**Stock Market Analysis**

In this lesson, you will learn about the stock market and how you can analyze its data to extract useful insights from it.

**We'll cover the following**

* + [Stock market](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#stock-market)
    - [Some stock market jargon](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#some-stock-market-jargon)
  + [Let’s dive in](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#lets-dive-in)
    - [Results?](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#results)

**Stock market**[#](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#stock-market)

A stock market represents the claims of companies and individuals on the shares of public companies. The prices of these shares vary depending on various factors, including inflation, demand, and reputation.

Buying and selling stocks is a major business, and people keep trying to predict the future stock behavior to earn large paybacks on their investments.

In this data analysis project, we try to analyze the data of various stocks to obtain valuable information. This information provides insights on which stocks generate more returns.

**Note:** There is no program in the world that can correctly predict future stock behavior. The techniques covered in this project only allow us to make an educated guess based on the previous behavior of a particular stock.

**Some stock market jargon**[#](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#some-stock-market-jargon)

* Closing price: The price of the stock when the market closes that day
* Daily returns: The increase or decrease in the percentage amount from the previous day
* Risk: The amount that can be lost
* Stock behavior: Whether the stock price would go up or down

**Let’s dive in**[#](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#lets-dive-in)

The stock data of the following four technology companies in Pakistan will be analyzed.

* **Systems Ltd**
* **NETSOL**
* **PTCL**
* **Avanceon**

First and foremost, we need the historical data of these companies. The data is available [here](https://dps.psx.com.pk/historical) and can be viewed by placing the relevant time period and company names in the search fields. A scraper will be used to pull all the data and put it into *CSV* files.

**Results?**[#](https://www.educative.io/courses/predictive-data-analysis-with-python/398GK0z4q0n#results)

We will try to answer the following questions after doing stock market data analysis:

1. How much did the stock price of each company change over time?
2. What were the daily returns for all the companies?
3. How are the stocks of the companies related to each other?
4. How much money do we risk losing by investing in a certain company?
5. Can we predict future stock behavior?

In the next lesson, the stocks data is explored along with some other analysis.

**Section 1: Price Trends**

In this lesson, stock price trends and the concept of moving averages are discussed.

**We'll cover the following**

* + [Stocks data](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#stocks-data)
  + [Stock price trend](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#stock-price-trend)
  + [Moving average](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#moving-average)
    - [Moving average comparisons](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#moving-average-comparisons)
  + [Comparison](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#comparison)

**Stocks data**[#](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#stocks-data)

The stock data for **Systems Ltd**, **NETSOL**, **PTCL**, and **Avanceon** for the year *2018* will be used for this analysis. Any stock data has six important columns.

* **Time**: The date and time of the day
* **Open**: The price of the stock at the start of the day when the market opens
* **High**: The highest price of the stock on that day
* **Low**: The lowest price of the stock on that day
* **Close**: The price of the stock at the end of the day when the market closes
* **Volume**: The number of stocks traded that day

Let’s look up the stocks data for **Systems Ltd**.

import pandas as pd

sys = pd.read\_csv('Year\_2018/SYS.csv')

#The "SYS" file name can be changed to "NETSOL", "AVN" and "PTC".

print(sys)

On **line 3**, the stock data for **Systems Ltd** is read in the DataFrame as a sys variable. The SYS.csv file contains this data. Similarly, by changing the SYS.csv to NETSOL.csv, PTC.csv, or AVN.csv, their data can also be obtained.

All the files are available for download below.

Year\_2018.zip

## Stock price trend [#](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#stock-price-trend)

For this analysis, the column Close will be of great importance. As mentioned above, this column contains the closing price of the stock of each day. So, let’s visualize this using a line graph how the closing price of **Systems Ltd** stock has varied throughout the year of 2018.

import pandas as pd

import matplotlib.pyplot as plt

sys = pd.read\_csv('Year\_2018/SYS.csv')

#The "SYS" file name can be changed to "NETSOL", "AVN" and "PTC".

sys['Time'] = pd.to\_datetime(sys.Time) # correct the format of date

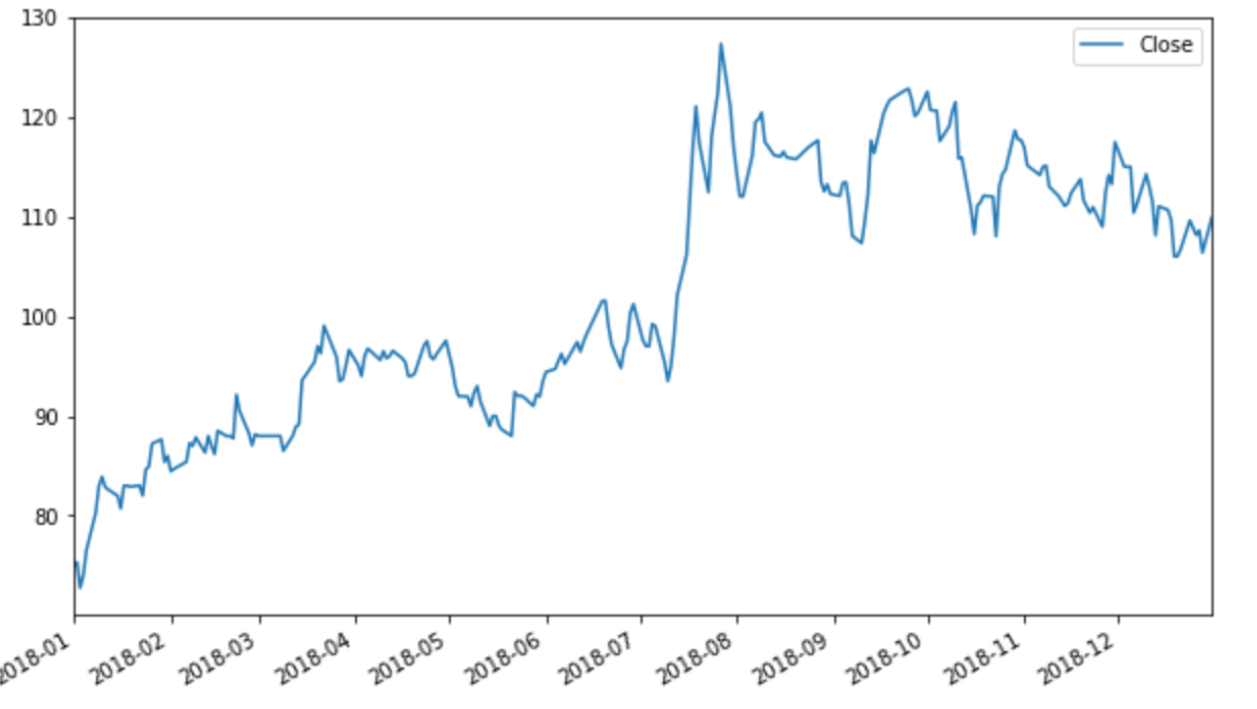
sys = sys.set\_index('Time') # Set Time column as row index

sys['Close'].plot(legend = True, figsize=(15,10))

On **line 7**, the date in the Time column is assigned to the correct date time format using the to\_datetime function of *Pandas*.

On **line 9**, the Time column is assigned as the row index as every other value of every column is uniquely associated with the Time column. Doing this automatically plots the graph according to different time frames.

On **line 11**, the values in the Close column are plotted against the index, which in this case is our Time column. The legend parameter shows the Close variable name using the *blue line* in the plot. The figsize parameter sets the dimensions of the plot.



The output of the code snippet generates the above graph. The x-axis represents the time frames and the y-axis represents the price.

It can be seen that the stock price of **Systems Ltd** is generally increasing for the year 2018. The stock price at the end of the year is higher than the price at the start of the year. The trends of the other three companies can also be viewed by replacing the SYS.csv file with the respective company file names.

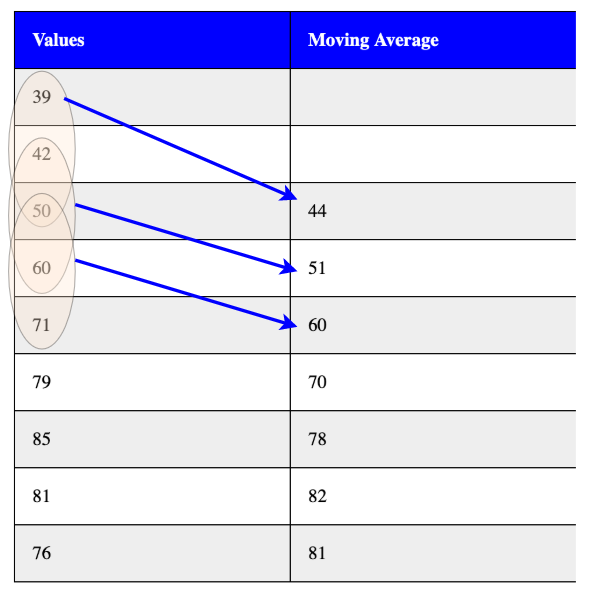
The trend graphs of other companies can be viewed in the output of the code by replacing the current company name with one of your choices.

