}(h) = \frac{\mathrm{Cov}(X_t, X_{t-h})}{\mathrm{Var}(X_t)}$$

Figure 2-4 denotes autocorrelation function plot. The vertical lines shown in the Figure 2-4 represents the correlation coefficients, the first line denotes the correlation of the series with its 0 lag, i.e., it is the correlation with itself. The second line indicates the correlation between series value at time t-1 and t. In the light of these, we can conclude that S&P-500 shows a serial dependence. It appears to be a strong dependence between the current value and lagged values of S&P-500 data because the correlation coefficients, represented by lines in the ACF plot, decays in a slow fashion.

```
In [8]: sm.graphics.tsa.plot_acf(SP_prices,lags=30)❶
        plt.xlabel('Number of Lags')
        plt.savefig('images/acf_SP.png')
        plt.show()
```

❶ Plotting Autocorrelation Function

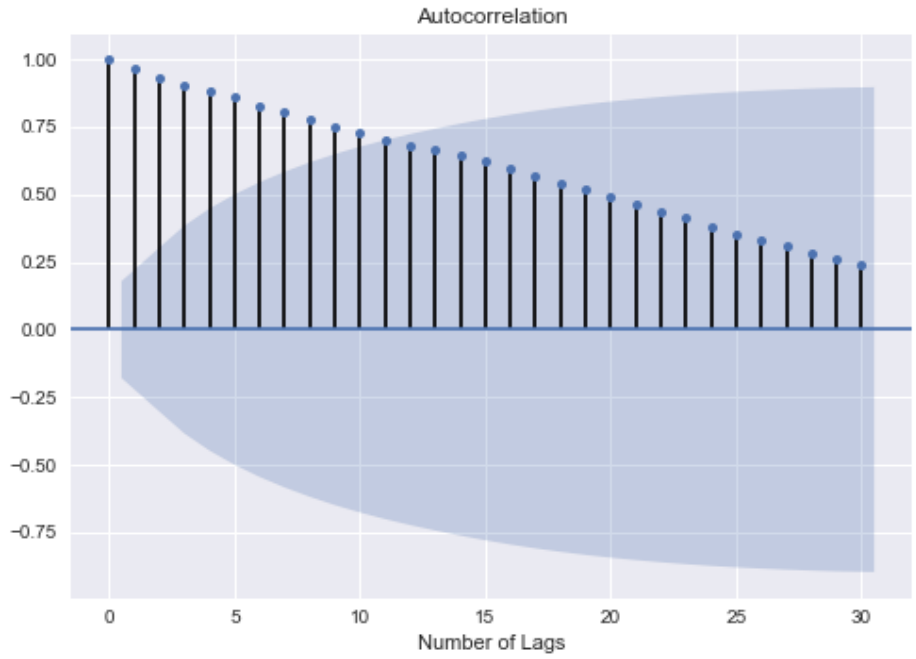


Figure 2-4. Autocorrelation Function Plot of S&P-500

Now, the question is what would be the likely sources of autocorrelations? Here are some reasons:

- The primary source of autocorrelation is "carryover" meaning that the preceding observation has an impact on the current one.
- Model misspecification.
- Measurement error, which is basically the difference between observed and actual values.
- Dropping a variable, which has an explanatory power.

Partial autocorrelation function (PACF) is another method to examine the relationship between X_t and $X_{t-p}, p \in \mathbb{Z}$. ACF is commonly considered as a useful tool in MA(q) model simply because PACF does not decay fast but approaches towards 0. However, the pattern of ACF is more applicable to MA. PACF is, on the other hand, work well with AR(p) process.

PACF provides information on correlation between current value of a time series and its lagged values controlling for the other correlations.

It is not easy to figure out what is going on at first glance. Let me give you an example. Suppose that we want to compute the partial correlation X_t and X_{t-h}. To do that, I take into account the correlation structure between X_t and X_{t-1} and X_{t-2}.

Put mathematically:

$$\hat{\rho}(h) = \frac{\text{Cov}(X_t, X_{t-h} | X_{t-1}, X_{t-2} \ldots X_{t-h-1})}{\sqrt{\text{Var}(X_t | X_{t-1}, X_{t-2}, \ldots, X_{t-h-1}) \text{Var}(X_{t-h} | X_{t-1}, X_{t-2}, \ldots, X_{t-h-1})}}$$

where h is the lag.

```
In [9]: sm.graphics.tsa.plot_pacf(SP_prices, lags=30)❶
        plt.xlabel('Number of Lags')
        plt.savefig('images/pacf_SP.png')
        plt.show()
```

❶ Plotting Partial Autocorrelation Function

Figure 2-5 exhibits the PACF of raw S&P-500 data. In interpreting the PACF, we focus on the spikes outside the dark region representing confidence interval. Figure 2-5 exhibits some spikes at different lags but the lag 16 is outside the confidence interval. So, it may be wise to select a model with 16 lags so that I am able to include all the lags up to lag 16.

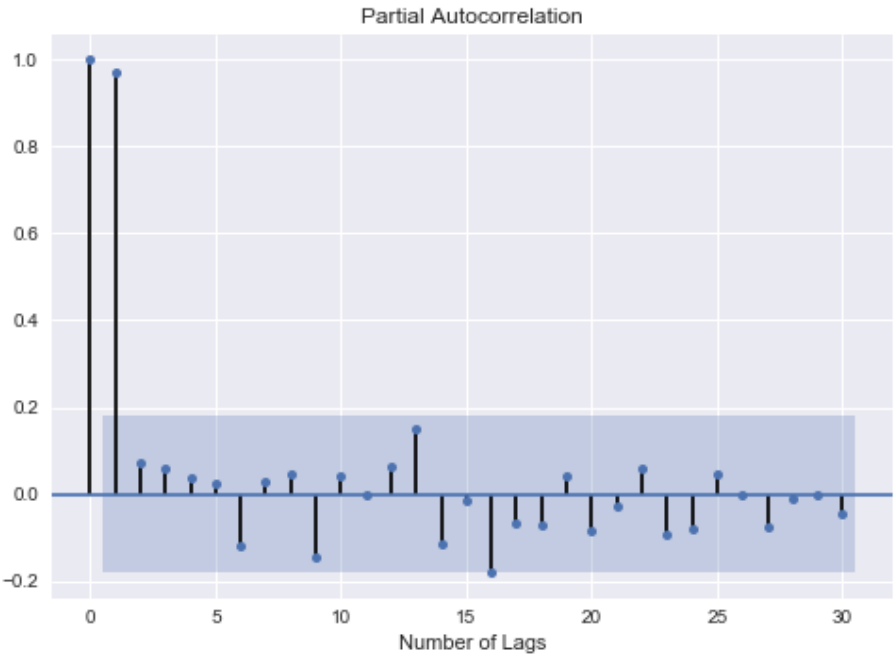


Figure 2-5. Partial Autocorrelation Function Plot of S&P-500

As discussed, PACF measures the correlation between current values of series and lagged values of it in a way to isolate in between effects.

Seasonality

Seasonality exists if there are regular fluctuations over a given period of time. For instance, energy usages can show seasonality characteristic. To be more specific, energy usage goes up and down in certain periods over a year.

To show how we can detect seasonality component, let us use FRED database, which includes more than 500,000 economic data series from over 80 sources covering issues and information relating to many areas such as banking, employment, exchange rates, gross domestic product, interest rates, trade and international transactions, and so on.

```
In [10]: from fredapi import Fred
         import statsmodels.api as sm

In [11]: fred = Fred(api_key='78b14ec6ba46f484b94db43694468bb1')#insert you api key

In [12]: energy = fred.get_series("CAPUTLG2211A2S",
                                  observation_start="2010-01-01",
```

```
                                    observation_end="2020-12-31")❶
         energy.head(12)
Out[12]: 2010-01-01    83.7028
         2010-02-01    84.9324
         2010-03-01    82.0379
         2010-04-01    79.5073
         2010-05-01    82.8055
         2010-06-01    84.4108
         2010-07-01    83.6338
         2010-08-01    83.7961
         2010-09-01    83.7459
         2010-10-01    80.8892
         2010-11-01    81.7758
         2010-12-01    85.9894
         dtype: float64

In [13]: plt.plot(energy)
         plt.title('Energy Capacity Utilization')
         plt.ylabel('$')
         plt.xlabel('Date')
         plt.savefig('images/energy.png')
         plt.show()
In [14]: sm.graphics.tsa.plot_acf(energy,lags=30)
         plt.xlabel('Number of Lags')
         plt.savefig('images/energy_acf.png')
         plt.show()
```

❶ Accessing the energy capacity utilization from Fred database for the period of 2010-2020.

Figure 2-6 indicates a periodic ups and downs over nearly 10-year period as to have high capacity utilization in the first months of every year and then goes down towards the end of year confirming that there is a seasonality in energy capacity utilization.

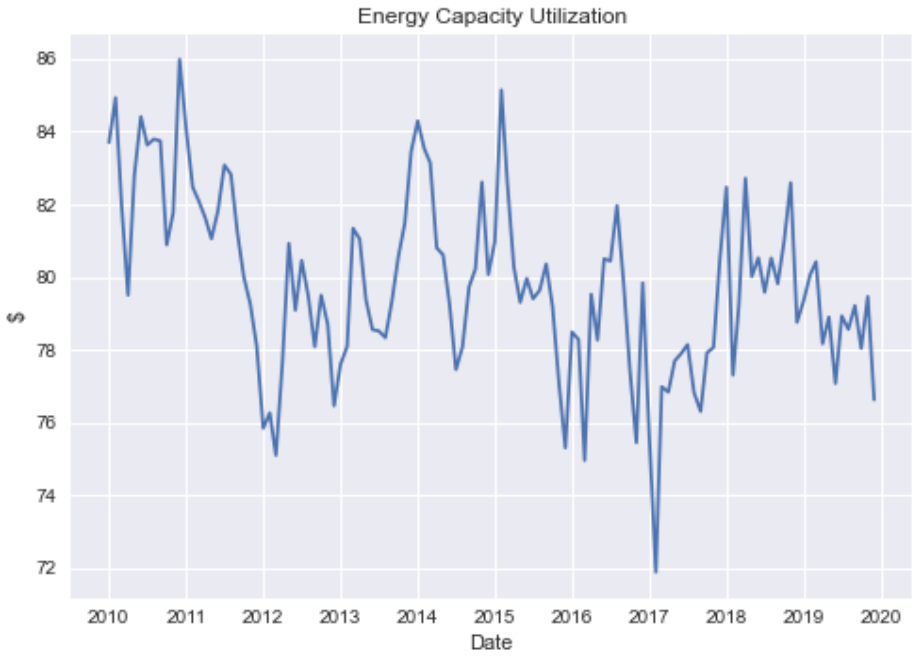


Figure 2-6. Seasonality in Energy Capacity Utilization

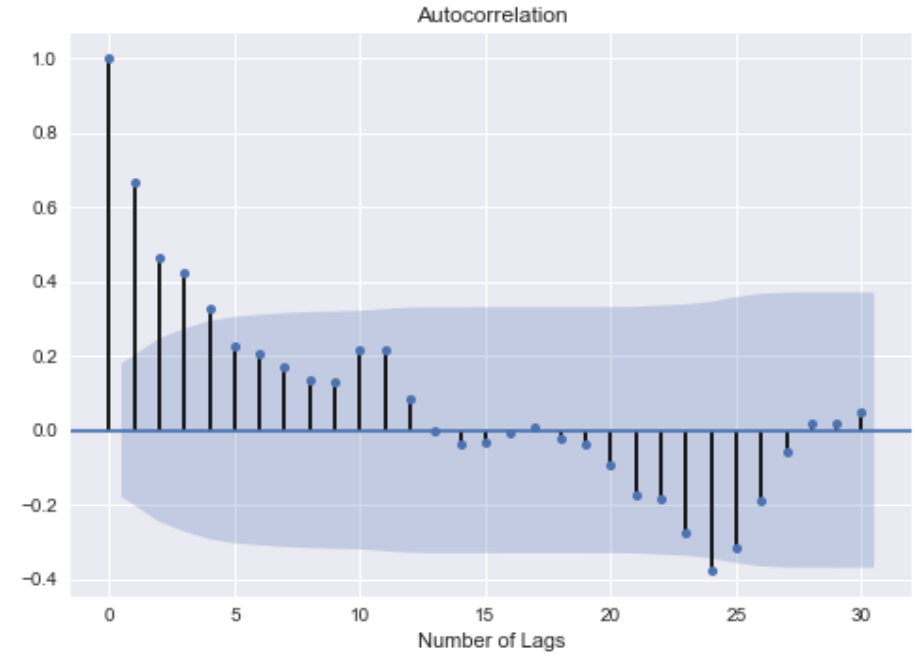


Figure 2-7. ACF of Energy Capacity Utilization

Cyclicality

What if data does not show fixed period movements? At this point, cyclicality comes into the picture. It exists when higher periodic variation than the trend emerges. Some confuse cyclicality and seasonality in a sense that they both exhibit expansion and contraction. We can, however, think of cyclicality as business cycles, which takes a long time to complete its cycle and the ups and downs over a long horizon. So, cyclicality is different from seasonality in the sense that there is no fluctuation in a fixed period. An example for cyclicality may be the house purchases (or sales) depending on mortgage rate. That is, when a mortgage rate is cut (or raise), it leads to a boost for house purchase (or sales).

Residual

Residual is known as irregular component of time series. Technically speaking, residual is equal to the difference between observations and related fitted values. So, we can think of it as a left over from the model.

As we have discussed before time series models lack in some core assumptions but it does not necessarily mean that time series models are free from assumptions. I would like to stress the most prominents one, which is called *stationarity*.

Stationarity means that statistical properties such as mean, variance, and covariance of the time series do not change over time.

There are two forms of stationarity:

1) Weak Stationarity: Time series, X_t, is said to be stationarity if

- X_t has finite variance, $\mathbb{E}\left(X_t^2\right) < \infty, \forall t \in \mathbb{Z}$
- Mean value of X_t is constant and does solely depend on time, $\mathbb{E}(X_t) = \mu, t\forall \in \mathbb{Z}$
- Covariance structure, $\gamma(t, t+h)$, depends on the time difference only:

$\gamma(h) = \gamma_h + \gamma(t+h, t)$

In words, time series should have finite variance with constant mean, and covariance structure that is a function of time difference.

2) Strong Stationarity: If the joint distribution of $X_{t1}, X_{t2}, \ldots X_{tk}$ is the same with the shifted version of set $X_{t1+h}, X_{t2+h}, \ldots X_{tk+h}$, it is referred to as strong stationarity. Thus, strong stationarity implies that distribution of random variable of a random process is the same with shifting time index.

The question is now why do we need stationarity? The reason is two fold.

First, in the estimation process, it is essential to have some distribution as time goes on. In other words, if distribution of a time series change over time, it becomes unpredictable and cannot be modeled.

The ultimate aim of time series models is forecasting. To do that we should estimate the coefficients first, which corresponds to learning in Machine Learning. Once we learn and conduct forecasting analysis, we assume that the distribution of the data in estimation stays the same in a way that we have the same estimated coefficients. If this is not the case, we should re-estimate the coefficients because we are unable to to forecast with the previous estimated coefficients.

Having structural break such as financial crisis generates a shift in distribution. We need to take care of this period cautiously and separately.

The other reason of having stationarity data is, by assumption, some statistical models require stationarity data, but it does not mean that some models requires stationary only. Instead, all models require stationary but event if you feed the model witn non-stationary data, some model, by design, turn it into stationary one and process it. Built-in functions in Python facilitates this analysis as follows:

```
In [15]: stat_test = adfuller(SP_prices)[0:2]❶
         print("The test statistic and p-value of ADF test are {}".format(stat_test))❷
         The test statistic and p-value of ADF test are (0.030295120072926063,
          0.9609669053518538)
```

❶ ADF test for stationarity

❷ Printing the first four decimals test statistic and p-value of ADF test

Figure 2-4 below shows the slow decaying lags amounting to non-stationary because persistence of the high correlation between lag of the time series continues.

There are, by and large, two ways to understand the non-stationarity: Visualization and statistical method. The latter, of course, has better and robust way of detecting the non-stationarity. However, to improve our understanding, let's start with the ACF. Slow-decaying ACF implies that the data is non-stationary because it presents a strong correlation in time. That is what I observe in S&P-500 data.

The statistical way of detecting non-stationarity is more reliable and *ADF* test is widely appreciated test. According to this test result given above, the p-value of 0.99 suggests that there is non-stationarity in the data, which we need to deal with before moving forward.

Taking difference is an efficient technique to remove the stationarity. Taking difference is nothing but to subtract current value of series from its first lagged value, i.e., $x_t - x_{t-1}$ and the following python code presents how to apply this technique:

```
In [16]: diff_SP_price = SP_prices.diff()❶

In [17]: plt.figure(figsize=(10, 6))
         plt.plot(diff_SP_price)
         plt.title('Differenced S&P-500 Price')
         plt.ylabel('$')
         plt.xlabel('Date')
         plt.savefig('images/diff_SP_price.png')
         plt.show()
In [18]: sm.graphics.tsa.plot_acf(diff_SP_price.dropna(),lags=30)
         plt.xlabel('Number of Lags')
         plt.savefig('images/diff_acf_SP.png')
         plt.show()
In [19]: stat_test2=adfuller(diff_SP_price.dropna())[0:2]❷
         print("The test statistic and p-value of ADF test after differencing are {}"\
               .format(stat_test2))
         The test statistic and p-value of ADF test after differencing are
          (-7.0951058730170855, 4.3095548146405375e-10)
```

❶ Taking difference of S&P-500 prices

❷ ADF test result based on differenced S&P-500 data

After taking first difference, we re-run the ADF test to see if it worked and yes it does. The very low p-value of ADF tells me that S&P-500 data is stationary now.

This can be observable from the line plot provided below. Unlike the raw S&P-500 plot, Figure 2-8 exhibits fluctuations around the mean with similar volatility meaning that we have a stationary series.

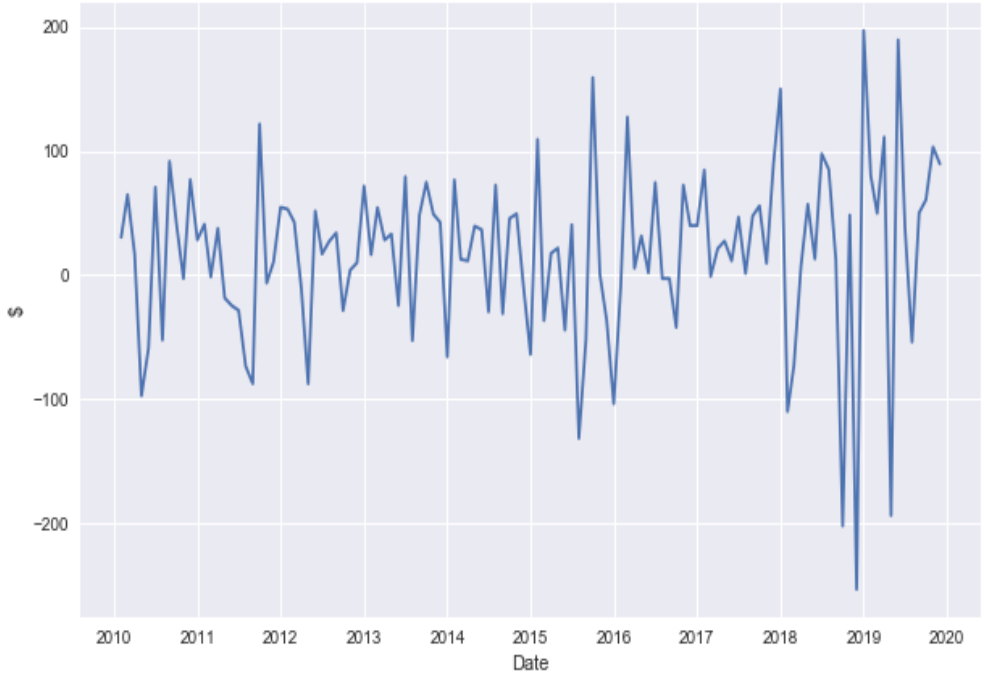


Figure 2-8. Detrended S&P-500 Price

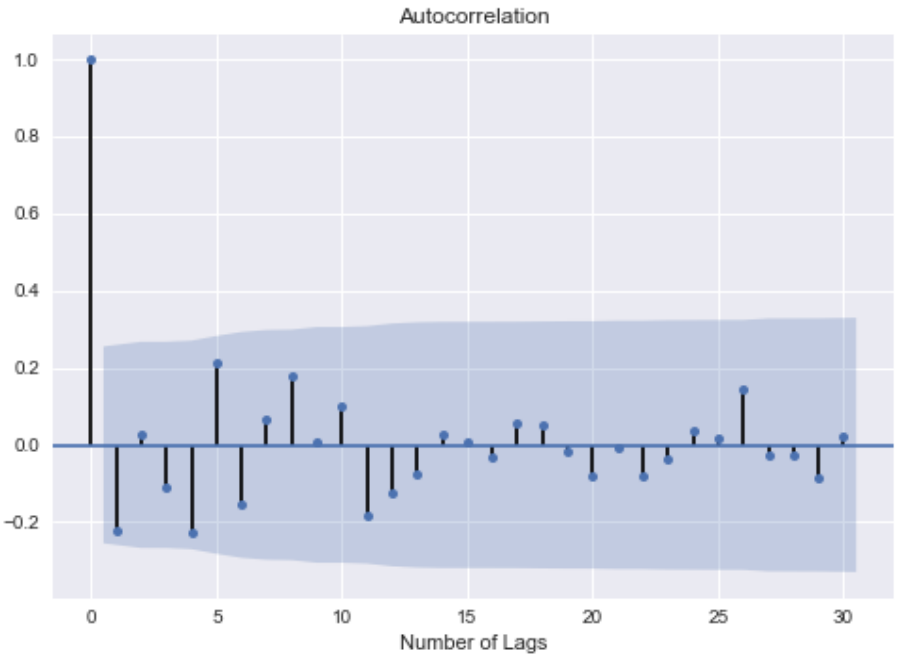


Figure 2-9. Detrended S&P-500 Price

Needless to say, trend is not the only indicator of non-stationarity. Seasonality is another source of it and now we are about to learn a method to deal with it.

First, let us observe the ACF of energy capacity utilization Figure 2-7 showing periodic ups and downs, which is a sign of non-stationarity.

In order to get rid of seasonality, we first apply *resample* method to calculate annual mean, which is used as denominator in the following formula:

$$\text{Seasonal Index} = \frac{\text{Value of a Seasonal Time Series}}{\text{Seasonal Average}}$$

Thus, the result of the application, *seasonal index*, gives us the de-seasonalized time series. The following code shows us how we code this formula in Python.

```
In [20]: seasonal_index = energy.resample('Q').mean()❶

In [21]: dates = energy.index.year.unique()❷
         deseasonalized = []
         for i in dates:
             for j in range(1, 13):
                 deseasonalized.append((energy[str(i)][energy[str(i)].index.month==j]))❸
         concat_deseasonalized = np.concatenate(deseasonalized)❹
```

```
In [22]: deseason_energy = []
         for i,s in zip(range(0, len(energy), 3), range(len(seasonal_index))):
             deseason_energy.append(concat_deseasonalized[i:i+3] /
seasonal_index.iloc[s])❺
         concat_deseason_energy = np.concatenate(deseason_energy)
         deseason_energy = pd.DataFrame(concat_deseason_energy, index=energy.index)
         deseason_energy.columns = ['Deaseasonalized Energy']
         deseason_energy.head()
Out[22]:            Deaseasonalized Energy
         2010-01-01                1.001737
         2010-02-01                1.016452
         2010-03-01                0.981811
         2010-04-01                0.966758
         2010-05-01                1.006862

In [23]: sm.graphics.tsa.plot_acf(deseason_energy, lags=10)
         plt.xlabel('Number of Lags')
         plt.savefig('images/deseason_energy_acf.png')
         plt.show()
In [24]: sm.graphics.tsa.plot_pacf(deseason_energy, lags=10)
         plt.xlabel('Number of Lags')
         plt.savefig('images/deseason_energy_pacf.png')
         plt.show()
```

❶ Calculation quarterly mean of energy utilization.

❷ Defining the years in which seasonality analysis is run.

❸ Computing the numerator of *Seasonal Index* formula.

❹ Concatenating the de-seasonalized energy utilization.

❺ Computing *Seasonal Index* using pre-defined formula.

Figure 2-10 suggests that there is a statistically significant correlation at lag 1 and 2 but ACF does not show any periodic characteristics, which is another way of saying de-seasonalization.

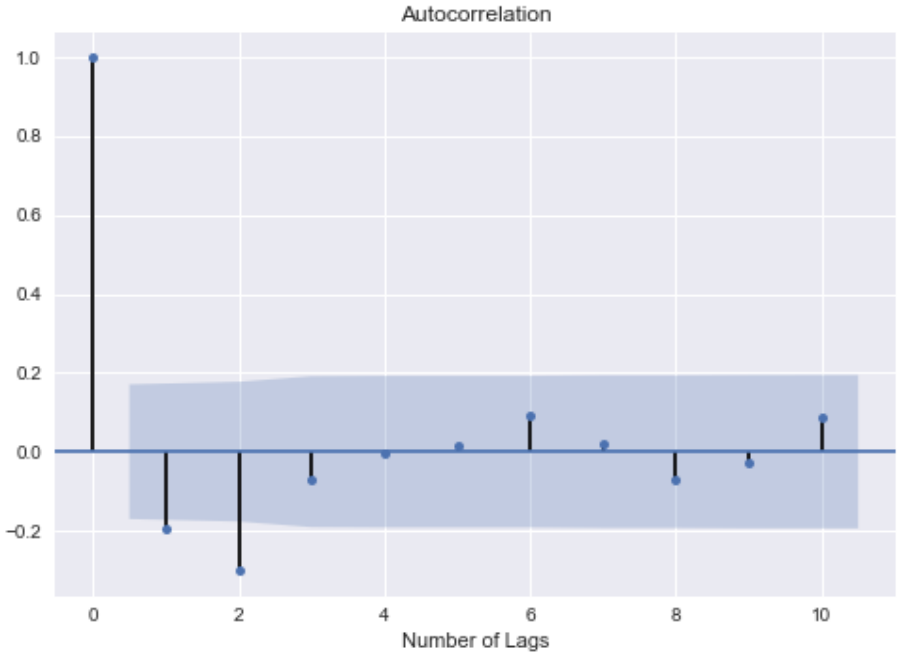


Figure 2-10. De-seasonalized ACF of Energy Utilization

Similarly, in the Figure 2-11, though we have spike at some lags, PACF do not show any periodic ups and downs. So, we can say that the data is de-seasonalized using seasonal index formula.

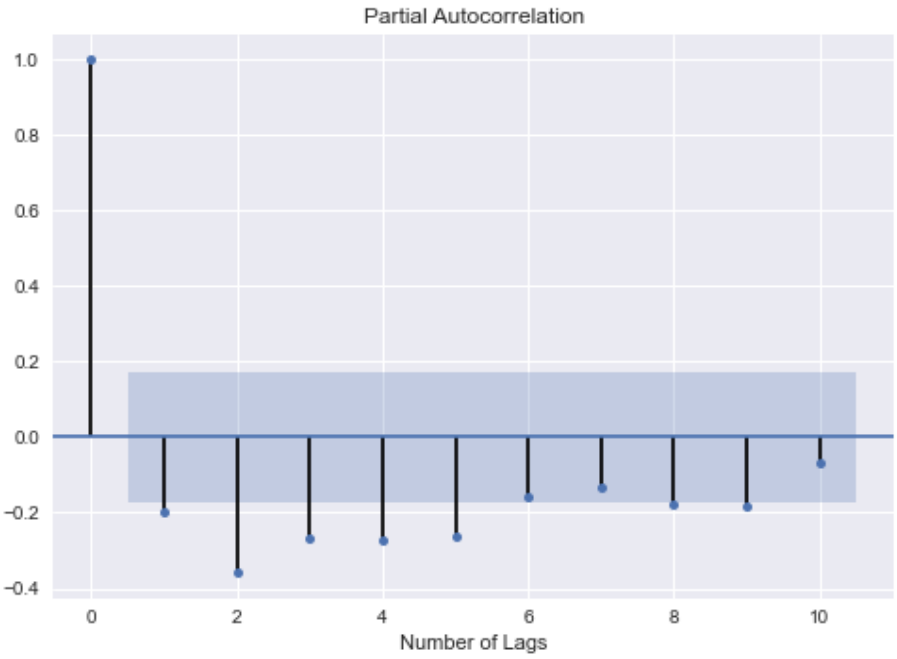


Figure 2-11. De-seasonalized PACF of Energy Utilization

What we have now is the less periodic ups and down in energy capacity utilization, meaning that the data turns out to be de-seasonalized.

Finally, we are ready to move forward and discuss the time series models.

Time Series Models

Traditional time series models are univariate models and they follow the following phases:

- Identification: In this process, we explore the data using ACF, PACF, identifying pattern, and conducting statistical tests.
- Estimation: This part belongs to estimation coefficients via proper optimization technique.
- Diagnostics: Having estimated, we need to check if information criteria or ACF/PACF suggest that the model is valid. If so, we move on to forecasting stage.

- Forecast: This part is more about the performance of the model. In forecasting, we predict future values based on our estimation.

Figure 2-12 indicates the modeling process. Accordingly, subsequent to identification of the variables and the estimation process, model is run. Only after running a proper diagnostics, we are able to perform the forecast analysis.

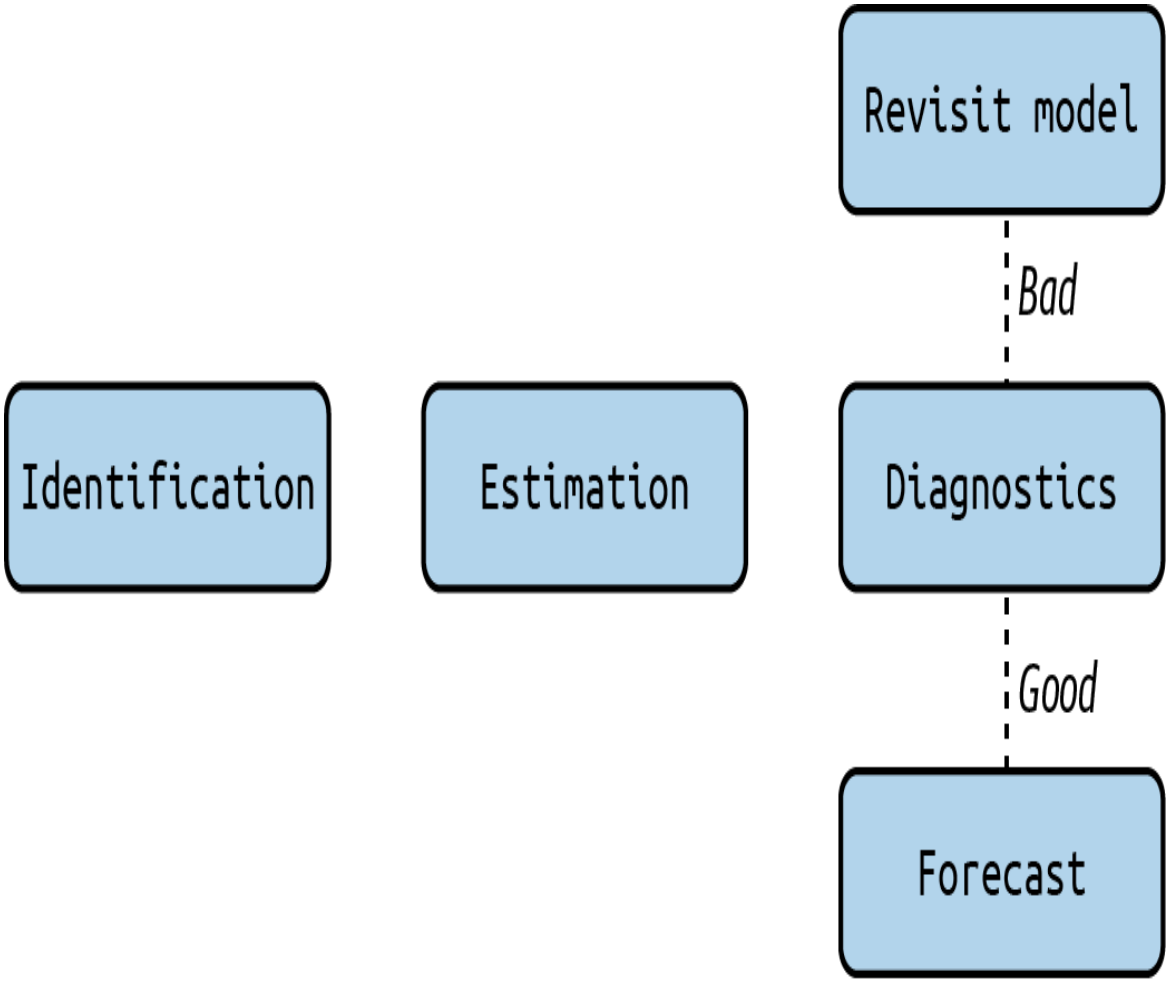


Figure 2-12. Modeling Process

In modeling a data with a time dimension, we should consider correlation in adjacent points in time. This consideration takes us to time series modeling as we discuss before. My aim, in modeling time series, is to fit a model and comprehend statistical character of a time series, which fluctuates randomly in time.

Recall the discussion about the IID process, which is the most basic time series model and is sometimes referred to as white noise. Let's touch the concept of white noise.

White Noise

The time series ϵ_t is said to be white noise if it satisfies the following:

$$\epsilon_t \sim WN\left(0, \sigma_\epsilon^2\right)$$

$$\text{Corr}(\epsilon_t, \epsilon_s) = 0, \forall t \neq s$$

In words, ϵ_t has mean of 0 and constant variance. Moreover, there is no correlation between successive terms of ϵ_t.

Well, it is easy to say that white noise process is stationary and plot of white noise exhibits fluctuations around mean in a random fashion in time.

However, as the white noise is formed by uncorrelated sequence, it is not an appealing model from forecasting standpoint. Differently, uncorrelation prevents us to forecast future values.

As we can observe from the Figure Figure 2-13 below, white noise oscillates around mean and it is completely erratic.

```
In [25]: mu = 0
         std = 1
         WN = np.random.normal(mu, std, 1000)

         plt.plot(WN)
         plt.xlabel('Number of Simulation')
         plt.savefig('images/WN.png')
         plt.show()
```

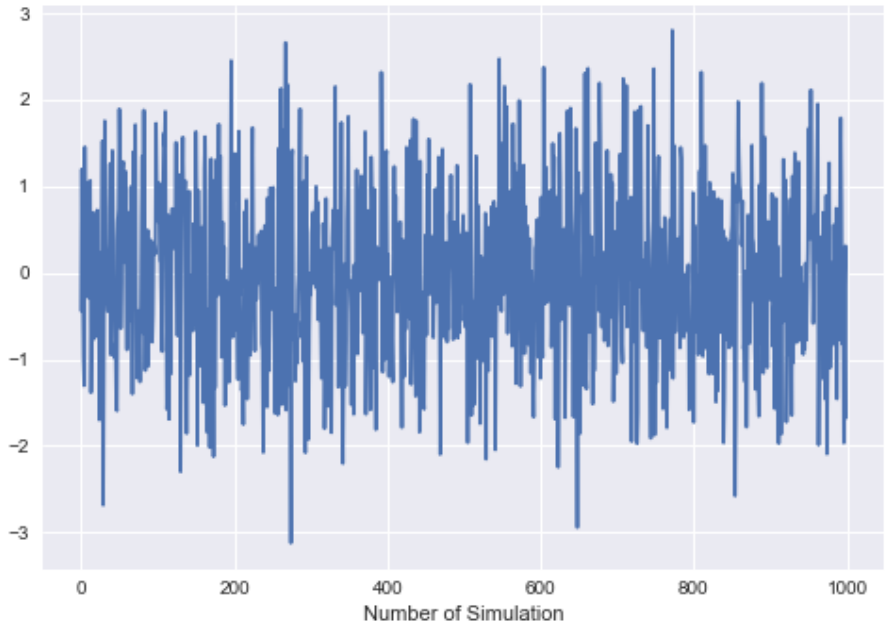


Figure 2-13. White Noise Process

From this point on, we need to identify the optimum number of lag before running the time series model. As you can imagine deciding the optimal number of lag is a challenging task. The most widely used methods are: *ACF*, *PACF*, and *information criteria*. ACF and PACF have been discussed and, for the sake of comprehensiveness, information criteria, AIC for short, will also be introduced.

