Chapter 7:
Predictive Analytics and Financial Forecasting

What is predictive analytics? In business, it uses various tools and methods to collect and analyze data and tries to answer the question of what is most likely to happen next based on given information. A business uses the predictions to make informed decisions about future business operations to achieve certain objectives.

Financial forecasting is one example of predictive analytics. In this chapter, you'll learn several methods you can use in the field of financial forecasting, without resorting to more state-of-the-art methods in Python. The idea is to lay out the basic logic behind predictive analytics.

Suppose your job is, at the end of each trading day, to forecast the next-day stock price of IBM. We'll briefly explain why you shouldn't predict price directly. Instead, you should predict the stock return first and then convert the return to the price.

There are at least two ways you can predict stock returns. The first is to use the moving average. You can look at the average returns on IBM in the last 21 trading days and use it as a forecast. You'll repeat the process each day, hence the term "moving average."

The second way you can use is to run a regression of the excess return of IBM on the market excess return to obtain the alpha and beta of the stock. The predicted next day return is calculated based on the estimated alpha and beta.

Statistically Stationary Variables

A time-series variable *x* is *statistically stationary* if the mean, variance, and autocorrelation of the variable *x* are constant over time.

The stock price is not statistically stationary, and we'll discuss it in greater details in Chapter 12 as to why. In a nutshell, nominal stock prices "jump" or "dive" due to various reasons (cash dividend payments, or stock splits). Therefore, the mean, variance, auto-correlation and other statistical properties of the stock price are not constant over time (even in relatively short period of time, say, a few days).

For example, Apple conducted a 4-for-1 stock split on August 28, 2020. The average stock price was around $499.23 before the split. However, after August 28, 2020, the average stock

price was $124.81. See this article https://www.marketwatch.com/story/3-things-to-know-about-apples-stock-split-2020-08-28 and Chapter 12 notes for details.

Stock returns, on the other hand, are more or less statistically stationary, especially in a short period of time (say, one to six months). For example, the average value of daily stock returns is usually close to 0 (or a very small positive number). Its variance and autocorrelation are also close to constant. That is, stock returns are considered statistically stationary.

Therefore, if your task is to predict the next-day IBM stock price, you'll first predict the next-day IBM stock return. You then look at today's IBM stock price at the stock exchange. You can forecast next-day stock price by using this formula:

```
Next day's price = today's price × (1 + predicted next day return)
```

We'll use the above steps for stock price forecasting in the chapter.

Moving Average

You can use the same method as in Chapter 5 to obtain historical daily stock prices, say, in the past six months. You then calculate the daily stock returns and use the average return in the last six months as your forecast.

The program *MovingAverage.py*, as shown in Listing 7-1, calculates the average return of IBM in the last one, three, and six months.

```
# import needed modules
from pandas_datareader import data as pdr

# set the start and end dates
start_date = "2021-02-01"    #❶
end_date = "2021-08-31"
# obtain stock price
ticker = "IBM"
stock = pdr.get_data_yahoo(ticker, start=start_date, end=end_date)
#calculate returns
stock['ret_stock'] = (stock['Adj Close']/stock['Adj Close'].shift(1))-1 #❷
#calculate moving averages
stock.dropna(inplace=True)
stock['avg6m']=stock['ret_stock'].rolling(126).mean()   #❸
```

```
# Use Aug 23 avg6m value to predict return on Aug 24
forecast_ret=stock['avg6m']["2021-08-23"]
print("the predicted return on Aug 24 is", forecast_ret)
# Find out the stock price on Aug 23
price_today=stock['Adj Close']["2021-08-23"]
print("the price on Aug 23 is", price_today)
# Predict tomorrow's price
pred=price_today*(1+forecast_ret)
print("the predicted price on Aug 24 is", pred)
```