In the graph above, the stock price goes up and down at random days. This creates a lot of noise in the trend. A more precise trend graph can be obtained by the moving average method.

## Moving average [#](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#moving-average)

A moving average method is used to smooth the short term random price changes by filtering unnecessary noise. As its name suggests, it takes a rolling average or mean of all prices in a specifically defined window. A single point of a moving average graph is plotted with information on the past prices; it is also called a trend indicator.

The working of the moving average can be understood from the following image.



import pandas as pd

import matplotlib.pyplot as plt

sys = pd.read\_csv('Year\_2018/SYS.csv') #The "SYS" file name can be changed to "NETSOL", "AVN" and "PTC".

sys['Time'] = pd.to\_datetime(sys.Time) # correct the format of date

sys = sys.set\_index('Time') # Set Time column as row index

days = 50 # Moving average window

col\_name = "mv\_avg for " + str(days) + " days" # New column to store moving average vlues

sys[col\_name] = sys['Close'].rolling(days).mean() #Calculating moving average

print(sys)

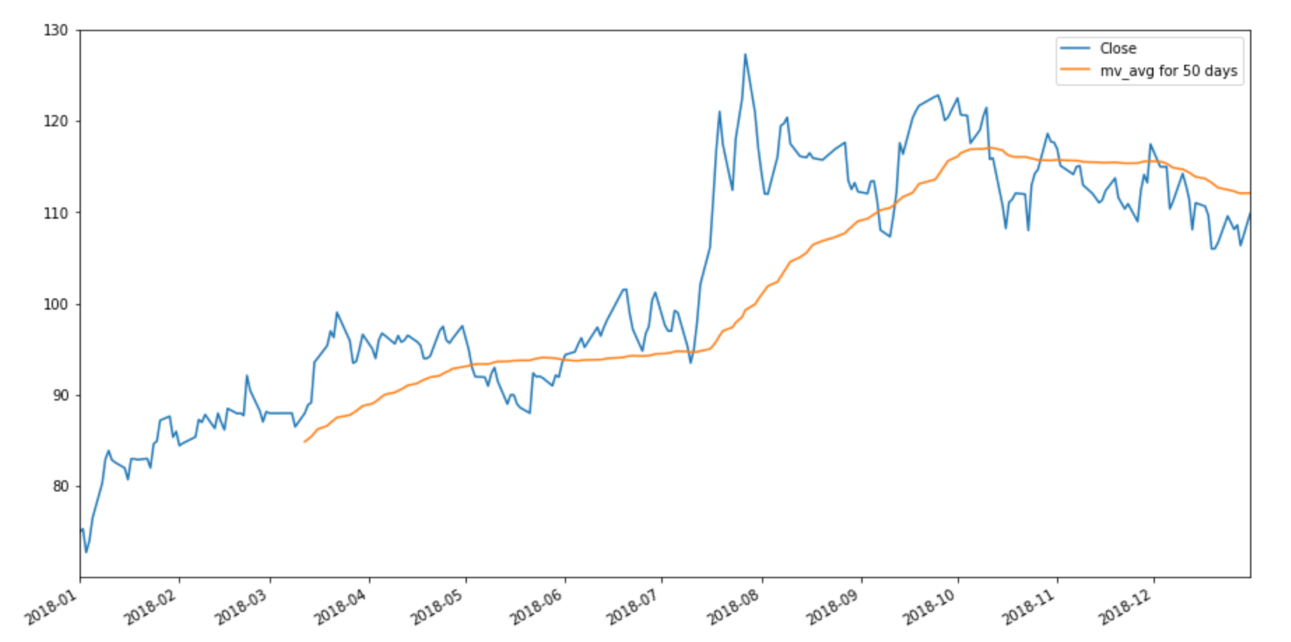
sys[['Close', 'mv\_avg for 50 days']].plot(figsize=(15,10)) # Plotting the closing price with moving avarage for comprison

On \*\* lines 8 & 9\*\*, the window length and the new column name are defined, respectively.

On **line 11**, the rolling function of *pandas* takes the days as a parameter and applies the mean, or average, operation on the values of the Close column. The rolling function automatically does the moving task.

On **line 15**, the Close column and the new *moving average* column are plotted together for comparison.

The values of the new *moving average* column can be seen in the **output** by clicking the **right** arrow. Each value of the new column is an average of fifty-row values of the Close column, so there won’t be any values for the first fifty rows as the first value will occur on the *fifty-first* row.

The output of the code snippet generates the above graph. The x-axis and y-axis represent the same values as before. It can be seen that, in the moving average graph, all the fluctuations are gone. A smooth curve represents the trend of stock behavior.

In the moving average, the length of the window or the number of days is of significant importance. A lower number of days is highly sensitive to fluctuation changes, while a higher number of days is less sensitive to these changes.

Keep in mind that less sensitivity offers more smooth curves.

### Moving average comparisons [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#moving-average-comparisons)

Let’s visualize how lower and higher numbers of days affect the moving average curve.

import pandas as pd

import matplotlib.pyplot as plt

sys = pd.read\_csv('Year\_2018/SYS.csv') #The "SYS" file name can be changed to "NETSOL", "AVN" and "PTC".

sys['Time'] = pd.to\_datetime(sys.Time)

sys = sys.set\_index('Time')

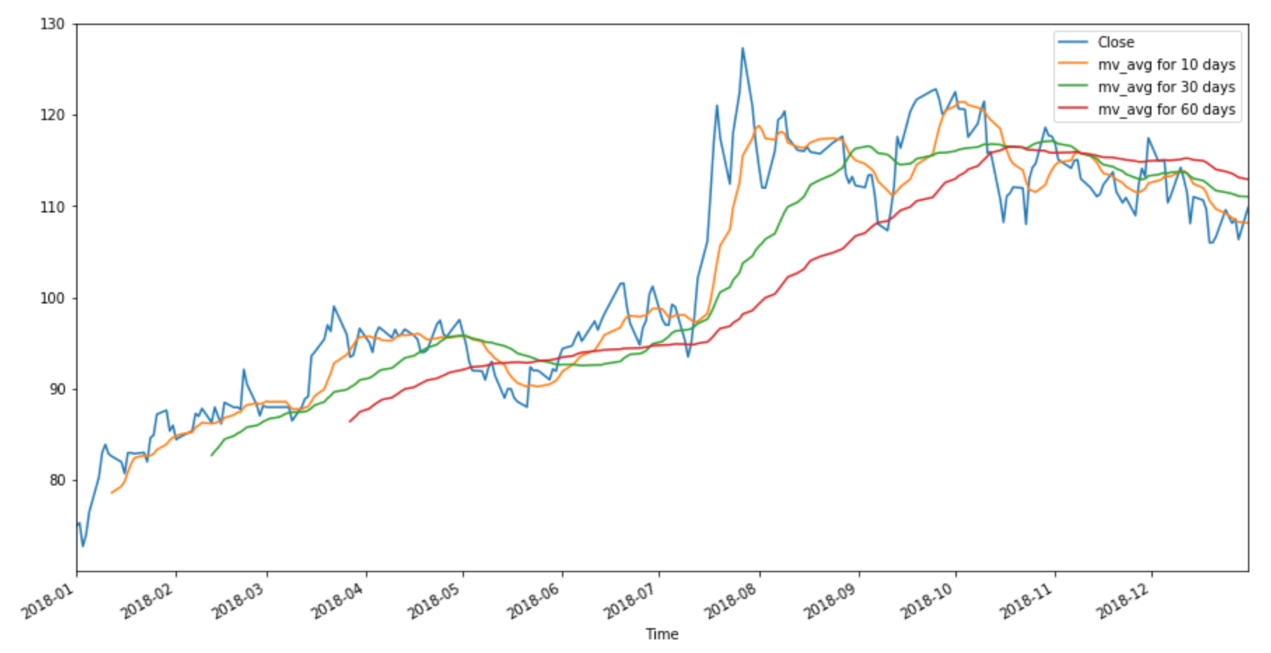
days = [10, 30 , 60] # Multiple number of days

for day in days:

    col\_name = "mv\_avg for " + str(day) + " days"

    sys[col\_name] = sys['Close'].rolling(day).mean()

sys[['Close', 'mv\_avg for 10 days', 'mv\_avg for 30 days', 'mv\_avg for 60 days']].plot(subplots = False, figsize=(15,10))



