Stock Market Analysis



Type Markdown and LaTeX: α2α2 add Codeadd Markdown

Type Markdown and LaTeX: α2α2 add Codeadd Markdown

Type Markdown and LaTeX: α2α2 add Codeadd Markdown

Type Markdown and LaTeX: α2α2 add Codeadd Markdown

Type Markdown and LaTeX: α2α2 add Codeadd Markdown

1. **What was the change in price of the stock overtime?** In this section we'll go over how to handle requesting stock information with pandas, and how to analyze basic attributes of a stock.

add Codeadd Markdown

**!**pip install **-**q yfinance

add Codeadd Markdown **import** pandas **as** pd **import** numpy **as** np

**import** matplotlib.pyplot **as** plt **import** seaborn **as** sns sns.set\_style('whitegrid') plt.style.use("fivethirtyeight")

**%**matplotlib inline

*# For reading stock data from yahoo*

**from** pandas\_datareader.data **import** DataReader

**import** yfinance **as** yf

**from** pandas\_datareader **import** data **as** pdr yf.pdr\_override()

*# For time stamps*

**from** datetime **import** datetime

*# The tech stocks we'll use for this analysis*

tech\_list **=** ['AAPL', 'GOOG', 'MSFT', 'AMZN']

*# Set up End and Start times for data grab*

tech\_list **=** ['AAPL', 'GOOG', 'MSFT', 'AMZN']

end **=** datetime.now()

start **=** datetime(end.year **-** 1, end.month, end.day)

**for** stock **in** tech\_list:

globals()[stock] **=** yf.download(stock, start, end)

company\_list **=** [AAPL, GOOG, MSFT, AMZN] company\_name **=** ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

**for** company, com\_name **in** zip(company\_list, company\_name): company["company\_name"] **=** com\_name

df **=** pd.concat(company\_list, axis**=**0) df.tail(10)

add Codeadd Markdown

Reviewing the content of our data, we can see that the data is numeric and the date is the index of the data. Notice also that weekends are missing from the records.

**Quick note:** Using globals() is a sloppy way of setting the DataFrame names, but it's simple. Now we have our data, let's perform some basic data analysis and check our data.

add Codeadd Markdown

Time Series data is a series of data points indexed in time order. Time series data is everywhere, so manipulating them is important for any data analyst or data scientist.

In this notebook, we will discover and explore data from the stock market, particularly some technology stocks (Apple, Amazon, Google, and Microsoft). We will learn how to use yfinance to get stock information, and visualize different aspects of it using Seaborn and Matplotlib. we will look at a few ways of analyzing the risk of a stock, based on its previous performance history. We will also be predicting future stock prices through a Long Short Term Memory (LSTM) method!

We'll be answering the following questions along the way:

1.) What was the change in price of the stock over time? 2.) What was the daily return of the stock on average?

3.) What was the moving average of the various stocks? 4.) What was the correlation between different stocks'?

5.) How much value do we put at risk by investing in a particular stock?

6.) How can we attempt to predict future stock behavior? (Predicting the closing price stock price of APPLE inc using LSTM)

# Getting the Data

The first step is to get the data and load it to memory. We will get our stock data from the Yahoo Finance website. Yahoo Finance is a rich resource of financial market data and tools to find compelling investments. To get the data from Yahoo Finance, we will be using yfinance library which offers a threaded and Pythonic way to download market data from Yahoo. Check this article to learn more about finance.

add Codeadd Markdown

# Descriptive Statistics about the Data

.describe() generates descriptive statistics. Descriptive statistics include those that summarize the central tendency, dispersion, and shape of a dataset’s distribution, excluding NaN values.

Analyzes both numeric and object series, as well

as DataFrame column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

add Codeadd Markdown

[ ]:

*# Summary Stats*

AAPL.describe()

add Codeadd Markdown

We have only 255 records in one year because weekends are not included in the data.

add Codeadd Markdown

# Information About the Data

.info() method prints information about a DataFrame including the index dtype and columns, non-null values, and memory usage.

add Codeadd Markdown

[ ]:

*# General info*

AAPL.info()

add Codeadd Markdown

# Closing Price

The closing price is the last price at which the stock is traded during the regular trading day. A stock’s closing price is the standard benchmark used by investors to track its performance over time.

add Codeadd Markdown

[ ]:

*# Let's see a historical view of the closing price* plt.figure(figsize**=**(15, 10)) plt.subplots\_adjust(top**=**1.25, bottom**=**1.2)