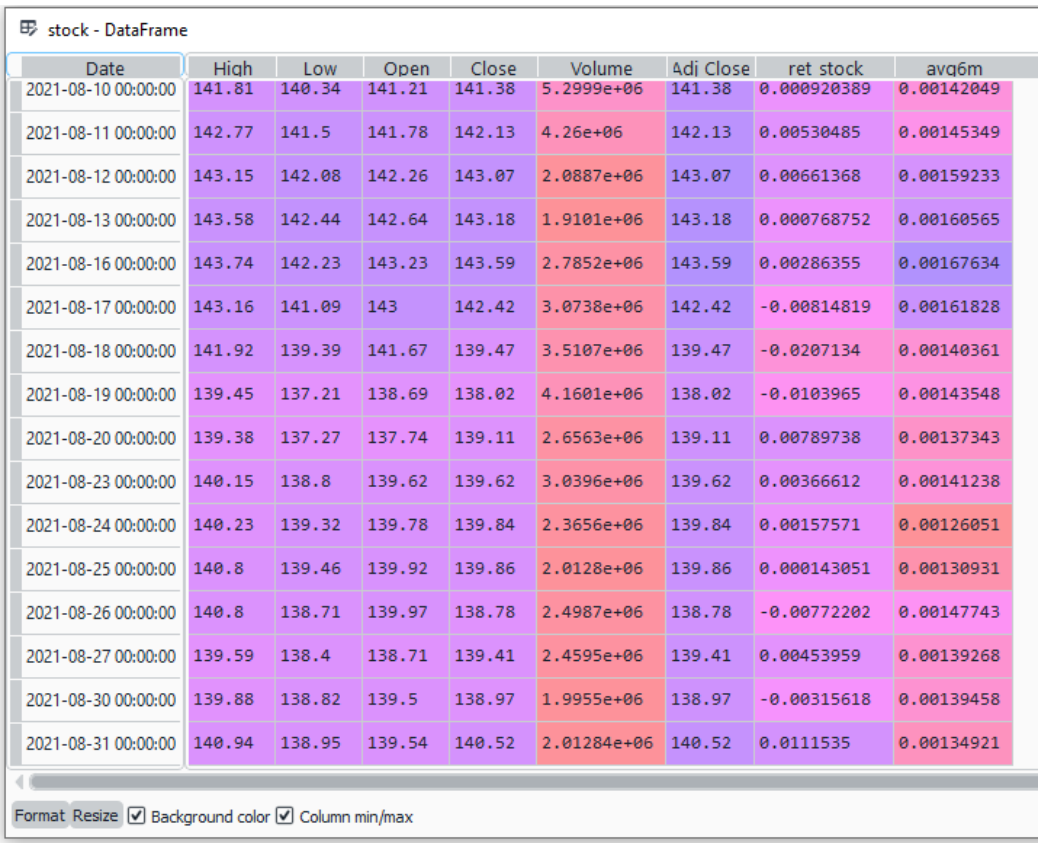
Listing 7-1: The program to calculate moving average

We first import the *pandas_datareader* module to retrieve daily stock price from Finance Yahoo. We then set the start and end date of the data we want to retrieve ❶. After that, we retrieve the daily stock information for IBM.

At ❷, we calculate the daily stock returns. At ❸, we calculate the 6-month moving average for IBM stock returns. Note that the `rolling()` function generates a data series from the existing data. We put 126 inside the `rolling()` function to indicate that the rolling window is six months (each month has an average of 21 trading days, and 21*6=126). The `mean()` function at the end is used to calculate the average value in each rolling window.

You can go to the **`Variable explorer`** pane of the Spyder IDE and double click on the variable *stock*, and you will see something similar to Figure 7-1, if you scroll down and scroll to the right.