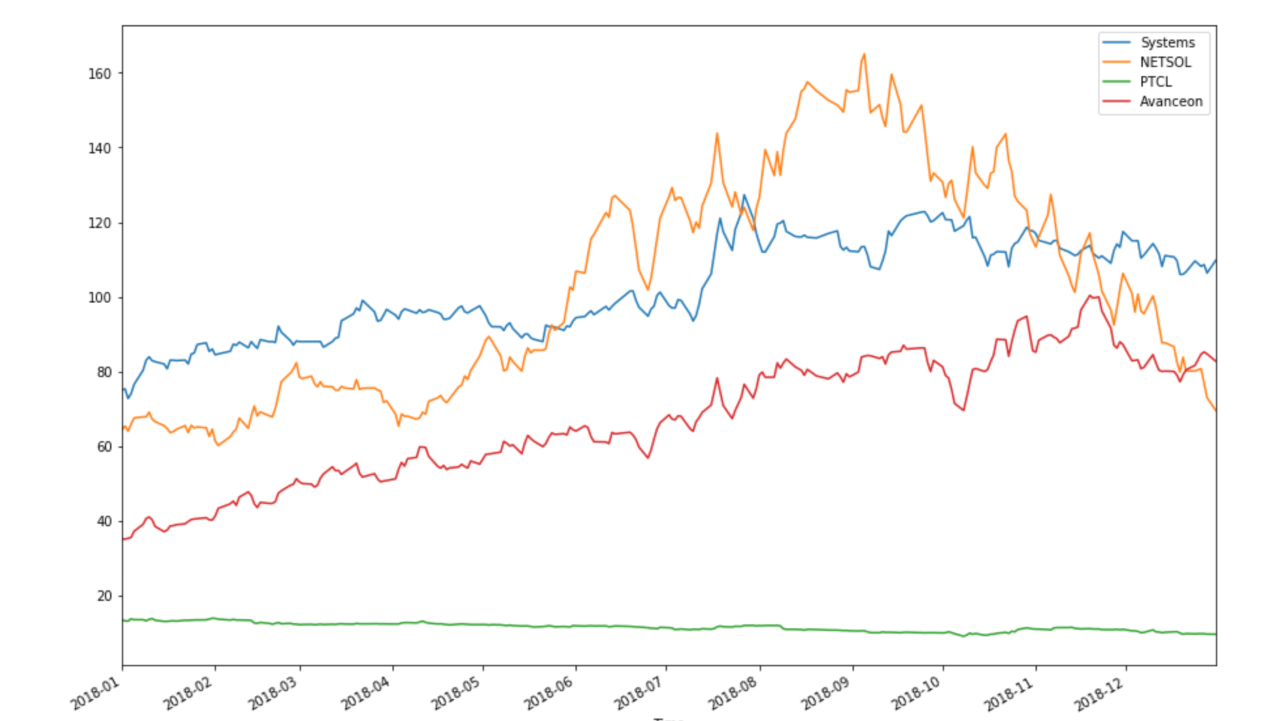
The trends for different days are clearly plotted and separated by different colors. We can see that as the number of days increases the moving average curve becomes smoother in determining the trend. Still, it should also be noted that all moving average curves give a better understanding of the stock trend than the regular plot.

For more information on how moving averages can help in determining stock trends, refer [here](https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp).

**NOTE**: All the above information can be generated for any of the other companies as well. Just replace the SYS.csv file name with the respective company’s file name, as described above.

## Comparison [#](https://www.educative.io/courses/predictive-data-analysis-with-python/gkPrrGKKvMk#comparison)

We can determine which of the four companies stocks did well for the year 2018 by plotting the closing prices of each company’s stock. The following image shows this plot.



It can be seen from the comparisons of the closing price of the four companies that 2018 was not a good year for **NETSOL** and **PTC**. The stocks of both **Systems Ltd** and **Avnceaon** did pretty good and showed steady growth.

# Section 2: Daily Returns

In this lesson, the daily returns of different stocks are calculated.

###### We'll cover the following

* + [Daily returns](https://www.educative.io/courses/predictive-data-analysis-with-python/7DRjqqBBwy8#daily-returns)
    - [Calculating daily returns](https://www.educative.io/courses/predictive-data-analysis-with-python/7DRjqqBBwy8#calculating-daily-returns)
  + [Estimating daily return](https://www.educative.io/courses/predictive-data-analysis-with-python/7DRjqqBBwy8#estimating-daily-return)

## Daily returns [#](https://www.educative.io/courses/predictive-data-analysis-with-python/7DRjqqBBwy8#daily-returns)

The price of stock changes on a daily basis. Daily return calculations tell us how much the current day stock value is different from the previous day stock value. A positive change indicates a rise in the value of the stock while a negative change indicates a fall in the value of the stock. A stock with minimal positive or negative change is considered to be a stable and good stock.

As seen in the previous lesson, only **Systems Ltd** and **Avanceon** stocks provide promising results for the year 2018. So, daily returns of **Systems Ltd** will be calculated and visualized in this lesson as this company had a higher value compared to **Avanceon**. Results for any company can be obtained by simply changing the name for the company file as discussed before.

### Calculating daily returns [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/7DRjqqBBwy8#calculating-daily-returns)

We don’t need to figure out the formula or steps for calculating the daily returns because pandas already provides a built-in function for it. The pct\_change function is called from a Series object, and it calculates the daily return for all rows based on the current and previous row value. This function returns a new Series with the calculated daily returns. The first row has no, or NaN, value as there is no previous value for it.

import pandas as pd

import matplotlib.pyplot as plt

sys = pd.read\_csv('Year\_2018/SYS.csv')

#The "SYS" file name can be changed to "NETSOL", "AVN" and "PTC".

sys['Time'] = pd.to\_datetime(sys.Time) # correct the format of date

sys = sys.set\_index('Time') # Set Time column as row index

daily\_return = sys['Close'].pct\_change() # Calculate the daily returns

sys['daily\_return'] = daily\_return # Create new column and assign daily return values to it

plt.ylabel('Percentage Change') # Assign a name to the y-axis of plot

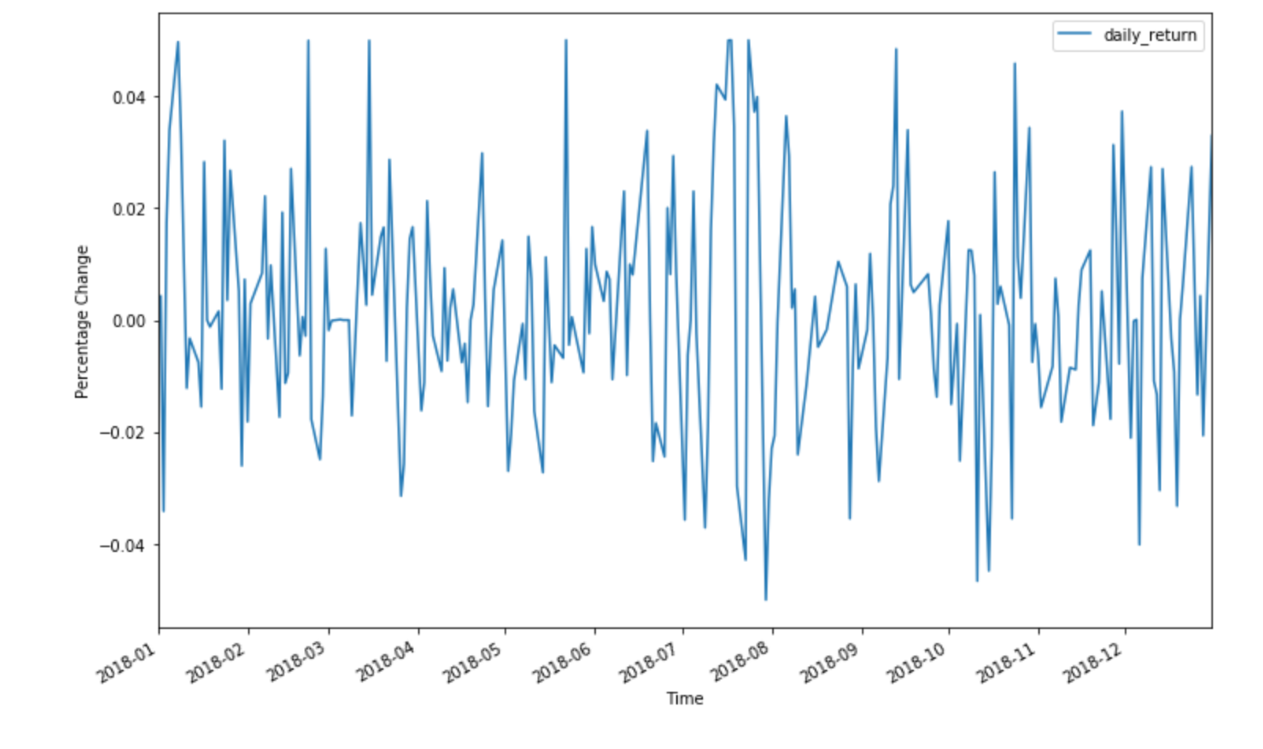
sys['daily\_return'].plot(legend = True, figsize=(15,10)) # plot the daily return values