INFORMATION CRITERIA

Detecting the optimal number of lag is a cumbersome task. We need to have a criteria to decide which model fits best to the data as there may be numerous potentially good models. Akaike Information Criteria, a.k.a AIC, as Cavanaugh and Neath (2019) denote that

> *AIC is introduced as an extension to the Maximum Likelihood Principle. Maximum likelihood is conventionally applied to estimate the parameters of a model once structure and dimension of the model have been formulated.*

AIC can be mathematically defined as:

$$AIC = -2ln(MaximumLikelihood) + 2d$$

where d is the total number of parameters. The last term, 2d in the equation aims at reducing the risk of overfitting. It is also called as *penalty term* by which I can filter out the unnecessary redundancy in the model.

BIC is the other information criteria used to select best model. The penalty term in BIC is larger than that of AIC.

$$BIC = -2ln(MaximumLikelihood) + ln(n) * d$$

where n is the number of observations.

Please note that you need to treat the AIC with caution if the proposed model is finite dimensional. This fact is well put by Clifford and Hurvich (1989):

> *If the true model is infinite dimensional, a case which seems most realistic in practice, AIC provides an asymptotically efficient selection of a finite dimensional approximating model. If the true model is finite dimensional, however, the asymptotically efficient methods, e.g., Akaike's FPE (Akaike, 1970), AIC, and Parzen's CAT (Parzen, 1977), do not provide consistent model order selections.*

Let's get started to visit classical time series model with Moving Average model.

Moving Average Model

Moving average (MA) and residuals are closely related models. Moving average can be considered as smoothing model as it tends to take into account the lag values of residual. For the sake of simplicity, let us start with MA(1):

$$X_t = \epsilon_t + \alpha\epsilon_{t-1}$$

As long as $\alpha \neq 0$, it has nontrivial correlation structure. Intuitively, MA(1) tells us that the time series has been affected by ϵ_t and ϵ_{t-1} only.

In general form, MA(q) equation becomes:

$$X_t = \epsilon_t + \alpha_1\epsilon_{t-1} + \alpha_2\epsilon_{t-2}... + \alpha_q\epsilon_{t-q}$$

From this point on, to be consistent, we will model the data of two major IT companies, namely *Apple* and *Microsoft*. *Yahoo Finance* provides a convenient tool to access closing price of the related stocks for the period of 01-01-2019 and 01-01-2021.

As a first step, we dropped the missing values and check if the data is stationary and it turns out neither Apple nor Microsoft stock prices have stationary structure as expected. Thus, taking first difference to make these data stationary and splitting the data as `train` and `test` are the steps to be taken at this point. The following code shows us how we can do these in Python.

```
In [26]: ticker = ['AAPL', 'MSFT']
         start = datetime.datetime(2019, 1, 1)
         end = datetime.datetime(2021, 1, 1)
         stock_prices = yf.download(ticker, start=start, end = end, interval='1d').Close❶
         [*********************100%***********************]  2 of 2 completed

In [27]: stock_prices = stock_prices.dropna()

In [28]: for i in ticker:
             stat_test = adfuller(stock_prices[i])[0:2]
             print("The ADF test statistic and p-value of {} are
{}".format(i,stat_test))
         The ADF test statistic and p-value of AAPL are  (0.29788764759932335,
          0.9772473651259085)
         The ADF test statistic and p-value of MSFT are  (-0.8345360070598484,
          0.8087663305296826)

In [29]: diff_stock_prices = stock_prices.diff().dropna()

In [30]: split = int(len(diff_stock_prices['AAPL'].values) * 0.95)❷
```

```
        diff_train_aapl = diff_stock_prices['AAPL'].iloc[:split]❸
        diff_test_aapl = diff_stock_prices['AAPL'].iloc[split:]❹
        diff_train_msft = diff_stock_prices['MSFT'].iloc[:split]❺
        diff_test_msft = diff_stock_prices['MSFT'].iloc[split:]❻

In [31]: diff_train_aapl.to_csv('diff_train_aapl.csv')❼
        diff_test_aapl.to_csv('diff_test_aapl.csv')
        diff_train_msft.to_csv('diff_train_msft.csv')
        diff_test_msft.to_csv('diff_test_msft.csv')

In [32]: fig, ax = plt.subplots(2, 1, figsize=(10, 6))
        plt.tight_layout()
        sm.graphics.tsa.plot_acf(diff_train_aapl,lags=30, ax=ax[0], title='ACF - Apple')
        sm.graphics.tsa.plot_acf(diff_train_msft,lags=30, ax=ax[1], title='ACF -
Microsoft')
        plt.savefig('images/acf_ma.png')
        plt.show()
```

❶ Retrieving monthly closing stock prices.

❷ Splitting data as 95% and 5%.

❸ Assigning 95% of the Apple stock price data to the train set.

❹ Assigning 5% of the Apple stock price data to the test set.

❺ Assigning 95% of the Microsoft stock price data to the train set.

❻ Assigning 5% of the Microsoft stock price data to the test set.

❼ Saving the data for future use.

Looking at the first panel of Figure 2-14, it can be observed that there is a significant spikes at some lags and, to be on the safe side, we choose lag 9 for short moving average model and 22 for long moving average. These implies that order of 9 will be our short-term order and 22 will become our long-term order in modeling MA.

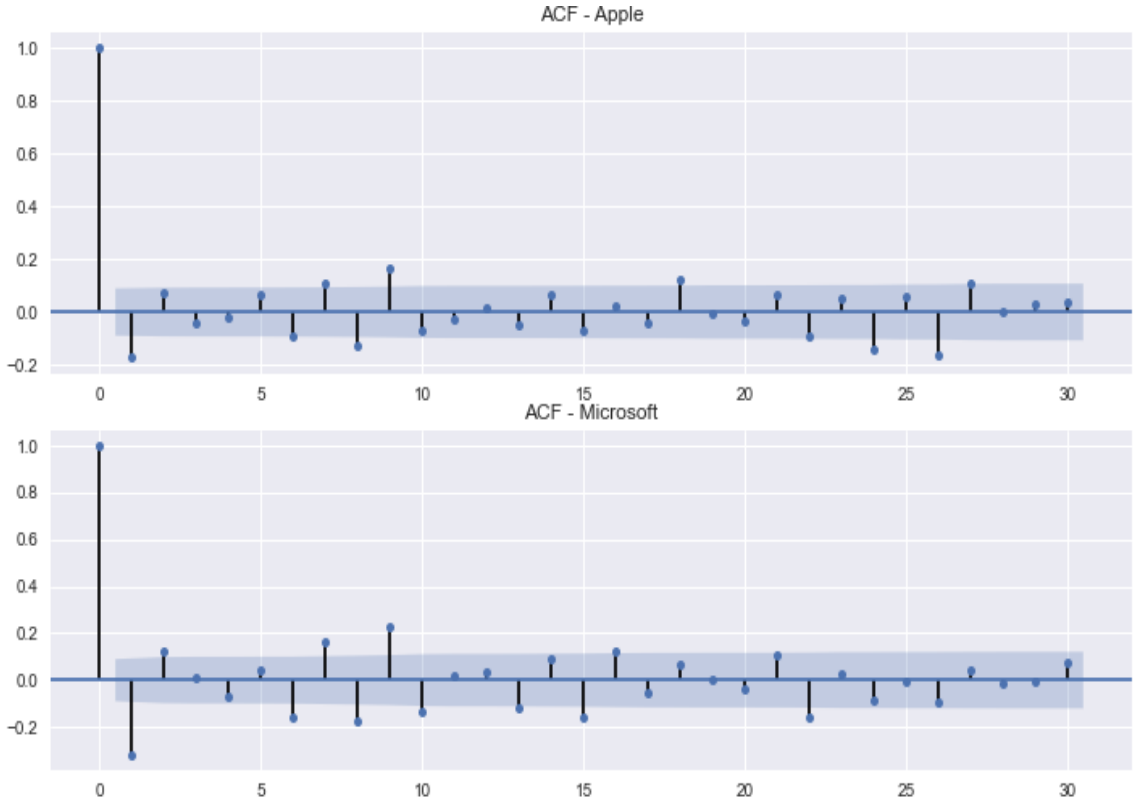


Figure 2-14. ACF After First Difference

```
In [33]: short_moving_average_appl = diff_train_aapl.rolling(window=9).mean()❶
         long_moving_average_appl = diff_train_aapl.rolling(window=22).mean()❷

In [34]: fig, ax = plt.subplots(figsize=(10, 6))
         ax.plot(diff_train_aapl.loc[start:end].index,
                 diff_train_aapl.loc[start:end],
                 label='Stock Price', c='b')❸
         ax.plot(short_moving_average_appl.loc[start:end].index,
                 short_moving_average_appl.loc[start:end],
                 label = 'Short MA',c='g')❹
         ax.plot(long_moving_average_appl.loc[start:end].index,
                 long_moving_average_appl.loc[start:end],
                 label = 'Long MA',c='r')❺
         ax.legend(loc='best')
         ax.set_ylabel('Price in $')
         ax.set_title('Stock Prediction-Apple')
         plt.savefig('images/ma_apple.png')
         plt.show()
```

❶ Moving average with short window for Apple stock

❷ Moving average with long window for Apple stock

❸ Line plot of first differenced Apple stock prices

❹ Visualization of short window Moving Average result for Apple

❺ Visualization of long window Moving Average result for Apple

Figure 2-15 exhibits the short-term moving average model result with green color and long-term moving average model result with red color. As expected, it turns out that short-term moving average tends to more reactive to daily Apple stock price change compared to long-term moving average. It makes sense because taking into account a long moving average generates smooth prediction.

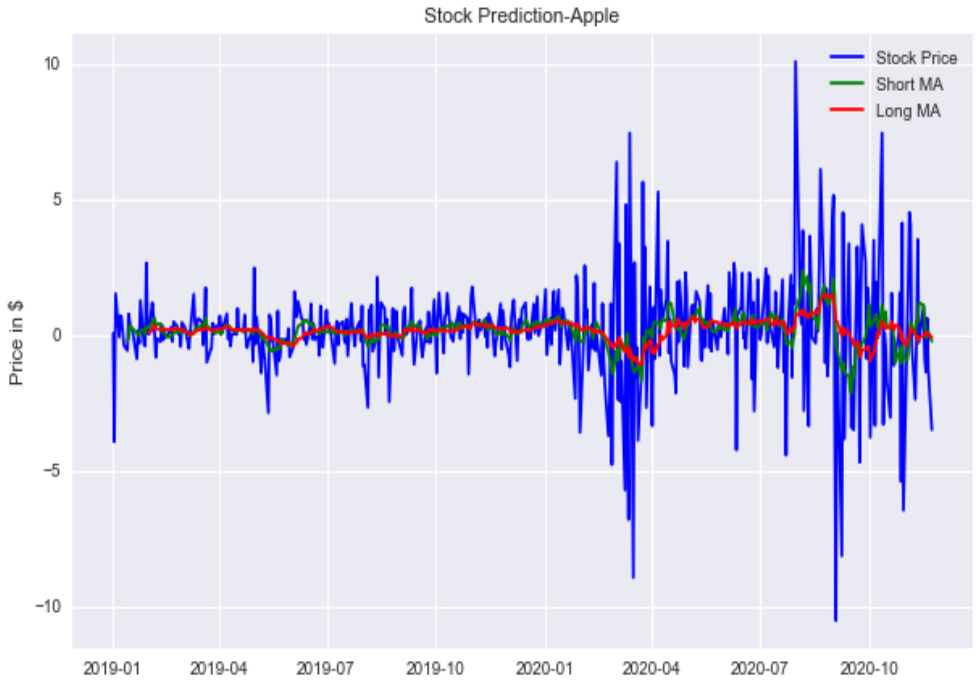


Figure 2-15. Moving Average Model Prediction Result for Apple

In the next step, we try to predict Microsoft stock price using moving average model with different window. But before proceeding, let me say that choosing proper window for short and long moving average analysis is a key to good modeling. In the second panel of Figure 2-14, it seems to have significant spikes at 2 and 23 and these lags are used in short and long moving average analysis,

respectively. After identifying the window, let us fit data to moving average model with the following application.

```
In [35]: short_moving_average_msft = diff_train_msft.rolling(window=2).mean()
         long_moving_average_msft = diff_train_msft.rolling(window=23).mean()

In [36]: fig, ax = plt.subplots(figsize=(10, 6))
         ax.plot(diff_train_msft.loc[start:end].index,
                 diff_train_msft.loc[start:end],
                 label='Stock Price',c='b')
         ax.plot(short_moving_average_msft.loc[start:end].index,
                 short_moving_average_msft.loc[start:end],
                 label = 'Short MA',c='g')
         ax.plot(long_moving_average_msft.loc[start:end].index,
                 long_moving_average_msft.loc[start:end],
                 label = 'Long MA',c='r')
         ax.legend(loc='best')
         ax.set_ylabel('$')
         ax.set_xlabel('Date')
         ax.set_title('Stock Prediction-Microsoft')
         plt.savefig('images/ma_msft.png')
         plt.show()
```

Similarly, predictions based on short moving average analysis tends to be more reactive than those of long moving average model in Figure 2-16. But, in Microsoft case, the short-term moving average prediction appears to be very close to the real data.

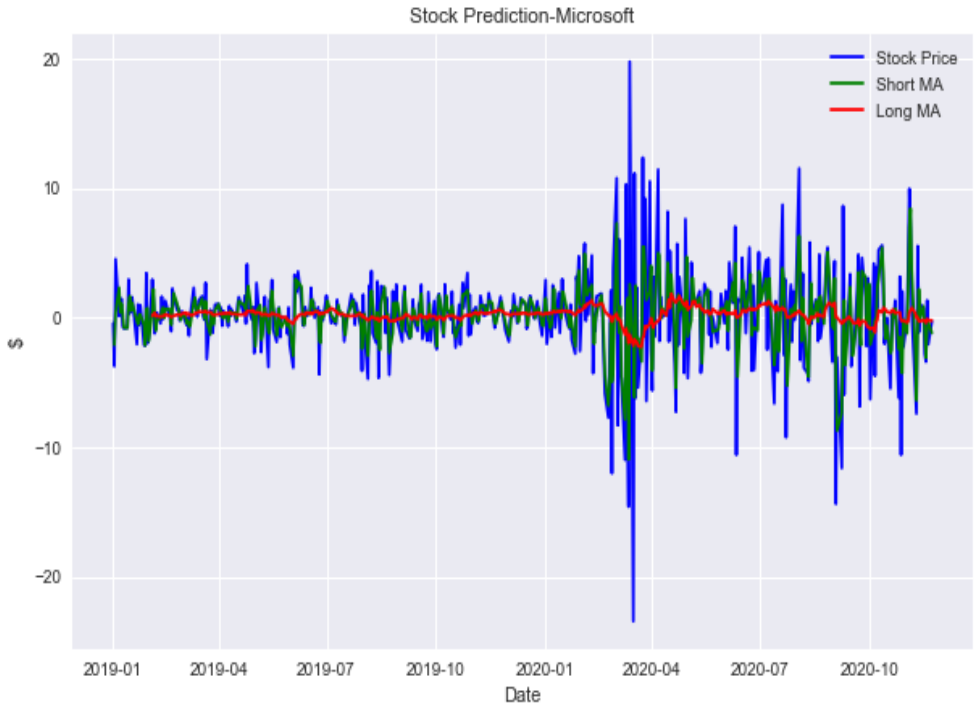


Figure 2-16. Moving Average Model Prediction Result for Microsoft

Autoregressive Model

Dependence structure of succesive terms is the most distinctive feature of the autoregressive model in the sense that current value is regressed over its own lag values in this model. So, we basically forecast the current value of the time series X_t by using a linear combination of its past values. Mathematically, the general form of AR(p) can be written as:

$$X_t = c + \alpha_1 X_{t-1} + \alpha_2 X_{t-2} ... + \alpha_p X_{t-p} + \epsilon_t$$

where ϵ_t denotes the residuals and c is the intercet term. AR(p) model implies that past values up to order p have somewhat explanatory power on X_t. If the relationship has shorter memory, then it is likely to model X_t with less number of lags.

We have discussed the one of the main properties in time series, which is *stationarity* and the other important property is *invertibility*. After introducing

AR model, it is time to show the invertibility of the MA process. It is said to be invertible if it can be converted to infinite AR model.

Differently, under some circumstances, MA can be written as an infinite AR process. These circumstances are to have stationary covariance structure, deterministic part, and invertible MA process. In doing so, we have another model called *infinite AR* thanks to the assumption of $|\alpha| < 1$.

$$X_t = \epsilon_t + \alpha\epsilon_{t-1}$$

$$= \epsilon_t + \alpha(X_{t-1} - \alpha\epsilon_{t-2})$$

$$= \epsilon_t + \alpha X_{t-1} - \alpha^2\epsilon_{t-2}$$

$$= \epsilon_t + \alpha X_{t-1} - \alpha^2(X_{t-2} + \alpha\epsilon_{t-3})$$

$$= \epsilon_t + \alpha X_{t-1} - \alpha^2 X_{t-2} + \alpha^3\epsilon_{t-3})$$

$$= ...$$

$$= \alpha X_{t-1} - \alpha^2 X_{t-2} + \alpha^3\epsilon_{t-3} - \alpha^4\epsilon_{t-4} + ... - (-\alpha)^n\epsilon_{t-n}$$

After doing necessary math equation gets the following form:

$$\alpha^n\epsilon_{t-n} = \epsilon_t - \sum_{i=0}^{n-1}\alpha^i X_{t-i}$$

In this case, if $|\alpha| < 1$. Then $n \to \infty$

$$\mathbb{E}\left(\epsilon_t - \sum_{i=0}^{n-1}\alpha^i X_{t-i}\right)^2) = \mathbb{E}\big(\alpha^{2n}\epsilon_{t-n}^2 \to \infty\big)$$

Finally, MA(1) process turns out to be:

$$\epsilon_t = \sum_{i=0}^{\infty}\alpha^i X_{t-i}$$

Due to the duality between AR and MA processes, it is possible to represent AR(1) as infinite MA, MA(∞). In other words, the AR(1) process can be

expressed as a function of past values of innovations.

$$X_t = \epsilon_t + \theta X_{t-1}$$

$$= \theta(\theta X_{t-2} + \epsilon_{t-1}) + \epsilon_t$$

$$= \theta^2 X_{t-2} + \theta\epsilon_{t-1} + \epsilon_t$$

$$= \theta^2(\theta X_{t-3} + \theta\epsilon_{t-2})\theta\epsilon_{t-1} + \epsilon_t$$

$$X_t = \epsilon_t + \epsilon_{t-1} + \theta^2\epsilon_{t-2} + ... + \theta^t X_t$$

As $n \to \infty$, $\theta^t \to 0$, so I can represent AR(1) as an infite MA process.

In the following analysis, we run autoregressive model to predict Apple and Microsoft stock prices. Unlike moving average, partial autocorrelation function is a useful tool to find out the optimum order in autoregressive model. This is because, in AR, we aim at finding out the relationship of a time series between two different time, say X_t and X_{t-k} and to do that I need to filter out the effect of other lags in between.

```
In [37]: sm.graphics.tsa.plot_pacf(diff_train_aapl,lags=30)
         plt.title('PACF of Apple')
         plt.xlabel('Number of Lags')
         plt.savefig('images/pacf_ar_aapl.png')
         plt.show()
In [38]: sm.graphics.tsa.plot_pacf(diff_train_msft,lags=30)
         plt.title('PACF of Microsoft')
         plt.xlabel('Number of Lags')
         plt.savefig('images/pacf_ar_msft.png')
         plt.show()
```

Based on Figure 2-17, obtained from first differenced of Apple stock price, we observe a significant spike at lag 29 and Figure 2-18 exhibits that we have a similar spike at lag 23. Thus, 29 and 23 are the lags that we am going to use in modeling AR for Apple and Microsoft, respectively.

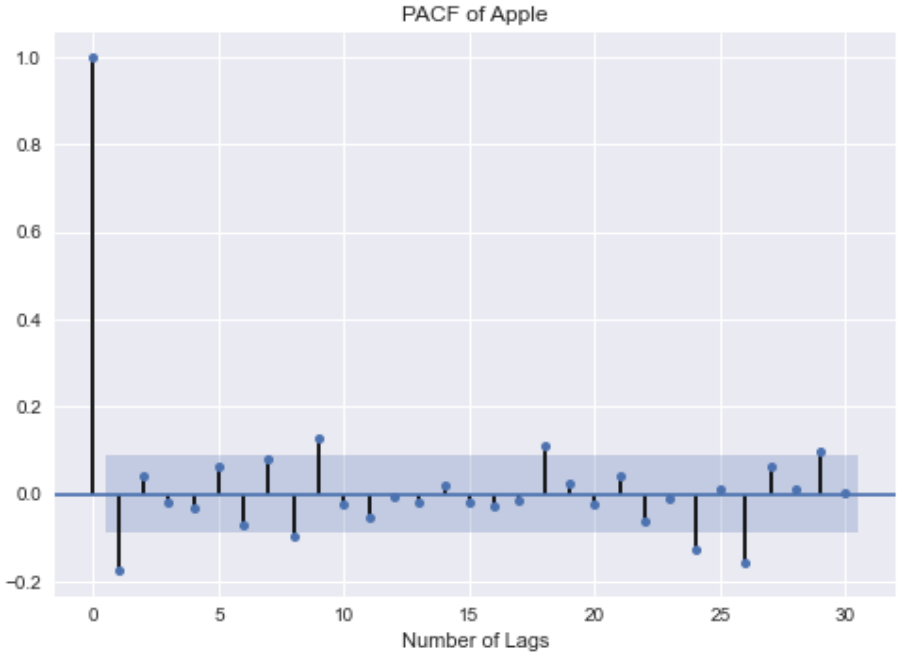


Figure 2-17. PACF for Apple

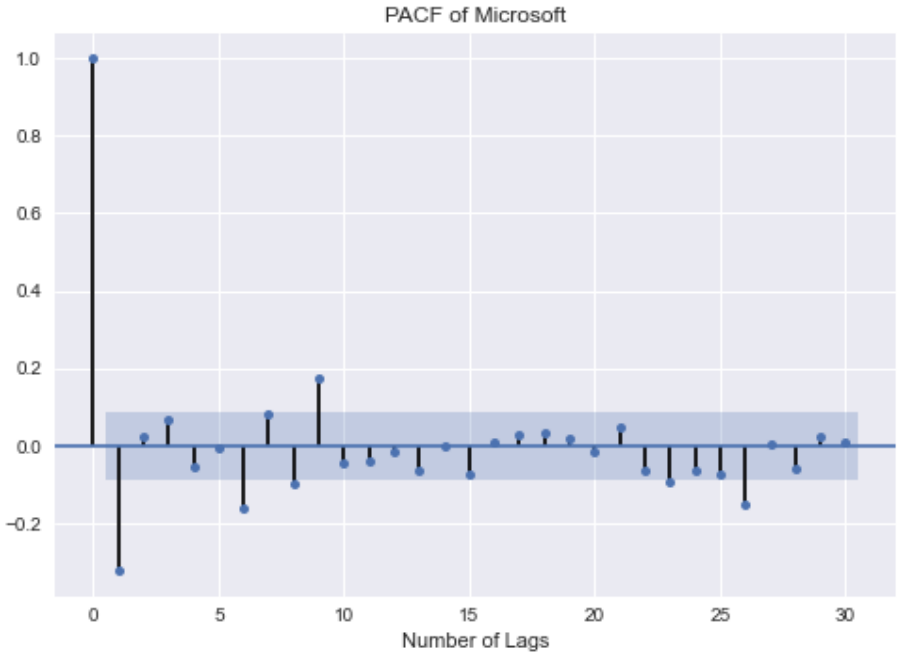


Figure 2-18. PACF for Microsoft

```
In [39]: from statsmodels.tsa.ar_model import AutoReg
         import warnings
         warnings.filterwarnings('ignore')

In [40]: ar_aapl = AutoReg(diff_train_aapl.values, lags=26)
         ar_fitted_aapl = ar_aapl.fit()❶

In [41]: ar_predictions_aapl = ar_fitted_aapl.predict(start=len(diff_train_aapl),
                                                       end=len(diff_train_aapl)\
                                                       + len(diff_test_aapl) - 1,
                                                       dynamic=False)❷

In [42]: for i in range(len(ar_predictions_aapl)):
             print('==' * 25)
             print('predicted values:{:.4f} & actual values:
{:.4f}'.format(ar_predictions_aapl[i],

diff_test_aapl[i]))❸
         ==================================================
         predicted values:1.9207 & actual values:1.3200
         ==================================================
         predicted values:-0.6051 & actual values:0.8600
         ==================================================
         predicted values:-0.5332 & actual values:0.5600
         ==================================================
         predicted values:1.2686 & actual values:2.4600
         ==================================================
         predicted values:-0.0181 & actual values:3.6700
         ==================================================
         predicted values:1.8889 & actual values:0.3600
         ==================================================
         predicted values:-0.6382 & actual values:-0.1400
         ==================================================
         predicted values:1.7444 & actual values:-0.6900
         ==================================================
         predicted values:-1.2651 & actual values:1.5000
         ==================================================
         predicted values:1.6208 & actual values:0.6300
         ==================================================
         predicted values:-0.4115 & actual values:-2.6000
         ==================================================
         predicted values:-0.7251 & actual values:1.4600
         ==================================================
         predicted values:0.4113 & actual values:-0.8300
         ==================================================
         predicted values:-0.9463 & actual values:-0.6300
         ==================================================
         predicted values:0.7367 & actual values:6.1000
         ==================================================
         predicted values:-0.0542 & actual values:-0.0700
         ==================================================
```

```
predicted values:0.1617 & actual values:0.8900
==================================================
predicted values:-1.0148 & actual values:-2.0400
==================================================
predicted values:0.5313 & actual values:1.5700
==================================================
predicted values:0.3299 & actual values:3.6500
==================================================
predicted values:-0.2970 & actual values:-0.9200
==================================================
predicted values:0.9842 & actual values:1.0100
==================================================
predicted values:0.3299 & actual values:4.7200
==================================================
predicted values:0.7565 & actual values:-1.8200
==================================================
predicted values:0.3012 & actual values:-1.1500
==================================================
predicted values:0.7847 & actual values:-1.0300
```

```
In [43]: ar_predictions_aapl = pd.DataFrame(ar_predictions_aapl)❹
         ar_predictions_aapl.index = diff_test_aapl.index❺

In [44]: ar_msft = AutoReg(diff_train_msft.values, lags=26)
         ar_fitted_msft = ar_msft.fit()❻

In [45]: ar_predictions_msft = ar_fitted_msft.predict(start=len(diff_train_msft),
                                                      end=len(diff_train_msft)\
                                                      +len(diff_test_msft) - 1,
                                                      dynamic=False)❼

In [46]: ar_predictions_msft = pd.DataFrame(ar_predictions_msft)❽
         ar_predictions_msft.index = diff_test_msft.index❾
```

❶ Fitting Apple stock data with AR model.

❷ Predicting the stock prices for Apple.

❸ Comparing the predicted and real observations.

❹ Turning array into dataframe to assign index.

❺ Assign test data indices to predicted values.

❻ Fitting Microsoft stock data with AR model.

❼ Predicting the stock prices for Microsoft.

❽ Turn array into dataframe to assign index.

❾ Assign test data indices to predicted values.

```
In [47]: fig, ax = plt.subplots(2,1, figsize=(18, 15))

         ax[0].plot(diff_test_aapl, label='Actual Stock Price', c='b')
         ax[0].plot(ar_predictions_aapl, c='r', label="Prediction")
         ax[0].set_title('Predicted Stock Price-Apple')
         ax[0].legend(loc='best')
         ax[1].plot(diff_test_msft, label='Actual Stock Price', c='b')
         ax[1].plot(ar_predictions_msft, c='r', label="Prediction")
         ax[1].set_title('Predicted Stock Price-Microsoft')
         ax[1].legend(loc='best')
         for ax in ax.flat:
             ax.set(xlabel='Date', ylabel='$')
         plt.savefig('images/ar.png')

         plt.show()
```

Figure 2-19 indicates the predictions based on AR model. Red lines represent the Apple and Microsoft stock price predictions and blue lines denote the real data. The result reveals that AR model is outperformed by MA model in capturing the stock price.

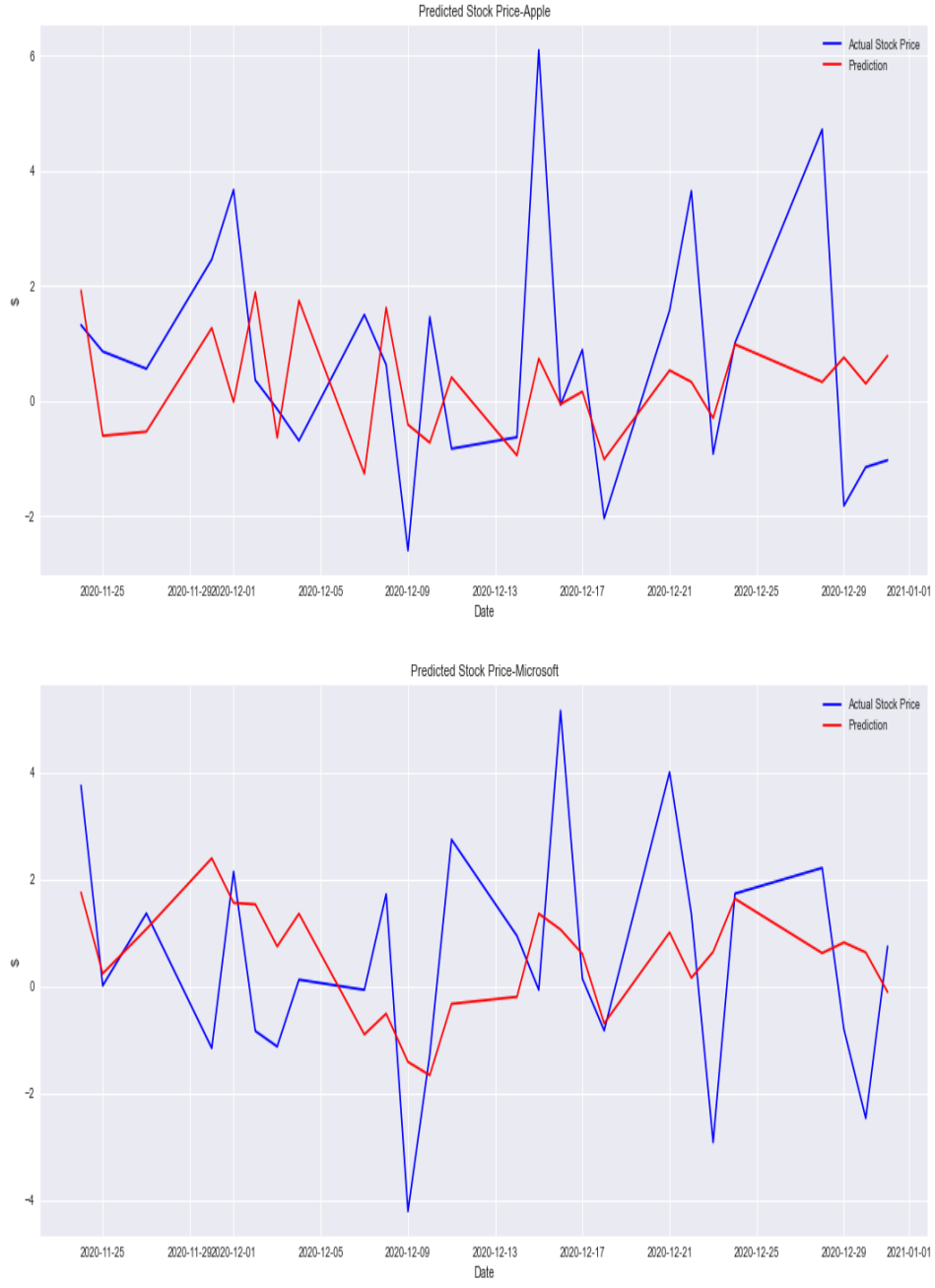


Figure 2-19. Autoregressive Model Prediction Results

Autoregressive Integrated Moving Average Model

Autoregressive Integrated Moving Average Model, ARIMA for short, is a function of past values of a time series and white noise. However, ARIMA is proposed as a generalization of AR and MA but they do not have intergration parameter, which helps us to feed model with the raw data. To this respect, even if we include non-stationary data, ARIMA makes it stationary by properly defining the integration parameter.

ARIMA has three parameters, namely p, d, q. As we are familiar from previous time series models p and q refer to order of AR and MA, respectively. But d controls for level difference. If d=1, it amounts to first difference and if it takes the value of 0, it means that the model is ARMA.

It is possible to have d greater than one but it is not as common as having d of 1. The ARIMA (p,1,q) equation has the following structure:

$$X_t = \alpha_1 dX_{t-1} + \alpha_2 dX_{t-2} ... + \alpha_p dX_{t-p} + \epsilon_t + \beta_1 d\epsilon_{t-1} + \beta_2 d\epsilon_{t-2} ... + \beta_q d\epsilon_{t-q}$$

where d refers to difference.

As it is widely embraced and applicable model, Let us discuss the pros and cons of the ARIMA model to get more familiar with the model.

Pros

- ARIMA allows us to work with raw data without considering if it is stationary.
- It performs well with high-frequent data.
- It is less sensitive to the fluctuation in the data compared to other models.

Cons

- ARIMA might fail in capturing seasonality.
- It work better with a long series and short-term (daily, hourly) data.
- As no automatic updating occurs in ARIMA, no structural break during the analysis period should be observed.
- Having no adjustment in the ARIMA process leads to instability.

Now, we will show how ARIMA works and performs using the same stocks, namely Apple and Microsoft.

```
In [48]: from statsmodels.tsa.arima_model import ARIMA

In [49]: split = int(len(stock_prices['AAPL'].values) * 0.95)
         train_aapl = stock_prices['AAPL'].iloc[:split]
         test_aapl = stock_prices['AAPL'].iloc[split:]
         train_msft = stock_prices['MSFT'].iloc[:split]
         test_msft = stock_prices['MSFT'].iloc[split:]

In [50]: arima_aapl = ARIMA(train_aapl,order=(9, 1, 9))❶
         arima_fitted_aapl = arima_aapl.fit()❷

In [51]: arima_msft = ARIMA(train_msft, order=(6, 1, 6))❸
         arima_fitted_msft = arima_msft.fit()❹

In [52]: arima_predictions_aapl = arima_fitted_aapl.predict(start=len(train_aapl),
                                                            end=len(train_aapl)\
                                                            + len(test_aapl) - 1,
                                                            dynamic=False)❺
         arima_predictions_msft = arima_fitted_msft.predict(start=len(train_msft),
                                                            end=len(train_msft)\
                                                            + len(test_msft) - 1,
                                                            dynamic=False)❻

In [53]: arima_predictions_aapl = pd.DataFrame(arima_predictions_aapl)
         arima_predictions_aapl.index = diff_test_aapl.index
         arima_predictions_msft = pd.DataFrame(arima_predictions_msft)
         arima_predictions_msft.index = diff_test_msft.index❼
```

❶ Configuring the ARIMA model for Apple stock.