**for** i, company **in** enumerate(company\_list, 1): plt.subplot(2, 2, i)

company['Adj Close'].plot() plt.ylabel('Adj Close') plt.xlabel(**None**)

plt.title(f"Closing Price of {tech\_list[i **-** 1]}")

plt.tight\_layout()

add Codeadd Markdown

# Volume of Sales

Volume is the amount of an asset or security that changes hands over some period of time, often over the course of a day. For instance, the

stock trading volume would refer to the number of shares of security traded between its daily open and close. Trading volume, and changes to volume over the course of time, are important inputs for technical traders.

add Codeadd Markdown

[ ]:

*# Now let's plot the total volume of stock being traded each day*

plt.figure(figsize**=**(15, 10)) plt.subplots\_adjust(top**=**1.25, bottom**=**1.2)

**for** i, company **in** enumerate(company\_list, 1): plt.subplot(2, 2, i) company['Volume'].plot() plt.ylabel('Volume')

plt.xlabel(**None**)

plt.title(f"Sales Volume for {tech\_list[i **-** 1]}")

plt.tight\_layout()

add Codeadd Markdown

Now that we've seen the visualizations for the closing price and the volume traded each day, let's go ahead and caculate the moving average for the stock.

add Codeadd Markdown

# What was the moving average of the various stocks?

The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 10 days, 20 minutes, 30 weeks, or any time period the trader chooses.

add Codeadd Markdown

[ ]:

ma\_day **=** [10, 20, 50]

**for** ma **in** ma\_day:

**for** company **in** company\_list:

column\_name **=** f"MA for {ma} days" company[column\_name] **=** company['Adj

Close'].rolling(ma).mean()

fig, axes **=** plt.subplots(nrows**=**2, ncols**=**2) fig.set\_figheight(10)

fig.set\_figwidth(15)

AAPL[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax**=**axes[0,0])

axes[0,0].set\_title('APPLE')

GOOG[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax**=**axes[0,1])

axes[0,1].set\_title('GOOGLE')

MSFT[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax**=**axes[1,0])

axes[1,0].set\_title('MICROSOFT')

AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50 days']].plot(ax**=**axes[1,1])

axes[1,1].set\_title('AMAZON')

fig.tight\_layout()

add Codeadd Markdown

We see in the graph that the best values to measure the moving average are 10 and 20 days because we still capture trends in the data without noise.

add Codeadd Markdown

# What was the daily return of the stock on average?

add Codeadd Markdown

Now that we've done some baseline analysis, let's go ahead and dive a little deeper. We're now going to analyze the risk of the stock. In order

to do so we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. Let's go ahead and use pandas to retrieve teh daily returns for the Apple stock.

add Codeadd Markdown [ ]:

*# We'll use pct\_change to find the percent change for each day*

**for** company **in** company\_list:

company['Daily Return'] **=** company['Adj Close'].pct\_change()

*# Then we'll plot the daily return percentage* fig, axes **=** plt.subplots(nrows**=**2, ncols**=**2) fig.set\_figheight(10)

fig.set\_figwidth(15)

AAPL['Daily Return'].plot(ax**=**axes[0,0], legend**=True**, linestyle**=**'--', marker**=**'o')

axes[0,0].set\_title('APPLE')

GOOG['Daily Return'].plot(ax**=**axes[0,1], legend**=True**, linestyle**=**'--', marker**=**'o')

axes[0,1].set\_title('GOOGLE')

MSFT['Daily Return'].plot(ax**=**axes[1,0], legend**=True**, linestyle**=**'--', marker**=**'o')

axes[1,0].set\_title('MICROSOFT')

AMZN['Daily Return'].plot(ax**=**axes[1,1], legend**=True**, linestyle**=**'--', marker**=**'o')

axes[1,1].set\_title('AMAZON')

fig.tight\_layout()

add Codeadd Markdown

Great, now let's get an overall look at the average daily return using a histogram. We'll use seaborn to create both a histogram and kde plot on the same figure.

add Codeadd Markdown

[ ]:

plt.figure(figsize**=**(12, 9))

**for** i, company **in** enumerate(company\_list, 1): plt.subplot(2, 2, i)

company['Daily Return'].hist(bins**=**50) plt.xlabel('Daily Return') plt.ylabel('Counts') plt.title(f'{company\_name[i **-** 1]}')

plt.tight\_layout()

add Codeadd Markdown

# What was the correlation between different stocks closing prices?

add Codeadd Markdown

Correlation is a statistic that measures the degree to which two variables move in relation to each other which has a value that must fall between -1.0 and +1.0. Correlation measures association, but doesn’t show if x causes y or vice versa — or if the association is caused by a third factor[1].