On **line 9**, the pct\_change function is called by the Close column. This returns a new Series in the daily\_returns variable. The initial DataFrame values are sorted on the Time, i.e., the data is arranged in ascending order of days, so it calculates the correct daily return for each day.

On **line 11**, a new column named daily\_returns is created in the sys dataframe and the Series variable daily\_returns is assigned to it. Now, every date has the daily return of that day assigned to it.

On **line 13**, the function plt.label of matplotlib is used to assign a name to the y-axis.

On **line 15**, the new daily\_returns column of the sys dataframe is plotted.

The output of the code snippet generates the above graph. The x-axis represents the time frames, and the y-axis represents the percentage change in daily stock values. It can be observed that the change in daily returns is between (-0.04%, 0.04%), which is not that big of a change.

As mentioned above, the stock with minimal change is considered a stable stock. So, the **Systems Ltd** stock is a good stock to hold according to the daily returns.

## Estimating daily return [#](https://www.educative.io/courses/predictive-data-analysis-with-python/7DRjqqBBwy8#estimating-daily-return)

Which daily return values are likely to occur using the probability density function can also be determined. This operation can easily be performed using the KDE plot, as mentioned in the previous [lesson](https://www.educative.io/collection/page/10370001/5982561936343040/5678975687852032#kde-kernel-density-estimation-plots).

The KDE plot plots the probability density function values on the y-axis with respect to the values of the x-axis. The histogram can be used to determine the amount of those likely occurring values by dividing them into continuous intervals. Let’s visualize this with an example for better understanding.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sys = pd.read\_csv('Year\_2018/SYS.csv')

#The "SYS" file name can be changed to "NETSOL", "AVN" and "PTC".

sys['Time'] = pd.to\_datetime(sys.Time) # correct the format of date

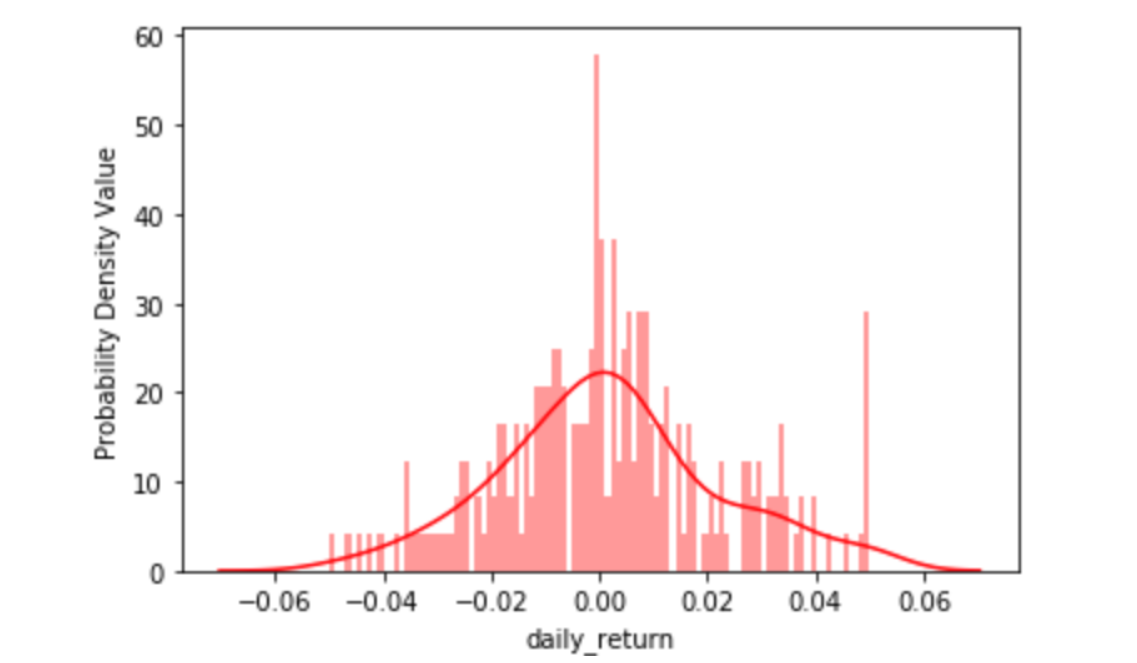
sys = sys.set\_index('Time') # Set Time column as row index

daily\_return = sys['Close'].pct\_change() # Calculate the daily returns

sys['daily\_return'] = daily\_return # Create new column and assign daily return values to it

plt.ylabel('Probability Density Value') # Assign a name to the y-axis of plot

sns.distplot(sys['daily\_return'].dropna(), bins = 100, color = 'red') # plots a distribution graph of KDE and histogram

On **line 16**, the distplot() function is used to plot a *KDE* over a histogram. The *KDE* is discussed extensively [here](https://www.educative.io/collection/page/10370001/5982561936343040/5678975687852032#kde-kernel-density-estimation-plots).

The x-axis has daily return values. The y-axis values represent how likely the value on the x-axis occurs. The higher values mean more likelihood of a value occurring on the x-axis. In the coming lessons, we will determine the average daily return percentage. The [histogram](https://www.educative.io/collection/page/10370001/5982561936343040/6367995153612800) shows how many values each of the *one-hundred* bins contain. The y-axis here does not provide any information about the histogram.

The *KDE* plot informs us that most of the daily returns for **System Ltd** are close to **zero**. This tells us that the changes in price of this stock are not drastic and we can assume that **System Ltd** has a stable stock.

Try changing the file name to other company’s names and observe how the plots change for them and what information can be extracted from them.

In the next lesson, we provide an explanation of how the stock behaviors of these different companies are related to each other using the correlation concepts.

# Section 3: Correlation in Stocks

In this lesson, the correlation between data of different companies is discussed.

###### We'll cover the following

* + [Correlation](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation)
    - [Correlation score](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation-score)
    - [Correlation with the closing price](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation-with-the-closing-price)
    - [Correlation with daily returns](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation-with-daily-returns)

The stock behavior of companies dealing in similar services is usually related, and this relation can be measured using correlation.

## Correlation [#](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation)

Correlation is a statistical technique that determines how strongly two variables are related to each other and how a change in one would affect the other. It can also be defined as a measure of dependence between two or more quantities.

The two types of correlation, in terms of stock behavior, can be described as follows:

* **Positive correlation**: The stock value of one company goes up, and in correlation with it, the stock values of other companies also go up.
* **Negative correlation**: The stock value of one company goes up, and in correlation with it, the stock values of other companies go down.

### Correlation score [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation-score)

For positive correlation, this score is between **0** and **1**, and for negative correlation, this score is between **-1** and **0** (inclusive). A strong positive correlation has a score above **0.4**. It is the same for the negative correlation; it’s strong below a **-0.4** value. A score of **1** represents a perfect positive relationship and usually occurs when the correlation is taken with itself.

Correlation for all companies will be calculated with each other to observe what kind of relationship exists between the data of different companies and to see if there is a strong or weak correlation. The **closing price** and **daily return** variables will be used as parameters to get the correlation score for every company.