❷ Fitting ARIMA model to Apple stock price.

❸ Configuring the ARIMA model for Apple stock.

❹ Fitting ARIMA model to Microsoft stock price.

❺ Predicting the Apple stock prices based on ARIMA.

❻ Predicting the Microsoft stock prices based on ARIMA.

❼ Forming index for predictions

```
In [54]: fig, ax = plt.subplots(2, 1, figsize=(18, 15))

         ax[0].plot(diff_test_aapl, label='Actual Stock Price', c='b')
         ax[0].plot(arima_predictions_aapl, c='r', label="Prediction")
         ax[0].set_title('Predicted Stock Price-Apple')
         ax[0].legend(loc='best')
         ax[1].plot(diff_test_msft, label='Actual Stock Price', c='b')
         ax[1].plot(arima_predictions_msft, c='r', label="Prediction")
         ax[1].set_title('Predicted Stock Price-Microsoft')
         ax[1].legend(loc='best')
         for ax in ax.flat:
             ax.set(xlabel='Date', ylabel='$')
         plt.savefig('images/ARIMA.png')
         plt.show()
```

Figure 2-20 exhibits the result of the prediction based on Apple and Microsoft stock price and as I employ the same order in AR and MA model, it turns out to be the same with these models.

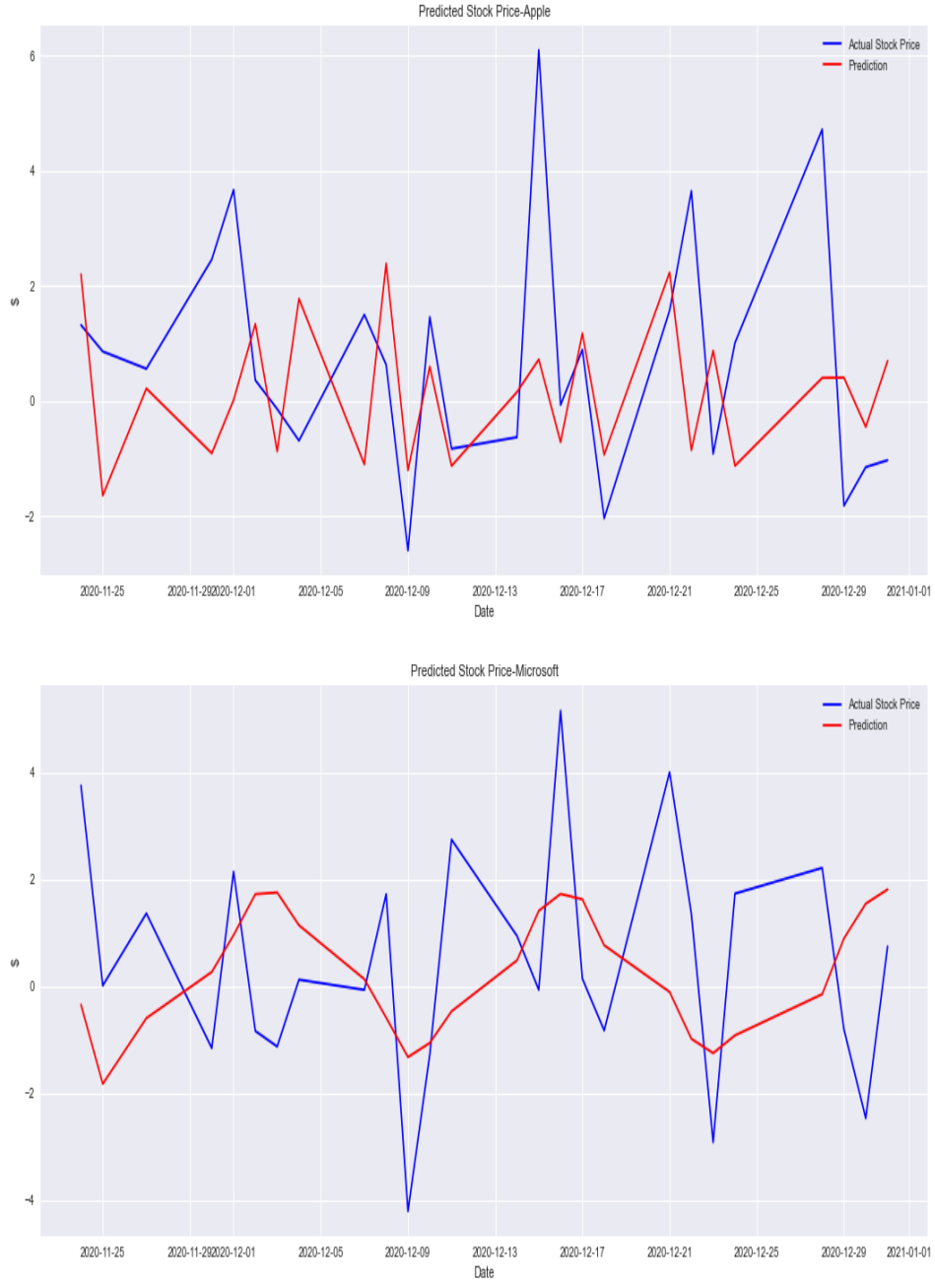


Figure 2-20. ARIMA Prediction Results

At this conjecture, it is worthwhile discussing the alternative method for optimum lag selection for time series models. AIC is the method that I apply here to select the proper number of lags. Please note that, even though the result of the AIC suggests (4, 0, 4), the model does not convergence with this orders. So, (4, 1, 4) is applied instead.

```
In [55]: import itertools

In [56]: p = q = range(0, 9)❶
         d = range(0, 3)❷
         pdq = list(itertools.product(p, d, q))❸
         arima_results_aapl = []❹
         for param_set in pdq:
             try:
                 arima_aapl = ARIMA(train_aapl, order=param_set)❺
                 arima_fitted_aapl = arima_aapl.fit()❻
                 arima_results_aapl.append(arima_fitted_aapl.aic)❼
             except:
                 continue
         print('**'*25)
         print('The Lowest AIC score is {:.4f} and the corresponding parameters are {}'
               .format(pd.DataFrame(arima_results_aapl)
                       .where(pd.DataFrame(arima_results_aapl).T.notnull().all()).min()
[0],
                       pdq[arima_results_aapl.index(min(arima_results_aapl))]))❽
         **************************************************
         The Lowest AIC score is 1951.9810 and the corresponding parameters are (4,
          0, 4)

In [57]: arima_aapl = ARIMA(train_aapl, order=(4, 1, 4))❾
         arima_fitted_aapl = arima_aapl.fit()❾

In [58]: p = q = range(0, 6)
         d = range(0, 3)
         pdq = list(itertools.product(p, d, q))
         arima_results_msft = []
         for param_set in pdq:
             try:
                 arima_msft = ARIMA(stock_prices['MSFT'], order=param_set)
                 arima_fitted_msft = arima_msft.fit()
                 arima_results_msft.append(arima_fitted_msft.aic)
             except:
                 continue
         print('**'*25)
         print('The Lowest AIC score is {:.4f} and the corresponding parameters are {}'
               .format(pd.DataFrame(arima_results_msft)
                       .where(pd.DataFrame(arima_results_msft).T.notnull().all()).min()
[0],
```

```
                    pdq[arima_results_msft.index(min(arima_results_msft))]))❿
          ************************************************
          The Lowest AIC score is 2640.6367 and the corresponding parameters are (4,
           2, 4)

In [59]: arima_msft = ARIMA(stock_prices['MSFT'], order=(4, 2 ,4))⓫
          arima_fitted_msft= arima_msft.fit()⓫

In [60]: arima_predictions_aapl = arima_fitted_aapl.predict(start=len(train_aapl),
                                                            end=len(train_aapl)\
                                                            +len(test_aapl) - 1,
                                                            dynamic=False)⓬
          arima_predictions_msft = arima_fitted_msft.predict(start=len(train_msft),
                                                            end=len(train_msft)\
                                                            + len(test_msft) - 1,
                                                            dynamic=False)⓬

In [61]: arima_predictions_aapl = pd.DataFrame(arima_predictions_aapl)
          arima_predictions_aapl.index = diff_test_aapl.index
          arima_predictions_msft = pd.DataFrame(arima_predictions_msft)
          arima_predictions_msft.index = diff_test_msft.index
```

❶ Defining a range for AR and MA orders.

❷ Defining a range difference term.

❸ Applying iteration over p, d, and q.

❹ Create an empty list to store AIC values.

❺ Configuring ARIMA model to fit Apple data.

❻ Running the ARIMA model with all possible lags.

❼ Storing AIC values into a list.

❽ Printing the lowest AIC value for Apple data.

❾ Configuring and fitting ARIMA model with optimum orders.

❿ Running ARIMA model with all possible lags for Microsoft data.

⓫ Fitting ARIMA model to Microsoft data with optimum orders.

- Predicting Apple and Microsoft stock prices.

```
In [62]: fig, ax = plt.subplots(2, 1, figsize=(18, 15))

         ax[0].plot(diff_test_aapl, label='Actual Stock Price', c='b')
         ax[0].plot(arima_predictions_aapl, c='r', label="Prediction")
         ax[0].set_title('Predicted Stock Price-Apple')
         ax[0].legend(loc='best')
         ax[1].plot(diff_test_msft, label='Actual Stock Price', c='b')
         ax[1].plot(arima_predictions_msft, c='r', label="Prediction")
         ax[1].set_title('Predicted Stock Price-Microsoft')
         ax[1].legend(loc='best')
         for ax in ax.flat:
             ax.set(xlabel='Date', ylabel='$')
         plt.savefig('images/ARIMA_AIC.png')
         plt.show()
```

Orders identified for Apple and Microsoft are (4, 1, 4) and (4, 2, 4), respectively. ARIMA does a good job in predicting the stock prices. However, please note that, improper identification of the orders results in a poor fit and this, in turn, produces predictions, which are far from being satisfactory.

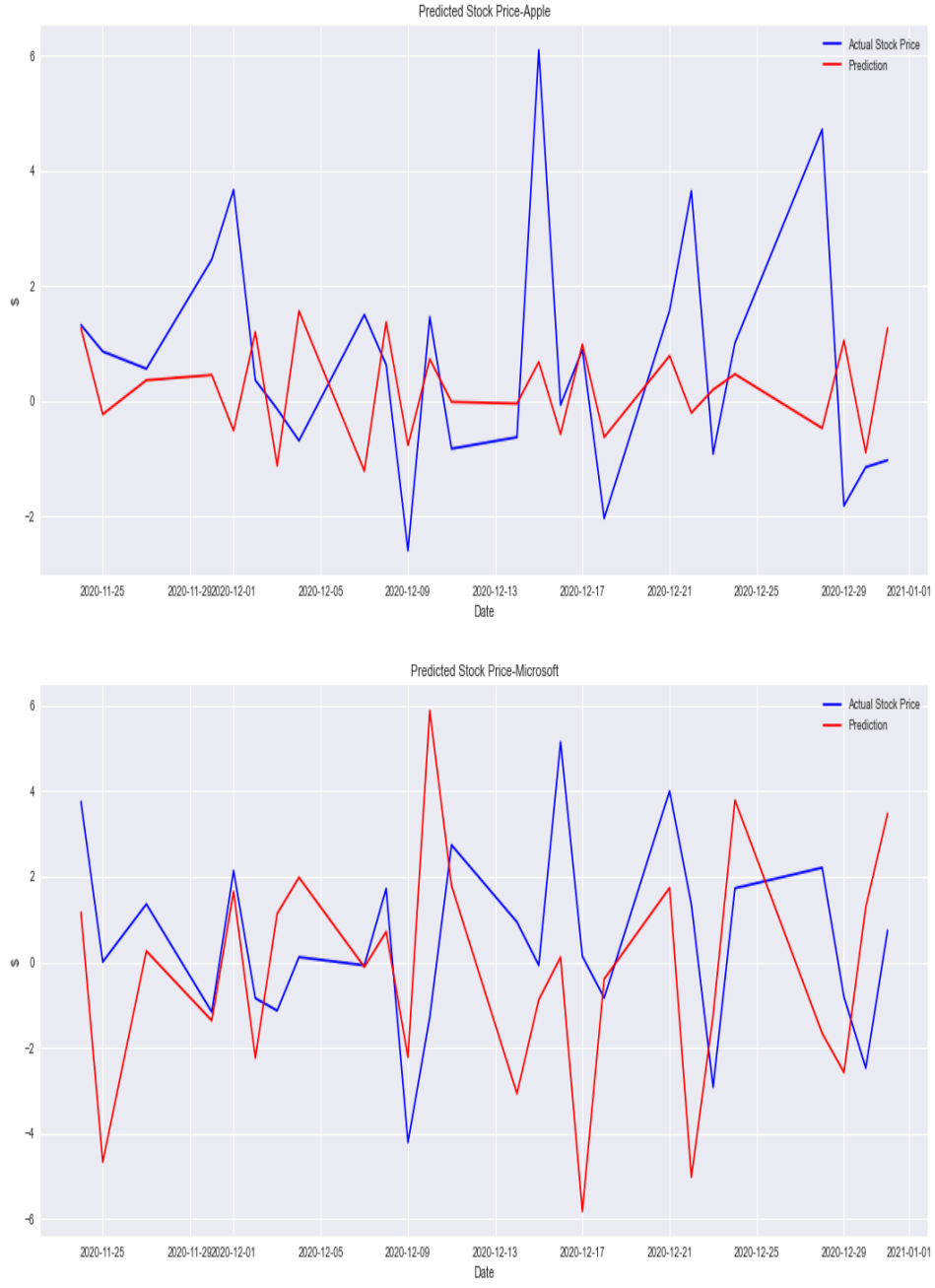


Figure 2-21. ARIMA Prediction Results

Conclusion

Time series analysis has a central role in financial analysis. It is simply because most of the financial data has time dimension and this type of data should be modeled cautiously. So, this chapter is the first attempt to model data with time dimension and to do that we have employed classical time series model, namely moving average, autoregressive model, and finally autoregressive integrated moving average. Do you think that that is the whole story? Absolutely, no! In the next chapter, we will see how a time series can be modeled using deep learning models.

Further Resources

Article cited in this chapter:

Hurvich, Clifford M., and Chih-Ling Tsai. “Regression and time series model selection in small samples.” Biometrika 76, no. 2 (1989): 297-30.

Books cited in this chapter:

- Buduma, N. and Locascio, N., 2017. Fundamentals of deep learning: Designing next-generation machine intelligence algorithms. O’Reilly Media, Inc.
- Brockwell, Peter J., and Richard A. Davis. Introduction to time series and forecasting. Springer, 2016.
- Walsh, Norman, and Leonard Muellner. DocBook: the definitive guide. O’Reilly Media, Inc., 1999.

Chapter 3. Deep Learning for Time Series Modeling

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

... Yes, it is true that a Turing machine can compute any computable function given enough memory and enough time, but nature had to solve problems in real time. To do this, it made use of the brain's neural networks that, like the most powerful computers on the planet, have massively parallel processors. Algorithms that run efficiently on them will eventually win out.

— Terrence J. Sejnowski (2018)

Deep Learning has recently become a buzzword for some good reasons though the attempt to improve deep learning practices are not the first of it kind. However, it is quite understandable why deep learning has been appreciated for nearly two decades. Deep learning is an abstract concept, which makes it hard to define in a couple of words.

Differently from neural network, deep learning has more complex structure and hidden layers define the complexity. Therefore, some researchers use

number of hidden layer as a comparison benchmark to distinguish the neural network and deep learning, it is useful but not a rigorous way to make this separation. So, a decent definition makes thing clear.

At a high level, deep learning can be defined as:

> *Deep-learning methods are representation-learningfootnote: [Representation learning helps us to define a concept in a unique way. For instance, if the task is to detect whether or not it is a circle, then edges play a key role as circle has no edge. So, using color, shape, size, we can create a representation for an object. In essence, this is how human brain works and we know that deep learning structure inspires from the functioning of it.] methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.*
>
> —Le Cunn et al. (2015)

Applications of deep learning dates back to 1940s in which Cybernetics is published and then Connectivist thinking dominates the years between 1980s and 1990s.Finally, recent development in deep learning such as backpropagation and neural network has created the field as we know as deep learning. Well, basically we are talking about three waves of deep learning and now, it is tempting to ask why deep learning is living its heyday? Goodfellow et al. (2016) list some plausible reasons, which are:

- Increasing Data Sizes
- Increasing Model Sizes
- Increasing Accuracy, Complexity, and Real World Impact

It seems like modern technology and data availability pave the way for deep learning era in which new data-driven methods are proposed so that we are able to model time series using unconventional models. This development has given rise a new wave of Deep Learning. In Deep Learning, two methods stand out with their ability to include longer time periods: “Recurrent Neural Network” (RNN) and “Long Short-Term Memory” (LSTM).

RNN and LSTM are two of them. In this part, we will concentrate on the practicality of these models in Python after briefly discussing the theoretical background.

Recurrent Neural Network

Recurrent Neural Network has a neural network structure with at least one feedback connection so that network can learn sequences. Feedback connection results in a loop enabling us unveil the non-linear characteristics. This type of connection brings us a new and quite useful property: *Memory*. Thus, RNN cannot only make use of the input data, but also the previous outputs, which sounds compelling when it comes to time series modeling.

RNN comes in many forms such as:

- One-to-One: It consists of single input and single output, which makes it most basic type of RNN.
- One-to-Many: In this form, RNN produces multiple outputs for a single input.
- Many-to-One: As opposed to One-to-Many structure, Many-to-one has multiple inputs for a singly output.
- Many-to-Many: It has multiple outputs and inputs and is known as most complicated structure of RNN.

Hidden unit in RNN feeds itself back into the neural network so that RNN has, unlike feed-forward neural network, recurrent layers, which makes it a suitable method to model a time series data. Therefore, in RNN, activations of a neuron comes from previous time step indication that RNN represents accumulating state of the network instance (Buduma, 2017)

As summarized by Nielsen (2019):

- RNN has time steps one at a time in an orderly fashion.
- The state of the network stays as it is from one time step to another.

- RNN updates its state based on the time step.

These dimensions are illustrated in Figure 3-1. As can be readily seen, RNN structure on the right hand side has time step, which is the main difference from feed forward network.

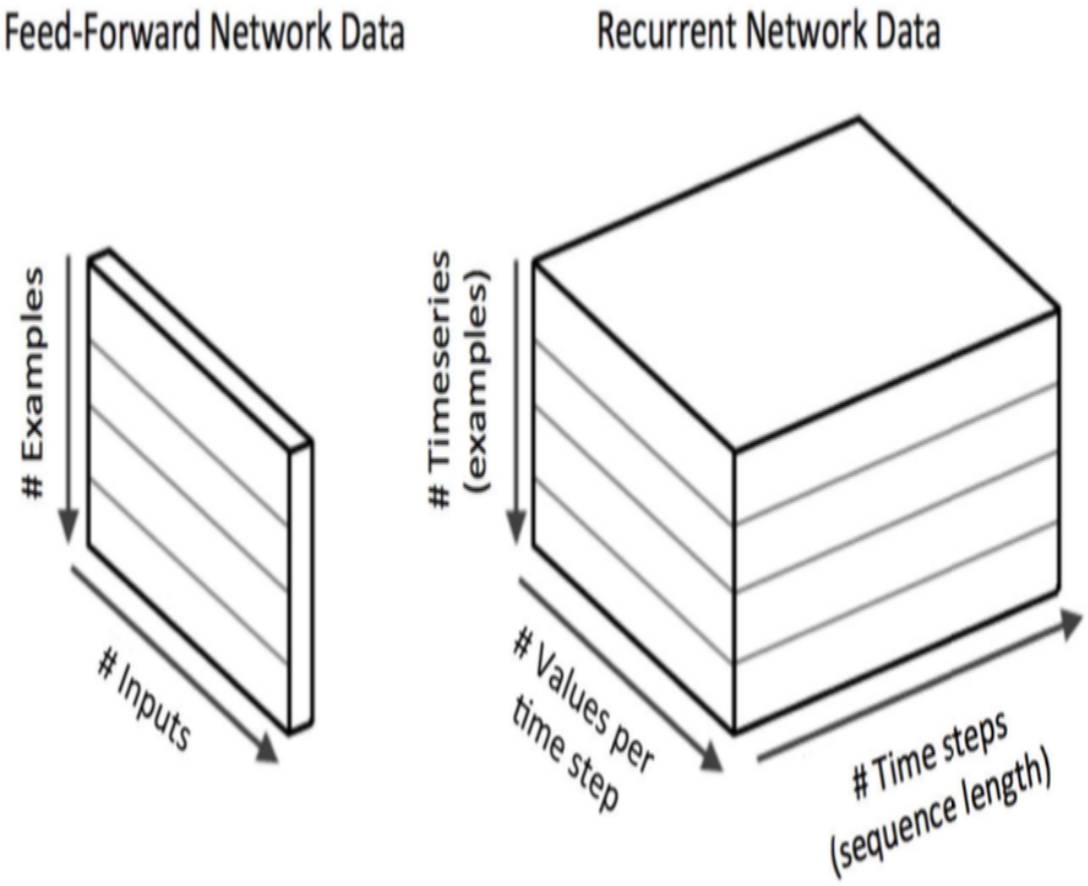


Figure 3-1. RNN Structure.[1]

RNN has three dimensional inpuy, which are:

- Batch size
- Time steps
- Number of feature

Batch size denotes the number of observations or number of rows of a data. Time steps is the number of times to feed the model. Finally, number of

feature is the number of columns of each sample.

```
In [1]: import numpy as np
        import pandas as pd
        import math
        import datetime
        import yfinance as yf
        import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.callbacks import EarlyStopping
        from tensorflow.keras.layers import (Dense, Dropout, Activation,
                                             Flatten, MaxPooling2D,SimpleRNN)
        from sklearn.model_selection import train_test_split
        import warnings
        warnings.filterwarnings('ignore')

In [2]: n_steps = 13❶
        n_features = 1❷

In [3]: model = Sequential()❸
        model.add(SimpleRNN(512, activation='relu',
                            input_shape=(n_steps, n_features),
                            return_sequences=True))❹
        model.add(Dropout(0.2))❺
        model.add(Dense(256, activation = 'relu'))❻
        model.add(Flatten())❼
        model.add(Dense(1, activation='linear'))❽

In [4]: model.compile(optimizer='rmsprop', loss='mean_squared_error',metrics=
['mse'])❾
```

❶ Defining the number of step to feed to RNN model.

❷ Defining number of feature as 1.

❸ Calling a Sequential model to run RNN.

❹ Identifying number of hidden neurons, and activation function, input shape.

❺ Putting a dropout layer to prevent overfitting.

❻ Adding one more hidden layer with 256 neuron with *relu* activation function.

❼ Flattenning the model to transform 3-dimensional matrix into a vector.

❽ Adding an output layer with *linear* activation function.

❾ Compiling RNN model.

```
In [5]: def split_sequence(sequence, n_steps):
            X, y = [], []
            for i in range(len(sequence)):
                end_ix = i + n_steps
                if end_ix > len(sequence) - 1:
                    break
                seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
                X.append(seq_x)
                y.append(seq_y)
            return np.array(X), np.array(y)❶
```

❶ Writing a function called *split_sequence* function to define look back period.

Figure 3-2 indicates the stock price prediction results for Apple and Microsoft. Although the prediction result do seem fine it fails to capture the extreme values.


Actual Stock Price
Prediction
6
4
2
0
−2
$
2019-12-13
2019-12-15
2019-12-17
2019-12-19
2019-12-21
2019-12-23
2019-12-25
2019-12-27
2019-12-29
2019-12-31
Date


Actual Stock Price
Prediction
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
$
2019-12-13
2019-12-15
2019-12-17
2019-12-19
2019-12-21
2019-12-23
2019-12-25
2019-12-27
2019-12-29
2019-12-31
Date

Figure 3-2. RNN Prediction Results

Even if we can have satisfactory predictive performance, the drawbacks of the RNN model should not be overlooked. The main drawbacks of the model are:

- Vanishing or exploding gradient problem. Please see the side note below for detailed explanation.
- Training an RNN is a very difficult task as it require consideranle amount of data.
- RNN is unable to process very long sequences when *tanh* activation function is used.

> **NOTE**
>
> Vanishing gradient is a commonplace problem in deep learning, which is not properly designed. Vanishing gradient problem arises if the gradient tends to get smaller as we are conducting backpropagation. It implies that neurons are learning so slow that optimization grind to a halt.
>
> Unlike vanishing gradient problem, exploding gradient problem occurs when small changes in the backpropagation result in huge updates in the weights during optimization process.

ACTIVATION FUNCTIONS

Activation functions are mathematical equations that are used to determine the output in neural network structure. Activation function is a tool to introduce non-linearity in the hidden layers so that we are able to model the non-linear issues.

Of the activation function, the followings are the most famous ones:

- Sigmoid: This activation function allows us to incorporate small amount of output as we introduce small changes in the model. It takes values between 0 and 1. The mathematical representation of *sigmoid* is:

$$\text{sigmoid}(x) = \frac{1}{1 + exp(-\sum_i w_i x_i - b)}$$

where w is weight, x denotes data, b represents bias, and subscript i shows features.

- Tanh: If you are handling with negative numbers, *tanh* is your activation function. As opposed to sigmoid function, it ranges between -1 and 1. The tanh formula is:

$$\tanh(x) = \frac{sinh(x)}{cosh(x)}$$

- Linear: Using *linear* activation function enables us to build linear relationship between independent and dependent variables. *Linear* activation function takes the inputs and multiplies by the weights and form the outputs. proportional to the input. So, it is convenient activation function for time-series models. Linear activation function takes the form of:

$$f(x) = wx$$

- Rectified Linear: *Rectified Linear* activation function, known as *ReLu*, can take 0 if the input is zero or below zero. If the input is greater than 0, it goes up in line with x. Mathematically:

$$\text{ReLu(x)} = \max(0, x)$$

- Softmax: It is widely applicable to classification problem as in sigmoid because *softmax* converts input into probabilistic distribution proportional to the exponential of the input numbers:

$$\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_i exp(x_i)}$$

Drawbacks of RNN are well-stated by Haviv et al. (2019) as:

> *This is due to the dependency of the network on its past states, and through them on the entire input history. This ability comes with a cost - RNNs are known to be hard to train (Pascanu et al., 2013a). This difficulty is commonly associated with the vanishing gradient that appears when trying to propagate errors over long times (Hochreiter, 1998). When training is successful, the network's hidden state represents these memories. Understanding how such representation forms throughout training can open new avenues for improving learning of memory-related tasks.*

Long-Short Term Memory

Long-Short Term Memory, LSTM for short, deep learning approach has been developed by Hochreiter and Schmidhuber (1997) and it is mainly based on Gated Recurrent Unit (GRU).

GRU is proposed to deal with vanishing gradient problem, which is a common problem in neural network structure and occurs when weight update becomes too small to create a significant change in the network. GRU

consists of two gates: *Update* and *reset*. When an early observation is detected as highly important, then we do not update the hidden state. In a similar fashion, when early observations are not significant, it leads to reset the state.

As we previously discussed, the one of the most appealing features of RNN is its ability to connect past and present information. However, this ability turns out to be a failure when "long-term dependencies" comes into the picture. Long-term dependencies mean that model learns from early observations.

For instance, let us examine the following sentence:

Countries have their own currencies as in USA where people transact with dollar

In the case of short-dependencies, it is known that the next predicted word is about a currency but if it is asked which currency is that? Then, things get complicated as we might have various currencies in the text implying long-term dependencies. It is needed to go way back to find something relevant about country using dollar.

LSTM tries to attack the weakness of RNN in long-term dependencies in a way that LSTM has a quite useful tool to get rid of the unnecessary information so that it works more efficiently. LSTM works with gates, enabling LSTM to forget irrelevant data. These are:

- Forget gates
- Input gates
- Output gates

Forget gates is created to sort out the necessary and unnecessary information so that LSTM performs more efficiently than RNN. In doing so, the value of activation function, sigmoid, becomes zero if the information is irrelevant. Forget gate can be formulated as:

$$F_t = \sigma(X_t W_I + h_{t-1} W_f + b_f)$$

where σ is activation function, h_{t-1} is previous hidden state, W_I and W_h are weights, and finally b_f is bias parameter in forget cell.

Input gate is fed by current timestep, X_t, and hidden state of the previous timestep $t-1$. The goal of input gate is to determine the extent to information that should be added to the long-term state. The input gate can be formulated like this:

$$I_t = \sigma(X_t W_I + h_{t-1} W_h + b_I)$$

Output gate basically determines the extent of the output that should be read and works as follows:

$$O_t = \sigma(X_t W_O + h_{t-1} W_O + b_I)$$

The gates are not the sole components of LSTM. The other components are:

- Candidate Memory cell
- Memory cell
- Hidden state

Candidate memory cell determines the extent to which information passes to the cell state. Differently, the activation function in the candidate cell is tanh and takes the following form:

$$\widehat{C_t} = \phi(X_t W_c + h_{t-1} W_c + b_c)$$

Memory cell allows LSTM to remember or to forget the information:

$$C_t = F_t \odot C + t - 1 + I_t \odot \widehat{C_t}$$

where $\odot$ is hadamard product.

In this recurrent network, hidden state is a tool to circulate information. Memory cell relates output gate to hidden state:

$$h_t = \phi(c_t) \odot O_t$$

Let us try to predict the Apple and Microsoft stock price using LSTM and see how it works:

```
In [18]: from tensorflow.keras.layers import LSTM


In [19]: n_steps = 13
         n_features = 1

In [20]: model = Sequential()
         model.add(LSTM(512, activation='relu',
                   input_shape=(n_steps, n_features),
                   return_sequences=True))❶
         model.add(Dropout(0.2))
         model.add(LSTM(256,activation='relu'))
         model.add(Flatten())
         model.add(Dense(1, activation='linear'))

In [21]: model.compile(optimizer='rmsprop', loss='mean_squared_error', metrics=
['mse'])
```

❶ Calling `LSTM` model and identifying the number of neurons and features

```
In [22]: history = model.fit(X_aapl, y_aapl,
                            epochs=400, batch_size=150, verbose=0,
                            validation_split = 0.10)

In [23]: start = X_aapl[X_aapl.shape[0] - 13]
         x_input = start
         x_input = x_input.reshape((1, n_steps, n_features))

In [24]: tempList_aapl = []
         for i in range(len(diff_test_aapl)):
             x_input = x_input.reshape((1, n_steps, n_features))
             yhat = model.predict(x_input, verbose=0)
             x_input = np.append(x_input, yhat)
             x_input = x_input[1:]
             tempList_aapl.append(yhat)
```

NOTE

The central idea of Root Mean Square Propagation, *RMSprop* for short, is a optimization method in which we calculate weighted average of squares gradients and exponential average of squares of gradients the moving average of the squared gradients for each weight. Then, find the difference of weight, which is to be used to compute new weight.

$$v_t = \rho_{v_{t-1}} + 1 - \rho g_t^2$$

$$\Delta w_t = -\frac{\nu}{\sqrt{\eta + \epsilon}} g_t$$

$$w_{t+1} = w_t + \Delta w_t$$

Figure 3-3 exhibits the prediction results and LSTM seems to outperform the RNN in particular fitting the extreme values.


Actual Stock Price
Prediction
6
5
4
3
2
1
0
−1
−2
$
2019-12-13
2019-12-15
2019-12-17
2019-12-19
2019-12-21
2019-12-23
2019-12-25
2019-12-27
2019-12-29
2019-12-31
Date


Actual Stock Price
Prediction
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
$
2019-12-13
2019-12-15
2019-12-17
2019-12-19
2019-12-21
2019-12-23
2019-12-25
2019-12-27
2019-12-29
2019-12-31
Date

Figure 3-3. LSTM Prediction Results

Conclusion

This chapter is dedicated to the deep learning-based stock price prediction. The models used are RNN and LSTM, which have ability on process longer time period. These models do not suggest remarkable improvement but still can be employed to model time series data. LSTM considers, in our case, 13-step look back period for prediction. For an extension, it would be wise approach to include multiple features into the Deep Learning-based models, which is not allowed in parametric time series models.

In the next part, volatility prediction will be discussed based on parametric and machine learning models so that we will be able to compare the performance of these models.

Further Resources

Articles cited in this chapter:

- Ding, Daizong, Mi Zhang, Xudong Pan, Min Yang, and Xiangnan He. "Modeling extreme events in time series prediction." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1114-1122. 2019.
- Haviv, Doron, Alexander Rivkind, and Omri Barak. "Understanding and controlling memory in recurrent neural networks." arXiv preprint arXiv:1902.07275 (2019).
- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436-444.

Books cited in this chapter:

- Nielsen, A. Practical Time Series Analysis: Prediction with Statistics and Machine Learning. (2019).
- Patterson, Josh, and Adam Gibson. Deep learning: A practitioner's approach. O'Reilly Media, Inc., 2017.
- Sejnowski, Terrence J. The deep learning revolution. Mit Press, 2018.

---

1 Patterson et. al, 2017. "Deep learning: A practitioner's approach."

Part II. Machine Learning for Market, Credit, Liquidity, and Operational Risks

Chapter 4. Machine Learning-Based Volatility Prediction

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

The most critical feature of the conditional return distribution is arguably its second moment structure, which is empirically the dominant time-varying characteristic of the distribution. This fact has spurred an enormous literature on the modeling and forecasting of return volatility.

—Andersen et al. (2003)

"Some concepts are easy to understand but hard to define. This also holds true for volatility" This could be a quote from someone living before Markowitz because the way he model the volatility is very clear and intuitive. Markowitz proposes his celebrated portfolio theory in which volatility is defined as standard deviation so that from then onward finance has become more intertwined with mathematics.

Volatility is the backbone of finance in the sense that it does not only provide information signal to investors but also inputs of various financial models.

What makes volatility so important? The answer stresses the importance of uncertainty, which is the main characteristic of the financial model.

Increased integration of financial markets has led to a prolonged uncertainty in financial market which in turn stresses the importance of volatility, degree at which values of financial assets changes. Volatility has been used as a proxy of risk that in an among the most important variable in many field such as asset pricing and risk management. Its strong presence and latency make it even compulsory to model. Basel Accord, therefore, came into effect in 1996, and volatility as a risk measure has taken the key role in risk management (Karasan and Gaygisiz, 2020).

There is a large and growing body of literature regarding the volatility estimation after the ground-breaking study of Black (1976), Raju and Ghosh (2004), Andersen and Bollerslev (1997), Dokuchaev (2014), and De Stefani et al. (2017). So, we are talking about a long tradition in volatility prediction using ARCH and GARCH-type models in which there are certain drawbacks that might cause failures, e.g., volatility clustering, information asymmetry and so on. Even though, this issues are addressed via differently models, the recent fluctuations in financial markets coupled with the development in machine learning make researchers to rethink volatility estimation.

In this chapter, our aim is to show how we can enhance the predictive performance using machine learning-based model. We will visit various machine learning algorithms, namely support vector regression, neural network, and deep learning, so that we are able to compare the predictive performance.