INSERT FIGURE 7-1 HERE


stock - DataFrame

Date	High	Low	Open	Close	Volume	Adj Close	ret stock	avg6m
2021-08-10 00:00:00	141.81	140.34	141.21	141.38	5.2999e+06	141.38	0.000920389	0.00142049
2021-08-11 00:00:00	142.77	141.5	141.78	142.13	4.26e+06	142.13	0.00530485	0.00145349
2021-08-12 00:00:00	143.15	142.08	142.26	143.07	2.0887e+06	143.07	0.00661368	0.00159233
2021-08-13 00:00:00	143.58	142.44	142.64	143.18	1.9101e+06	143.18	0.000768752	0.00160565
2021-08-16 00:00:00	143.74	142.23	143.23	143.59	2.7852e+06	143.59	0.00286355	0.00167634
2021-08-17 00:00:00	143.16	141.09	143	142.42	3.0738e+06	142.42	-0.00814819	0.00161828
2021-08-18 00:00:00	141.92	139.39	141.67	139.47	3.5107e+06	139.47	-0.0207134	0.00140361
2021-08-19 00:00:00	139.45	137.21	138.69	138.02	4.1601e+06	138.02	-0.0103965	0.00143548
2021-08-20 00:00:00	139.38	137.27	137.74	139.11	2.6563e+06	139.11	0.00789738	0.00137343
2021-08-23 00:00:00	140.15	138.8	139.62	139.62	3.0396e+06	139.62	0.00366612	0.00141238
2021-08-24 00:00:00	140.23	139.32	139.78	139.84	2.3656e+06	139.84	0.00157571	0.00126051
2021-08-25 00:00:00	140.8	139.46	139.92	139.86	2.0128e+06	139.86	0.000143051	0.00130931
2021-08-26 00:00:00	140.8	138.71	139.97	138.78	2.4987e+06	138.78	-0.00772202	0.00147743
2021-08-27 00:00:00	139.59	138.4	138.71	139.41	2.4595e+06	139.41	0.00453959	0.00139268
2021-08-30 00:00:00	139.88	138.82	139.5	138.97	1.9955e+06	138.97	-0.00315618	0.00139458
2021-08-31 00:00:00	140.94	138.95	139.54	140.52	2.01284e+06	140.52	0.0111535	0.00134921

Format Resize Background color Column min/max

Figure 7-1: The moving average returns for IBM

From the data, you can see that the moving average on August 23, 2021, is 0.00141238. Therefore, you can use that value to forecast the stock return for IBM on August 24, 2021.

To predict IBM stock price on August 24, 2021, we convert next day stock return to stock price:

```
IBM Stock Price on 08/24/2021 = Price on 08/23/2021 × (1 + predicted return)
```

Plus in the values, we have

```
IBM Stock Price on 08/24/2021 = $139.62 × (1 + 0.00141238) = $140.13
```

Therefore, your forecast for IBM stock price on August 24, 2021, is $140.13.

WARNING: The adjusted closing price may change if IBM pays cash dividends or if it splits its stock. I'll explain why in Chapter 12. As a result, you'll likely see a different Adjust Close than the above. But the return should be the same.

Evaluate the Accuracy of Your Forecasts

In the above example, we use the six-month moving average of daily stock returns to forecast the next-day stock return.

You may ask: why not use the three-month or 12-month moving average to forecast? The answer is you can. Generally speaking, you don't want the rolling window to be too short, because there is randomness in daily stock price movements. By using a longer window, you can reduce the randomness in your forecast and make it more accurate.

On the other hand, you don't want the rolling window to be too long either. The market condition and the firm's financial health can change over time. The conditions one or two years ago may be quite different from what they are today. By using a shorter rolling window, you can ensure that your forecast tracks what's currently going on in the market and in the company.

Therefore, there is a trade-off between *smoothing* and *tracking* in terms of using moving averages to forecast. You can evaluate the accuracies of several different forecasting methods.

One way to evaluate is to use the *least squared errors* approach.

The idea behind the least squared errors approach is as follows. You compare the forecasts you made to the actual outcomes. The differences are called forecast errors. The least squared errors approach is to minimize the average squared forecast errors.