Before starting, all the company files need to be read in as variables, and their Time column needs to be set as the index. The following piece of code does the preprocessing before we actually observe the correlation.

import numpy as np

from scipy import stats

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

sys = pd.read\_csv('Year\_2018/SYS.csv')

ns = pd.read\_csv('Year\_2018/NETSOL.csv')

ptc = pd.read\_csv('Year\_2018/PTC.csv')

avn = pd.read\_csv('Year\_2018/AVN.csv')

sys['Time'] = pd.to\_datetime(sys.Time)

ns['Time'] = pd.to\_datetime(ns.Time)

ptc['Time'] = pd.to\_datetime(ptc.Time)

avn['Time'] = pd.to\_datetime(avn.Time)

sys = sys.set\_index('Time')

ns = ns.set\_index('Time')

ptc = ptc.set\_index('Time')

avn = avn.set\_index('Time')

## Try printing the data of any company

Now that the preprocessing is done and all the data is ready, the correlation can be found between these companies stock behavior. As mentioned above, the **closing price** and **daily return** values will be used to find this correlation.

### Correlation with the closing price [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation-with-the-closing-price)

The corr() function of a DataFrame can easily find correlations between its columns. So first, the closing prices of all the companies need to be in one DataFrame, on which the corr() function will be applied.

df = pd.DataFrame({'SYS': sys['Close'],

                   'NETSOL': ns['Close'],

                   'PTC': ptc['Close'],

                   'AVN': avn['Close']})

print("The New DataFrame\n", df.head())

corr = (df.dropna()).corr() # Calculating correlation after dropping null values

print("The correlations\n", corr)

On **line 1-4**, a new DataFrame is created using the dictionary method, which takes the Close column of all four companies.

On **line 8**, the corr() function is used to find a correlation between the columns.

The output shows that the correlation values of all the companies, with respect to the closing price, have been calculated against each other. Also, we can observe that the correlation of the data of a company with itself is **1**. The values below zero represent negative correlation, and the values above zero represent positive correlation.

Let’s visualize these values with a heatmap.

df = pd.DataFrame({'SYS': sys['Close'],

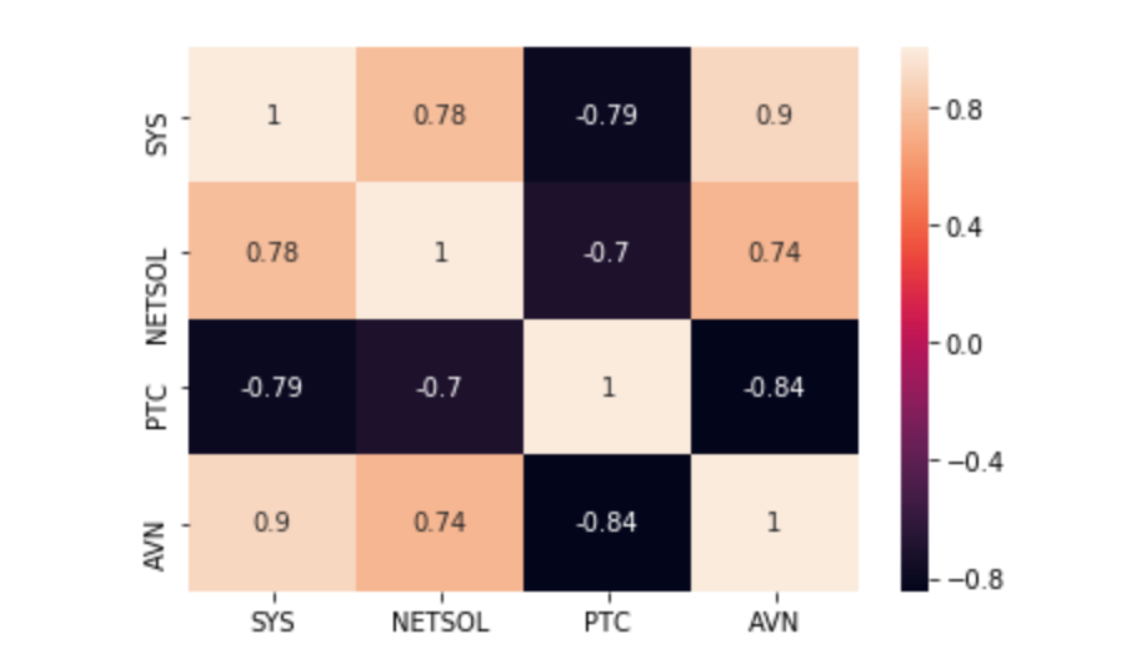
                   'NETSOL': ns['Close'],

                   'PTC': ptc['Close'],

                   'AVN': avn['Close']})

corr = (df.dropna()).corr()

sns.heatmap(corr, annot = True) # Plotting heatmap

On **line 8**, the heatmap function of the seaborn package is used to plot correlation values on a heatmap with the annot=True parameter as discussed in this [lesson](https://www.educative.io/collection/page/10370001/5982561936343040/6239215827288064).

The correlation values are properly separated and easy to comprehend on a heatmap. The highest correlation value is of **Systems Ltd** and **Avanceon**, which is **0.9**. This means that these two companies are highly correlated in a positive way. So, if the stock value of one company goes up, the value of the other company also goes up and vice versa.

**Try to infer more important information about other companies as well!**

### Correlation with daily returns [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/xoqjBNvNQnq#correlation-with-daily-returns)

The same corr() function and technique are used here, but instead of the closing price, the correlation is calculated on daily returns.

df = pd.DataFrame({'SYS': sys['Close'],

                   'NETSOL': ns['Close'],

                   'PTC': ptc['Close'],

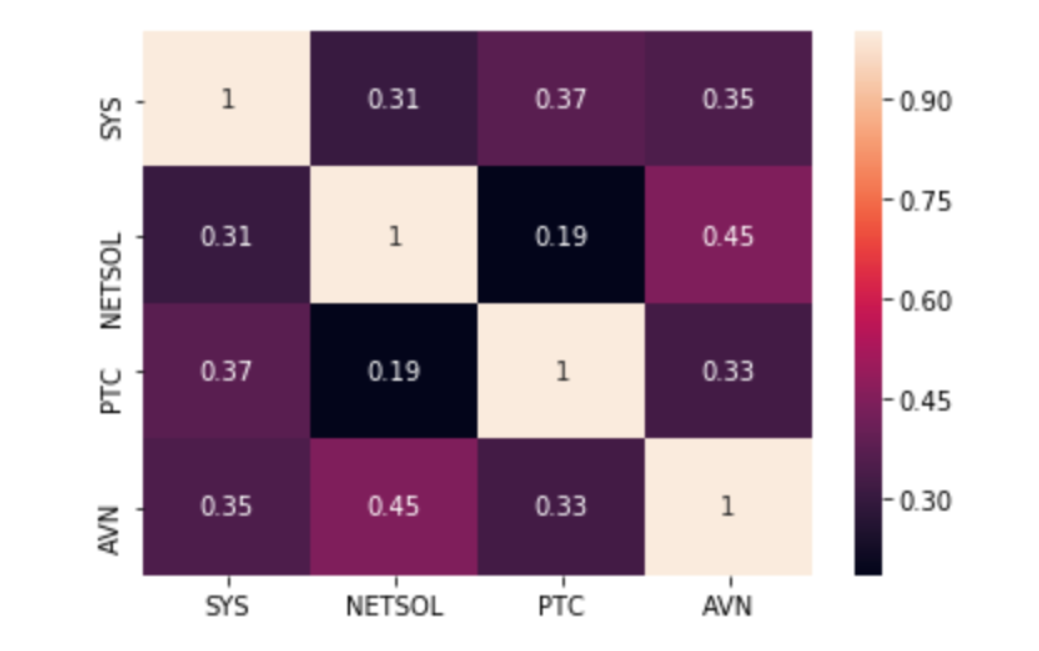
                   'AVN': avn['Close']})

all\_returns = df.pct\_change()

print(all\_returns)

corr = (all\_returns.dropna()).corr()

sns.heatmap(corr, annot = True)

On **line 6**, the pct\_change function is used to calculate daily returns just like before. Each column is the closing price of the companies, so, it calculates the daily returns for all companies in one go.

The daily return values can be viewed by clicking the right button on the output panel.

On **line 10**, the same heatmap function of seaborn is used to plot correlation values on a heatmap.

From this correlation, we can infer that **NETSOL** and **Avanceon** are correlated in a positive way. Because the score is above **0.4**, it is a strong correlation.

# Section 4: Risk Estimation

In this lesson, risk estimation for the stock of different companies is discussed.