Modeling volatility amounts to modeling uncertainty so that we better understand and approach the uncertainty enabling us to have good enough approximation to the real world. In order to gauge the extent to which proposed model accounts for the real situation, we need to calculate the return volatility, which is also known as *realized volatility*. Realized volatility is the square root of realized variance, which is the sum of squared return. Realized volatility is used to calculate the performance of the volatility prediction method. Here is the formula for return volatility:

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{n=1}^{N}(r_n - \mu)^2}$$

where r and μ are return and mean of return, n is number of observations.

Let's see how return volatility is computed in Python:

```
In [1]: import numpy as np
        from scipy.stats import norm
        import scipy.optimize as opt
        import yfinance as yf
        import pandas as pd
        import datetime
        import time
        from arch import arch_model
        import matplotlib.pyplot as plt
        from numba import jit
        from sklearn.metrics import mean_squared_error
        import warnings
        warnings.filterwarnings('ignore')

In [2]: stocks = '^GSPC'
        start = datetime.datetime(2010, 1, 1)
        end = datetime.datetime(2021, 8, 1)
        s_p500 = yf.download(stocks, start=start, end = end, interval='1d')
        [**********************100%**********************]  1 of 1 completed

In [3]: ret = 100 * (s_p500.pct_change()[1:]['Adj Close'])❶
        realized_vol = ret.rolling(5).std()

In [4]: plt.figure(figsize=(10, 6))
        plt.plot(realized_vol.index,realized_vol)
        plt.title('Realized Volatility- S&P-500')
        plt.ylabel('Volatility')
        plt.xlabel('Date')
        plt.savefig('images/realized_vol.png')
        plt.show()
```

❶ Calculating the returns of S&P-500 based on adjusted closing prices.

Figure 4-1 shows the realized volatility of S&P-500 over the period of 2010-2021. The most striking observations is the spikes around Covid-19 pandemic.

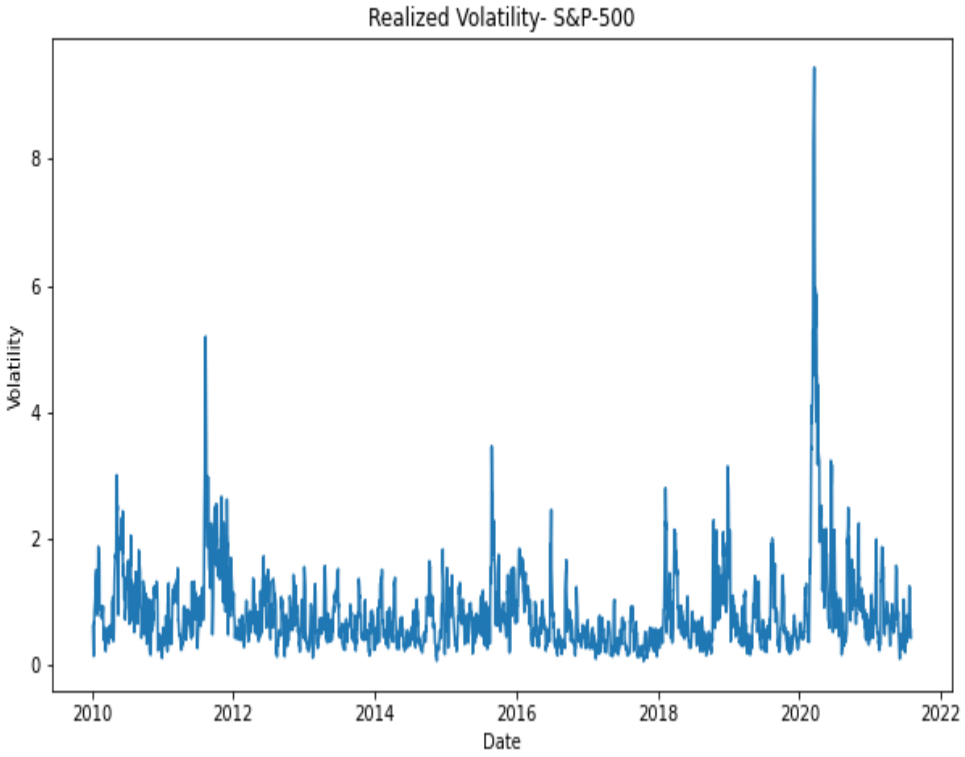


Figure 4-1. Realized Volatility- S&P-500

The way volatility is estimated has an undeniable impact on the reliablity and accuracy of the related analysis. So, this chapter deals with both classical and ML-based volatility prediction techniques with a view to show the superior prediction performance of the ML-based models. In order to compare the brand new ML-based models, we start with modeling the classical volatility models. Some very well known classical volatility models are, but not limited to:

- ARCH
- GARCH
- GJR-GARCH
- EGARCH

It is time to dig into the classical volatilility models. Let's start off with ARCH model.

ARCH Model

One of the early attempt to model the volatility was proposed by Engel (1982) and it is known as ARCH model. ARCH model is a univariate model and it is based on the historical asset returns. The ARCH(p) model has the following form:

$$\sigma_t^2 = \omega + \sum_{k=1}^{p} \alpha_k (r_{t-k})^2$$

where

$$r_t = \sigma_t \epsilon_t$$

where ϵ_t is assumed to be normally distributed. In this parametric model, we need to satisfy some assumptions to have strictly positive variance. To this respect, following condition should hold:

- $\omega > 0$
- $\alpha_k \geq 0$

All these equations tell us that ARCH is a univariate and non-linear model in which volatility is estimated with squared of past returns. The one of the most distinctive feature of ARCH is that it has the property of the time-varying conditional variance[1] so that ARCH is able to model the phenomenon known as volatility clustering, that is large changes tend to be followed by large changes of either sign, and small changes tend to be followed by small changes as put by Benoit Mandelbrot (1963). Hence, once an important announcement arrives into the market, it might results in a huge volatility.

The following code block shows how to plot clustering and what it looks like:

```
In [5]: retv = ret.values❶

In [6]: plt.figure(figsize=(10, 6))
        plt.plot(s_p500.index[1:], ret)
        plt.title('Volatility clustering of S&P-500')
        plt.ylabel('Daily returns')
        plt.xlabel('Date')
        plt.savefig('images/vol_clustering.png')
        plt.show()
```

❶ Return dataframe into numpy representation.

Similar to spikes in realized volatility, Figure 4-2 suggests some large movements and, unsurprisingly, these ups and downs happen around important events such as Covid-19 pandemic in the mid-2020.

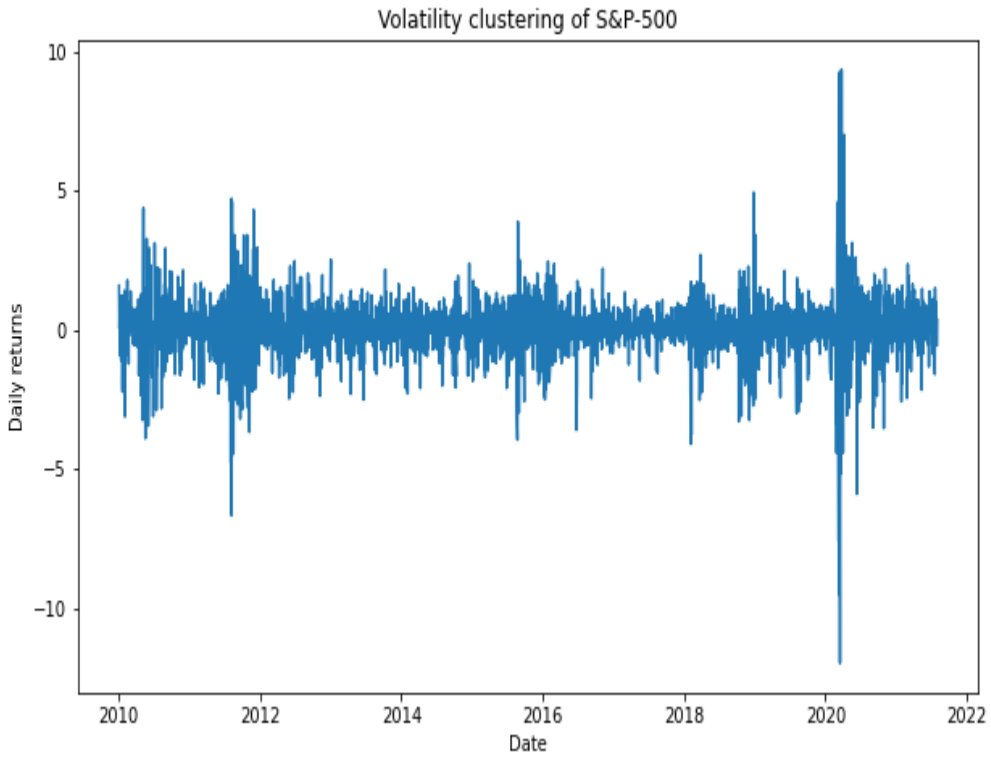


Figure 4-2. Volatility Clustering- S&P-500

Despite its appealing features such as simplicity, non-linearity, easiness, and adjustment for forecast, it has certain drawbacks, which can be listed as:

- Equal response to the positive and negative shocks.
- Strong assumptions such as restrictions on parameters.
- Possible misprediction due to slow-adjustment to large movements.

These drawbacks motivate researchers to work on extensions of ARCH model and Bollerslev (1986) and Taylor (2008) proposed GARCH model.

Now, we will employ ARCH model to predict volatility but first let us generate our own Python code and then compare it to see the difference with the built-in Python code.

```
In [7]: n = 252❶
        split_date = ret.iloc[-n:].index❷
```

```
In [8]: sgm2 = ret.var()❸
        K = ret.kurtosis()❹
        alpha = (-3.0 * sgm2 + np.sqrt(9.0 * sgm2 ** 2 - 12.0 *
                                       (3.0 * sgm2 - K) * K)) / (6 * K)❺
        omega = (1 - alpha) * sgm2❻
        initial_parameters = [alpha, omega]
        omega,alpha
Out[8]: (0.6345749196895419, 0.46656704131150534)

In [9]: @jit(nopython=True, parallel=True)❼
        def arch_likelihood(initial_parameters, retv):
            omega = abs(initial_parameters[0])❽
            alpha = abs(initial_parameters[1])❽
            T = len(retv)
            logliks = 0
            sigma2 = np.zeros(T)
            sigma2[0] = np.var(retv)❾
            for t in range(1, T):
                sigma2[t] = omega + alpha * (retv[t - 1]) ** 2 ❿
            logliks = np.sum(0.5 * (np.log(sigma2)+retv ** 2 / sigma2))⓫
            return logliks


In [10]: logliks = arch_likelihood(initial_parameters, retv)
         logliks
Out[10]: 1453.127184488521

In [11]: def opt_params(x0, retv):
             opt_result = opt.minimize(arch_likelihood, x0=x0, args = (retv),
                                       method='Nelder-Mead',
                                       options={'maxiter': 5000})⓬
             params = opt_result.x⓭
             print('\nResults of Nelder-Mead minimization\n{}\n{}'
                   .format(''.join(['-'] * 28), opt_result))
             print('\nResulting params = {}'.format(params))
             return params

In [12]: params = opt_params(initial_parameters, retv)

         Results of Nelder-Mead minimization
         ---------------------------
          final_simplex: (array([[0.70168795, 0.39039044],
                [0.70163494, 0.3904423 ],
         [0.70163928, 0.39033154]]), array([1385.79241695, 1385.792417  ,
          1385.79241907]))
                    fun: 1385.7924169507244
                message: 'Optimization terminated successfully.'
```

```
                  nfev: 62
                   nit: 33
                status: 0
               success: True
                     x: array([0.70168795, 0.39039044])

        Resulting params = [0.70168795 0.39039044]

In [13]: def arch_apply(ret):
             omega = params[0]
             alpha = params[1]
             T = len(ret)
             sigma2_arch = np.zeros(T + 1)
             sigma2_arch[0] = np.var(ret)
             for t in range(1, T):
                 sigma2_arch[t] = omega + alpha * ret[t - 1] ** 2
             return sigma2_arch

In [14]: sigma2_arch = arch_apply(ret)
```

❶ Defining the split location and assign the splitted data to *split* variable.

❷ Calculating variance of S&P-500.

❸ Calculating kurtosis of S&P-500.

❹ Identifying the initial value for slope coefficient α.

❺ Identifying the initial value for constant term ω.

❻ Using paralel processing to decrease the processing time.

❼ Taking absolute values and assigning the initial values into related variables.

❽ Identifying the initial values of volatility.

❾ Iterating the variance of S&P-500.

❿ Calculation log-likelihood.

11. Calling the function.
12. Minimizing the log-likelihood function.
13. Creating a variable *params* for optimized parameters.

Well, we model volatility via ARCH using our own optimization method and ARCH equation. How about comparing it with the built-in Python code. This built-in code can be imported from ARCH library and it is extremely easy-to-apply. The result of built-in code is provided below and it turns out that these two results are very similar to each other.

```
In [15]: arch = arch_model(ret, mean='zero', vol='ARCH', p=1).fit(disp='off')
         print(arch.summary())
         Zero Mean - ARCH Model Results

         ==============================================================================
          ========
         Dep. Variable:              Adj Close   R-squared:
          0.000
         Mean Model:                 Zero Mean   Adj. R-squared:
          0.000
         Vol Model:                      ARCH   Log-Likelihood:
          -4063.63
         Distribution:                  Normal   AIC:
          8131.25
         Method:            Maximum Likelihood   BIC:
          8143.21
         No. Observations:                 2914

         Date:                Tue, Sep 07 2021   Df Residuals:
          2914
         Time:                        11:19:08   Df Model:
            0
                                  Volatility Model
         ==============================================================================
                         coef    std err          t      P>|t|  95.0% Conf. Int.
         ------------------------------------------------------------------------------
         omega          0.7018  5.006e-02     14.018  1.214e-44 [  0.604,  0.800]
         alpha[1]       0.3910  7.016e-02      5.573  2.506e-08 [  0.253,  0.529]
         ==============================================================================

         Covariance estimator: robust
```

Although developing our own code is always helpful and improve our understanding, the beauty of built-in code is not only restricted to its simplicity. Finding the optimal lag value using built-in code is another advantage of it along with the optimized running procedure.

All we need is to create a for loop and define a proper information criteria. Below, Bayesian Information Criteria (BIC) is chosen as the model selection method and in order to select lag. The reason why BIC is picked is that as long as we have large enough samples, BIC is a reliable tool for model selection as discussed by Burnham and Anderson (2002 and 2004). Now, we iterate ARCH model from 1 to 5 lags.

```
In [16]: bic_arch = []

         for p in range(1,5):❶
                 arch = arch_model(ret, mean='zero', vol='ARCH', p=p)\
                         .fit(disp='off')❷
                 bic_arch.append(arch.bic)
                 if arch.bic == np.min(bic_arch):❸
                     best_param = p
         arch = arch_model(ret, mean='Constant', vol='ARCH', p=p)\
                 .fit(disp='off')❹
         print(arch.summary())
         forecast = arch.forecast(start=split_date[0])❺
         forecast_arch = forecast
         Constant Mean - ARCH Model Results

         ==============================================================================
          ========
         Dep. Variable:              Adj Close   R-squared:
          0.000
         Mean Model:             Constant Mean   Adj. R-squared:
          0.000
         Vol Model:                       ARCH   Log-Likelihood:
          -3691.03
         Distribution:                  Normal   AIC:
          7394.06
         Method:            Maximum Likelihood   BIC:
          7429.92
         No. Observations:                 2914

         Date:                Tue, Sep 07 2021   Df Residuals:
          2913
         Time:                        11:19:11   Df Model:
```

```
                   1
Mean Model

================================================================================
 ====
coef    std err          t      P>|t|    95.0% Conf. Int.

--------------------------------------------------------------------------------
 ----
mu             0.0852  1.391e-02      6.128  8.877e-10 [5.798e-02,
 0.113]
Volatility Model

================================================================================
 ====
coef    std err          t      P>|t|    95.0% Conf. Int.

--------------------------------------------------------------------------------
 ----
omega          0.2652  2.618e-02     10.130  4.052e-24   [  0.214,
 0.317]
alpha[1]       0.1674  3.849e-02      4.350  1.361e-05 [9.198e-02,
 0.243]
alpha[2]       0.2357  3.679e-02      6.406  1.496e-10   [  0.164,
 0.308]
alpha[3]       0.2029  3.959e-02      5.127  2.950e-07   [  0.125,
 0.281]
alpha[4]       0.1910  4.270e-02      4.474  7.687e-06   [  0.107,
 0.275]
================================================================================
 ====

Covariance estimator: robust
```

```
In [17]: rmse_arch = np.sqrt(mean_squared_error(realized_vol[-n:] / 100,
                                                np.sqrt(forecast_arch\
                                                        .variance.iloc[-
len(split_date):]
                                                                       / 100)))❻
         print('The RMSE value of ARCH model is {:.4f}'.format(rmse_arch))
         The RMSE value of ARCH model is 0.0896

In [18]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(forecast_arch.variance.iloc[-len(split_date):] / 100,
                  label='Volatility Prediction-ARCH')
         plt.title('Volatility Prediction with ARCH', fontsize=12)
         plt.legend()
```

```
plt.savefig('images/arch.png')
plt.show()
```

❶ Iterating ARCH parameter p over specified interval.

❷ Running ARCH model with different p values.

❸ Finding the minimum Bayesian Information Criteria score to select the best model.

❹ Running ARCH model with the best p value.

❺ Forecasting the volatility based on the optimized ARCH model.

❻ Calculating the RMSE score.

The result of volatility prediction based on our first model is shown in Figure 4-3.

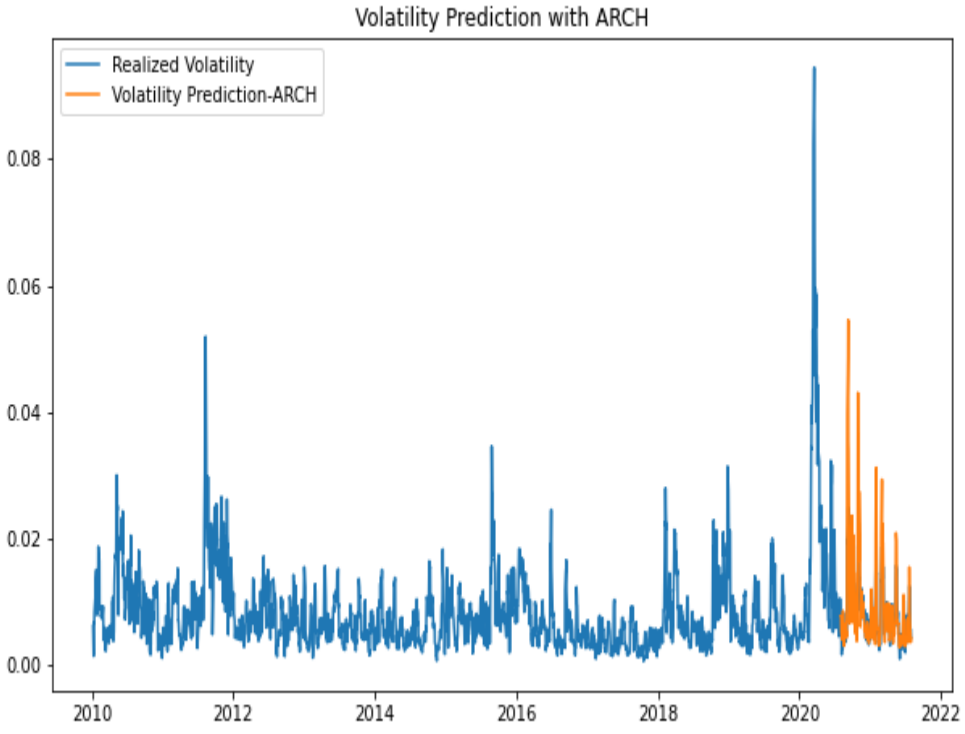


Figure 4-3. Volatility Prediction with ARCH

GARCH Model

GARCH model is an extension of ARCH model incorporating lagged conditional variance. So, ARCH is improved by adding p number of delated conditional variance, which makes GARCH model a multivariate one in the sense that it is a autoregressive moving average models for conditional variance with p number of lagged squared returns and q number of lagged conditional variance. GARCH (p, q) can be formulated as:

$$\sigma_t^2 = \omega + \sum_{k=1}^{q} \alpha_k r_{t-k}^2 + \sum_{k=1}^{p} \beta_k \sigma_{t-k}^2$$

where ω, β, and α are parameters to be estimated and q and p are maximum lag in the model. In order to have consistent GARCH, following conditions

should hold:

- $\omega > 0$
- $\beta \geq 0$
- $\alpha \geq 0$
- $\beta + \alpha < 1$

ARCH model is unable to capture the influence of historical innovations. However, as a more parsimonious model, GARCH model can account for the change in historical innovations because GARCH models can be expressed as an infinite-order ARCH. Let's show how GARCH can be shown as infinite order of ARCH.

$$\sigma_t^2 = \omega + \alpha r_{t-1}^2 + \beta \sigma_{t-1}^2$$

Then replace σ_{t-1}^2 by $\omega + \alpha r_{t-2}^2 + \beta \sigma_{t-2}^2$

$$\sigma_t^2 = \omega + \alpha r_{t-1}^2 + \beta\left(\omega + \alpha r_{t-2}^2 \sigma_{t-2}^2\right)$$

$$= \omega(1+\beta) + \alpha r_{t-1}^2 + \beta \alpha r_{t-2}^2 + \beta^2 \sigma_{t-2}^2\big)$$

Now, let us substitute σ_{t-2}^2 by $\omega + \alpha r_{t-3}^2 + \beta \sigma_{t-3}^2$ and do the necessary math, we end up with:

$$\sigma_t^2 = \omega\left(1 + \beta + \beta^2 + \ldots\right) + \alpha \sum_{k=1}^{\infty} \beta^{k-1} r_{t-k}$$

Similar to ARCH model, there are more than one way to model volatility using GARCH in Python. Let us try to develop our own Python-based code using optimization technique first. In what follows, *arch* library will be used to predict volatility.

```
In [19]: a0 = 0.0001
         sgm2 = ret.var()
         K = ret.kurtosis()
```

```
        h = 1 - alpha / sgm2
        alpha = np.sqrt(K * (1 - h ** 2) / (2.0 * (K + 3)))
        beta = np.abs(h - omega)
        omega = (1 - omega) * sgm2
        initial_parameters = np.array([omega, alpha, beta])
        print('Initial parameters for omega, alpha, and beta  are \n{}\n{}\n{}'
              .format(omega, alpha, beta))
        Initial parameters for omega, alpha, and beta  are
        0.43471178001576827
        0.512827280537482
        0.02677799855546381

In [20]: retv = ret.values

In [21]: @jit(nopython=True, parallel=True)
         def garch_likelihood(initial_parameters, retv):
             omega = initial_parameters[0]
             alpha = initial_parameters[1]
             beta = initial_parameters[2]
             T =  len(retv)
             logliks = 0
             sigma2 = np.zeros(T)
             sigma2[0] = np.var(retv)
             for t in range(1, T):
                 sigma2[t] = omega + alpha * (retv[t - 1]) ** 2 + beta * sigma2[t-1]
             logliks = np.sum(0.5 * (np.log(sigma2) + retv ** 2 / sigma2))
             return logliks

In [22]: logliks = garch_likelihood(initial_parameters, retv)
         print('The Log likelihood  is {:.4f}'.format(logliks))
         The Log likelihood  is 1387.7215

In [23]: def garch_constraint(initial_parameters):
             alpha = initial_parameters[0]
             gamma = initial_parameters[1]
             beta = initial_parameters[2]
             return np.array([1 - alpha - beta])

In [24]: bounds = [(0.0, 1.0), (0.0, 1.0), (0.0, 1.0)]

In [25]: def opt_paramsG(initial_parameters, retv):
             opt_result = opt.minimize(garch_likelihood,
                                       x0=initial_parameters,
                                       constraints=np.array([1 - alpha - beta]),
                                       bounds=bounds, args = (retv),
                                       method='Nelder-Mead',
                                       options={'maxiter': 5000})
             params = opt_result.x
```

```
            print('\nResults of Nelder-Mead minimization\n{}\n{}'\
                  .format('-' * 35, opt_result))
            print('-' * 35)
            print('\nResulting parameters = {}'.format(params))
            return params

In [26]: params = opt_paramsG(initial_parameters, retv)

         Results of Nelder-Mead minimization
         ----------------------------------
          final_simplex: (array([[0.03918956, 0.17370549, 0.78991502],
                [0.03920507, 0.17374466, 0.78987403],
                [0.03916671, 0.17377319, 0.78993078],
         [0.03917324, 0.17364595, 0.78998753]]), array([979.87109624, 979.8710967 ,
          979.87109865, 979.8711147 ]))
                    fun: 979.8710962352685
                message: 'Optimization terminated successfully.'
                   nfev: 178
                    nit: 102
                 status: 0
                success: True
                      x: array([0.03918956, 0.17370549, 0.78991502])
         ----------------------------------

         Resulting parameters = [0.03918956 0.17370549 0.78991502]

In [27]: def garch_apply(ret):
                 omega = params[0]
                 alpha = params[1]
                 beta = params[2]
                 T = len(ret)
                 sigma2 = np.zeros(T + 1)
                 sigma2[0] = np.var(ret)
                 for t in range(1, T):
                     sigma2[t] = omega + alpha * ret[t - 1] ** 2 + beta * sigma2[t-
1]
                 return sigma2
```

The parameters we get from our own GARCH code are approximately:

- $\omega = 0.0375$
- $\alpha = 0.1724$
- $\beta = 0.7913$

The following built-in Python code confirms that we did a great job on as the parameters obtained via the built-in code is almost the same with ours. So, we have learned how to code GARCH and ARCH models to predict volatility.

```
In [28]: garch = arch_model(ret, mean='zero', vol='GARCH', p=1, o=0, q=1)\
                .fit(disp='off')
         print(garch.summary())
         Zero Mean - GARCH Model Results

         ==============================================================================
          ========
         Dep. Variable:              Adj Close   R-squared:
          0.000
         Mean Model:                 Zero Mean   Adj. R-squared:
          0.000
         Vol Model:                      GARCH   Log-Likelihood:
          -3657.62
         Distribution:                  Normal   AIC:
          7321.23
         Method:            Maximum Likelihood   BIC:
          7339.16
         No. Observations:                 2914

         Date:                Tue, Sep 07 2021   Df Residuals:
          2914
         Time:                        11:19:28   Df Model:
            0
         Volatility Model

         ==============================================================================
          ======
         coef    std err          t      P>|t|      95.0% Conf. Int.

         ------------------------------------------------------------------------------
          ------
         omega          0.0392  8.422e-03      4.652  3.280e-06
          [2.268e-02,5.569e-02]
         alpha[1]       0.1738  2.275e-02      7.637  2.225e-14     [  0.129,
          0.218]
         beta[1]        0.7899  2.275e-02     34.715 4.607e-264     [  0.745,
          0.835]
         ==============================================================================
          ======

         Covariance estimator: robust
```

It is apparent that it is easy to work with GARCH(1, 1) but how do we know that theses parameters are the optimum one. Let us decide the optimum parameter set given the lowest BIC value.

```
In [29]: bic_garch = []

         for p in range(1, 5):
             for q in range(1, 5):
                 garch = arch_model(ret, mean='zero',vol='GARCH', p=p, o=0, q=q)\
                         .fit(disp='off')
                 bic_garch.append(garch.bic)
                 if garch.bic == np.min(bic_garch):
                     best_param = p, q
         garch = arch_model(ret, mean='zero', vol='GARCH', p=p, o=0, q=q)\
                 .fit(disp='off')
         print(garch.summary())
         forecast = garch.forecast(start=split_date[0])
         forecast_garch = forecast
         Zero Mean - GARCH Model Results

         ==============================================================================
          ========
         Dep. Variable:              Adj Close   R-squared:
          0.000
         Mean Model:                 Zero Mean   Adj. R-squared:
          0.000
         Vol Model:                      GARCH   Log-Likelihood:
          -3653.12
         Distribution:                  Normal   AIC:
          7324.23
         Method:            Maximum Likelihood   BIC:
          7378.03
         No. Observations:                 2914

         Date:                Tue, Sep 07 2021   Df Residuals:
          2914
         Time:                        11:19:30   Df Model:
            0
         Volatility Model

         ==============================================================================
          =====
         coef    std err          t      P>|t|     95.0% Conf. Int.

         ------------------------------------------------------------------------------
          -----
```

```
omega          0.1013  6.895e-02      1.469      0.142 [-3.388e-02,
 0.236]
alpha[1]       0.1347  3.680e-02      3.660  2.525e-04  [6.255e-02,
 0.207]
alpha[2]       0.1750      0.103      1.707  8.781e-02 [-2.593e-02,
 0.376]
alpha[3]       0.0627      0.163      0.386      0.700    [ -0.256,
 0.382]
alpha[4]       0.0655  8.901e-02      0.736      0.462    [ -0.109,
 0.240]
beta[1]    1.7151e-15      0.734  2.337e-15      1.000    [ -1.438,
 1.438]
beta[2]        0.2104      0.298      0.707      0.480    [ -0.373,
 0.794]
beta[3]        0.2145      0.547      0.392      0.695    [ -0.857,
 1.286]
beta[4]        0.0440      0.233      0.189      0.850    [ -0.412,
 0.500]
================================================================
 =====

Covariance estimator: robust
```

```
In [30]: rmse_garch = np.sqrt(mean_squared_error(realized_vol[-n:] / 100,
                                                 np.sqrt(forecast_garch\
                                                         .variance.iloc[-
len(split_date):]
                                                                 / 100)))
         print('The RMSE value of GARCH model is {:.4f}'.format(rmse_garch))
         The RMSE value of GARCH model is 0.0878

In [31]: plt.figure(figsize=(10,6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(forecast_garch.variance.iloc[-len(split_date):] / 100,
                  label='Volatility Prediction-GARCH')
         plt.title('Volatility Prediction with GARCH', fontsize=12)
         plt.legend()
         plt.savefig('images/garch.png')
         plt.show()
```

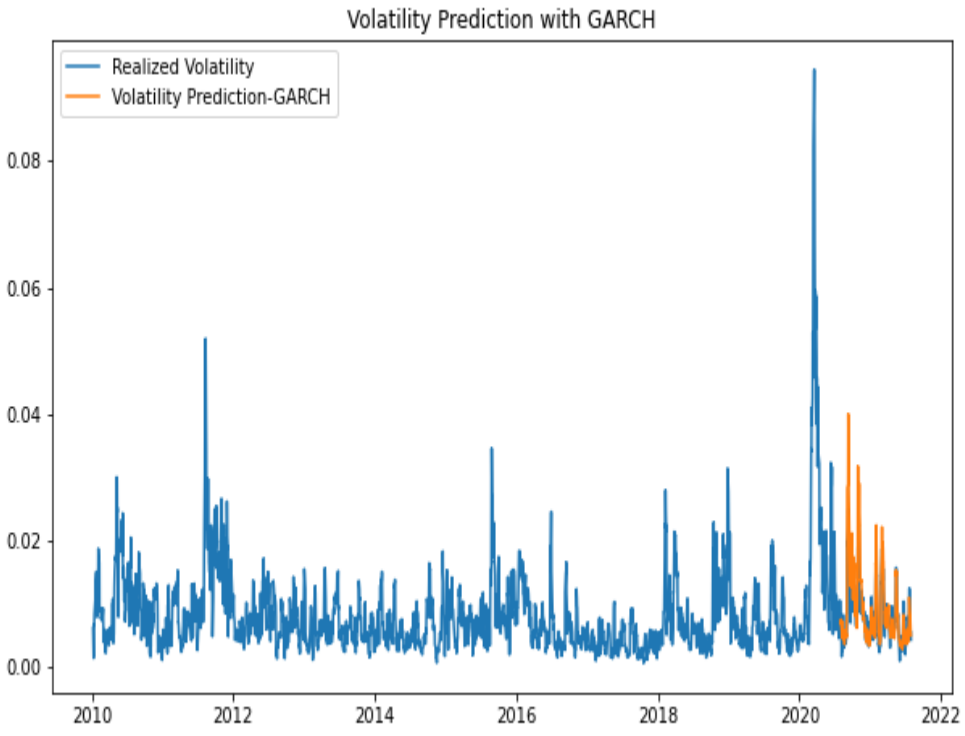


Figure 4-4. Volatility Prediction with GARCH

The main reasons of applying GARCH to volatility modeling are returns are well fitted by GARCH model partly due to the volatility clustering and GARCH does not assume that the returns are independent that allows modeling the leptokurtic property of returns. However, despite these useful properties and intuitiveness, GARCH is not able to asymmetric response of the shocks (Karasan and Gaygisiz (2020)). To remedy this issue, GJR-GARCH was proposed by Glosten, Jagannathan and Runkle (1993).