The program *LeastSquareErrors.py*, as shown in Listing 7-2, evaluates the forecast accuracies using one-month, three-month, and six-month moving averages.

```
# import needed modules
from pandas_datareader import data as pdr

# set the start and end dates
start_date = "2019-07-01"
end_date = "2020-06-30"
# obtain stock price
ticker = "TSLA"  #1
stock = pdr.get_data_yahoo(ticker, start=start_date, end=end_date)
#calculate returns
stock['ret_stock'] =(stock['Adj Close']/stock['Adj Close'].shift(1))-1  #2
#calculate moving averages
stock.dropna(inplace=True)
```

```
stock['avg1m']=stock['ret_stock'].rolling(21).mean()  #❸
stock['avg3m']=stock['ret_stock'].rolling(63).mean()
stock['avg6m']=stock['ret_stock'].rolling(126).mean()
#calculate squared forecast errors
stock['err1m']=stock['ret_stock']-stock['avg1m'].shift(1) #❹
stock['sqerr1m']=stock['err1m']*stock['err1m']  #❺
stock['err3m']=stock['ret_stock']-stock['avg3m'].shift(1)
stock['sqerr3m']=stock['err3m']*stock['err3m']
stock['err6m']=stock['ret_stock']-stock['avg6m'].shift(1)
stock['sqerr6m']=stock['err6m']*stock['err6m']
#calcualte avearage squared error for each method
stock.dropna(inplace=True)
print('the squared error using 1-month moving average is',
stock['sqerr1m'].mean())  #❻
print('the squared error using 3-month moving average is',
stock['sqerr3m'].mean())
print('the squared error using 6-month moving average is',
stock['sqerr6m'].mean())
```

Listing 7-2: The program to measure forecasting accuracy

We use Tesla as our example this time ❶. We retrieve the daily stock price information from July 2019 to June 2020 and calculate daily stock returns ❷. Starting at ❸, we calculate the 1-month, 3-month, and 6-month moving average for Tesla stock returns. The forecast error is defined as the difference between the actual stock return and the forecast ❹. The squared forecast error is the value of the forecast error multiplied by itself ❺.

Finally, we calculate the average squared forecast error for each forecasting method ❻.

The output from the program is as follows.

```
the squared error using 1-month moving average is 0.004048739397893946
the squared error using 3-month moving average is 0.0038756573932600667
the squared error using 6-month moving average is 0.003817939105382213
```

As you can see, the average squared forecast errors for the three methods are about 0.00405, 0.00388, and 0.00382, respectively. Therefore, using 6-month moving average to forecast is the most accurate, and using 1-month moving average to forecast is the least accurate.

Facing the choices, you'll probably choose the 6-month average as your forecast.

Use Regressions to Forecast

Another way to forecast is to use the regression analysis.

Assume that you want to forecast the value of y. At the same time, you know that y is related to another variable x, with a linear relation $y=a+bx$, where a and b are constants.

Suppose you are at time t, and you want to predict the value y at time $t+1$. You can follow these three steps to forecast:

1. Run a regression of $y=a+bx$ using historical data, i.e., observations from t, $t-1$, $t-2$, etc. From this step, you obtain the estimated values of a and b, and let's call them $a^$ and $b^$, respectively.
2. Estimate the value of x at time $t+1$, $E(x_{t+1})$. If the value of x at time $t+1$ is observable at time t, that's even better: you don't even need to estimate. For example, interest rates are pre-determined if a company borrows money from a bank, so companies know what interest rate they'll have to pay next day or next month. Sometimes the value of x at time $t+1$ is unobservable at time t. You then estimate it based on historical values or using other methods. For example, you don't know what the actual inflation rate will be next month and you have to estimate it.
3. The forecast for y at time $t+1$ is, $E(y_{t+1}) = a^+b^*E(x_{t+1})$.

Below, we use Capital Asset Pricing Model (CAPM) as a concrete example of the above regression analysis approach.

Use CAPM to Forecast Stock Returns

Let's use an example to illustrate how regression analyses can be used to forecast.

Suppose you are at the end of the trading day, June 25, 2020. You want to forecast the stock return of, say, Apple for the next trading day, June 26, 2020. You believe that Apple stock return is related to the market return as described in the Capital Asset Pricing Model (CAPM). Specifically, $E(r_{t+1}) = alpha+beta*E(rm_{t+1})$, where $E(r_{t+1})$ is the expected excess return on the stock that you are interested in, $E(rm_{t+1})$ is the expected excess market return (say, the excess return on S&P500 index), and $alpha$ and $beta$ are constants.

Since we are using daily returns and the risk-free rate is small, we ignore the risk-free rate. That is, we treat excess returns and returns the same.