###### We'll cover the following

* + [Risk](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#risk)
    - [Risk/Return plot](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#riskreturn-plot)
      * [Results](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#results)
    - [Quantiles](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#quantiles)

## Risk [#](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#risk)

For the scope of our project, the risk relates to the amount of capital we could lose on our investment on a daily basis.

The daily risk for the companies stock prices can be calculated by taking the standard deviation of the daily returns. The daily returns and risks can be visualized using a scatter plot, which can give us a better understanding of the return vs. risk ratio of each company’s stock.

Let’s first calculate the average daily return and the risk. Here the risk is the standard deviation of all the daily return values. As mentioned in the statistical features [lesson](https://www.educative.io/collection/page/10370001/5982561936343040/5493110550298624#standard-deviation-std), the standard deviation measures how far the values are from the mean or average value. In our case, this distance from the mean value is the value or amount that we are putting at risk by investing in a certain stock.

To perform the following steps, all the preprocessing of reading the companies as variables and setting Time as the index needs to be done again. So, for simplicity, these steps have been hidden in the following example.

ret = all\_returns.dropna() # drop the null values

avg\_daily\_return = ret.mean() # Take mean of the daily return of all companies

print("Averge daily return of companies\n", avg\_daily\_return)

daily\_risk = ret.std() # Take standard deviation of the daily return of all companies

print("\nDaily Risk or standard deviation of companies\n", daily\_risk)

On **line 1**, the null values are dropped using the dropna() function.

On **line 3**, the built-in mean() function is used to calculate the average of each column.

On **line 6**, the built-in std() function is used to calculate the standard deviation of each column.

### Risk/Return plot [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#riskreturn-plot)

Now that std and mean are calculated, the scatter plot representing the risk and return can be plotted.

ret = all\_returns.dropna() # drop the null values

avg\_daily\_return = ret.mean() # Take mean of the daily return of all companies

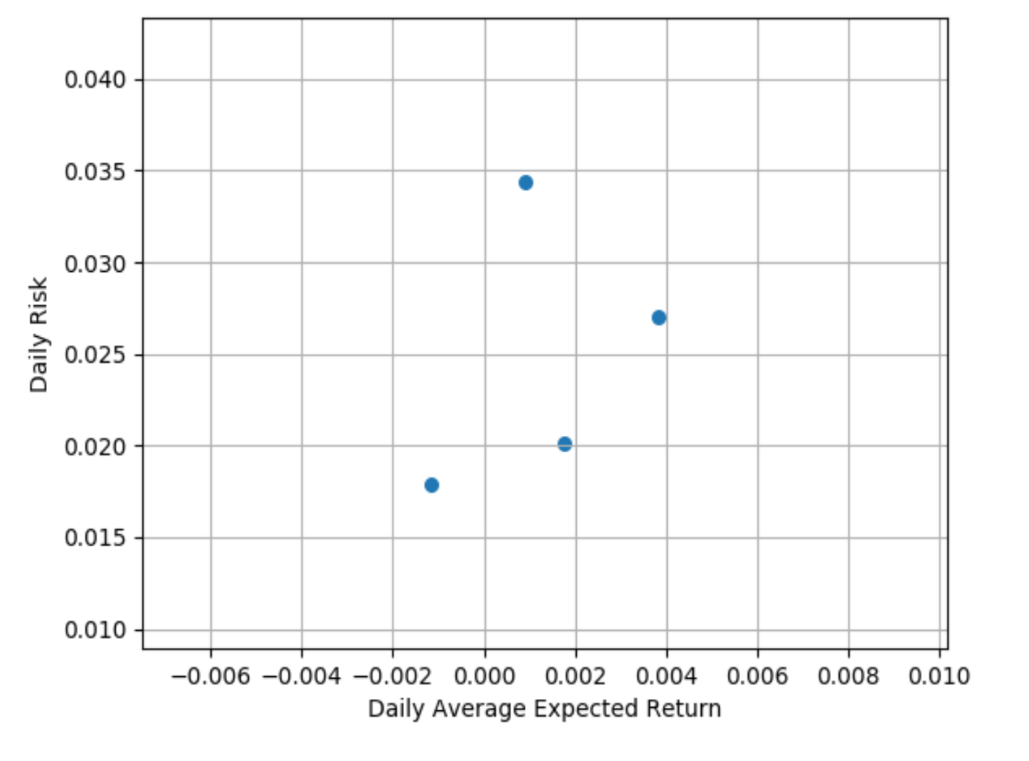
daily\_risk = ret.std() # Take standard deviation of the daily return of all companies

plt.xlabel("Daily Average Expected Return") # Name the x-axis

plt.ylabel("Daily Risk") # Name the y-axis

plt.grid() # Add grid lines on the plot

plt.scatter(avg\_daily\_return, daily\_risk, s = 30) # Plot the scatter plot for risk & return

The above plot is obtained after running the program.

On **lines 6 & 7**, the names of x-axis and y-axis are defined using the xlabel() and ylabel() functions, respectively.

On **line 9**, grid lines are drawn on the graph to better assess the position of points. The grid() function is used for it.

On **line 11**, the scatter() function of matplotlib is used to draw a scatter plot with the average daily returns on the x-axis and standard deviation on the y-axis as the daily risk value. The s parameter defines the size of the dots in the plot.

It looks like the plot is missing something. Names of companies are not associated with the dots on the plot. Let’s add some annotations to make it visually understandable.

ret = all\_returns.dropna()

avg\_daily\_return = ret.mean()

daily\_risk = ret.std()

plt.xlabel("Daily Average Expected Return")

plt.ylabel("Daily Risk")

plt.xlim(ret.mean().min() + ret.mean().min()\*2, ret.mean().max() + ret.mean().max()\*2)

for label, x, y in zip(ret.columns, ret.mean(), ret.std()):

    plt.annotate(

        label,

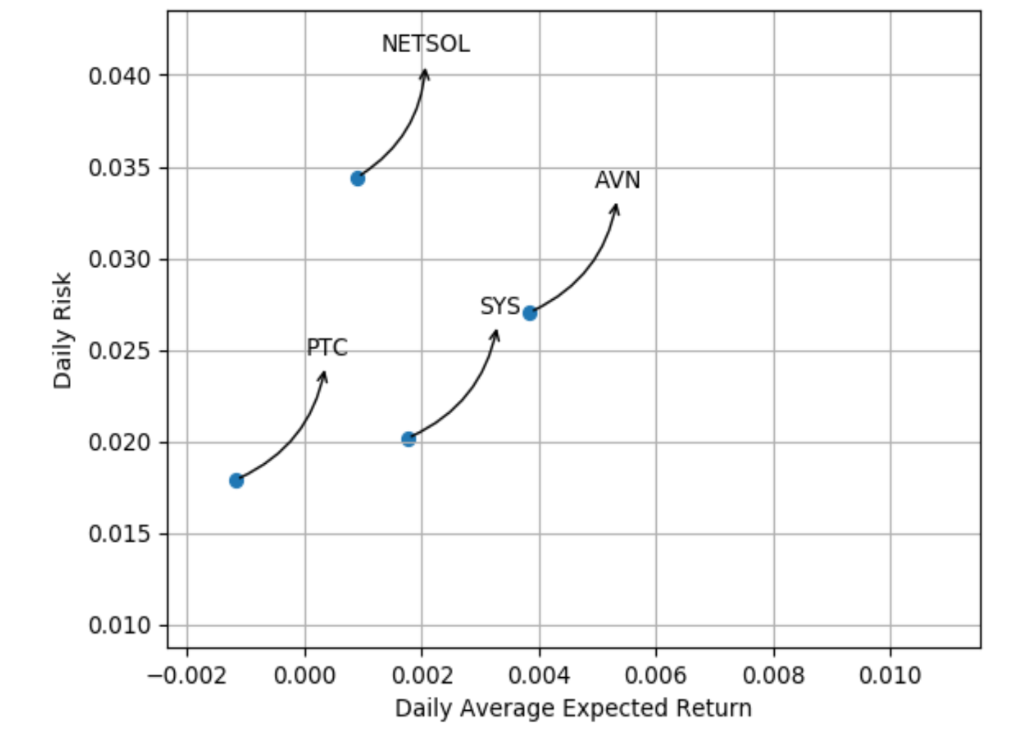
        xy = (x,y), xytext = (50, 50),

        textcoords = 'offset points', ha = 'right', va = 'bottom',

        arrowprops = dict(arrowstyle = '<-', connectionstyle = 'arc3,rad=-0.4'))

plt.grid()

plt.scatter(avg\_daily\_return, daily\_risk, s = 30)

Now, the above plot can provide some relevant information about how much risk a particular stock holds for how much of an average return.