GJR-GARCH

This model performs well in modeling the asymmetric effects of the announcements in a way that bad news has larger impact than that of good news. In other words, in the presence of asymmetry, distribution of losses has a fatter tail than the distribution of gains. The equation of the model includes one more parameter γ and it takes the following form:

$$\sigma_t^2 = \omega + \sum_{k=1}^{q} \left(\alpha_k r_{t-k}^2 + \gamma r_{t-k}^2 I(\epsilon_{t-1} < 0)\right) + \sum_{k=1}^{p} \beta_k \sigma_{t-k}^2$$

where γ controls for the asymmetry of the announcements and if

- $\gamma = 0$, then the response to the past shock is the same
- $\gamma > 0$, then the response to the past negative shock is stronger than that of a positive one
- $\gamma < 0$, then the response to the past positive shock is stronger than that of a negative one

```
In [32]: bic_gjr_garch = []

         for p in range(1, 5):
             for q in range(1, 5):
                 gjrgarch = arch_model(ret,mean='zero', p=p, o=1, q=q)\
                             .fit(disp='off')
                 bic_gjr_garch.append(gjrgarch.bic)
                 if gjrgarch.bic == np.min(bic_gjr_garch):
                     best_param = p, q
         gjrgarch = arch_model(ret,mean='zero', p=p, o=1, q=q)\
                    .fit(disp='off')
         print(gjrgarch.summary())
         forecast = gjrgarch.forecast(start=split_date[0])
         forecast_gjrgarch = forecast
         Zero Mean - GJR-GARCH Model Results

         ==============================================================================
          ========
         Dep. Variable:              Adj Close   R-squared:
          0.000
         Mean Model:                 Zero Mean   Adj. R-squared:
          0.000
         Vol Model:                  GJR-GARCH   Log-Likelihood:
          -3588.89
         Distribution:                  Normal   AIC:
          7197.78
         Method:            Maximum Likelihood   BIC:
          7257.55
         No. Observations:                 2914

         Date:                Tue, Sep 07 2021   Df Residuals:
```

```
 2914
Time:                        11:19:35   Df Model:
   0
Volatility Model

================================================================================
 ======
coef    std err          t      P>|t|      95.0% Conf. Int.

--------------------------------------------------------------------------------
 ------
omega          0.0538  1.510e-02      3.560  3.712e-04
 [2.416e-02,8.336e-02]
alpha[1]   8.9183e-03  5.189e-02      0.172      0.864  [-9.277e-02,
 0.111]
alpha[2]       0.0000  6.279e-02      0.000      1.000     [ -0.123,
 0.123]
alpha[3]       0.0000  6.520e-02      0.000      1.000     [ -0.128,
 0.128]
alpha[4]       0.0480  4.966e-02      0.966      0.334  [-4.937e-02,
 0.145]
gamma[1]       0.3271  7.842e-02      4.171  3.029e-05     [  0.173,
 0.481]
beta[1]        0.7296      0.488      1.494      0.135     [ -0.228,
 1.687]
beta[2]        0.0000      0.462      0.000      1.000     [ -0.905,
 0.905]
beta[3]        0.0000      0.370      0.000      1.000     [ -0.725,
 0.725]
beta[4]        0.0103      0.293  3.512e-02      0.972     [ -0.563,
 0.584]
================================================================================
 ======

Covariance estimator: robust
```

```
In [33]: rmse_gjr_garch = np.sqrt(mean_squared_error(realized_vol[-n:] / 100,
                                                    np.sqrt(forecast_gjrgarch\
                                                            .variance.iloc[-
len(split_date):]
                                                                           / 100)))
         print('The RMSE value of GJR-GARCH models is
{:.4f}'.format(rmse_gjr_garch))
```

```
         The RMSE value of GJR-GARCH models is 0.0880
```

```
In [34]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(forecast_gjrgarch.variance.iloc[-len(split_date):] / 100,
```

```
        label='Volatility Prediction-GJR-GARCH')
plt.title('Volatility Prediction with GJR-GARCH', fontsize=12)
plt.legend()
plt.savefig('images/gjr_garch.png')
plt.show()
```

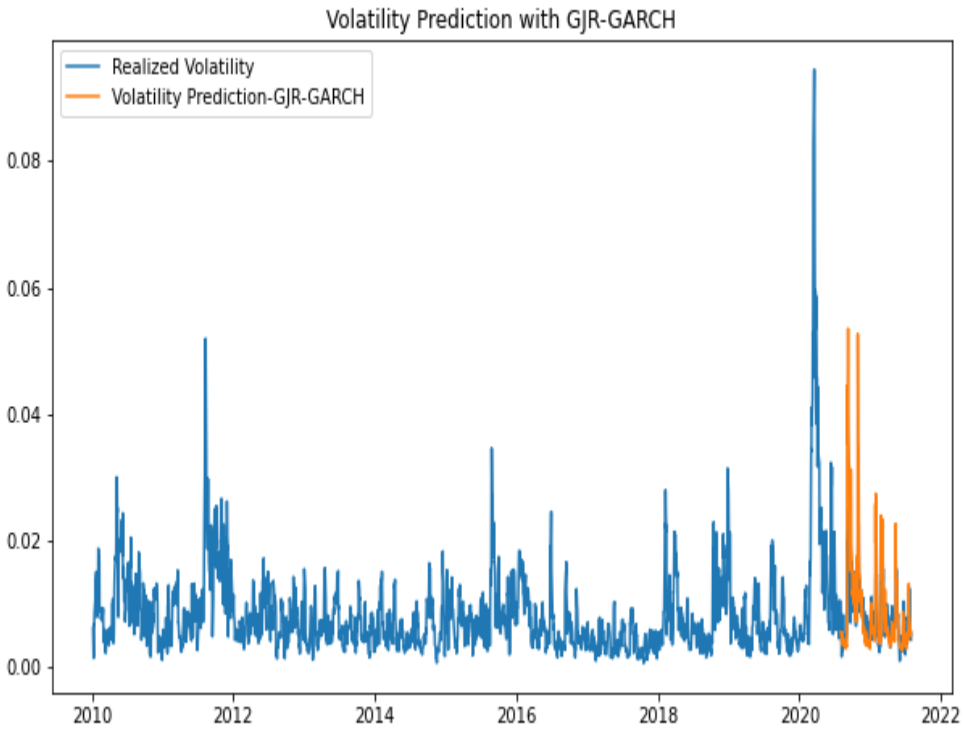


Figure 4-5. Volatility Prediction with GJR-GARCH

EGARCH

Together with the GJR-GARCH model, EGARCH proposed by Nelson (1991), is a also tool for controlling for the effect of the asymmetric announcements and additionally it is specified in logarithmic form, there is no need to put restriction to avoid negative volatility.

$$\log\left(\sigma_t^2\right) = \omega + \sum_{k=1}^{p} \beta_k \log\sigma_{t-k}^2 + \sum_{k=1}^{q} \alpha_i \frac{|r_{k-1}|}{\sqrt{\sigma_{t-k}^2}} + \sum_{k=1}^{q} \gamma_k \frac{r_{t-k}}{\sqrt{\sigma_{t-k}^2}}$$

The main difference of EGARCH equation is that logarithm is taken of the variance on the left-hand-side of the equation. This indicates the leverage effect meaning that there exists a negative correlation between past asset returns and volatility. If $\gamma < 0$, it implies leverage effect and if $\gamma \neq 0$, it shows asymmetry in volatility.

```
In [35]: bic_egarch = []

         for p in range(1, 5):
             for q in range(1, 5):
                 egarch = arch_model(ret, mean='zero',vol='EGARCH', p=p, o=1, q=q)\
                          .fit(disp='off')
                 bic_egarch.append(egarch.bic)
                 if egarch.bic == np.min(bic_egarch):
                     best_param = p, q
         egarch = arch_model(ret, mean='zero', vol='EGARCH', p=p, o=1, q=q)\
                  .fit(disp='off')
         print(egarch.summary())
         forecast = egarch.forecast(start=split_date[0])
         forecast_egarch = forecast
         Zero Mean - EGARCH Model Results

         ==============================================================================
          ========
         Dep. Variable:              Adj Close   R-squared:
          0.000
         Mean Model:                 Zero Mean   Adj. R-squared:
          0.000
         Vol Model:                      EGARCH   Log-Likelihood:
          -3575.09
         Distribution:                   Normal   AIC:
          7170.17
         Method:            Maximum Likelihood   BIC:
          7229.94
         No. Observations:                 2914

         Date:                Tue, Sep 07 2021   Df Residuals:
          2914
         Time:                        11:19:38   Df Model:
            0
         Volatility Model

         ==============================================================================
          ========
         coef    std err          t      P>|t|       95.0% Conf. Int.
```

```
------------------------------------------------------------------------
 -------
omega      -1.9786e-03  8.231e-03     -0.240      0.810
 [-1.811e-02,1.415e-02]
alpha[1]       0.1879  8.662e-02      2.170  3.004e-02    [1.816e-02,
 0.358]
alpha[2]       0.0661      0.130      0.508      0.612      [ -0.189,
 0.321]
alpha[3]      -0.0129      0.212 -6.081e-02      0.952      [ -0.429,
 0.403]
alpha[4]       0.0936  9.336e-02      1.003      0.316   [-8.936e-02,
 0.277]
gamma[1]      -0.2230  3.726e-02     -5.986  2.149e-09      [ -0.296,
 -0.150]
beta[1]        0.8400      0.333      2.522  1.168e-02      [  0.187,
 1.493]
beta[2]    4.1051e-14      0.658  6.235e-14      1.000      [ -1.290,
 1.290]
beta[3]    1.2375e-14      0.465  2.659e-14      1.000      [ -0.912,
 0.912]
beta[4]        0.0816      0.202      0.404      0.686      [ -0.314,
 0.478]
========================================================================
 ========

Covariance estimator: robust
```

```
In [36]: rmse_egarch = np.sqrt(mean_squared_error(realized_vol[-n:] / 100,
                                                 np.sqrt(forecast_egarch.variance\
                                                         .iloc[-len(split_date):]
                                                         / 100)))
         print('The RMSE value of EGARCH models is {:.4f}'.format(rmse_egarch))
```

```
The RMSE value of EGARCH models is 0.0895
```

```
In [37]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(forecast_egarch.variance.iloc[-len(split_date):] / 100,
                  label='Volatility Prediction-EGARCH')
         plt.title('Volatility Prediction with EGARCH', fontsize=12)
         plt.legend()
         plt.savefig('images/egarch.png')
         plt.show()
```

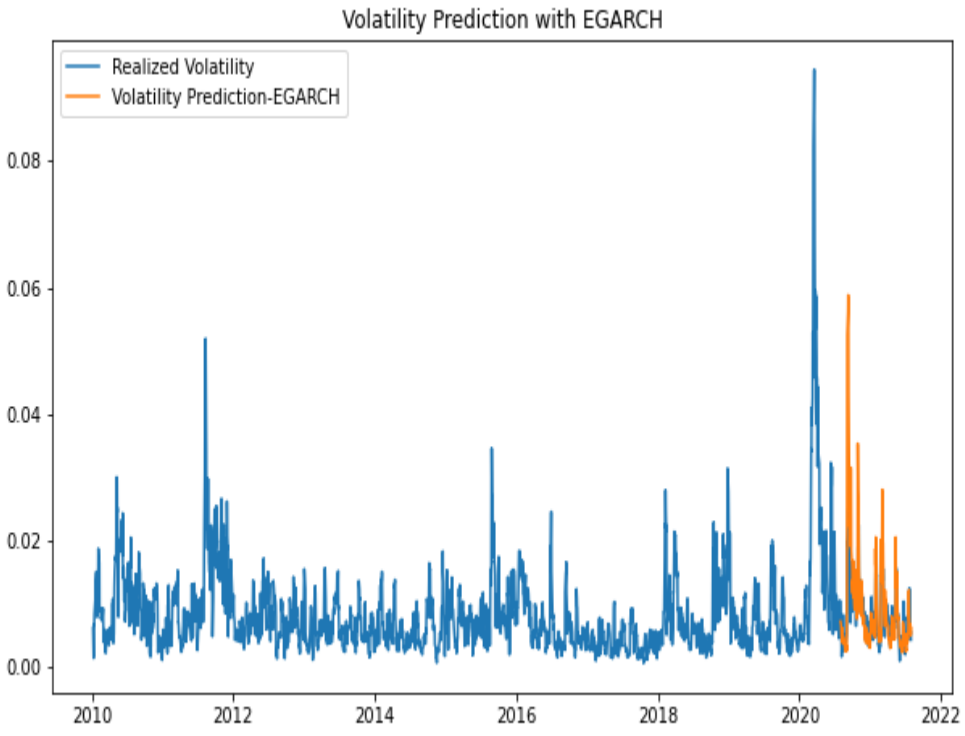


Figure 4-6. Volatility Prediction with GJR-GARCH

Given the RMSE result, the best and worst performing models are GARCH and ARCH, respectively. But there is no big differences among the performance of the model we have used above. In particular, during bad/good news announcement, the performances of EGARCH and GJR-GARCH might be different due to the asymmetry in the market.

Model	RMSE
ARCH	0.0896
GARCH	0.0878
GJR-GARCH	0.0880
EGARCH	0.0895

Up to now, we have discussed the classical volatility models but from this point on we will see how Machine Learning and Bayesian Approach can be used to model volatility. In the context of Machine Learning, Support Vector Machines and Neural Network will be the first models to visit. Let's get started.

Support Vector Regression-GARCH

Support Vector Machines (SVM) is a supervised learning algorithm, which can be applicable to both classification and regression. The aim in SVM is to find a line that separate two classes. It sounds easy but here is the challenging part: There are almost infinitely many lines that can be used to distinguish the classes. But we are looking for the optimal line by which the classes can be perfectly discriminated.

In linear algebra, the optimal line is called *hyperplane*, which maximize the distance between the points, which are closest to the hyperplane but belonging to different classes. The distance between the two points, i.e., support vectors, is known as *margin*. So, in SVM, what we are trying to do is to maximize the margin between support vectors.

SVM for classification is labeled as Support Vector Classification (SVC). Keeping all characteristics of SVM, it can be applicable to regression. Again, in regression, the aim is to find the hyperplane that minimize the error and maximize the margin. This method is called Support Vector Regression (SVR) and, in this part, we will apply this method to GARCH model. Combining these two models comes up with a different name: *SVR-GARCH*.

KERNEL FUNCTION

What happens if the data we are working on cannot be linearly separable. That would be a huge headache for us but no worries we have Kernel functions to remedy this problem. It is a nice and easy method to model non-linear and high dimensional data. The steps we take in Kernel SVM are:

1. Move the data into high dimension
2. Find suitable hyperplane
3. Go back to initial data

To do that we use kernel functions. Using the idea of feature map indicating that our original variables mapped to new set of quantities and passed to learning algorithm.

Finally, instead of input data we use following main Kernel functions in optimization procedures:

- Polynomial Kernel: $K(x, z) = \left(x^T z + b\right)$.
- Radial Basis (Gaussian) Kernel: $K(x, z) = \exp\left(-\frac{|x-z|^2}{2\sigma^2}\right)$.
- Exponential Kernel: $K(x, z) = exp\left(-\frac{|x-z|}{\sigma}\right)$. where x is input, b is bias or constant, and z is linear combination of x^2.

The following code shows us the preparations before running then SVR-GARCH in Python. The most crucial step here is to obtain independent variables, which are realized volatility and square of historical returns.

```
In [38]: from sklearn.svm import SVR
         from scipy.stats import uniform as sp_rand
         from sklearn.model_selection import RandomizedSearchCV
         from sklearn.metrics import mean_squared_error
```

```
In [39]: realized_vol = ret.rolling(5).std()❶
         realized_vol = pd.DataFrame(realized_vol)
         realized_vol.reset_index(drop=True, inplace=True)

In [40]: returns_svm = ret ** 2
         returns_svm = returns_svm.reset_index()
         del returns_svm['Date']

In [41]: X = pd.concat([realized_vol, returns_svm], axis=1, ignore_index=True)
         X = X[4:].copy()
         X = X.reset_index()
         X.drop('index', axis=1, inplace=True)

In [42]: realized_vol = realized_vol.dropna().reset_index()
         realized_vol.drop('index', axis=1, inplace=True)

In [43]: svr_poly = SVR(kernel='poly')❷
         svr_lin = SVR(kernel='linear')❷
         svr_rbf = SVR(kernel='rbf')❷
```

❶ Computing realized volatility and assign a new variable to it named *realized_vol*.

❷ Creating a new variables for different SVR kernel.

Let us run and see our first SVR-GARCH application with linear kernel. Root mean squared error (RMSE) is the metric to be used to compare.

```
In [44]: para_grid = {'gamma': sp_rand(),
                      'C': sp_rand(),
                      'epsilon': sp_rand()}❶
         clf = RandomizedSearchCV(svr_lin, para_grid)❷
         clf.fit(X.iloc[:-n].values,
                 realized_vol.iloc[1:-(n-1)].values.reshape(-1,))❸
         predict_svr_lin = clf.predict(X.iloc[-n:])❹

In [45]: predict_svr_lin = pd.DataFrame(predict_svr_lin)
         predict_svr_lin.index = ret.iloc[-n:].index

In [46]: rmse_svr = np.sqrt(mean_squared_error(realized_vol.iloc[-n:] / 100,
                                               predict_svr_lin / 100))
         print('The RMSE value of SVR with Linear Kernel is
{:.6f}'.format(rmse_svr))
```

```
        The RMSE value of SVR with Linear Kernel is 0.000648

In [47]: realized_vol.index = ret.iloc[4:].index

In [48]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(predict_svr_lin / 100, label='Volatility Prediction-SVR-GARCH')
         plt.title('Volatility Prediction with SVR-GARCH (Linear)', fontsize=12)
         plt.legend()
         plt.savefig('images/svr_garch_linear.png')
         plt.show()
```

❶ Identifying the hyperparameter space for tuning.

❷ Applying hyperparameter tuning with *RandomizedSearchCV*.

❸ Fitting SVR-GARCH with linear kernel to data.

❹ Predicting the volatilities based on the last 252 observations and store them in the *predict_svr_lin*.

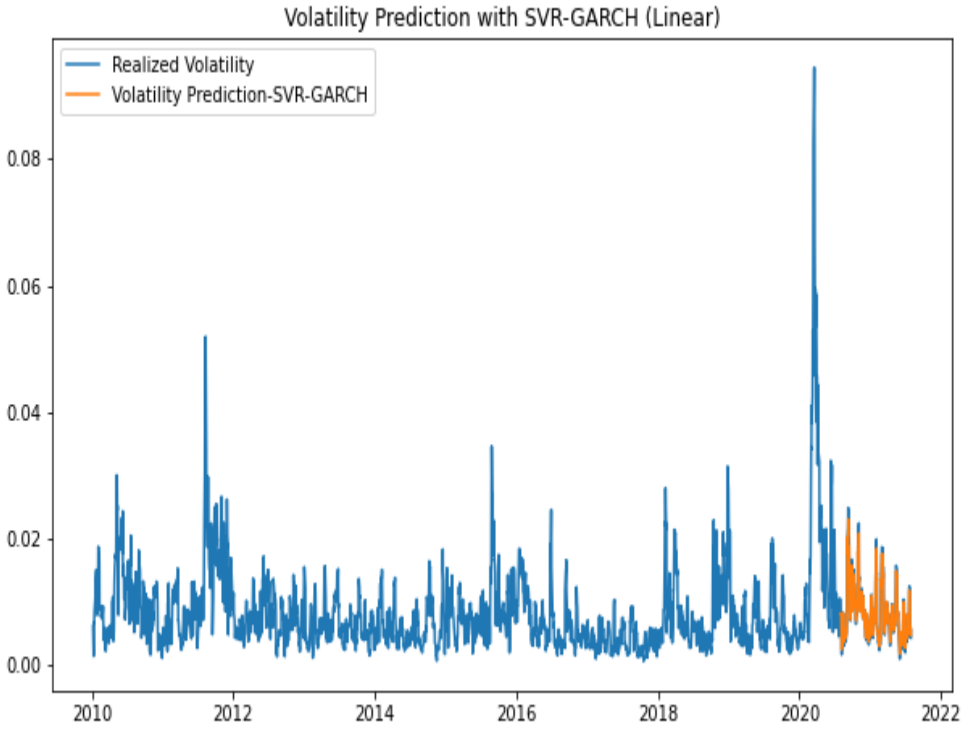


Figure 4-7. Volatility Prediction with SVR-GARCH Linear Kernel

Figure 4-7 exhibits the predicted values and actual observation. By eyeballing, one can tell that SVR-GARCH perform well. As you can guess, linear kernel works fine if the dataset is linearly separable and it is also the suggestion of _Occam's Razor_[3]. What if it does not? Let's continue with RBF and Polynomial kernels. The former one uses elliptical curves around the observations and the latter, differently from the first two, focuses on the combinations of samples, too. Let's now see how they work.

SVR-GARCH application with RBF kernel, a function that projections data into a new vector space, can be found below. From the practical standpoint, SVR-GARCH application with different kernels is not a labor-intensive process, all we need to switch the kernel name.

```
In [49]: para_grid ={'gamma': sp_rand(),
                     'C': sp_rand(),
                     'epsilon': sp_rand()}
         clf = RandomizedSearchCV(svr_rbf, para_grid)
```

```
        clf.fit(X.iloc[:-n].values,
                realized_vol.iloc[1:-(n-1)].values.reshape(-1,))
        predict_svr_rbf = clf.predict(X.iloc[-n:])

In [50]: predict_svr_rbf = pd.DataFrame(predict_svr_rbf)
         predict_svr_rbf.index = ret.iloc[-n:].index

In [51]: rmse_svr_rbf = np.sqrt(mean_squared_error(realized_vol.iloc[-n:] / 100,
                                                   predict_svr_rbf / 100))
         print('The RMSE value of SVR with RBF Kernel is
{:.6f}'.format(rmse_svr_rbf))
         The RMSE value of SVR with RBF Kernel is  0.000942

In [52]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(predict_svr_rbf / 100, label='Volatility Prediction-SVR_GARCH')
         plt.title('Volatility Prediction with SVR-GARCH (RBF)', fontsize=12)
         plt.legend()
         plt.savefig('images/svr_garch_rbf.png')
         plt.show()
```

Both RMSE score and the visualization suggests that SVR-GARCH with linear kernel outperforms SVR-GARCH with RBF kernel. The RMSE of SVR-GARCH with linear and RBF kernels are 0.000453 and 0.000986, respectively. So, SVR with linear kernel does performs well.

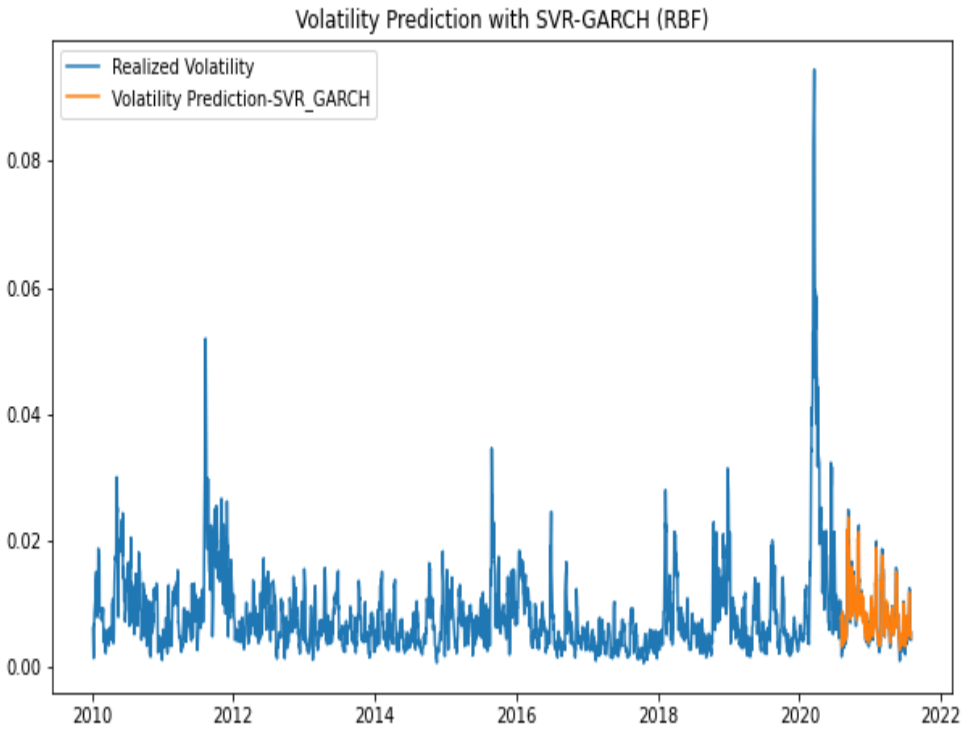


Figure 4-8. Volatility Prediction with SVR-GARCH RBF Kernel

Lastly, SVR-GARCH with polynomial kernel is employed but it turns out that it has the lowest RMSE implying that it is the worst performing kernel among these three different applications.

```
In [53]: para_grid = {'gamma': sp_rand(),
                      'C': sp_rand(),
                      'epsilon': sp_rand()}
         clf = RandomizedSearchCV(svr_poly, para_grid)
         clf.fit(X.iloc[:-n].values,
                 realized_vol.iloc[1:-(n-1)].values.reshape(-1,))
         predict_svr_poly = clf.predict(X.iloc[-n:])

In [54]: predict_svr_poly = pd.DataFrame(predict_svr_poly)
         predict_svr_poly.index = ret.iloc[-n:].index

In [55]: rmse_svr_poly = np.sqrt(mean_squared_error(realized_vol.iloc[-n:] / 100,
                                                    predict_svr_poly / 100))
         print('The RMSE value of SVR with Polynomial Kernel is {:.6f}'\
               .format(rmse_svr_poly))
         The RMSE value of SVR with Polynomial Kernel is 0.005291
```

```
In [56]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol/100, label='Realized Volatility')
         plt.plot(predict_svr_poly/100, label='Volatility Prediction-SVR-GARCH')
         plt.title('Volatility Prediction with SVR-GARCH (Polynomial)', fontsize=12)
         plt.legend()
         plt.savefig('images/svr_garch_poly.png')
         plt.show()
```

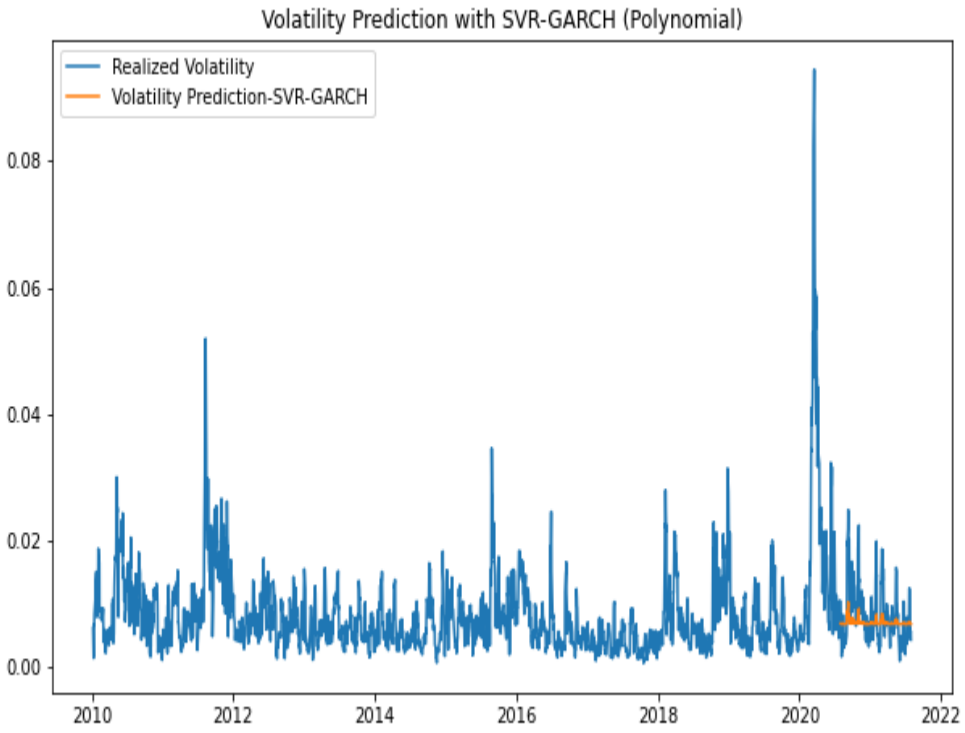


Figure 4-9. Volatility Prediction with SVR-GARCH Polynomial Kernel

Neural Network

Neural Network (NN) is the building block for deep learning. In NN, data is processed by multiple stages in a way to make a decision. Each neuron takes a result of a dot product as input and use it as input in activation function to make a decision.

$$z = w_1 x_1 + w_2 x_2 + b$$

where b is bias, w is weight, and x is input data.

During this process, input data is undertaken various mathematical manipulation in hidden and output layers. Generally speaking, NN has three types of layers:

- Input layer
- Hidden layer
- Output layer

Input layer includes raw data. In going from input layer to hidden layer, we learn coefficients. There may be one or more than one hidden layers depending on the network structuree. The more hidden layer the network has, the more complicated it is. Hidden layer, locating between inout and output layers, perform nonlinear transformation via activation function.

Finally, output layer is the layer in which output is produced and decision is made.

In Machine Learning, Gradient Descent is the tool applied to minimize the cost function but employing only gradient descent in neural network is not feasible due to the chain-like structure in neural network. Thus, a new concept known as backpropagation is proposed to minimize the cost function. The idea of backpropagation rest upon the calculating error between observed and actual output and pass this error to the hidden layer. So, we move backward and the main equation takes the form of:

$$\delta^l = \frac{\delta J}{\delta z_j^l}$$

where z is linear transformation and δ represents error. There is much more to say but to keep myself on track I stop here. For those who wants to dig more into math behind Neural Network please refer to Wilmott (2013) and Alpaydin (2020).

GRADIENT DESCENT

Suppose that we are at the top of the hill and try to reach to the plateau at which we minimize the cost function. Formally, Gradient Descent is an optimization algorithm used to search for best parameter space (w, b) which minimize cost function via following update rule:

$$\theta_{t+1} = \theta_t - \lambda \frac{\delta J}{\delta \theta_t}$$

where $\theta(w, b)$ is the function of weight, w, and bias, b. J is cost function, and λ is the learning rate, which is a constant number deciding how fast we want to minimize the cost function. Well, at each iteration, we update the parameters to minimize the error.

Shortly, The gradient descent algorithm works in the following way:

1. Select initial values for w and b.
2. Take an λ step in the direction opposite to where the gradient points.
3. Update w and b at each iteration.
4. Repeat from step 2 until convergence.

Now, we apply neural network based volatility prediction using *MLPRegressor* module from scikit-learn Python even though we have various options[4] to run neural network in Python. Given the neural network structure we introduce, the result is given below.

```
In [57]: from sklearn.neural_network import MLPRegressor❶
         NN_vol = MLPRegressor(learning_rate_init=0.001, random_state=1)
         para_grid_NN = {'hidden_layer_sizes': [(100, 50), (50, 50), (10, 100)],
                         'max_iter': [500, 1000],
                         'alpha': [0.00005, 0.0005 ]}❷
         clf = RandomizedSearchCV(NN_vol, para_grid_NN)
         clf.fit(X.iloc[:-n].values,
```

```
                realized_vol.iloc[1:-(n-1)].values.reshape(-1, ))❸
        NN_predictions = clf.predict(X.iloc[-n:])❹

In [58]: NN_predictions = pd.DataFrame(NN_predictions)
         NN_predictions.index = ret.iloc[-n:].index

In [59]: rmse_NN = np.sqrt(mean_squared_error(realized_vol.iloc[-n:] / 100,
                                               NN_predictions / 100))
         print('The RMSE value of NN is {:.6f}'.format(rmse_NN))
         The RMSE value of NN is 0.000583

In [60]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100, label='Realized Volatility')
         plt.plot(NN_predictions / 100, label='Volatility Prediction-NN')
         plt.title('Volatility Prediction with Neural Network', fontsize=12)
         plt.legend()
         plt.savefig('images/NN.png')
         plt.show()
```

❶ Importing *MLPRegressor* library.

❷ Configuring Neural Network model.

❸ Fitting Neural Network model to the training data[5].

❹ Predicting the volatilities based on the last 252 observations and store them in the *NN_predictions* variable.

Figure 4-10 shows the volatility prediction result based on neural network model. Despite its reasonable performance, we can play with the number of hidden neurons to find generate a deep learning model. To do that, we can apply Keras library, Python interface for artificial neural networks.

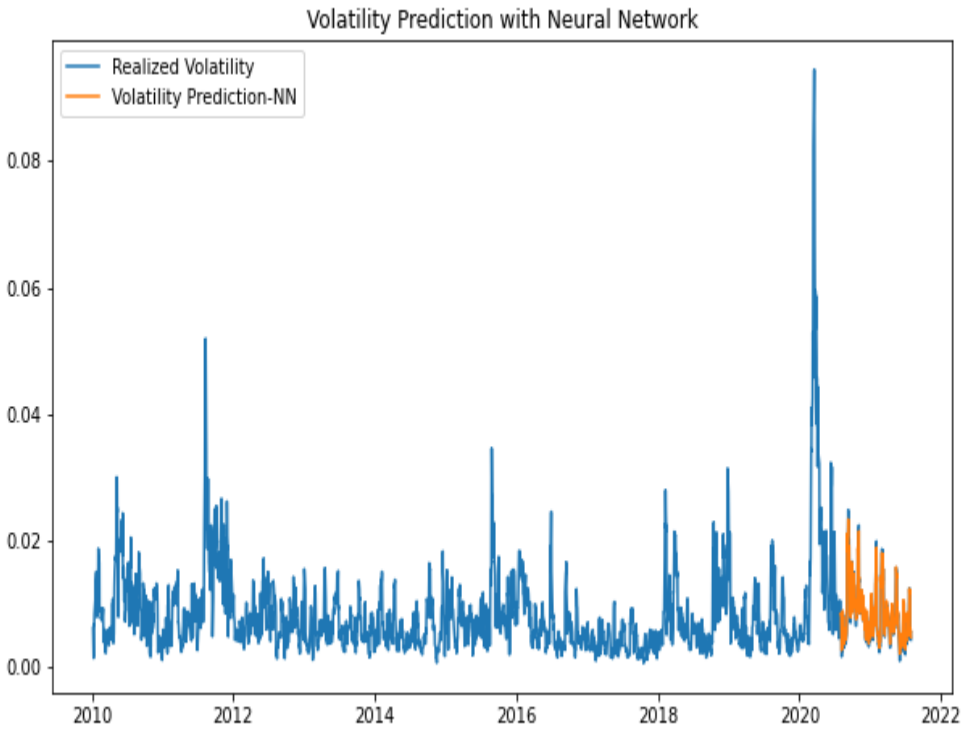


Figure 4-10. Volatility Prediction with Neural Network

Now, it is time to predict volatility using deep learning. Based on Keras, it is easy to configure the network structure. All we need is to determine the number of neuron of the specific layer. Here, the number of neuron for first and second hidden layers are 256 and 128, respectively. As volatility has a continuous type, we have only one output neuron.

```
In [61]: import tensorflow as tf
         from tensorflow import keras
         from tensorflow.keras import layers

In [62]: model = keras.Sequential(
             [layers.Dense(256, activation="relu"),
              layers.Dense(128, activation="relu"),
              layers.Dense(1, activation="linear"),])❶

In [63]: model.compile(loss='mse', optimizer='rmsprop')❷

In [64]: epochs_trial = np.arange(100, 400, 4)❸
         batch_trial = np.arange(100, 400, 4)❸
```

```
DL_pred = []
DL_RMSE = []
for i, j, k in zip(range(4), epochs_trial, batch_trial):
    model.fit(X.iloc[:-n].values,
              realized_vol.iloc[1:-(n-1)].values.reshape(-1,),
              batch_size=k, epochs=j, verbose=False)❹
    DL_predict = model.predict(np.asarray(X.iloc[-n:]))❺
    DL_RMSE.append(np.sqrt(mean_squared_error(realized_vol.iloc[-n:] / 100,
                            DL_predict.flatten() / 100)))❻
    DL_pred.append(DL_predict)
    print('DL_RMSE_{}:{:.6f}'.format(i+1, DL_RMSE[i]))
DL_RMSE_1:0.000524
DL_RMSE_2:0.001118
DL_RMSE_3:0.000676
DL_RMSE_4:0.000757
```

```
In [65]: DL_predict = pd.DataFrame(DL_pred[DL_RMSE.index(min(DL_RMSE))])
         DL_predict.index = ret.iloc[-n:].index
```

```
In [66]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100,label='Realized Volatility')
         plt.plot(DL_predict / 100,label='Volatility Prediction-DL')
         plt.title('Volatility Prediction with Deep Learning',  fontsize=12)
         plt.legend()
         plt.savefig('images/DL.png')
         plt.show()
```

❶ Configuring network structure by deciding number of layers and neurons.

❷ Compiling model with loss and optimizer.

❸ Deciding the epoch and batch size using *np.arange*.

❹ Fitting the deep learning model.

❺ Predicting the volatility based on the weights obtained from the training phase.

❻ Calculating RMSE score by flattening the predictions.

It turns out that we get minimum RMSE score as we increase the layer size, which is quite understandable because most of the time number of layers and

model performance goes hand in hand up a point in which model tends to overfit. Figuring out the proper number of layer for a specific data is a key in deep learning in the sense that we stop adding more layer before model get into the overfitting problem.

Figure 4-11 shows the volatility prediction result derived from the following code and it implies that deep learning provides a strong tool in modeling volatility, too.