NOTE: You can read more about CAPM here

https://en.wikipedia.org/wiki/Capital_asset_pricing_model

The program *CAPM.py*, as shown in Listing 7-3, predicts the Apple stock return.

```
# import needed modules
import statsmodels.api as sm   #❶
from pandas_datareader import data as pdr

# set the start and end dates
start_date = "2019-12-25"  #❷
end_date = "2020-06-25"
# choose market index (S&P500) and stock ticker symbols
market = "^GSPC"
ticker = "AAPL"
sp = pdr.get_data_yahoo( market, start=start_date, end=end_date)
stock = pdr.get_data_yahoo(ticker, start=start_date, end=end_date)
#calculate returns for sp500 and the stock
sp['ret_sp'] =(sp['Adj Close']/sp['Adj Close'].shift(1))-1
stock['ret_stock'] =(stock['Adj Close']/stock['Adj Close'].shift(1))-1
# merge the two datasets, keep only returns
df = sp[['ret_sp']].merge(stock[['ret_stock']], left_index=True,
right_index=True)    #❸
# we need a constant to run regressions
df['const'] = 1   #❹
# remove missing values
df.dropna(inplace=True)
# Calculate the stock's alpha and Beta
reg = sm.OLS(endog=df['ret_stock'], exog=df[['const', 'ret_sp']],
missing='drop')   #❺
results = reg.fit()
alpha=results.params['const']
print('alpha is', alpha)
beta=results.params['ret_sp']
print('beta is', beta)
#calculate the expected market return
mkt_ret=df['ret_sp'].mean()   #❻
print('the market return is', mkt_ret)
```

```
# forecast
expected_ret=alpha+beta*mkt_ret    #❼
print('the forecast for Apple stock return on June 26, 2020, is',
expected_ret)
```

Listing 7-3: The program to forecast stock returns using CAPM

We first import needed modules. In particular, we import the *api* module from the *statsmodels* package to run regressions ❶. We retrieve the daily price information from July 2019 to June 2020 for both the S&P500 index and the Apple stock and calculate their daily returns ❷. At ❸, we merge the two datasets (the return on the S&P500 index and the return on the Apple stock) into one dataset. We also add a constant to the dataset because it's needed in the regression ❹.

We use the *api* module from the *statsmodels* package to run a regression ❺. The intercept from the regression is the value of *alpha*, and the coefficient on the market return is *beta*. We calculate the average return on the S&P500 index and use it as the expected rate of return on the market ❻.

To forecast the next-day return on the Apple stock, we use the formula *E(r) = alpha+beta*E(rm)* and plug in the corresponding values ❼.

The output from the program is as follows.

```
alpha is 0.0024050362677401653
beta is 1.0544868592390895
the market return is 2.341585216178732e-05
the forecast for Apple stock return on June 26, 2020, is 0.002429727976142655
```

As you can see, the program predicts that the Apple stock return on June 26, 2020, is about 0.243%.

You can then obtain the predicted Apple stock price based on current stock price and the predicted stock return, by using this formula:

```
Next day's price = today's price × (1 + predicted next day return)
```

I'll leave that as a homework for you to finish yourself.

Summary

In this chapter, you first learned how to use moving average to forecast. You also learned the trade-off between smoothing and tracking in order to choose the right window to calculate the moving average. You then use the least squared errors method to test the accuracy of your forecasts.

You also learned to use regression analyses to forecast. Your first learned the idea behind the regression analyses. You then use CAPM as a concrete example to forecast stock returns.

End of Chapter Exercises

1: Modify *MovingAverage.py* to forecast the stock price of Kroger on Jan 31, 2023, based on the moving average in the previous three months. Show the modified script. Note you need to convert return to price.

2: Modify *LeastSquareErrors.py* so that you use 1-month, 3-month, and six-month moving averages to forecast the stock return and price of Walmart on Jan 31, 2023. Then calculate the accuracy of each method. Which method is the most accurate based on your results? Show the script and results.

3: Modify *CAPM.py* to forecast the stock return of General Motors on Jan 31, 2023, based on regression analyses using daily prices from the previous six months. Show the modified script. What's the alpha and beta based on the regression?

4: Add a few lines of code at the end of *CAPM.py* to print out the adjusted closing price of Apple on Aug 25, 2020. Then calculate the predicted price on Aug 26, 2020, and print it out (hint, see the last few lines of code in *MovingAverage.py*)