The code on **line 9** only extends the value limits of the x-axis.

The code on **lines 11-16** adds the relevant names and lines while structuring them. This part needs to be custom made for every instance of data so that if this code runs on more data or for different companies, this structure is not maintained. For information on how to make your own custom design to represent data, refer [here](https://matplotlib.org/tutorials/text/annotations.html).

#### Results [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#results)

So, from the above graph, the following results can be inferred:

1. **NETSOL** stock has a little positive daily return and a very high risk, which means the loss is also high.
2. **Avanceon** has a high positive daily return, and the risk value is also less than **NETSOL**.
3. **Systems Ltd** has a positive daily return, but it is less than **Avanceon**. The risk value is less than **NETSOL** and **Avanceon** so it can be considered a good stock.
4. The **PTC** stock has the lowest risk value, but its average daily return value is negative. This indicates that even though our losses would be little, there would not be a positive return for our investment.

These results support the assumptions we made in the previous [lesson](https://www.educative.io/courses/predictive-data-analysis-with-python/educative.io/collection/page/10370001/5982561936343040/5253052144549888#comparison) that **Systems Ltd** and **Avanceon** stocks are better for investments than others.

### Quantiles [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/7A373JMQm2A#quantiles)

This method can also be used to assess the risk associated with a stock. This method can determine the maximum loss over an investment with high certainty. The quantile method is explained [here](https://www.educative.io/collection/page/10370001/5982561936343040/5493110550298624#quantiles).

Fortunately, pandas DataFrame provides the quantile(n) function to calculate the value at a given quartile n. Let’s discuss this using an example.

df = pd.DataFrame({'SYS': sys['Close'],

                   'NETSOL': ns['Close'],

                   'PTC': ptc['Close'],

                   'AVN': avn['Close']})

all\_returns = df.pct\_change()

investment = 100000

loss = (abs(all\_returns.quantile(0.1))) \* investment

print(loss)

The new DataFrame is created with only the Close prices, and their daily returns are calculated using the pct\_change() method.

Now, let’s suppose we want to invest **100,000** in some stocks and want to know what our maximum loss for a day is. On **line 10**, the quantile() function is used with a parameter value of 0.1. The absolute values returned from the quantile() function are then multiplied with our initial investment. The 0.1 value gives us the loss values with **90%** accuracy. More on quantiles are explained [here](https://en.wikipedia.org/wiki/Quantile).

In the output, four values are obtained for each company. Each number signifies that the total loss for a single day will not exceed this value. Considering our initial investment, this loss is not very much.

It can also be observed that if sorted, the number of values at risk is in the same order as detected by the scatter plot. **NETSOL** has the highest loss and **PTC** has the lowest loss value with other companies between them.

# Section 5: Predicting Future Stock Behavior

In this lesson, you will try to predict future stock behavior.

###### We'll cover the following

* + [Random walk theory](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#random-walk-theory)
  + [MonteCarlo simulations](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#montecarlo-simulations)
    - [Implementation](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#implementation)
      * [Calculating drift](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#calculating-drift)
      * [Calculating Rv](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#calculating-rv)
      * [Summing it up](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#summing-it-up)

There are two main techniques used to analyze stock behavior.

1. **Fundamental analysis**: This mostly deals with the intrinsic value of companies based on the various changes in their financials on a regular basis.
2. **Technical analysis**: This provides results based on the historical data of a company’s stock.

Fundamental analysis is beyond the scope of this course, and the information required for it is also not easily accessible.

Until now, our focus has been on technical analysis, as we calculated various results from the historical data of the companies. However, these techniques won’t help us predict the random and irregular behavior of stocks.

## Random walk theory [#](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#random-walk-theory)

Many analysts believe that the stock market prices follow the **random walk theory**. This theory states the following:

***The stock market may take random, irregular, and unpredictable paths in determining the prices of stocks. It also assumes that past trends are useless in predicting future prices and that the future price only depends on the current price of the stock.***

Detailed information about this theory can be found [here](https://www.investopedia.com/terms/r/randomwalktheory.asp). This theory rejects both fundamental and technical analysis techniques with rational arguments.

We will also use the random walk theory to determine the future behavior of stocks. The Monte Carlo simulations will be deployed to assess the results using this theory.

## MonteCarlo simulations [#](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#montecarlo-simulations)

This method uses randomness to solve problems. It converts the randomness in the variables into probability distributions. It then generates a range of future price values in a normal distribution instead of just one value. More detailed information about this can be found [here](https://www.investopedia.com/terms/m/montecarlosimulation.asp).

The following is the formula for MonteCarlo simulations:

S\_{t+1}=S\_{t}\*e^{Drift+Rv}*S*​*t*+1​​=*S*​*t*​​∗*e*​*Drift*+*Rv*​​

Drift=AveragedDailyReturn-DailyReturnVariance/2*Drift*=*AveragedDailyReturn*−*DailyReturnVariance*/2

Rv=STD\*NORMSINV(RAND())*Rv*=*STD*∗*NORMSINV*(*RAND*())

Here, S\_{t+1}*S*​*t*+1​​ is the future price of a stock. S\_{t}*S*​*t*​​ is the current price of a stock. **e** is the universal constant. Drift and Rv are represented by their formulas. The Drift component represents the direction of stock, whether it’ll go up or down. The Rv is our random variable; which either pushes the stock price up or down. More information on the NORMSINV() function can be found [here](https://www.isixsigma.com/dictionary/normsinv/).

### Implementation [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#implementation)

Now, let’s see monte-carlo simulations in action by predicting future stock price values of the **Systems Ltd** company. Our final range of predictions will be in a multidimensional NumPy array that will be plotted to get the predicted range of future prices.

Before computing the results from the formula, some extra variables need to be calculated. We learned how to calculate daily returns for our stocks in the previous [lesson](https://www.educative.io/collection/page/10370001/5982561936343040/6629670062653440). For this exercise, logarithmic returns will be calculated instead of daily returns, as they provide more concise information.

#### Calculating drift [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#calculating-drift)

Here we show how drift is calculated using logarithmic daily returns.

# Importing the packages

import numpy as np

import pandas as pd

# Preprocessing steps as before

sys = pd.read\_csv('Year\_2018/SYS.csv')

sys['Time'] = pd.to\_datetime(sys.Time)

sys = sys.set\_index('Time')

daily\_returns = sys['Close'].pct\_change() # Calculating daily returns

# Calculating log returns from daily returns

log\_returns = np.log(1 + daily\_returns)

avg = log\_returns.mean() # Calculating average of log returns

var = log\_returns.var() # Calculating variance

drift = avg - (var / 2.0) # Calculating drift

drift = np.array(drift) # Convert to array

print("The calculated Drift is:", drift)

After the preprocessing, the daily returns are calculated using the pct\_change() method as discussed before.

On **line 15**, the log returns are calculated using the np.log() function of the NumPy package. The log of every value in the daily return Series is calculated after adding **1** to it and another Series is returned.

On **lines 17 & 18**, the respective average and variance of the log-returns are calculated.

On **line 20**, the drift component is calculated according to the above-described formula.

On **line 22**, the drift is converted to a NumPy array to simplify further calculations with arrays.

#### Calculating Rv [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#calculating-rv)

In this step, the random values are defined in accordance with the random walk theory.

# Importing the packages

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.stats import norm

# Preprocessing steps as before

sys = pd.read\_csv('Year\_2018/SYS.csv')

sys['Time'] = pd.to\_datetime(sys.Time)

sys = sys.set\_index('Time')

daily\_returns = sys['Close'].pct\_change() # Calculating daily returns

log\_returns = np.log(1 + daily\_returns) # Calculating log returns from daily returns

avg = log\_returns.mean() # Calculating average of log returns

var = log\_returns.var() # Calculating variance

drift = avg - (var / 2.0) # Calculating drift

drift = np.array(drift) # Convert to array

pred\_price\_overDays = 60 # Number of days

pred\_count = 10 # Range of prediction

std = log\_returns.std() # Calculating STD

std = np.array(std) # Convert to array

x = np.random.rand(pred\_price\_overDays, pred\_count) # get random multidimensional array

Rv = std \* norm.ppf(x) # Calculating Rv

print("The required Rv array is:\n", Rv)

On **line 21**, a variable pred\_price\_overDays is defined; it determines the number of days for which we need to forecast the price.

On **line 22**, a variable pred\_count is defined; it determines how many predictions we need.