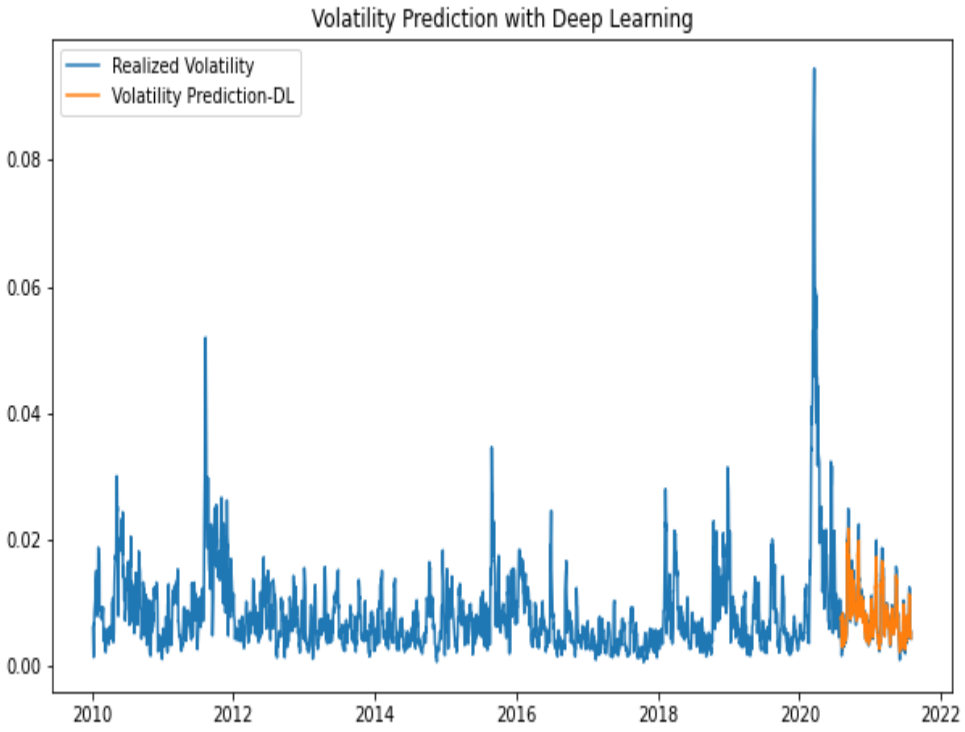


Figure 4-11. Volatility Prediction with Deep Learning

Bayesian Approach

The way we approach to the probability is of central importance in the sense that it distinquishes the classical (or frequentist) and Bayesian approach. According to the former method, the relative frequency will converge to the true probability. However, Bayesian application is based on the subjective

interpretation. Unlike frequentists, Bayesian statisticians consider the probability distribution as uncertain and it is revised as new information comes in.

Due to the different interpretation of the probability of these two approaches, likelihood, defined as, given a set of parameters, the probability of observed event, is computed differently.

Starting from joint density function, we can give the mathematical representation of likelihood function:

$$\mathscr{L}(\theta|x_1, x_2, \ldots, x_p) = \Pr(x_1, x_2, \ldots, x_p|\theta)$$

Among possible θ values, what we are trying to do is to decide which one is more likely. Under the statistical model proposed by likelihood function, the observed data $x_1, \ldots, x_p$ is the most probable.

In fact, you are familiar with the method based on the approach, which is maximum likelihood estimation. Having defined the main difference between Bayesian and Frequentist approaches, it is time to delve more into the Bayes' Theorem.

Bayes' Theorem

Bayesian approach is based on conditional distribution, which states that probability gauges the extent to which one has about a uncertain event. So, Bayesian application suggests a rule that can be used to update the beliefs that one holds in light of new information:

> *Bayesian estimation is used when we have some prior information regarding a parameter. For example, before looking at a sample to estimate the mean of a distribution, we may have some prior belief that it is close to 2, between 1 and 3. Such prior beliefs are especially important when we have a small sample. In such a case, we are interested in combining what the data tells us, namely, the value calculated from the sample, and our prior information.*
>
> — (Rachev et al., 2008)

Similar to the frequentist application, Bayesian estimation is based on probability density $\Pr(x|\theta)$. However, as we have discussed before, the way Bayesian and frequentist methods treat parameter set θ differently. Frequentist assumes θ to be fixed whereas it is, in Bayesian setting, taken as random variable, whose probability is known as prior density $\Pr(\theta)$. Oh no! We have another different term but no worries it is easy to understand.

In the light of this information, we can estimate $\mathscr{L}(x|\theta)$ using prior density $\Pr(\theta)$ and we come up with the following formula. Prior is employed when we need to estimate the conditional distribution of the parameters given observations.

$$\Pr(\theta|x_1, x_2, \ldots, x_p) = \frac{\mathscr{L}(x_1, x_2, \ldots, x_p|\theta)\ \Pr(\theta)}{\Pr(x_1, x_2, \ldots, x_p)}$$

or

$$\Pr(\theta|data) = \frac{\mathscr{L}(data|\theta)\ \Pr(\theta)}{\Pr(data)}$$

where

- $\Pr(\theta|data)$ is the posterior density,which gives us the information about the parameters given observed data.
- $\mathscr{L}(data|\theta)$ is the likelihood function, which estimates the probability of the data given parameters.
- $\Pr(\theta)$ is prior probability. It is the probability of the parameteres. Prior is basicallt the initial beliefs about estimates.
- Finally, Pr is the evidence, which is used to update the prior.

Consequently, Bayes' Theorem suggests that the posterior density is directly proportional to the prior and likelihood terms but inverserly related to the evidence term. As the evidence is there for scaling, we can describe this process as:

$$Posterior \propto Likelihood x prior$$

where $\propto$ means "is proportional to"

Within this context, Bayes' Theorem sounds attractive, doesn't it? Well, it does but it comes with a cost, which is analytical intractability. Even if Bayes' Theorem is theoretically intuitive, it is, by and large, hard to solve analytically. This is the major drawback in wide applicability of Bayes' Theorem. However, good news is that numerical methods provide solid methods to solve this probabilistic model.

So, some methods proposed to deal with the computational issue in Bayes' Theorem. These methods provides solution with approximation, which can be listed as:

- Quadrature approximation
- Maximum a posteriori estimation
- Grid Approach
- Sampling Based Approach
- Metropolis-Hastings
- Gibbs Sampler
- No U-Turn Sampler

Of these approaches, let us restrict our attention to the Metropolis-Hastings algorithm, which will be our method to be used in modeling Bayes' Theorem. Metropolis-Hastings (M-H) method is rest upon the Markov Chain Monte Carlo (MCMC). Alos, maximum a posteriori estimation will be discussed in the [Link to Come]. So, before moving forward, it would be better to talk about MCMC method.

Markov Chain Monte Carlo

Markov Chain is a model for us to describe the transition probabilities among states, which is a rule of a game. A chain is called Markovian if the

probability of current state s_t depends only on the most recent state s_{t-1}.

$$\Pr\left(s_t|s_{t-1}, s_{t-2}, \ldots, s_{t-p}\right) = \Pr\left(s_t|s_{t-1}\right)$$

Thus, MCMC relies on Markov Chain to find the parameter space θ with highest posterior probability. As the sample size grows, parameter values approximate to the posterior density.

$$\lim_{j \to +\infty} \theta^j \xrightarrow{D} \Pr\left(\theta|x\right)$$

where D refers to distributional approximation. Realized values of parameter space can be used to make inference about posterior. In a nutshell, MCMC method helps us to gather i.i.d sample from posterior density so that we can calculate the posterior probability.

To illustrate, we can refer to Figure 4-12. This figure tells us the probability of moving from one state to another. For the sake of simplicity, we set the probability to be 0.2 indicating, for instance, that transition from study to sleeping has a probability of 0.2.

```
In [67]: import quantecon as qe
         from quantecon import MarkovChain
         import networkx as nx
         from pprint import pprint

In [68]: P = [[0.5, 0.2, 0.3],
              [0.2, 0.3, 0.5],
              [0.2, 0.2, 0.6]]

         mc = qe.MarkovChain(P, ('studying', 'travelling', 'sleeping'))
         mc.is_irreducible
Out[68]: True

In [69]: states = ['studying', 'travelling', 'sleeping']
         initial_probs = [0.5, 0.3, 0.6]
         state_space = pd.Series(initial_probs, index=states, name='states')

In [70]: q_df = pd.DataFrame(columns=states, index=states)
         q_df = pd.DataFrame(columns=states, index=states)
         q_df.loc[states[0]] = [0.5, 0.2, 0.3]
         q_df.loc[states[1]] = [0.2, 0.3, 0.5]
```

```
        q_df.loc[states[2]] = [0.2, 0.2, 0.6]
```

```
In [71]: def _get_markov_edges(Q):
             edges = {}
             for col in Q.columns:
                 for idx in Q.index:
                     edges[(idx,col)] = Q.loc[idx,col]
             return edges
         edges_wts = _get_markov_edges(q_df)
         pprint(edges_wts)
         {('sleeping', 'sleeping'): 0.6,
          ('sleeping', 'studying'): 0.2,
          ('sleeping', 'travelling'): 0.2,
          ('studying', 'sleeping'): 0.3,
          ('studying', 'studying'): 0.5,
          ('studying', 'travelling'): 0.2,
          ('travelling', 'sleeping'): 0.5,
          ('travelling', 'studying'): 0.2,
          ('travelling', 'travelling'): 0.3}
```

```
In [72]: G = nx.MultiDiGraph()
         G.add_nodes_from(states)
         for k, v in edges_wts.items():
             tmp_origin, tmp_destination = k[0], k[1]
             G.add_edge(tmp_origin, tmp_destination, weight=v, label=v)

         pos = nx.drawing.nx_pydot.graphviz_layout(G, prog='dot')
         nx.draw_networkx(G, pos)
         edge_labels = {(n1, n2):d['label'] for n1, n2, d in G.edges(data=True)}
         nx.draw_networkx_edge_labels(G , pos, edge_labels=edge_labels)
         nx.drawing.nx_pydot.write_dot(G, 'mc_states.dot')
```

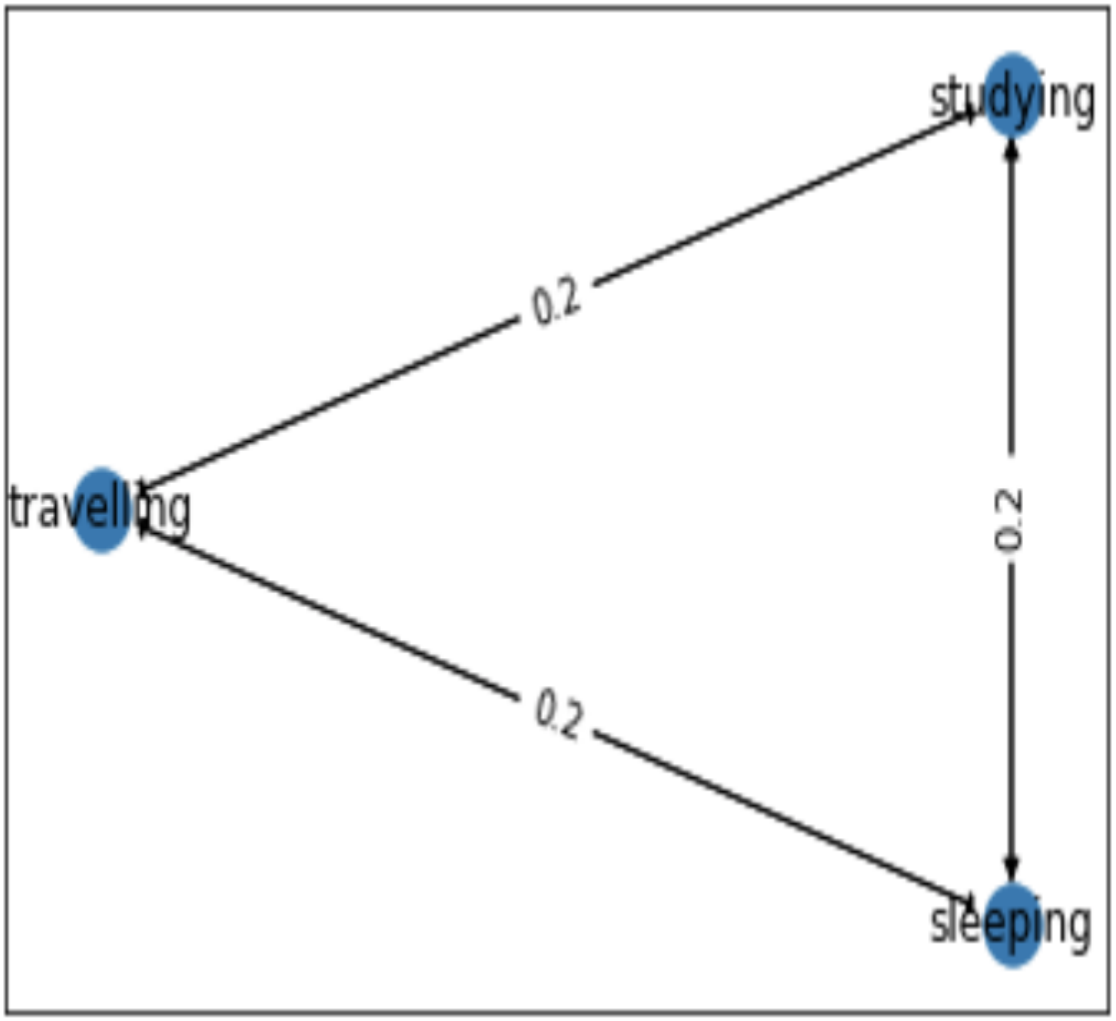


Figure 4-12. Interactions of different states

There are two common MCMC methods: Metropolis-Hastings and Gibbs Sampler. Here, we delve into the former one.

Metropolis-Hastings

Metropolis-Hastings (M-H) allows us to have efficient sampling procedure with two steps: First we draw sample from proposal density and, in the second step, we decide either to accept or reject.

Let $q\left(\theta \middle| \theta^{t-1}\right)$ be a proposal density and θ be a parameter space. The entire algorithm of M-H can be summarized as:

1. Select initial value for θ^1 from parameter space θ

2. Select a new parameter value θ^2 from proposal density, which can be, for the sake of easiness, Gaussian or Uniform distribution.

3. Compute the following acceptance probability:

$$\Pr_a\left(\theta^*, \theta^{t-1}\right) = min\left(1, \frac{p(\theta^*)/q(\theta^*|\theta^{t-1})}{p(\theta^{t-1})/q(\theta^{t-1}|\theta^*)}\right)$$

4. If $\Pr_a\left(\theta^*, \theta^{t-1}\right)$ is greater than a sample value drawn from uniform distribution U(0,1).

5. Repeat from step 2.

Well, it appears intimidating but don't be. We have built-in code in Python makes the applicability of the M-H algorithm way easier. We use *PyFlux* library to make use of Bayes' Theorem. Let's go and apply M-H algorithm to predict volatility.

```
In [73]: import pyflux as pf
         from scipy.stats import kurtosis

In [74]: model = pf.GARCH(ret.values, p=1, q=1)❶
         print(model.latent_variables)❷
         model.adjust_prior(1, pf.Normal())❸
         model.adjust_prior(2, pf.Normal())❸
         x = model.fit(method='M-H', iterations='1000')❹
         print(x.summary())
         Index    Latent Variable          Prior           Prior Hyperparameters
            V.I. Dist  Transform
         ======== ======================== ==============
          ======================== ========== ==========
         0        Vol Constant             Normal          mu0: 0, sigma0: 3
            Normal     exp
         1        q(1)                     Normal          mu0: 0, sigma0: 0.5
            Normal     logit
         2        p(1)                     Normal          mu0: 0, sigma0: 0.5
            Normal     logit
         3        Returns Constant         Normal          mu0: 0, sigma0: 3
            Normal     None
         Acceptance rate of Metropolis-Hastings is 0.00285
         Acceptance rate of Metropolis-Hastings is 0.2444
```

```
Tuning complete! Now sampling.
Acceptance rate of Metropolis-Hastings is 0.237925
GARCH(1,1)

============================================================
 ============================================
Dependent Variable: Series                              Method: Metropolis
 Hastings
Start Date: 1                                           Unnormalized Log
 Posterior: -3635.1999
End Date: 2913                                          AIC:
 7278.39981852521
Number of observations: 2913                            BIC:
 7302.307573553048
============================================================================
 ==================================
Latent Variable                          Median            Mean
     95% Credibility Interval
======================================== ================
 ================ ======================
Vol Constant                             0.0425            0.0425
     (0.0342 | 0.0522)
q(1)                                     0.1966            0.1981
     (0.1676 | 0.2319)
p(1)                                     0.7679            0.767
     (0.7344 | 0.7963)
Returns Constant                         0.0854            0.0839
     (0.0579 | 0.1032)
============================================================================
 ==================================
None
```

```
In [75]: model.plot_z([1, 2])❺
         model.plot_fit(figsize=(15, 5))❻
         model.plot_ppc(T=kurtosis, nsims=1000)❼
```

❶ Configuring GARCH model using PyFlux library.

❷ Printing the estimation of latent variables (parameters).

❸ Adjusting the priors for the model latent variables.

❹ Fitting the model using M-H process.

❺ Plotting the latent variables.

❻ Plotting the fitted model.

❼ Plotting the histogram for posterior check.

It is worthile to visualize the results of what we have done so far for volatility prediction with Bayesian-based Garch Model.

Figure 4-13 exhibits the distribution of latent variables. Latent variable q gathers around 0.2 and the other latent variable p mostly takes values between 0.7 and 0.8.

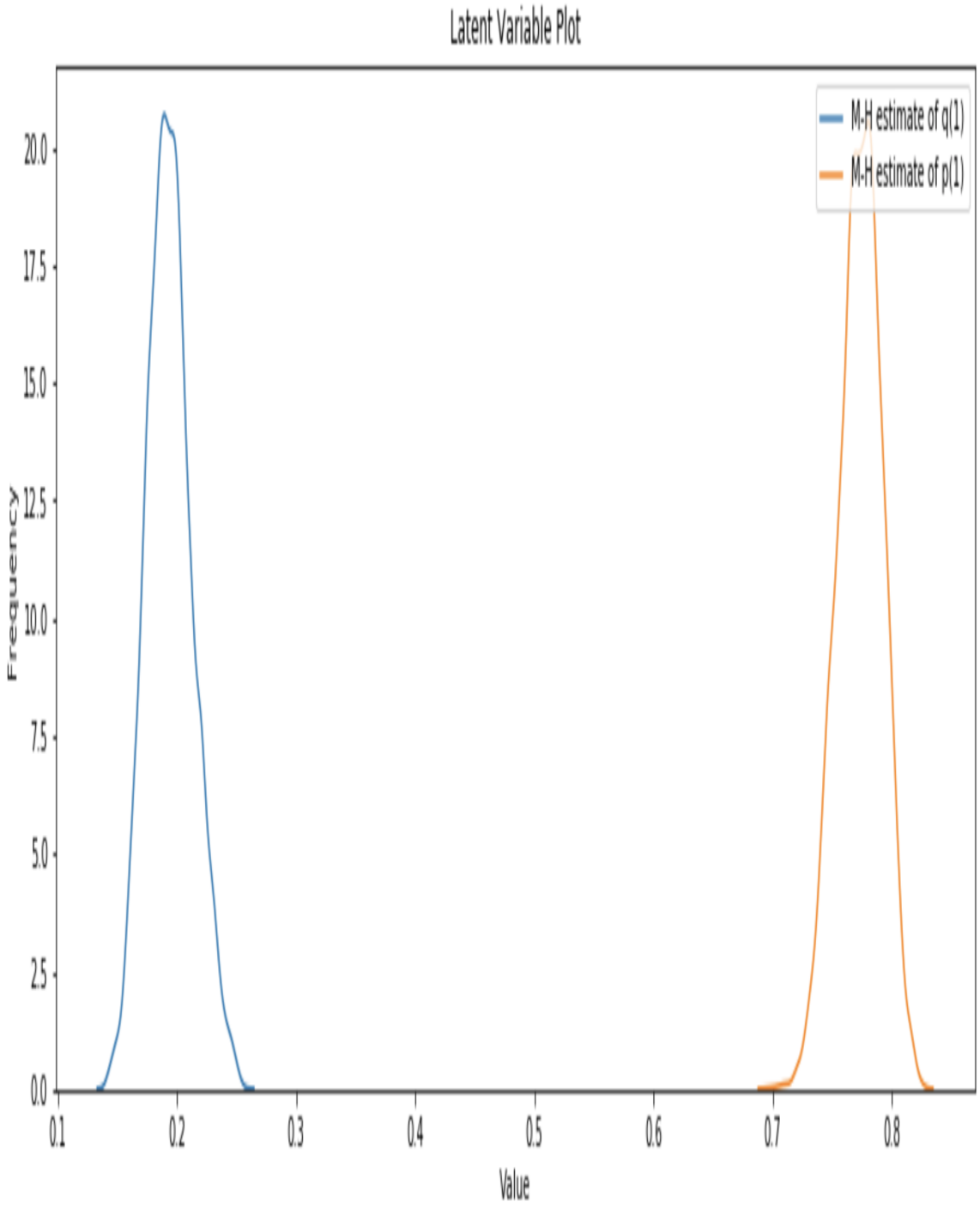


Figure 4-13. Latent Variables

Figure 4-14 indicates the demeaned volatility series and the GARCH prediction result based on Bayesian approach.

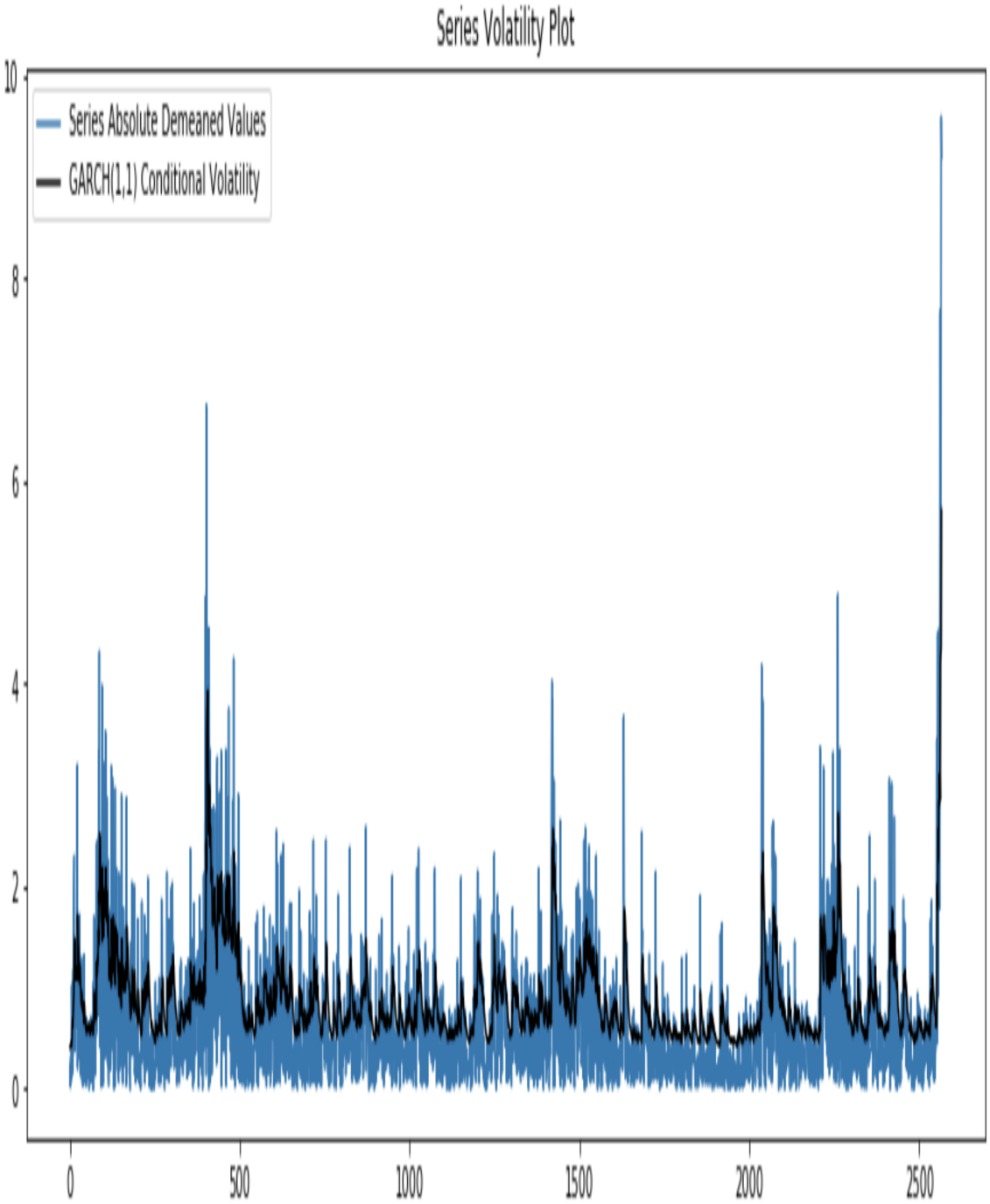


Figure 4-14. Model Fit

Figure 4-15 visualizes the posterior predictions of the Bayesian model with the data so that we are able to detect systematic discrepancies, if any. The

vertical line represents the test statistic and it turns out the observed value is larger than that of our model.

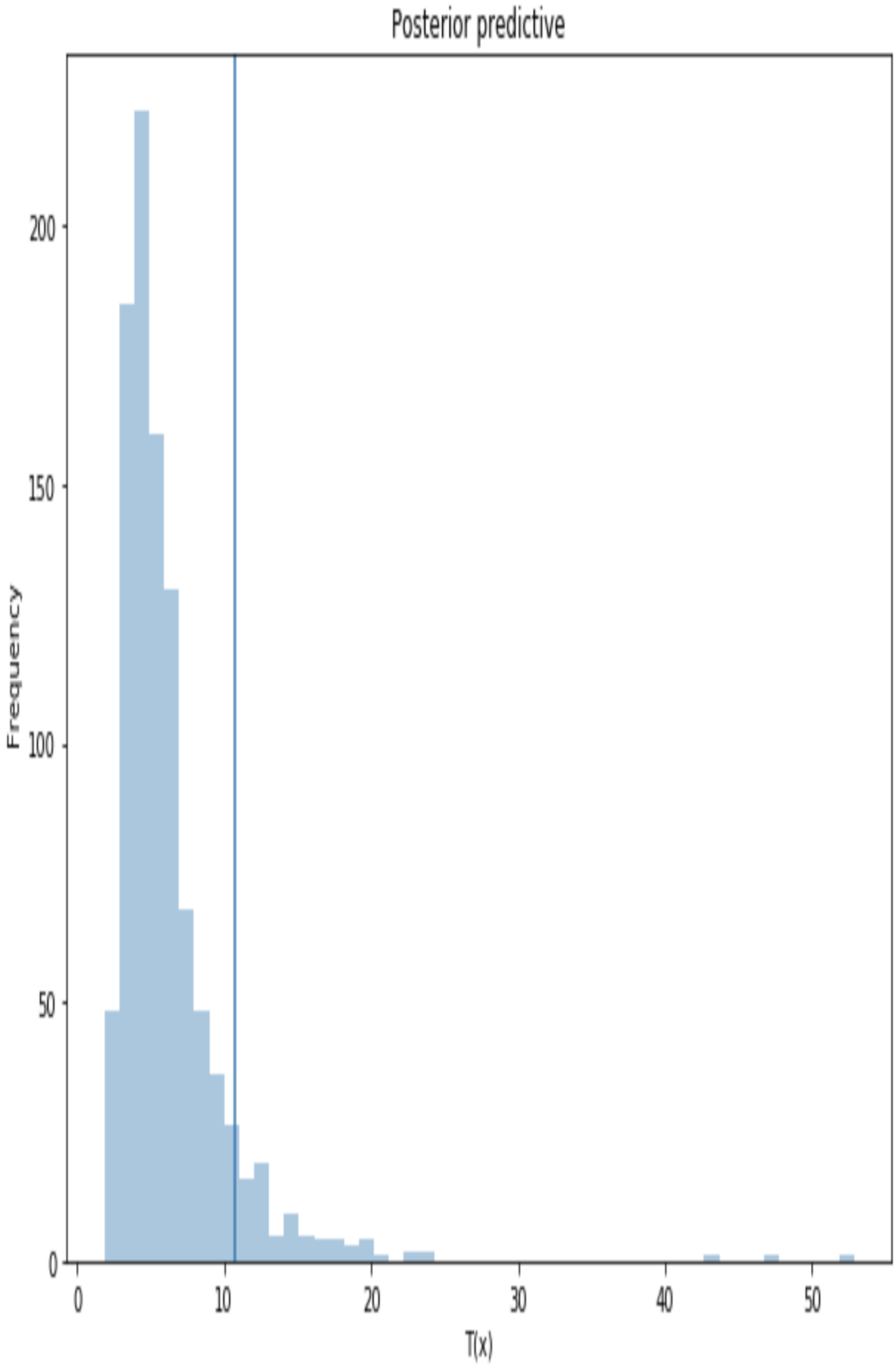

Posterior predictive
Frequency
0
50
100
150
200
0
10
20
30
40
50
T(x)

Figure 4-15. Posterior Prediction

After we are done with the training part, we all set to move on to the next phase, which is prediction. Prediction analysis is done for the 252 step ahead and the RMSE is calculated given the realized volatility.

```
In [76]: bayesian_prediction = model.predict_is(n, fit_method='M-H')❶
         Acceptance rate of Metropolis-Hastings is 0.1127
         Acceptance rate of Metropolis-Hastings is 0.17795
         Acceptance rate of Metropolis-Hastings is 0.2532

         Tuning complete! Now sampling.
         Acceptance rate of Metropolis-Hastings is 0.255425

In [77]: bayesian_RMSE = np.sqrt(mean_squared_error(realized_vol.iloc[-n:] / 100,
                                  bayesian_prediction.values / 100))❷
         print('The RMSE of Bayesian model is {:.6f}'.format(bayesian_RMSE))
         The RMSE of Bayesian model is 0.004090

In [78]: bayesian_prediction.index = ret.iloc[-n:].index

In [79]: plt.figure(figsize=(10, 6))
         plt.plot(realized_vol / 100,
                  label='Realized Volatility')
         plt.plot(bayesian_prediction['Series'] / 100,
                  label='Volatility Prediction-Bayesian')
         plt.title('Volatility Prediction with M-H Approach', fontsize=12)
         plt.legend()
         plt.savefig('images/bayesian.png')
         plt.show()
```

❶ In-sample volatility prediction.

❷ Calculating the RMSE score.

Eventually, we are ready to observe the prediction result of the Bayesian approach and the following code does it for us.

Figure 4-16 visualizes the volatility prediction based on Metropolis-Hasting based Bayesian approach and it seems to overshot towards the end of 2020

and overall performance of this method shows that it is not among the best methods as it just outperforms SVR-GARCH with polynomial kernel.

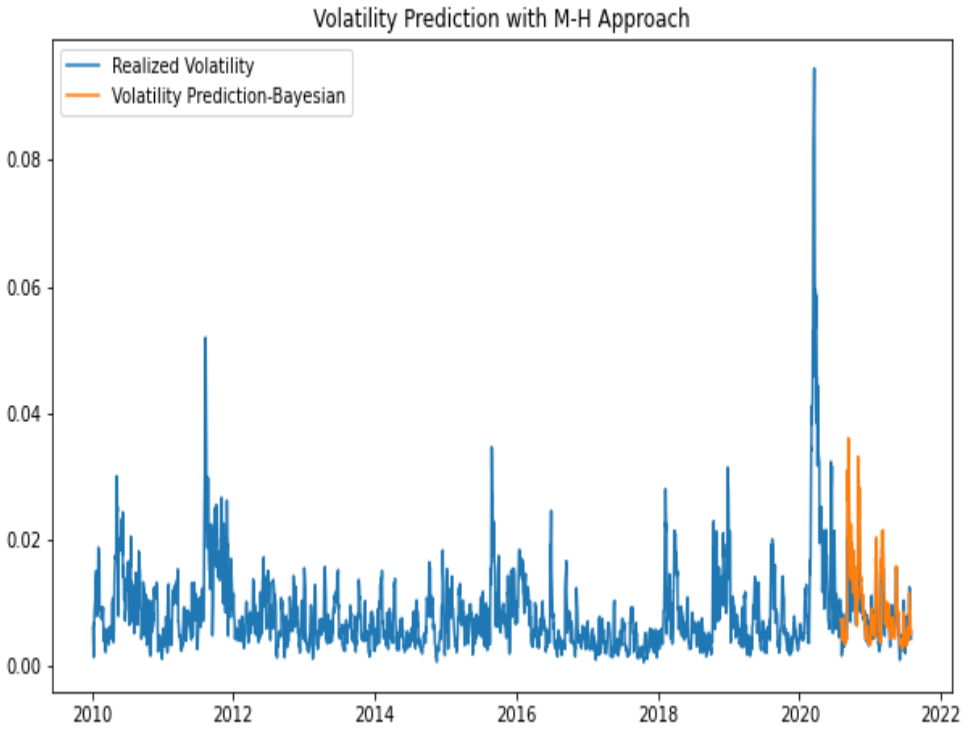


Figure 4-16. Bayesian Volatility Prediction

Conclusion

Volatility prediction is a key to understand the dynamics of financial market in the sense that it helps us to gauge the uncertainty. With that being said, it is used as input in many financial model including risk models. These facts emphasize the importance of having accurate volatility prediction. Traditionally, parametric methods such ARCH, GARCH and their extensions have been extensively used but these models suffer from being inflexible. To remedy this issue, data-driven models are found promising and this chapter attempts to make use of these models, namely, Support Vector Machines, Neural Network, and Deep Learning-based models, and it turns out data-driven model outperforms the parametric models.

In the next chapter, market risk, a core financial risk topic, will be discussed both from theoretical and empirical standpoints and the machine learning models will be incorporate to further improve the estimation of this risk.

Further Resources

Articles cited in this chapter:

- Andersen, Torben G., Tim Bollerslev, Francis X. Diebold, and Paul Labys. "Modeling and forecasting realized volatility." Econometrica 71, no. 2 (2003): 579-625.
- Andersen, Torben G., and Tim Bollerslev. "Intraday periodicity and volatility persistence in financial markets." Journal of empirical finance 4, no. 2-3 (1997): 115-158.
- Black, Fischer. "Studies of stock market volatility changes." 1976 Proceedings of the American statistical association bisiness and economic statistics section (1976).
- Burnham, Kenneth P., and David R. Anderson. "Multimodel inference: understanding AIC and BIC in model selection." Sociological methods & research 33, no. 2 (2004): 261-304.
- Eagle, Robert F. "Autoregressive conditional heteroskedasticity with estimates of the variance of UK inflation." Econometrica 50, no. 4 (1982): 987-1008.
- De Stefani, Jacopo, Olivier Caelen, Dalila Hattab, and Gianluca Bontempi. "Machine Learning for Multi-step Ahead Forecasting of Volatility Proxies." In MIDAS@ PKDD/ECML, pp. 17-28. 2017.
- Dokuchaev, Nikolai. "Volatility estimation from short time series of stock prices." Journal of Nonparametric statistics 26, no. 2 (2014): 373-384.
- Karasan, Abdullah, and Esma Gaygisiz. "Volatility Prediction and Risk Management: An SVR-GARCH Approach." The Journal of

Financial Data Science 2, no. 4 (2020): 85-104.

- Mandelbrot, Benoit. “New methods in statistical economics.” Journal of political economy 71, no. 5 (1963): 421-440.
- Raju, M. T., and Anirban Ghosh. “Stock market volatility–An international comparison.” Securities and Exchange Board of India (2004).

Books cited in this chapter:

- Alpaydin, E., 2020. Introduction to machine learning. MIT press.
- Burnham, Kenneth P., and David R. Anderson. “A practical information-theoretic approach.” Model selection and multimodel inference (2002).
- Focardi, Sergio M. Modeling the market: New theories and techniques. Vol. 14. John Wiley & Sons, 1997.
- Rachev, Svetlozar T., John SJ Hsu, Biliana S. Bagasheva, and Frank J. Fabozzi. Bayesian methods in finance. Vol. 153. John Wiley & Sons, 2008.
- Wilmott, Paul. Machine Learning-An Applied Mathematics Introduction. John Wiley & Sons, 2013.