Now what if we wanted to analyze the returns of all the stocks in our list? Let's go ahead and build a DataFrame with all the ['Close'] columns for each of the stocks dataframes.

add Codeadd Markdown [ ]:

*# Grab all the closing prices for the tech stock list into one DataFrame*

closing\_df **=** pdr.get\_data\_yahoo(tech\_list, start**=**start, end**=**end)['Adj Close']

*# Make a new tech returns DataFrame* tech\_rets **=** closing\_df.pct\_change() tech\_rets.head()

add Codeadd Markdown

Now we can compare the daily percentage return of two stocks to check how correlated. First let's see a sotck compared to itself.

add Codeadd Markdown [ ]:

*# Comparing Google to itself should show a perfectly linear relationship*

sns.jointplot(x**=**'GOOG', y**=**'GOOG', data**=**tech\_rets, kind**=**'scatter', color**=**'seagreen')

add Codeadd Markdown [ ]:

*# We'll use joinplot to compare the daily returns of Google and Microsoft*

sns.jointplot(x**=**'GOOG', y**=**'MSFT', data**=**tech\_rets, kind**=**'scatter') add Codeadd Markdown

So now we can see that if two stocks are perfectly (and positivley) correlated with each other a linear relationship bewteen its daily return values should occur.

Seaborn and pandas make it very easy to repeat this comparison analysis for every possible combination of stocks in our technology stock ticker list. We can use sns.pairplot() to automatically create this plot

add Codeadd Markdown [ ]:

*# We can simply call pairplot on our DataFrame for an automatic visual analysis*

*# of all the comparisons*

sns.pairplot(tech\_rets, kind**=**'reg') add Codeadd Markdown

Above we can see all the relationships on daily returns between all the stocks. A quick glance shows an interesting correlation between Google and Amazon daily returns. It might be interesting to investigate that individual comaprison.

While the simplicity of just calling sns.pairplot() is fantastic we can also use sns.PairGrid() for full control of the figure, including what

kind of plots go in the diagonal, the upper triangle, and the lower triangle. Below is an example of utilizing the full power of seaborn to achieve this result.

add Codeadd Markdown [ ]:

*# Set up our figure by naming it returns\_fig, call PairPLot on the DataFrame*

return\_fig **=** sns.PairGrid(tech\_rets.dropna())

*# Using map\_upper we can specify what the upper triangle will look like.*

return\_fig.map\_upper(plt.scatter, color**=**'purple')

*# We can also define the lower triangle in the figure, inclufing the plot type (kde)*

*# or the color map (BluePurple)*

return\_fig.map\_lower(sns.kdeplot, cmap**=**'cool\_d')

*# Finally we'll define the diagonal as a series of histogram plots of the daily return*

return\_fig.map\_diag(plt.hist, bins**=**30) add Codeadd Markdown

[ ]:

*# Set up our figure by naming it returns\_fig, call PairPLot on the DataFrame*

returns\_fig **=** sns.PairGrid(closing\_df)

*# Using map\_upper we can specify what the upper triangle will look like.*

returns\_fig.map\_upper(plt.scatter,color**=**'purple')

*# We can also define the lower triangle in the figure, inclufing the plot type (kde) or the color map (BluePurple)* returns\_fig.map\_lower(sns.kdeplot,cmap**=**'cool\_d')

*# Finally we'll define the diagonal as a series of histogram plots of the daily return*

returns\_fig.map\_diag(plt.hist,bins**=**30) add Codeadd Markdown

Finally, we could also do a correlation plot, to get actual numerical values for the correlation between the stocks' daily return values. By comparing the closing prices, we see an interesting relationship between Microsoft and Apple.

add Codeadd Markdown [ ]:

plt.figure(figsize**=**(12, 10))

plt.subplot(2, 2, 1)

sns.heatmap(tech\_rets.corr(), annot**=True**, cmap**=**'summer') plt.title('Correlation of stock return')

plt.subplot(2, 2, 2)

sns.heatmap(closing\_df.corr(), annot**=True**, cmap**=**'summer') plt.title('Correlation of stock closing price')

add Codeadd Markdown

Just like we suspected in our PairPlot we see here numerically and visually that Microsoft and Amazon had the strongest correlation of daily stock return. It's also interesting to see that all the technology comapnies are positively correlated.

add Codeadd Markdown

# How much value do we put at risk by investing in a particular stock?

add Codeadd Markdown

There are many ways we can quantify risk, one of the most basic ways using the information we've gathered on daily percentage returns is by comparing the expected return with the standard deviation of the daily returns.