On **lines 24 & 25**, the standard deviation is calculated and converted to a NumPy array object to simplify further calculations with arrays.

On **line 27**, the random values are generated using the rand function of NumPy. As two parameters are given to the function, a multidimensional array is generated. An array with **sixty** rows and **ten** columns is produced as described by the two variables. This array acts as a random input, as mentioned above.

On **line 29**, the function norm.ppf() of the scipy python package takes the inverse of the normal distribution. As mentioned in the above formula **NORMSINV(RAND())**, the random numbers are already calculated, and the ppf() function performs the inverse normal distribution function on those random values. In simple terms, this function takes the values generated by rand() and converts them to distances measured from their mean. The documentation is available [here](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html). After its calculation, the array is multiplied by the standard deviation to get the Rv value.

As all the components of the original formula, i.e., S\_{t+1}=S\_{t}\*e^{Drift+Rv}*S*​*t*+1​​=*S*​*t*​​∗*e*​*Drift*+*Rv*​​ are calculated, the final future prices can finally be generated.

#### Summing it up [**#**](https://www.educative.io/courses/predictive-data-analysis-with-python/B8nDPk5o0y2#summing-it-up)

Now, only the current stock price and the **e** value needs to be fetched and calculated, and the corresponding future prices get generated.

e\_value = np.exp(drift + Rv) # Calculating the E value

current\_price = sys['Close'].iloc[-1] # Selecting last price of the year

new\_prices = np.zeros\_like(e\_value) # create array to store the results

new\_prices[0] = current\_price

print(new\_prices)

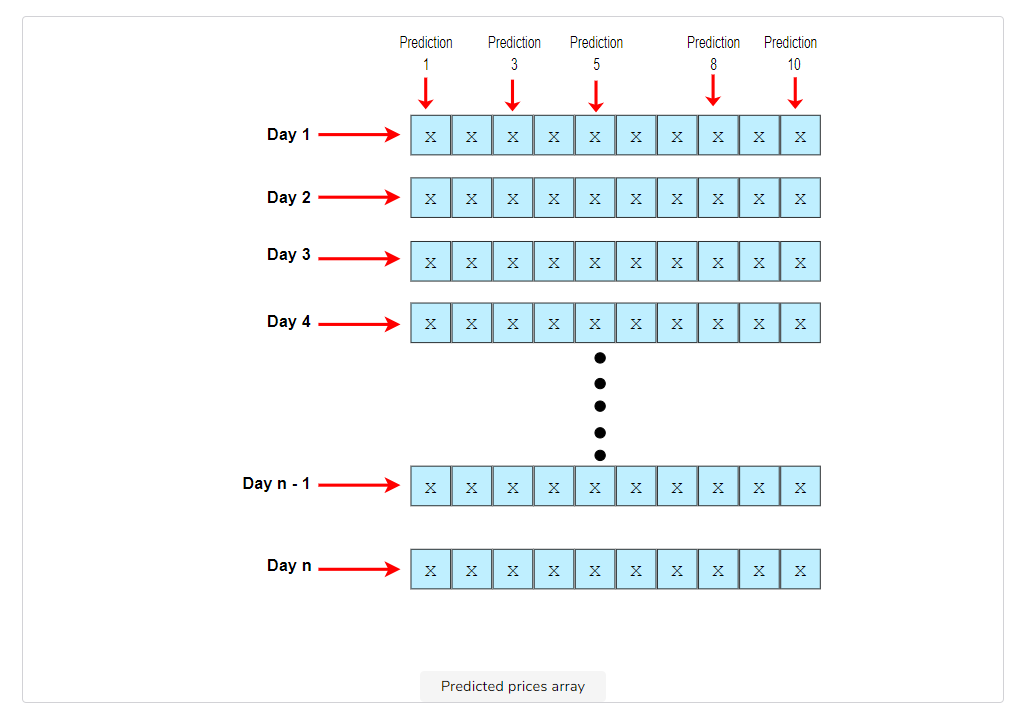
On **line 1**, the exp() function of NumPy is used. It takes e to the power of any number assigned in the parameters. The drift and Rv values are added, and then the exp() function is evaluated. This returns a 60 x 10 array with **60** as the number of days for which the price is forecasted and **10** as the number of predictions.

On **line 3**, the current price is fetched from the DataFrame. As the DataFrame is indexed on Time, it is sorted, and the last closing price is the most recent and current price of the stock.

On **line 5**, a new array new\_prices with the exact shape as the e\_value array is declared to store the new predicted prices. The zeros\_like function declares this array and instantiates it with zeroes.

Now, each row value of the new\_prices array is a different day containing the new predicted price, and each column of the new\_prices array is a new prediction for all the days. This concept is shown in the illustration below.

On **line 7**, the first row of the new\_prices array is assigned the current stock value. Now, the new prices can generate in **ten** paths according to the pred\_count variable with the same current starting price.

The above figure displays how the predicted prices are stored after calculation. Now, let’s loop over the number of days and predict the price for each one.

for i in range(1, pred\_price\_overDays): # Loop over all the days to find their prices

    new\_prices[i] = new\_prices[i - 1] \* e\_value[i] # Calculating the future price with formula

print("The Minimum Predicted Price:", new\_prices[pred\_price\_overDays-1].min()) # Get minimum price

print("The Maximum Predicted Price:", new\_prices[pred\_price\_overDays-1].max()) # Get maximum price

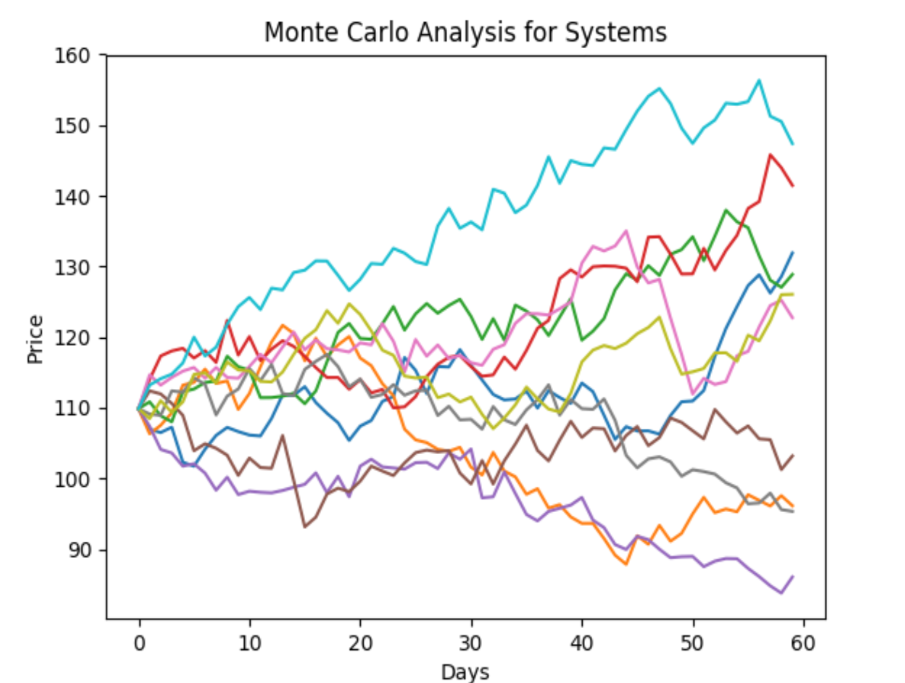
plt.xlabel('Days') # Assign name to x-axis

plt.ylabel('Price') # Assign name to y-axis

plt.title('Monte Carlo Analysis for Systems') # Assign name to the plot

plt.plot(new\_prices)# plot the figure

print("\nThe price array:\n", new\_prices)

A plot like above, is generated from the output. It shows that from the current price, the stock can take *ten* different routes to predict the prices.

If you click the right arrow in the output, the minimum and maximum prices of the last day from all routes are shown with the whole price array with the predicted prices. According to the *monte-carlo method*, the original price should be close to the mean of each path that the stock takes.

On **lines 1 & 2**, the program loops over the total number of days defined and applies the initial formula. The new price is equal to the product of the price of the previous day and the **e** value calculated for that day. The previous day’s price is set in the above code snippet.

On **lines 4 & 5**, the maximum and minimum prices for the last day are shown.

The plot will be different every time the program is run. So you might get a different plot at different prices. The program can also be executed multiple times to compare the *mean* obtained in each run to figure out a more concise future price.

This marks the end of the stock market analysis project.