---

1 Conditional variance means that volatility estimation is a function of the past values of asset returns.

2 For more information please see this note: *http://cs229.stanford.edu/notes/cs229-notes3.pdf*

3 Occam’s Razor, also known as law of parsimony, states that given a set of explanations, simpler explanation is the most plausible and likely one.

4 Of these alternatives, Tensorflow, PyTorch, and Neurolab are the most prominent libraries.

5 Please see this manual: *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html*

Chapter 5. Market Risk

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 5th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at mcronin@oreilly.com.

Risk is ubiquitous in finance but it is hard to quantify. The first and foremost thing to know is differentiating the sources of financial risks on the grounds that it might not be a wise move to utilize same tools against financial risk arising from different sources.

Thus, treating different sources of financial risk is crucial as the impact of different financial risks as well as tools developed to mitigate risk are completely different. Pretending that firms are subject to large market fluctuations, all assets in the portfolio of the firms are susceptible to risk originating from these fluctuations. However, a different tool should be developed to cope with a risk emanating from customer profile. In addition, it should be kept in mind that different risk factors contribute significantly to the asset prices. All these examples implies that assessing risk factors need careful consideration in finance.

As is briefly discussed previously, these are, mainly, market, credit, liquidity, and operational risks. It is evident that some other types can be added to this list of financial risks but they can be thought of a subset of these

main four risk types. So, these types of risks will be our focus throughout this chapter.

Market risk is the risk arising from changes on financial indicators such as exchange rate, interest rate, inflation and so on. Differently, market risk can be referred to a risk of losses in on and off-balance-sheet positions arising from movements in market prices (BIS, 2020). Let us now see how these factors affect market risk. Suppose that a raise in inflation rate might pose a threat to current profitability of the financial institutions with a view to inflation creates pressure to interest rate. This, in turn, affects borrower's cost of funds. These instances can be amplified but we should also note the interactions of these financial risk sources. That is, while a single source of financial risk changes, other risk sources cannot stay constant. With that being said, to some extent, financial indicators are interrelated, meaning that interactions of these risks sources should be taken into account.

As you can imagine, there are different tools to measure market risk. Of them, the most prominent and widely acceptable tools are Value-at-Risk (VaR) and Expected Shortfall (ES). The ultimate aim of this chapter is to augment these approaches using recent developments in Machine Learning. At this juncture, it would be tempting to ask: Does the traditional model fail in finance? And what makes the ML-based model different?

I will start tackling the first question. The first and foremost challenge that traditional models unable to address is the complexity of the financial system. Due either to the some strong assumptions or simply inability to capture the complexity introduced by the data, long-standing traditional models has been started to be replaced by ML-based models.

This fact is well put by Prado:

> *Considering the complexity of modern financial systems, it is unlikely that a researcher will be able to uncover the ingredients of a theory by visual inspection of the data or by running a few regressions.*
>
> —Prado (Machine Learning for Asset Managers, 2020, p.4)

To address the second question, it would be wise to think about the working logic of ML models. ML models, as opposed to old statistical methods, tries to unveil the association between variables, identify key variables, and enable us to to find out the impact of the variables on the dependent variable without a need of a well established theory. This is, in fact, the beauty of ML model as ML models, without requiring many restrictive and strong assumptions, allow us to discover theories, let alone require them.

> *Many methods from statistics and machine learning (ML) may, in principle, be used for both prediction and inference. However, statistical methods have a long-standing focus on inference, which is achieved through the creation and fitting of a project-specific probability model... By contrast, ML concentrates on prediction by using general-pur- pose learning algorithms to find patterns in often rich and unwieldy data.*
>
> —Bzdok (Statistics versus machine learning, 2018, p.232)

In the following part, we initiate our discussion on the market risk models. We, first, talk about the application of Value-at-Risk (VaR) and Expected Shortfall (ES). Subsequent to the traditional application of these models, we will learn how to them using the ML-based approach. Let's jump in.

Value-at-Risk

The emergence of VaR model rests upon a request for a JP Morgan executive who wanted to have a summary report showing possible losses as well as risks in one day that JP Morgan is exposed to. In this report, executives are informed about the risk assumed by the institution in an aggregated manner. The method by which market risk is computed is known as VaR. So, it is the starting point of VaR and, now, it has become so widespread that its adoption has been forced by regulators.

The adoption of VaR dates back to 1990s and despite numerous extensions to VaR and new proposed models, it is still in use. So, what makes it so appealing? maybe the question to be addressed. The answer comes from Kevin Dowd:

The VaR figure has two important characteristics. The first is that it provides a common consistent measure of risk across different positions and risk factors. It enables us to measure the risk associated with a fixed-income position, say, in a way that is comparable to and consistent with a measure of the risk associated with equity positions. VaR provides us with a common risk yardstick, and this yardstick makes it possible for institutions to manage their risks in new ways that were not possible before. The other characteristic of VaR is that it takes account of the correlations between different risk factors. If two risks offset each other, the VaR allows for this offset and tells us that the overall risk is fairly low.

—Dowd (2002, p.10)

In fact, VaR basically address one of the most common question of an investor:

Given the risk level, what is the maximum expected loss of my investment?

VaR provides a very intuitive and practical answer to this question. In this regard, it is used to measure the worst expected loss of a company over a given period and a pre-defined confidence interval. Suppose that a daily VaR of an investment is $10 million with 95% confidence interval. This reads as: There is a 5% chance that investor can incur a loss greater than $10 million loss in a day.

Based on the above-given definition, it turns out that the components of VaR are confidence interval, time period, value of an asset or portfolio and standard deviation as we are talking about risk.

In summary, there are some important points in VaR analysis that needs to be highlighted:

- VaR needs an estimation of the probability of loss
- VaR concentrates on the potential losses. We are not talking about actual or realized losses rather VaR is a kind of loss projection
- VaR has three key ingredients:

- Standard deviation that defines the level of loss
- Fixed time horizon over which risk is assessed
- Confidence Interval

Well, VaR can be measured via three different approaches:

- Variance-Covariance VaR
- Historical Simulation VaR
- Monte-Carlo VaR

Variance-Covariance Method

Variance-Covariance Method is also known as parametric method, because the data is assumed to be normally distributed. Variance-Covariance method is commonplace due to this assumption, however it is worth noting that returns are not normal distributed. Parametric form assumption makes the application of Variance-Covariance method practical and easy-to-apply.

As in all VaR approaches, we can either work with single asset or a portfolio. However, working with portfolio requires careful treatment in the sense that correlation structure and portfolio variance need to be estimated. Exactly at this point, correlation comes into the picture and historical data is used to calculate correlation, mean, and standard deviation. While augmenting this with ML-based approach, correlation structure will be our main focus.

Suppose that we have a single asset, in Figure 5-1, it is shown that the mean of this asset is zero and standard deviation is 1 and if the holding period is 1, the corresponding VaR value can be computed the value of the asset by corresponding Z-value and standard deviation. Hence, normality assumption makes things easier but it is a strong assumption because there is no guarantee that asset returns are normally distributed rather most of the asset returns do not follow normal distribution. Moreover, due to the normality

assumption, potential risk in tail might not be captured. Therefore normality assumption comes with a cost.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import datetime
        import yfinance as yf
        from scipy.stats import norm
        import requests
        from io import StringIO
        import seaborn as sns; sns.set()
        import warnings
        warnings.filterwarnings('ignore')
        plt.rcParams['figure.figsize'] = (10,6)

In [2]: mean = 0
        std_dev = 1
        x = np.arange(-5, 5, 0.01)
        y = norm.pdf(x, mean, std_dev)
        pdf = plt.plot(x, y)
        min_ylim, max_ylim = plt.ylim()
        plt.text(np.percentile(x, 5), max_ylim * 0.9, '95%:${:.4f}'
                 .format(np.percentile(x, 5)))
        plt.axvline(np.percentile(x, 5), color='r', linestyle='dashed', linewidth=4)
        plt.title('Value-at-Risk Illustration')
        plt.savefig('images/VaR_Illustration.png')
        plt.show()
In [3]: mean = 0
        std_dev = 1
        x = np.arange(-5, 5, 0.01)
        y = norm.pdf(x, mean, std_dev)❶
        pdf = plt.plot(x, y)
        min_ylim, max_ylim = plt.ylim()❷
        plt.text(np.percentile(x, 5), max_ylim * 0.9, '95%:${:.4f}'
                 .format(np.percentile(x, 5)))❸
        plt.axvline(np.percentile(x, 5), color='r', linestyle='dashed', linewidth=4)
        plt.title('Value-at-Risk Illustration')
        plt.savefig('images/VaR_Illustration.png')
        plt.show()
```

❶ Generating probability density function based on given x, mean, and standard deviation.

❷ Limiting the x-axis and y-axis.

❸ Specifying the location of x at 5% percentile of the x data.

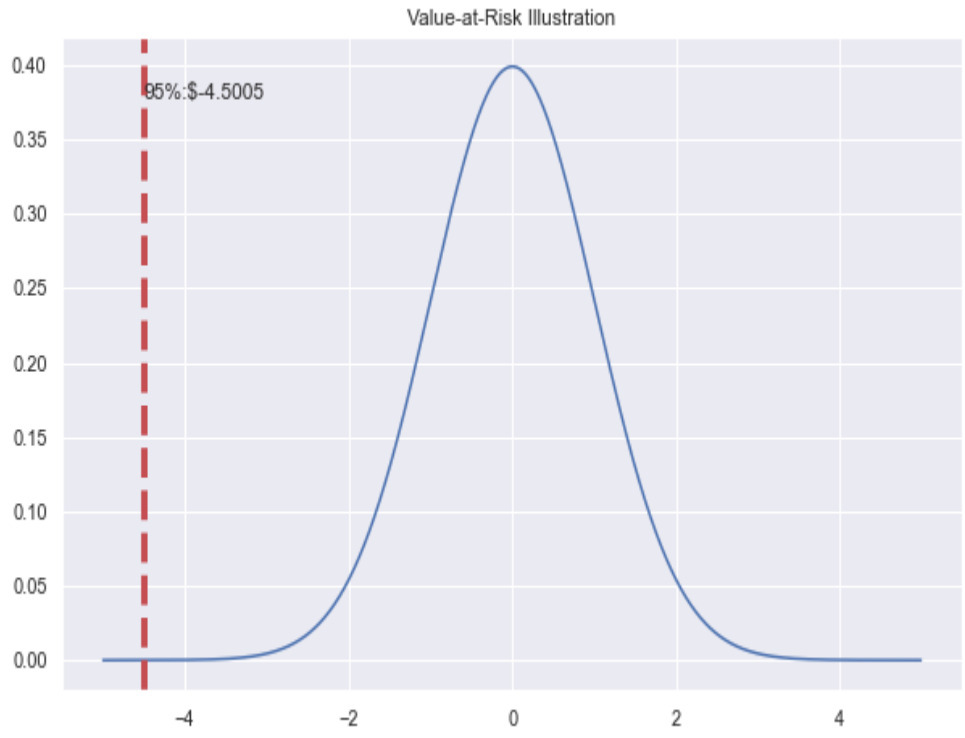


Figure 5-1. Value-at-Risk Illustration

> **NOTE**
>
> Following Fama (1965), it is realized that some stock price returns do not follow normal distribution due to fat tail and asymmetry. Rather they follow leptokurtic distribution. This empirical observation implies stock returns have higher kurtosis than that of normal distribution.
>
> Having high kurtosis amounts to fat-tail and this might results in extreme negative returns. As the Variance-Covariance method is unable to capture fat-tail, it cannot, therefore, estimate extreme negative return, which is likely to occur especially in the crisis period.

Let us see how we apply Variance-Covariance VaR in Python. To illustrate, 2-asset portfolio is considered and the formula of Variance-Covarince VaR is as follows:

$$\text{VaR} = V\sigma_p\sqrt{t}Z_\alpha$$

$$\sigma_p = \sqrt{w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + \rho w_1 w_2 \sigma_1 \sigma_2}$$

$$\sigma_p = \sqrt{w_1\sigma_1 + w_2 + \sigma + 2w_1 w_2 \sum_{1,2}}$$

```
In [4]: def getDailyData(symbol):
                parameters = {'function': 'TIME_SERIES_DAILY_ADJUSTED',
                              'symbol': symbol,
                              'outputsize':'full',
                              'datatype': 'csv',
                              'apikey': 'LL1WA15IW41XV2T2'}❶

                response = requests.get('https://www.alphavantage.co/query',
                                        params=parameters)❷

                # Process the CSV file retrieved
                csvText = StringIO(response.text)❸
                data = pd.read_csv(csvText, index_col='timestamp')
                return data

In [5]: symbols = ["IBM", "MSFT", "INTC"]
        data3 = []
        for symbol in symbols:
            data3.append(getDailyData(symbol)[::-1]['close']['2020-01-01': '2020-12-
31'])❹
        stocks = pd.DataFrame(data3).T
        stocks.columns = symbols

In [6]: stocks.head()
Out[6]:               IBM    MSFT   INTC
        timestamp
        2020-01-02  135.42  160.62  60.84
        2020-01-03  134.34  158.62  60.10
        2020-01-06  134.10  159.03  59.93
        2020-01-07  134.19  157.58  58.93
        2020-01-08  135.31  160.09  58.97
```

❶ Identifying the parameters to be used in extracting the data from Alpha Vantage.

❷ Making a request to Alpha Vantage website.

❸ Open the response file, which is in the text format.

❹ Reversing the data that covers the period of examination and append the daily stock prices of IBM, MSFT, and INTC.

> **NOTE**
>
> Alpha Vantage is a data providing company partnered with major exchanges and institutions. Using Alpha Vantage API, it is possible to access stock prices with different time interval- i.e., intraday, daily, weekly, and so on, stock fundamentals, and Forex information. For more information, please see Alpha Vantage's website.

```
In [7]: stocks_returns = (np.log(stocks) - np.log(stocks.shift(1))).dropna()❶
        stocks_returns
Out[7]:                 IBM      MSFT      INTC
        timestamp
        2020-01-03 -0.008007 -0.012530 -0.012238
        2020-01-06 -0.001788  0.002581 -0.002833
        2020-01-07  0.000671 -0.009160 -0.016827
        2020-01-08  0.008312  0.015803  0.000679
        2020-01-09  0.010513  0.012416  0.005580
        ...              ...       ...       ...
        2020-12-24  0.006356  0.007797  0.010679
        2020-12-28  0.001042  0.009873  0.000000
        2020-12-29 -0.008205 -0.003607  0.048112
        2020-12-30  0.004352 -0.011081 -0.013043
        2020-12-31  0.012309  0.003333  0.021711

        [252 rows x 3 columns]

In [8]: stocks_returns_mean = stocks_returns.mean()
        weights  = np.random.random(len(stocks_returns.columns))❷
        weights /= np.sum(weights)❸
        cov_var = stocks_returns.cov()❹
        port_std = np.sqrt(weights.T.dot(cov_var).dot(weights))❺
```

```
In [9]: initial_investment = 1e6
        conf_level = 0.95

In [10]: def VaR_parametric(initial_investment, conf_level):
             alpha = norm.ppf(1 - conf_level, stocks_returns_mean, port_std)❻
             for i, j in zip(stocks.columns, range(len(stocks.columns))):
                 VaR_param = (initial_investment - initial_investment * (1 + alpha))
[j]❼
                 print("Parametric VaR result for {} is {} ".format(i, VaR_param))
             VaR_param = (initial_investment-initial_investment * (1 + alpha))
             print('--' * 25)
             return VaR_param

In [11]: VaR_param = VaR_parametric(initial_investment, conf_level)
         VaR_param
         Parametric VaR result for IBM is 42199.839069714886
         Parametric VaR result for MSFT is 40618.179754271754
         Parametric VaR result for INTC is 42702.930219301255
         --------------------------------------------------

Out[11]: array([42199.83906971, 40618.17975427, 42702.9302193 ])
```

❶ Calculating logarithmic return.

❷ Drawing random numbers for weights.

❸ Generating weights.

❹ Calculating covariance matrix.

❺ Finding the portfolio standard deviation.

❻ Computing the Z-score for a specific value using percent point function (ppf).

❼ Estimating the Variance-Covariance VaR model.

Given the time horizon, the result of the Value-at-Risk changes due to the fact that holding asset for a longer period makes investor more susceptible to risk. As it is shown in the Figure 5-2, VaR increases in relation to holding

time by the amount of $\sqrt{t}$. Additionally, the holding period is the longest for portfolio liquidation. Taking into account the reporting purpose, a 30-day period may be more suitable one for an investor and it is illustrated in Figure 5-2.

```
In [12]: var_horizon = []
         time_horizon = 30
         for j in range(len(stocks_returns.columns)):
             for i in range(1, time_horizon + 1):
                 var_horizon.append(VaR_param[j] * np.sqrt(i))
         plt.plot(var_horizon[:time_horizon], "o",
                  c='blue', marker='*', label='IBM')
         plt.plot(var_horizon[time_horizon:time_horizon + 30], "o",
                  c='green', marker='o', label='MSFT')
         plt.plot(var_horizon[time_horizon + 30:time_horizon + 60], "o",
                  c='red', marker='v', label='INTC')
         plt.xlabel("Days")
         plt.ylabel("USD")
         plt.title("VaR over 30-day period")
         plt.savefig('images/VaR_30_day.png')
         plt.legend()
         plt.show()
```

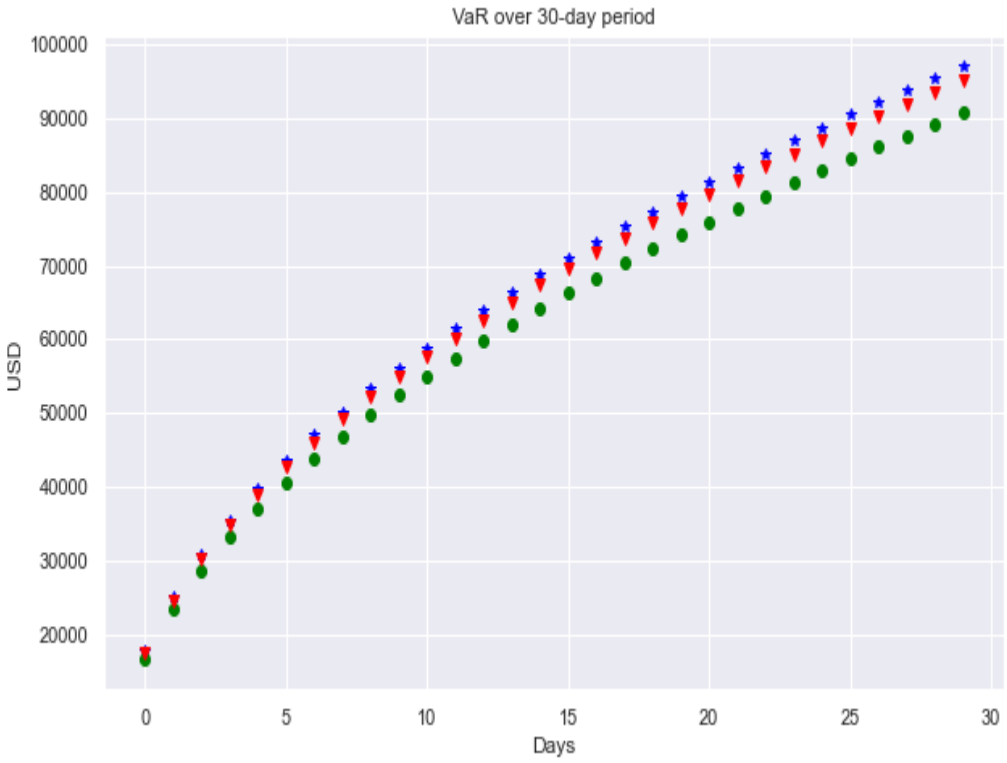


Figure 5-2. Value-at-Risk over Different Horizon

We conclude this part listing the pros and cons of Variance-Covariance method.

Pros

- Easy to calculate
- It does not require large number of samples

Cons

- Observations are normally distributed
- It does not work well with non-linear structure
- Requires computation of covariance matrix

So, even though assuming normality sounds appealing, it may not be the best way to estimate VaR especially in the case where the asset return do not have

normal distribution. Luckily, there are another methods not requiring normality assumption and the models is known as *Historical simulation VaR* model.

Historical-Simulation Method

Having strong assumption, like normal distribution, might be the cause of inaccurate estimation. A solution to this issue is referred to as Historical Simulation VaR. This is an empirical method and instead of using parametric approach, what we do is to find the percentile, which is the Z-table equivelant of Variance-Covariance method. Pretend that the confidence interval is 95%, then 5% will be used in liue of Z-table value and all we need to do is to multiply this percentile by initial investment.

Here are the steps taken in Historical Simulation VaR:

- Obtain asset returns of the portfolio (or individual asset).
- Find the corresponding return percentile based on confidence interval.
- Multiply this percentile by initial investment.

```
In [13]: def VaR_historical(initial_investment, conf_level):
             Hist_percentile95 = []
             for i, j in zip(stocks_returns.columns, ❶
                             range(len(stocks_returns.columns))):
                 Hist_percentile95.append(np.percentile(stocks_returns.loc[:, i],
5))
                 print("Based on historical values 95% of {}'s return is {:.4f}"
                       .format(i, Hist_percentile95[j]))
                 VaR_historical = (initial_investment-initial_investment *
                                   (1 + Hist_percentile95[j]))
                 print("Historical VaR result for {} is {:.2f} "
                       .format(i, VaR_historical))
                 print('--' * 35)

In [14]: VaR_historical(initial_investment,conf_level) ❷
         Based on historical values 95% of IBM's return is -0.0371
         Historical VaR result for IBM is 37081.53
         ----------------------------------------------------------------------
         Based on historical values 95% of MSFT's return is -0.0426
```

```
Historical VaR result for MSFT is 42583.68
----------------------------------------------------------------------
Based on historical values 95% of INTC's return is -0.0425
Historical VaR result for INTC is 42485.39
----------------------------------------------------------------------
```

❶ Calculating the 95% percentile of stock returns

❷ Estimating the historical simulation VaR

Historical simulation VaR method implicitly assumes that historical price changes have similar pattern, i.e., there is no structural break. The pros and cons of this method are as follows:

Pros

- No distributional assumption
- Work well with non-linear structure
- Easy to calculate

Cons

- Require large sample
- In need of high computing power
- Mislead if a company subject to ambiguity like growth company stocks

Monte Carlo-Simulation VaR

Before delving into the Monte Carlo simulation VaR estimation, it would be better to make a brief introduction about Monte Carlo simulation. Monte Carlo is a computerized mathematical method used to make an estimation in the case where there is no closed-form solution. So, it is a highly efficient tool for numerical approximation. Monte Carlo relies on repeated random sample form a given distribution.

The logic behind Monte Carlo is well-defined by Glasserman:

> *Monte Carlo methods are based on the analogy between probability and volume. The mathematics of measure formalizes the intuitive notion of probability, associating an event with a set of outcomes and defining the probability of the event to be its volume or measure relative to that of a universe of possible outcomes. Monte Carlo uses this identity in reverse, calculating the volume of a set by interpreting the volume as a probability.*
>
> —Glasserman (Monte Carlo Methods in Financial Engineering, 2003, p.11)

From the application standpoint, Monte Carlo is very similar to Historical Simulation VaR but it does not use historical observations. Rather, it generates random samples from a given distribution. So, Monte Carlo helps decision makers by providing link between possible outcomes and probabilities, which makes it a efficient and applicable tool in finance.

Mathematical Monte Carlo can be defined as:

Let $X_1, X_2, \cdots \ldots X_n$ are independent and identically distributed random variables and f(x) is a real-valued function. Then, Law of Large Number states that:

$$\mathsf{E}(f(X)) \approx \frac{1}{N} \sum_i^N f(X_i)$$

In a nutshell, Monte Carlo simulation is nothing but generating random samples and calculating its mean. Computationally, it follows the following steps:

- Define the domain
- Generate random numbers
- Iteration and aggregation the result

Determination of mathematical π is a toy but illustrative example for Monte Carlo application.

Suppose we have a circle with radius r = 1 and area=4. Area of a circle is π and area of a square in which we try to fit circle is 4. The ratio turns out to be:

$\frac{\pi}{4}$ eq.1

To leave π alone, the proportion between circle and area can be defined as:

$$\frac{Circumference_{circle}}{Area_{square}} = \frac{m}{n} \text{ eq.2}$$

Once we equalize `eq.1` and `eq.2`, it turns out:

$$\pi = 4x\frac{m}{n}$$

If we go step by step, the first one is to define domain which is [-1,1]. So, the numbers inside the circle satisfy: $x^2 + y^2 \leq 1$.

Second step is to generate random numbers to meet the above-given condition. That is to say, we need to have uniformly distributed random samples, which is a rather easy task in Python. For the sake of practice, I will generate 100 uniformly distributed random numbers by using numpy library:

```
In [15]: x = np.random.uniform(-1, 1, 100) ❶
         y = np.random.uniform(-1, 1, 100)

In [16]: sample = 100
         def pi_calc(x, y):
             point_inside_circle = 0
             for i in range(sample):
                 if np.sqrt(x[i] ** 2 + y[i] ** 2) <= 1: ❷
                     point_inside_circle += 1
             print('pi value is {}'.format(4 * point_inside_circle/sample))

In [17]: pi_calc(x,y)
         pi value is 3.36

In [18]: x = np.random.uniform(-1, 1, 1000000)
         y = np.random.uniform(-1, 1, 1000000)

In [19]: sample = 1000000

         def pi_calc(x, y):
             point_inside_circle = 0
```

```
            for i in range(sample):
                if np.sqrt(x[i] ** 2 + y[i] ** 2) < 1:
                    point_inside_circle += 1
            print('pi value is {:.2f}'.format(4 * point_inside_circle/sample))
```

```
In [20]: sim_data = pd.DataFrame([])
         num_reps = 1000
         mean = np.random.random(3)
         std = np.random.random(3)
         for i in range(len(stocks.columns)):
             temp = pd.DataFrame(np.random.normal(mean[i], std[i], num_reps))
             sim_data = pd.concat([sim_data, temp],axis=1)
         sim_data.columns = ['Simulation 1', 'Simulation 2', 'Simulation 3']
```

```
In [21]: sim_data
Out[21]:      Simulation 1  Simulation 2  Simulation 3
         0        0.475188      1.587426      1.334594
         1        0.547615     -0.169073      0.684107
         2        0.486227      1.533680      0.755307
         3        0.494880     -0.230544      0.471358
         4        0.580477      0.388032      0.493490
         ..            ...           ...           ...
         995      0.517479      0.387384      0.539328
         996      0.286267      0.680610      0.409003
         997      0.398601      0.733176      0.820090
         998      0.624548      0.482050      0.821792
         999      0.627089      1.405230      1.275521

         [1000 rows x 3 columns]
```

```
In [22]: def MC_VaR(initial_investment, conf_level):
             MC_percentile95 = []
             for i, j in zip(sim_data.columns, range(len(sim_data.columns))):
                 MC_percentile95.append(np.percentile(sim_data.loc[:, i], 5))❸
                 print("Based on simulation 95% of {}'s return is {:.4f}"
                       .format(i, MC_percentile95[j]))
                 VaR_MC = (initial_investment-initial_investment *
                           (1 + MC_percentile95[j]))❹
                 print("Simulation VaR result for {} is {:.2f} ".format(i, VaR_MC))
                 print('--'*35)
```

```
In [23]: MC_VaR(initial_investment, conf_level)
         Based on simulation 95% of Simulation 1's return is 0.3294
         Simulation VaR result for Simulation 1 is -329409.17
         ----------------------------------------------------------------------
         Based on simulation 95% of Simulation 2's return is 0.0847
         Simulation VaR result for Simulation 2 is -84730.05
         ----------------------------------------------------------------------
```

```
Based on simulation 95% of Simulation 3's return is 0.1814
Simulation VaR result for Simulation 3 is -181376.94
----------------------------------------------------------------
```

❶ Generating random numbers from uniform distribution.

❷ Checking if points are inside the circle, which has a radius of 1.

❸ Calculating 95% of every stock returns and append the result in the list named *MC_percentile95*.

❹ Estimating Monte Carlo VaR.

Denoising

Volatility is everywhere but it is formidable task to find out what kind of volatility is most valuable. In general, there are two types of information in the market: *noise* and *signal*. The former generates nothing but random information but the latter equip us with a valuable information by which investor can make money. To illustrate, consider that there are two main players in the market the one use noisy information called noise trader and informed trader who exploits signal or insider information. Noise traders trading motivation is driven by random behavior. So, information flow to the market signals are thought to be as buying signal for some noise traders and selling for others.

However, informed trader is considered to be a rational one in the sense that insider informed trader is able to assess a signal because she knows that it is a private information.

Consequently, continuous flow of information should be treated with caution. In short, information coming from noise trader can be considered as noise and information coming from insider can be taken as signal and this is the sort of information that matters. Investor who cannot distinguish noise and signal can fail to gain profit and/or assess the risk in a proper way.

Now, the problem turns out to be the differentiating the flow of information to the financial markets. How can we differentiate noise from signal? and how can we utilize this information.

It is now appropriate to discuss the Marcenko Pastur Theorem that helps us to have homogeneous covariance matrix. The Marcenko-Pastur theorem allows us to etract signal from noise using eigenvalues of covariance matrix.

NOTE

Let $A \in \mathbb{R}^{nxn}$ be a square matrix. Then, $\lambda \in \mathbb{R}$ is an eigenvalue of A and $x \in mathbbR^n\{0\}$ is the corresponding eigenvector of A if

$Ax = \lambda x$

Eigenvalue and eigenvector have special meaning in financial context. Eigenvector corresponds the variance in covariance matrix while eigenvalue shows the magnitude of the eigenvector. Specifically, largest eigenvector corresponds to largest variance and the magnitude of this equals to the corresponding eigenvalue. Due to noise in the data some eigenvalues can be thought of as random and it makes sense to detect and filter these eigenvalues in order to retain signals only.

To differentiate noise and signal, we fit the Marcenko Pastur Theorem PDF to the noisy covariance. The PDF of Marcenko Pastur Theorem takes the following form (Prado, 2020):

$$f(\lambda) = \begin{cases} \frac{T}{N}\sqrt{(\lambda_t - \lambda)(\lambda - \lambda_-)} & \text{if } \lambda \in [\lambda - \lambda_-] \\ 0, & \text{if } \lambda \notin [\lambda - \lambda_-] \end{cases}$$

where $\lambda_+ and \lambda_-$ are maximum and minimum eigenvalues.respectively.

In the following code block, which is slight modification of the codes provided by Prado (2020), we will generate probability density function of Marchenko-Pastur distribution and kernel density that allows us to model a

random variable in a non-parametric approach. Then, Marchenko-Pastur distribution will be fitted to the data.