add Codeadd Markdown [ ]:

rets **=** tech\_rets.dropna()

area **=** np.pi **\*** 20

plt.figure(figsize**=**(10, 8)) plt.scatter(rets.mean(), rets.std(), s**=**area) plt.xlabel('Expected return') plt.ylabel('Risk')

**for** label, x, y **in** zip(rets.columns, rets.mean(), rets.std()): plt.annotate(label, xy**=**(x, y), xytext**=**(50, 50), textcoords**=**'offset

points', ha**=**'right', va**=**'bottom',

arrowprops**=**dict(arrowstyle**=**'-', color**=**'blue', connectionstyle**=**'arc3,rad=-0.3'))

add Codeadd Markdown

# Predicting the closing price stock price of APPLE inc:

add Codeadd Markdown

*# Get the stock quote*

df **=** pdr.get\_data\_yahoo('AAPL', start**=**'2012-01-01', end**=**datetime.now())

*# Show teh data*

df

add Codeadd Markdown plt.figure(figsize**=**(16,6)) plt.title('Close Price History') plt.plot(df['Close']) plt.xlabel('Date', fontsize**=**18)

plt.ylabel('Close Price USD ($)', fontsize**=**18) plt.show()

add Codeadd Markdown

*# Create a new dataframe with only the 'Close column*

data **=** df.filter(['Close'])

*# Convert the dataframe to a numpy array*

dataset **=** data.values

*# Get the number of rows to train the model on*

training\_data\_len **=** int(np.ceil( len(dataset) **\*** .95 ))

training\_data\_len

add Codeadd Markdown

*# Scale the data*

**from** sklearn.preprocessing **import** MinMaxScaler

scaler **=** MinMaxScaler(feature\_range**=**(0,1)) scaled\_data **=** scaler.fit\_transform(dataset)

scaled\_data

add Codeadd Markdown

*# Create the training data set*

*# Create the scaled training data set*

train\_data **=** scaled\_data[0:int(training\_data\_len), :] *# Split the data into x\_train and y\_train data sets* x\_train **=** []

y\_train **=** []

**for** i **in** range(60, len(train\_data)): x\_train.append(train\_data[i**-**60:i, 0])

y\_train.append(train\_data[i, 0])

**if** i**<=** 61: print(x\_train) print(y\_train) print()

*# Convert the x\_train and y\_train to numpy arrays*

x\_train, y\_train **=** np.array(x\_train), np.array(y\_train)

*# Reshape the data*

x\_train **=** np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))

*# x\_train.shape*

add Codeadd Markdown

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, LSTM

*# Build the LSTM model*

model **=** Sequential()

model.add(LSTM(128, return\_sequences**=True**, input\_shape**=** (x\_train.shape[1], 1)))

model.add(LSTM(64, return\_sequences**=False**)) model.add(Dense(25))

model.add(Dense(1))

*# Compile the model*

model.compile(optimizer**=**'adam', loss**=**'mean\_squared\_error')

*# Train the model*

model.fit(x\_train, y\_train, batch\_size**=**1, epochs**=**1) add Codeadd Markdown

*# Create the testing data set*

*# Create a new array containing scaled values from index 1543 to 2002*

test\_data **=** scaled\_data[training\_data\_len **-** 60: , :]

*# Create the data sets x\_test and y\_test*

x\_test **=** []

y\_test **=** dataset[training\_data\_len:, :]

**for** i **in** range(60, len(test\_data)): x\_test.append(test\_data[i**-**60:i, 0])

*# Convert the data to a numpy array*

x\_test **=** np.array(x\_test)

*# Reshape the data*

x\_test **=** np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1 ))

*# Get the models predicted price values*

predictions **=** model.predict(x\_test)

predictions **=** scaler.inverse\_transform(predictions)

*# Get the root mean squared error (RMSE)*

rmse **=** np.sqrt(np.mean(((predictions **-** y\_test) **\*\*** 2))) rmse

add Codeadd Markdown

*# Plot the data*

train **=** data[:training\_data\_len]

valid **=** data[training\_data\_len:] valid['Predictions'] **=** predictions *# Visualize the data* plt.figure(figsize**=**(16,6)) plt.title('Model') plt.xlabel('Date', fontsize**=**18)

plt.ylabel('Close Price USD ($)', fontsize**=**18) plt.plot(train['Close'])

plt.plot(valid[['Close', 'Predictions']])

plt.legend(['Train', 'Val', 'Predictions'], loc**=**'lower right') plt.show()

add Codeadd Markdown

*# Show the valid and predicted prices*

valid

add Codeadd Markdown