```
In [24]: def mp_pdf(sigma2, q, obs):
             lambda_plus = sigma2 * (1 + q ** 0.5) ** 2❶
             lambda_minus = sigma2 * (1 - q ** 0.5) ** 2❷
             l = np.linspace(lambda_minus, lambda_plus, obs)
             pdf_mp = 1 / (2 * np.pi * sigma2 * q * l) * np.sqrt((lambda_plus  - l)
                                                                 *  (l -
lambda_minus))❸
             pdf_mp = pd.Series(pdf_mp, index=l)
             return pdf_mp

In [25]: from sklearn.neighbors import KernelDensity

         def kde_fit(bandwidth,obs,x=None):
             kde = KernelDensity(bandwidth, kernel='gaussian')❹
             if len(obs.shape) == 1:
                 kde_fit=kde.fit(np.array(obs).reshape(-1, 1))❺
             if x is None:
                 x=np.unique(obs).reshape(-1, 1)
             if len(x.shape) == 1:
                 x = x.reshape(-1, 1)
             logprob = kde_fit.score_samples(x)❻
             pdf_kde = pd.Series(np.exp(logprob), index=x.flatten())
             return pdf_kde

In [26]: corr_mat = np.random.normal(size=(10000, 1000))❼
         corr_coef = np.corrcoef(corr_mat, rowvar=0)❽
         sigma2 = 1
         obs = corr_mat.shape[0]
         q = corr_mat.shape[0] / corr_mat.shape[1]

         def plotting(corr_coef, q):
             ev, _ = np.linalg.eigh(corr_coef)❾
             idx = ev.argsort()[::-1]
             eigen_val = np.diagflat(ev[idx])❿
             pdf_mp = mp_pdf(1., q=corr_mat.shape[1] / corr_mat.shape[0], obs=1000)
⓫
             kde_pdf = kde_fit(0.01, np.diag(eigen_val))⓬
             ax = pdf_mp.plot(title="Marchenko-Pastur Theorem",
                              label="M-P", style='r--')
             kde_pdf.plot(label="Empirical Density", style='o-', alpha=0.3)
             ax.set(xlabel="Eigenvalue", ylabel="Frequency")
             ax.legend(loc="upper right")
             plt.savefig('images/MP_fit.png')
```

```
            plt.show()
            return plt

In [27]: plotting(corr_coef, q);
```

❶ Calculating maximum expected eigenvalue.

❷ Calculating minimum expected eigenvalue

❸ Generating probability density function of Marchenko-Pastur distribution.

❹ Initiating kernel density estimation.

❺ Fitting kernel density to the observations.

❻ Assessing the log density model on observations.

❼ Generating random samples from normal distribution.

❽ Converting covariance matrix into correlation matrix.

❾ Calculating eigenvalues of the correlation matrix.

❿ Turning numpy array into diagonal matrix.

⓫ Calling the *mp_pdf* function to estimate the pdf of Marchenko-Pastur distribution.

⓬ Calling the *kde_fit* function to fit kernel distribution to the data.

Figure 5-3 exhibits that Marchenko-Pastur distribution fit well to the data. Thanks to Marchenko-Pastur theorem, we are able to differentiate the noise and signal and the data in which noise is filtered is refered to as denoised.

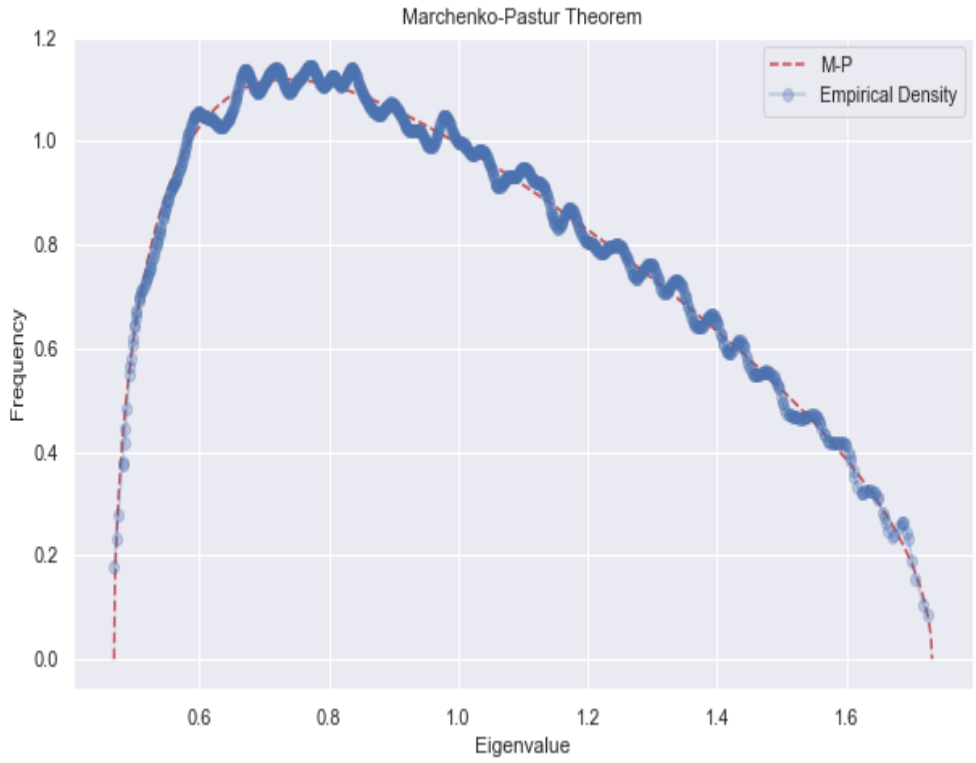


Figure 5-3. Fitting Marchenko-Pastur Distribution

So far, we have discussed the main steps to take to denoising the covariance matrix so that we can plug it into the VaR model, which is called *Denoised VaR* estimation. Denoising covariance matrix is nothing but taking unnecessary information (noise) out of the data. So, we make use of signal from the market, which tells us something important is going on in the market.

Denoising the covariance matrix includes following stages[1]:

- Calculating eigenvalues and eigenvectors based on correlation matrix.
- Using Kernel Density Estimation, find eigenvector for a specific eigenvalue.
- Fitting Marchenko-Pastor distribution to Kernel Density Estimation.
- Finding maximum theoretical eigenvalue using Marchenko-Pastur distribution.

- Average of eigenvalues greater than theoretical value are calculated.
- These new eigenvalues and eigenvectors are used to calculate denoised correlation matrix.
- Denoised covariance matrix is calculated by the new correlation matrix.

In the Appendix, you can find the algorithm including all these steps, but here I would like to show you how easy to apply finding denoised covariance matrix with a few lines of code utilizing the *portfoliolab* library in Python:

```
In [28]: import portfoliolab as pl

In [29]: risk_estimators = pl.estimators.RiskEstimators()

In [30]: stock_prices = stocks.copy()

In [31]: cov_matrix = stocks_returns.cov()
         cov_matrix
Out[31]:           IBM      MSFT      INTC
         IBM   0.000672  0.000465  0.000569
         MSFT  0.000465  0.000770  0.000679
         INTC  0.000569  0.000679  0.001158

In [32]: tn_relation = stock_prices.shape[0] / stock_prices.shape[1]❶
         kde_bwidth = 0.25❷
         cov_matrix_denoised = risk_estimators.denoise_covariance(cov_matrix,
                                                                  tn_relation,
                                                                  kde_bwidth)❸
         cov_matrix_denoised = pd.DataFrame(cov_matrix_denoised,
                                            index=cov_matrix.index,
                                            columns=cov_matrix.columns)
         cov_matrix_denoised
Out[32]:           IBM      MSFT      INTC
         IBM   0.000672  0.000480  0.000589
         MSFT  0.000480  0.000770  0.000638
         INTC  0.000589  0.000638  0.001158

In [33]: def VaR_parametric_denoised(initial_investment, conf_level):
             port_std = np.sqrt(weights.T.dot(cov_matrix_denoised).dot(weights))❹
             alpha = norm.ppf(1 - conf_level, stocks_returns_mean, port_std)
             for i,j in zip(stocks.columns,range(len(stocks.columns))):
                 print("Parametric VaR result for {} is {} ".format(i,VaR_param))
             VaR_params = (initial_investment - initial_investment * (1 + alpha))
```

```
            print('--' * 25)
            return VaR_params

In [34]: VaR_parametric_denoised(initial_investment, conf_level)
         Parametric VaR result for IBM is [42199.83906971 40618.17975427
          42702.9302193 ]
         Parametric VaR result for MSFT is [42199.83906971 40618.17975427
          42702.9302193 ]
         Parametric VaR result for INTC is [42199.83906971 40618.17975427
          42702.9302193 ]
         --------------------------------------------------

Out[34]: array([42259.80830949, 40678.14899405, 42762.89945907])

In [35]: symbols = ["IBM", "MSFT", "INTC"]
         data3 = []
         for symbol in symbols:
             data3.append(getDailyData(symbol)[::-1]['close']['2007-04-01': '2009-
02-01'])
         stocks_crisis = pd.DataFrame(data3).T
         stocks_crisis.columns = symbols

In [36]: stock_prices = stocks_crisis.copy()

In [37]: stocks_returns = (np.log(stocks) - np.log(stocks.shift(1))).dropna()

In [38]: cov_matrix = stocks_returns.cov()

In [39]: VaR_parametric(initial_investment, conf_level)
         Parametric VaR result for IBM is 42199.839069714886
         Parametric VaR result for MSFT is 40618.179754271754
         Parametric VaR result for INTC is 42702.930219301255
         --------------------------------------------------

Out[39]: array([42199.83906971, 40618.17975427, 42702.9302193 ])

In [40]: VaR_parametric_denoised(initial_investment, conf_level)
         Parametric VaR result for IBM is [42199.83906971 40618.17975427
          42702.9302193 ]
         Parametric VaR result for MSFT is [42199.83906971 40618.17975427
          42702.9302193 ]
         Parametric VaR result for INTC is [42199.83906971 40618.17975427
          42702.9302193 ]
         --------------------------------------------------

Out[40]: array([42259.80830949, 40678.14899405, 42762.89945907])
```

1. Relation of number of observations T to the number of variables N.
2. Identifying the bandwidth for kernel density estimation.
3. Generating the denoised covariance matrix.
4. Incoporating the denoised covariance matrix into the VaR formula.

The difference of the traditionally applied VaR and the denoised VaR are even more pronounced in the crisis period. During crisis period, correlation among assets becomes higher, it is sometimes referred to as *correlation breakdown*. The likely consequences of correlation breakdown can be twofold:

- An increase on market risk that makes diversification becomes less effective.
- Making hedging even more difficult.

In order to check this phenomenon, we consider the 2017-2018 financial crisis period and the exact crisis period is obtained from NBER, which announces business cycles (see [*https://www.nber.org/research/data/us-business-cycle-expansions-and-contractions*] for further information).

The result confirms that the correlation and thereby VaR becomes higher during crisis period.

Now, we manage to obtained ML-based VaR using denoised covariance matrix in lieu of empirical matrix that we calculate directly from the data. Despite its appealing and easiness VaR is not a coherent risk measure. Being a coherent risk measure requires certain conditions or axioms to satisfy. You can think of these axioms as a technical requirements for a risk measure.

Let $\alpha \in (0,1)$ be fixed confidence level and $(\omega \mathscr{F} \mathsf{P})$ be a probability space in which ω represents a sample space, $\mathscr{F}$ denotes subset of sample space, and P is probability measure.

NOTE

To illustrate, say ω is the set of all possible outcomes in the event of tossing a coin, ω= {H,T}. $\mathscr{F}$ can be treated as tossing a coin twice, F=$2^{\{\omega\}=2}$2. Finally, probability measure, P, is the odds of getting tails is 0.5.

Here is the four axioms of a coherent risk measure:

1) Translation Invariance: For all outcomes Y and a constant $a \in \mathscr{R}$, we have

$$VaR(Y_1 + Y_2) = VaR(Y) + a$$

It means that if a riskless amount of a is added to the portfolio, it results in lower VaR by the amount of a.

2) Sub-additivity: For all Y_1 and Y_2 we have

$$VaR(Y_1 + Y_2) \leq VaR(Y_1) + VaR(Y_2)$$

This axiom stress the importance of diversification in risk management. Take Y_1 and Y_2 as two assets if you both are included in the portfolio, then it results in lower Value-at-Risk than including them separately. Let's check whether or not VaR satisfy sub-additivity assumption:

```
In [41]: asset1 = [-0.5, 0, 0.1, 0.4]❶
         VaR1 = np.percentile(asset1, 90)
         print('VaR for the Asset 1 is {:.4f}'.format(VaR1))
         asset2 = [0, -0.5, 0.01, 0.4]❷
         VaR2 = np.percentile(asset2, 90)
         print('VaR for the Asset 2 is {:.4f}'.format(VaR2))
         VaR_all = np.percentile(asset1 + asset2, 90)
         print('VaR for the portfolio is {:.4f}'.format(VaR_all))
         VaR for the Asset 1 is 0.3100
         VaR for the Asset 2 is 0.2830
         VaR for the portfolio is 0.4000

In [42]: asset1 = [-0.5, 0, 0.05, 0.03]❶
         VaR1 = np.percentile(asset1, 90)
         print('VaR for the Asset 1 is {:.4f}'.format(VaR1))
         asset2 = [0, -0.5, 0.02, 0.8]❷
         VaR2 = np.percentile(asset2,90)
```

```
print('VaR for the Asset 2 is {:.4f}'.format(VaR2))
VaR_all = np.percentile(asset1 + asset2 , 90)
print('VaR for the portfolio is {:.4f}'.format(VaR_all))
VaR for the Asset 1 is 0.0440
VaR for the Asset 2 is 0.5660
VaR for the portfolio is 0.2750
```

❶ Asset return for the first asset

❷ Asset return for the second asset

It turns out portfolio VaR is less that the sum of individual VaRs, which makes no sense due to the risk mitigation throght diversification. More elaborately, portfolio VaR should be lower than the sum of individual VaRs via divesification as divesification mitigates risk, which in turn have a reduce the portfolio VaR.

3) Positive homogeneity: For all outcomes Y and a>0, we have

$$VaR(aY) = aVaR(L)$$

It implies that risk and value of the portfolio go in tandem, that is, if value of a portfolio increases by an amount of a, the risk goes up by an amount of a.

4) Monotonicity: For any 2 outcomes, Y_1 and Y_2 if $Y_1 \leq Y_2$, then

$$VaR(Y_1) \geq VaR(Y_2)$$

At first, it may seem puzzling but it is intuitive. Monotonicity indicates that an asset with higher return has a less VaR.

In the light of above discussion, we now know that VaR is not a coherent risk measure. But do not worry, VaR is not the only tool by which we estimate market risk. Expected shortfall is another and coherent market risk measure.

Expected Shortfall

Unlike VaR, Expected Shortfall focuses on the tail of the distribution. More elaboratively, Expected Shortfall enable us to take into account the

unexpected risks in the market. However, it does not mean that Expected Shortfall and VaR are two entirely different concepts. Rather, they are related, that is, it is possible to express Expected Shortfall using VaR.

Let us now assume that loss distribution is continuous, then Expected Shortfall can be mathematically defined as:

$$ES_\alpha = \frac{1}{1-\alpha}\int_\alpha^1 q_u du$$

where q denotes quantile of the loss distribution. Expected Shortfall formula suggests that it is nothing but a probability weights average of $(1-\alpha)\%$ of losses.

Let us substitute q_u and VaR, which gives us the following equation:

$$ES_\alpha = \frac{1}{1-\alpha}\int_\alpha^1 VaR_u du$$

Alternatively, it is mean of losses exceeding VaR as shown below:

$$ES_\alpha = \mathsf{E}(L|L > VaR_\alpha)$$

Loss distribution can be continuous or discrete and, as you can imagine, if it takes the discrete form, the Expected Shortfall becomes different as given below:

$$ES_\alpha = \frac{1}{1-\alpha}\sum_{n=0}^{1} max(L_n) * \mathsf{P}(L_n)$$

where $max(L_n)$ shows the highest nth loss and $Prob(L_n)$ indicates probability of pth highest loss.

```
In [43]: def ES_parametric(initial_investment , conf_level):
             alpha = - norm.ppf(1 - conf_level,stocks_returns_mean,port_std)
             for i,j in zip(stocks.columns, range(len(stocks.columns))):
                 VaR_param = (initial_investment * alpha)[j]❶
                 ES_param = (1 / (1 - conf_level)) * norm.expect(lambda x:
VaR_param,
                                                                 lb = conf_level)❷
                 print(f"Parametric ES result for {i} is {ES_param}")

In [44]: ES_parametric(initial_investment, conf_level)
         Parametric ES result for IBM is 144370.82004213505
```

```
Parametric ES result for MSFT is 138959.76972934653
Parametric ES result for INTC is 146091.9565067016
```

❶ Estimating the Variance-Covaraince VaR.

❷ Given confidence interval, estimating the ES based on VaR.

ES can also be computed based on the historical observations. Similar to historical simulation VaR method, parametric assumption can be relaxed. To do that, first return (or loss) corresponding to the 95% is found and then the mean of the observations greater than the 95% gives us the result. Here, this is what we do:

```
In [45]: def ES_historical(initial_investment, conf_level):
             for i, j in zip(stocks_returns.columns,
range(len(stocks_returns.columns))):
                 ES_hist_percentile95 = np.percentile(stocks_returns.loc[:, i], 5)❶
                 ES_historical = stocks_returns[str(i)][stocks_returns[str(i)] <=
                                                        ES_hist_percentile95].mean()
❷
                 print("Historical ES result for {} is {:.4f} "
                       .format(i, initial_investment * ES_historical))

In [46]: ES_historical(initial_investment, conf_level)
         Historical ES result for IBM is -64802.3898
         Historical ES result for MSFT is -65765.0848
         Historical ES result for INTC is -88462.7404
```

❶ Calculating the 95% of the returns.

❷ Estimating the ES based on the historical observations.

Thus far, we have seen how to model the expected shortfall in a traditional way. Now, it is time to introduce ML-based approach to further enhance the estimation performance and reliability of the ES model.

Liquidity Augmented Expected Shortfall

As is discussed, ES provides us a coherent risk measure to gauge market risk. However, even though we differentiate financial risk as market, credit, liquidity, and operational, it does not necessarily mean that these risks are entirely unrelated to each other. Rather, they are, to some extent, correlated. That is, once a financial crisis hit the market, market risk surges align with the drawdown on lines of credit, which in turn increase liquidity risk.

Given the situation of the economy, the effect of illiquidity varies as it becomes commonplace during huge market downturns but it is more manageable during normal times. Therefore, its impact will be more distinct in down market.

This fact is supported by Antoniades stating that

> *Common pool of liquid assets is the resource constraint through which liquidity risk can affect the supply of mortgage credit. During the financial crisis of 2007–2008 the primary source of stresses to bank funding conditions arose from the funding illiquidity experienced in the markets for wholesale funding.*
>
> —Antoniades (Liquidity Risk and the Credit Crunch of 2007-2008: Evidence from Micro-Level Data on Mortgage Loan Applications, 2014, p.6)

Ignoring liquidity dimension of risk may cause underestimating the market risk. Therefore, augmenting ES with liquidity risk may result in more accurate and reliable estimation. Well, it sounds appealing but how can we find a proxy for liquidity?

NOTE

Liquidity can be defined as the easiness of the trasaction with which assets can be sold in a very short time period without a significant impact on market price. There are two main measures of liquidity:

- Market liquidity: The ease with which an asset is traded.
- Funding liquidity: The ease with which investor can obtain funding.

Liquidity and the risk arising from it will be discussed in a greater detail in Chapter-7. So, much of the discussion is left to this chapter.

In the literature, bid-ask spread measures are commonly used for modeling liquidity. Shortly, bid-ask spread is the difference of bid-ask prices. Put differently, it is the difference between the highest available price (bid price) that a buyer is willing to pay and the lowest price (ask price) that a seller is willing to get. So, bid-ask spread gives a tool to measure the transaction cost.

To the extent that bid-ask spread is a good indicator of transaction cost, it is also a good proxy of liquidity in the sense that transaction cost is one of the components of liquidity. Spreads can be defined various ways depending on their focus. Here is the bid-ask spreads that we will use to incorporate liquidity risk into ES model.

- Effective Spread

$$\text{Effective Spread} = 2 * |(P_t - P_{mid})|$$

where P_t is the price of trade at time t and P_{mid} is the midpoint of the bid-ask offer ($(P_{ask} - P_{bid})/2$) prevailing at the time of the t.

- Proportional Quoted Spread

$$\text{Proportional Quoted Spread} = (P_{ask} - P_{bid})/P_{mid}$$

where P_{ask} is the ask price and P_{bid} and P_{mid} are bid price and mid price, respectively.

- Quoted Spread

$$\text{Quoted Spread} = P_{ask} - P_{bid}$$

- Proportional Effective Spread:

$$\text{Proportional Effective Spread} = 2 * (|P_t - P_{mid}|)/P_{mid}$$

Effective Cost

$$\text{Effective Cost} = \begin{cases} (P_t - P_{mid})/P_{mid} \text{ for buyer initiated} \\ (P_{mid}/P_t)/P_{mid} \text{ for seller initiated} \end{cases}$$

Buyer-initiated trade occurs when a trade is executed at a price abover quoted mid price. Similarly, seller-initiated trade occurs when a trade is executed at a price below than the quoted mid price.

Now, we need to find a way to incorporate these bid-ask spreads into the ES model so that we are able to account for the liquidity risk as well as market risk. We employ two different methods to accomplish this task. First one is to take the cross-sectional mean of the bid ask spread as suggested by Chordia et al., (2000) and Pastor and Stambaugh (2003). The second method is to apply Principal Component Analysis (PCA) as proposed by Mancini et al. (2013).

Cross-sectional mean is nothing but averaging the bid-ask spread. Using this method, we are able to generate a measure for market-wide liquidity. The averaging formula is as follows:

$L_{M,t} = \frac{1}{N} \sum_i^N L_{i,t}$

where $L_{M,t}$ is the market liquidity and $L_{i,t}$ is the individual liqudity measure, namely bid-ask spread in our case.

$$ES_L = ES + \text{Liquidity Cost}$$

$$ES_L = \frac{1}{1-\alpha} \int_\alpha^1 VaR_u du + \frac{1}{2} P_{last}(\mu + k\sigma)$$

where

- P_{last} is the closing stock price.
- μ is the mean of spread.
- k is the scaling factor to accommodate fat-tail.
- σ is the standard deviation of the spread.

```
In [47]: bid_ask = pd.read_csv('bid_ask.csv')❶

In [48]: bid_ask['mid_price'] = (bid_ask['ASKHI'] + bid_ask['BIDLO']) / 2❷
         buyer_seller_initiated = []
         for i in range(len(bid_ask)):
             if bid_ask['PRC'][i] > bid_ask['mid_price'][i]:❸
                 buyer_seller_initiated.append(1)❹
             else:
                 buyer_seller_initiated.append(0)❺

         bid_ask['buyer_seller_init'] = buyer_seller_initiated

In [49]: effective_cost = []
         for i in range(len(bid_ask)):
             if bid_ask['buyer_seller_init'][i] == 1:
                 effective_cost.append((bid_ask['PRC'][i] - bid_ask['mid_price'][i])
/
                                       bid_ask['mid_price'][i])❻
             else:
                 effective_cost.append((bid_ask['mid_price'][i] - bid_ask['PRC']
[i])/
                                       bid_ask['mid_price'][i])❼
         bid_ask['effective_cost'] = effective_cost

In [50]: bid_ask['quoted'] = bid_ask['ASKHI'] - bid_ask['BIDLO']❽
         bid_ask['prop_quoted'] = (bid_ask['ASKHI'] - bid_ask['BIDLO']) / \
                                  bid_ask['mid_price']❽
         bid_ask['effective'] = 2 * abs(bid_ask['PRC'] - bid_ask['mid_price'])❽
         bid_ask['prop_effective'] = 2 * abs(bid_ask['PRC'] - bid_ask['mid_price'])
/ \
                                     bid_ask['PRC']❽

In [51]: spread_measures = bid_ask.iloc[:, -5:]
         spread_measures.corr()
Out[51]:                 effective_cost    quoted  prop_quoted  effective  \
         effective_cost        1.000000  0.441290     0.727917   0.800894
         quoted                0.441290  1.000000     0.628526   0.717246
         prop_quoted           0.727917  0.628526     1.000000   0.514979
```

```
effective             0.800894  0.717246     0.514979   1.000000
prop_effective        0.999847  0.442053     0.728687   0.800713

                  prop_effective
effective_cost          0.999847
quoted                  0.442053
prop_quoted             0.728687
effective               0.800713
prop_effective          1.000000

In [52]: spread_measures.describe()
Out[52]: effective_cost      quoted  prop_quoted   effective  prop_effective
         count      756.000000  756.000000   756.000000  756.000000
          756.000000
         mean         0.004247    1.592583     0.015869    0.844314
          0.008484
         std          0.003633    0.921321     0.007791    0.768363
          0.007257
         min          0.000000    0.320000     0.003780    0.000000
          0.000000
         25%          0.001517    0.979975     0.010530    0.300007
          0.003029
         50%          0.003438    1.400000     0.013943    0.610000
          0.006874
         75%          0.005854    1.962508     0.019133    1.180005
          0.011646
         max          0.023283    8.110000     0.055451    6.750000
          0.047677

In [53]: high_corr = spread_measures.corr().unstack()\
                     .sort_values(ascending=False).drop_duplicates()❾
         high_corr[(high_corr > 0.80) & (high_corr != 1)]❿
Out[53]: effective_cost  prop_effective    0.999847
         effective       effective_cost    0.800894
         prop_effective  effective         0.800713
         dtype: float64

In [54]: sorted_spread_measures = bid_ask.iloc[:, -5:-2]

In [55]: cross_sec_mean_corr = sorted_spread_measures.mean(axis=1).mean()⓫
         std_corr = sorted_spread_measures.std().sum() / 3⓬

In [56]: df = pd.DataFrame(index=stocks.columns)
         last_prices = []
         for i in symbols:
             last_prices.append(stocks[i].iloc[-1])⓭
         df['last_prices'] = last_prices
```

```
In [57]: def ES_parametric(initial_investment, conf_level):
             ES_params = []
             alpha = - norm.ppf(1 - conf_level, stocks_returns_mean, port_std)
             for i,j in zip(stocks.columns,range(len(stocks.columns))):
                 VaR_param = (initial_investment * alpha)[j]
                 ES_param = (1 / (1 - conf_level)) * norm.expect(lambda x:
VaR_param,
                                                                 lb = conf_level)
                 ES_params.append(ES_param)
             return ES_params

In [58]: ES_params = ES_parametric(initial_investment,conf_level)
         for i in range(len(symbols)):
             print(f'The Expected Shortfall result for {symbols[i]} is
{ES_params[i]}')
         The Expected Shortfall result for IBM is 144370.82004213505
         The Expected Shortfall result for MSFT is 138959.76972934653
         The Expected Shortfall result for INTC is 146091.9565067016

In [59]: k = 1.96
         for i, j in zip(range(len(symbols)), symbols):
             print('The liquidity Adjusted ES of {} is {}'
                   .format(j, ES_params[i] + (df.loc[j].values[0] / 2) *
                           (cross_sec_mean_corr + k * std_corr)))❹
         The liquidity Adjusted ES of IBM is 144443.0096816674
         The liquidity Adjusted ES of MSFT is 139087.3231105412
         The liquidity Adjusted ES of INTC is 146120.5272712512
```

❶ Importing the *bid_ask* data.

❷ Calculating the mid price.

❸ Defining condition for buyer and seller initiated trade.

❹ If the above-given condition holds, it returns 1 and it is appended into *buyer_seller_initiated* list.

❺ If the above-given condition does not hold, it returns 0 and it is appended into *buyer_seller_initiated* list.

❻ If *buyer_seller_initiated* variable takes the value of 1, corresponding effective cost formula is run.

❼ If *buyer_seller_initiated* variable takes the value of 0, corresponding effective cost formula is run.

❽ Quoted, proportional quoted, effective , and proportional effective spreads are computed.

❾ Correlation matrix are obtained and list them column-wise.

❿ Sorting out the correlation greater than 80%.

⓫ Calculating the cross-sectional mean of spread measures.

⓬ Obtaining the standard deviation of spreads.

⓭ Filtering the last observed stock prices from the *stocks* data.

⓮ Estimating the liquidity-adjusted ES.

PCA is a method used to reduce dimensionality. It is used to extract as much as information as possible using as few component as possible. That is, in this example, out of 5 features, we pick 2 components relying on the Figure 5-4 given below. So, we reduce dimensionality at the expense of losing information. Because, depending on the cut-off point that we decide, we pick the number of components and we lose information as much as how many components we left off.

To be more specific, point at which Figure 5-4 gets flatter implies that we retain less information and this is the cut-off point for the PCA. However, it is not an easy call in that there is a trade-off between the cut-off point and information retained. That is, on the one hand, the higher the cut-off point (the higher number the components) we have, the more information we retain (the less dimensionality we reduce). On the other hand, the less the cut-off point (the less number of components we have), the less information we retain (the higher dimensionality we reduce). Getting flatter scree plot is not the mere criteria for selecting suitable number of components. So, what would be the

possible criteria to picking proper number of components. Here is the possible cut-off criteria for PCA:

- Greater than 80% explained variance
- Greater than 1 eigenvalues
- The point at which scree plot get flatter

However, dimensionality reduction is not the only thing that we take advantage from. In this study, we apply PCA to benefit of getting peculiar features of liquidity. Because, PCA filter the most important information from the data for us.

The mathematics behind the PCA is discussed in detail in Appendix.

> **WARNING**
>
> Please note that liqudity adjustment can be applied to VAR, too. The same procedure applies to VaR. Mathematically,
>
> $$VaR_L = \sigma_p \sqrt{t} Z_\alpha + \frac{1}{2} P_{last}(\mu + k\sigma)$$
>
> This application is left to the reader.

```
In [60]: from sklearn.decomposition import PCA
         from sklearn.preprocessing import StandardScaler

In [61]: scaler = StandardScaler()
         spread_measures_scaled = scaler.fit_transform(np.abs(spread_measures))❶
         pca = PCA(n_components=5)❷
         prin_comp = pca.fit_transform(spread_measures_scaled)❸

In [62]: var_expl = np.round(pca.explained_variance_ratio_, decimals=4)❹
         cum_var = np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4))❺
         print('Individually Explained Variances are:\n{}'.format(var_expl))
         print('=='*30)
         print('Cumulative Explained Variances are: {}'.format(cum_var))
         Individually Explained Variances are:
         [0.7494 0.1461 0.0983 0.0062 0.    ]
         ============================================================
         Cumulative Explained Variances are: [0.7494 0.8955 0.9938 1.     1.    ]
```

```
In [63]: plt.plot(pca.explained_variance_ratio_)❻
         plt.xlabel('Number of Components')
         plt.ylabel('Variance Explained')
         plt.title('Scree Plot')
         plt.savefig('Scree_plot.png')
         plt.show()
In [64]: pca = PCA(n_components=2)❼
         pca.fit(np.abs(spread_measures_scaled))
         prin_comp = pca.transform(np.abs(spread_measures_scaled))
         prin_comp = pd.DataFrame(np.abs(prin_comp), columns = ['Component 1',
                                                                'Component 2'])
         print(pca.explained_variance_ratio_*100)
         [65.65640435 19.29704671]

In [65]: def myplot(score, coeff, labels=None):
             xs = score[:, 0]
             ys = score[:, 1]
             n = coeff.shape[0]
             scalex = 1.0 / (xs.max() - xs.min())
             scaley = 1.0 / (ys.max() - ys.min())
             plt.scatter(xs * scalex * 4, ys * scaley * 4, s=5)
             for i in range(n):
                 plt.arrow(0, 0, coeff[i, 0], coeff[i, 1], color = 'r', alpha=0.5)
                 if labels is None:
                     plt.text(coeff[i, 0], coeff[i, 1], "Var"+str(i), color='black')
                 else:
                     plt.text(coeff[i,0 ], coeff[i, 1], labels[i], color='black')

             plt.xlabel("PC{}".format(1))
             plt.ylabel("PC{}".format(2))
             plt.grid()

In [66]: spread_measures_scaled_df = pd.DataFrame(spread_measures_scaled,
                                                  columns=spread_measures.columns)

In [67]: myplot(np.array(spread_measures_scaled_df)[:, 0:2],
                np.transpose(pca.components_[0:2,:]),
                list(spread_measures_scaled_df.columns))❽
         plt.savefig('Bi_plot.png')
         plt.show()
```

❶ Standardization of the spread measures.

❷ Identifying the number of principal components as 5.

❸ Applying the principal component to the *spread_measures_scaled*.

❹ Observing the explained variance of the five principal components.

❺ Observing the cumulative explained variance of the five principal components

❻ Drawing the *scree plot*.

❼ Based on scree plot, deciding the number of component as 2 to be used in PCA analysis.

❽ Drawing the *biplot* to observe the relationship between components and features.

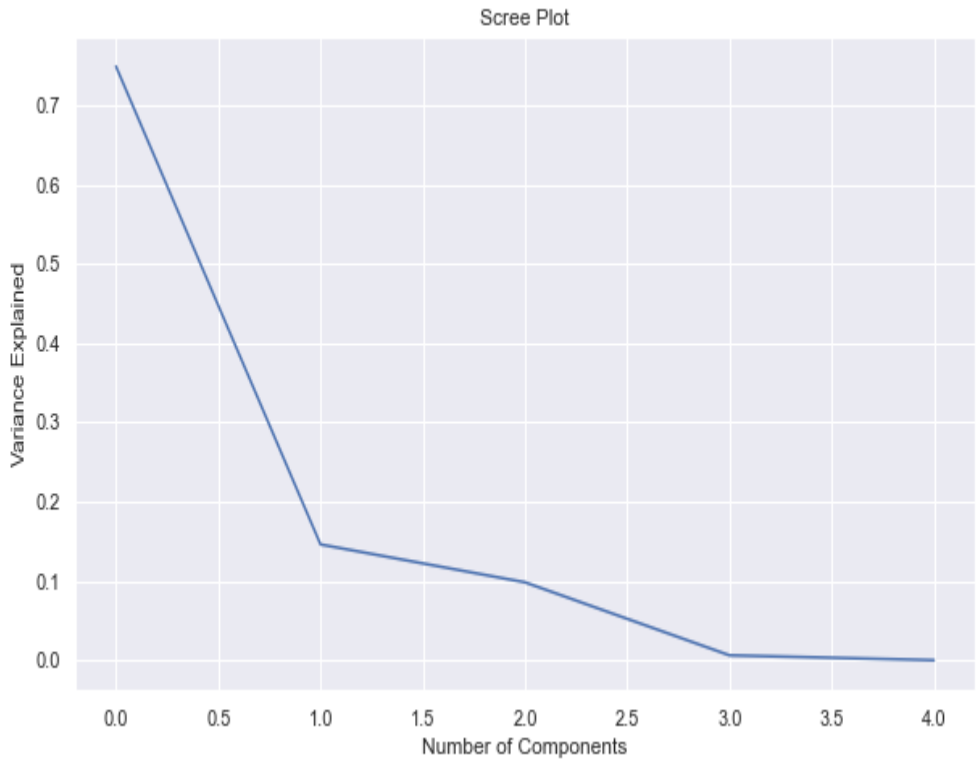


Figure 5-4. PCA Scree Plot

Figure 5-5. PCA Biplot

All right, we have all the necessary information now and incorporating this information, we are able to calculate liquidity adjusted ES. Unsurprisingly, the following code reveals that the liquidity adjusted ES provides larger values compared to standard ES application. This implies that including liquidity dimension in ES estimation results in higher risk. Please note that the difference will be more pronounced during crisis.

```
In [68]: prin_comp1_rescaled = prin_comp.iloc[:,0] * prin_comp.iloc[:,0].std()\
                              + prin_comp.iloc[:, 0].mean()❶
         prin_comp2_rescaled = prin_comp.iloc[:,1] * prin_comp.iloc[:,1].std()\
                              + prin_comp.iloc[:, 1].mean()❷
         prin_comp_rescaled = pd.concat([prin_comp1_rescaled, prin_comp2_rescaled],
                                        axis=1)
         prin_comp_rescaled.head()
Out[68]:    Component 1  Component 2
         0     1.766661     1.256192
         1     4.835170     1.939466
         2     3.611486     1.551059
```

```
          3     0.962666     0.601529
          4     0.831065     0.734612

In [69]: mean_pca_liq = prin_comp_rescaled.mean(axis=1).mean()❸
         mean_pca_liq
Out[69]: 1.0647130086973815

In [70]: k = 1.96
         for i, j in zip(range(len(symbols)), symbols):
             print('The liquidity Adjusted ES of {} is {}'
                   .format(j, ES_params[i] + (df.loc[j].values[0] / 2) *
                           (mean_pca_liq + k * std_corr)))❹
         The liquidity Adjusted ES of IBM is 144476.18830537834
         The liquidity Adjusted ES of MSFT is 139145.94711344707
         The liquidity Adjusted ES of INTC is 146133.65849966934
```

❶ Calculating the liquidity part of liquidity-adjusted ES formula for the first principal component.

❷ Calculating the liquidity part of liquidity-adjusted ES formula for the first principal component.

❸ Calculating cross-sectional mean of the two principal components.

❹ Estimating the liquidity-adjusted ES.

Conclusion

Market risk has been always under scrutiny as it gives us the extent to which a company is vulnerable to risk emanating from market events. In a financial risk management textbook, it is customary to find a VaR and ES model, which are two prominent and commonly applied model in theory and practice. In this chapter, after providing an introduction to these models, a cutting edge models are introduced to revisit and improve model estimation. To this end, first we try to distinguish information flowing in the form of noise and signal, which is called denoising. In what follows, denoised covariance matrix is employed to improve the VaR estimation.

Then, ES model are discussed as a coherent risk measure. The method that we applied to improve this model is liquidity-based approach by which we revisit eS model and augment using liquidity component so that it is possible to consider liquidity risk in estimating ES.

Further improvements in market risk estimation are also possible but the aim is to give an idea and tools to provide a decent ground for ML-based market risk approaches. However, you can go further and apply different tools. In the next chapter, we will discuss the credit risk modeling suggested by regulatory bodies such as the Basel Committee on Banking Supervision (BCBS) and enrich this model using ML-based approach.

Further Resources

Articles cited in this chapter:

- Antoniades, Adonis. "Liquidity Risk and the Credit Crunch of 2007-2008: Evidence from Micro-Level Data on Mortgage Loan Applications." Journal of Financial and Quantitative Analysis (2016): 1795-1822.
- Bzdok, D., N. Altman, and M. Krzywinski. "Points of significance: statistics versus machine learning." Nature Methods (2018): 1-7.
- BIS, Calculation of RWA for market risk, 2020.
- Chordia, Tarun, Richard Roll, and Avanidhar Subrahmanyam. "Commonality in liquidity." Journal of financial economics 56, no. 1 (2000): 3-28.
- Mancini, Loriano, Angelo Ranaldo, and Jan Wrampelmeyer. "Liquidity in the foreign exchange market: Measurement, commonality, and risk premiums." The Journal of Finance 68, no. 5 (2013): 1805-1841.
- Pástor, Ľuboš, and Robert F. Stambaugh. "Liquidity risk and expected stock returns." Journal of Political economy 111, no. 3

(2003): 642-685.

Books cited in this chapter:

- Dowd, Kevin. An introduction to market risk measurement. John Wiley & Sons, 2003.
- De Prado ML. Machine learning for asset managers. Cambridge University Press, 2020.

---

1 The details of the procedure can be found in this link [*https://hudsonthames.org/portfolio-optimisation-with-portfoliolab-estimation-of-risk/*

About the Author

Abdullah Karasan was born in Berlin, Germany. After he studied Economics and Business Administration at Gazi University-Ankara, he obtained his master's degree from the University of Michigan-Ann Arbor and his PhD in Financial Mathematics from Middle East Technical University (METU)-Ankara. He worked as a Treasury Controller at the Undersecretariat of Treasury of Turkey. More recently, he has started to work as a Senior Data Science consultant and instructor for companies in Turkey and the USA. Currently, he is a Data Science consultant at Datajarlabs and Data Science mentor at Thinkful.