# Artificial Intelligence

## Project:stock market analysis and prediction

Name:B.vinaykumar

Reg no:11804196

Sec:K18HKB

Roll no:63

**Introduction:** Stock market plays a major role in the world for their country Economy.In india stock market runs under the Securities and exchange board of india [SEBI] Stock Market Analysis and Prediction is the project on technical analysis, visualization, and prediction using data provided by Google Finance. By looking at data from the stock market, particularly some giant technology stocks and others. Used pandas to get stock information, visualized different aspects of it, and finally looked at a few ways of analyzing the risk of a stock, based on its previous performance history. Predicted future stock prices.Analysing technically we can predict the stock price and finally we get analysis on a stock to buy or sell in our world stock market plays a major role.

**Purpose:** The purpose of this project is to comparatively analyze the effectiveness of prediction algorithms on stock market data and get general insight on this data through visualization to predict future stock behavior and value at risk for each stock. The project encompasses the concept of Data Mining and Statistics. This project makes heavy use of NumPy,

Pandas, and Data Visualization Libraries.by using previous historical data of a particularstock we can analysis the data.

# Code:

```
#Stock market analysis and prediction
# For Data Processing
# For division
from __future__ import division
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
get_ipython().magic('matplotlib inline')
# For reading stock data from yahoo
from pandas_datareader import DataReader
# For time stamps
from datetime import datetime
# List of Tech_stocks for analytics
tech_list = ['AAPL','GOOGL','MSFT','AMZN']
# set up Start and End time for data grab
end = datetime.now()
start = datetime(end.year-1,end.month,end.day)
#For-loop for grabing google finance data and setting as a dataframe
# Set DataFrame as the Stock Ticker
for stock in tech_list:
    globals()[stock] = DataReader(stock,'yahoo',start,end)
AAPL.head()
# Summery stats for Apple Stock
AAPL.describe()
# General Info
AAPL.info()
# Let's see a historical view of the closing price
AAPL['Close'].plot(legend=True, figsize=(10,4))
# Now let's plot the total volume of stock being traded each day over the past year
AAPL['Volume'].plot(legend=True, figsize=(10,4))
# We can see that on Feb'2017 was the higher for AAPL stock being traded.
# Now that we've seen the visualizations for the closing price and the volume traded each day for AAPL stock.
# Let's go ahead and caculate the moving average for the AAPL stock.
# Let's go ahead and plot out several moving averages
MA_day = [10,20,50,100]
for ma in MA_day:
    column_name = 'MA for %s days' %(str(ma))
    AAPL[column_name] = AAPL['Adj Close'].rolling(window=ma).mean()
AAPL[['Close','MA for 10 days','MA for 20 days','MA for 50 days','MA for 100 days']].plot(subplots=False,figsize=(10,4))
# We'll use pct_change to find the percent change for each day
AAPL['Daily Return'] = AAPL['Close'].pct_change()
# Lets plot the daily return percentage
AAPL['Daily Return'].plot(figsize=(12,4), legend=True, linestyle='--', marker='o')
# only with histogram
AAPL['Daily Return'].hist(bins=100)
# Note the use of dropna() here, otherwise the NaN values can't be read by seaborn
sns.distplot(AAPL['Daily Return'].dropna(), bins=100, color='magenta')
# Grab all the closing prices for the tech stock list into one DataFrame
closingprice_df = DataReader(tech_list, 'yahoo', start, end)['Close']
```

```
closingprice_df.head(10)
# make a new tech returns DataFrame
tech_returns = closingprice_df.pct_change()
tech_returns.head()
# Now we can compare the daily percentage return of two stocks to check how correlated.
#First let's see a stock compared to itself.
# Comparing Google to itself should show a perfectly linear relationship
sns.jointplot('GOOGL','GOOGL',tech_returns,kind='scatter',color='orange')
# 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.
# We'll use joinplot to compare the daily returns of Google and Amazon.
sns.jointplot('GOOGL','AMZN',tech_returns, kind='scatter',size=8, color='skyblue')
# with Hex plot
sns.jointplot('GOOGL','AMZN',tech_returns, kind='hex',size=8, color='skyblue')
# Lets check out for Apple and Microsoft with reg jointplot
sns.jointplot('AAPL','MSFT',tech_returns, kind='reg', size=8, color='skyblue')
# Intersting, the pearsonr value (officially known as the Pearson product-moment correlation coefficient) can give you a sense of how correlated the daily percentage returns are. You can find more information about it at this link:
from IPython.display import SVG
SVG(url='http://upload.wikimedia.org/wikipedia/commons/d/d4/Correlation_examples2.svg')
# 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
# We can simply call pairplot on our DataFrame for an automatic visual analysis of all the comparisons
sns.pairplot(tech_returns.dropna(),size=3)
# 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.
# Set up the figure by naming it returns_fig, call PairGrid on the DataFrame
returns_fig = sns.PairGrid(tech_returns.dropna())
# 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, including the plot type (kde) & 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)
# Set up the figure by naming it returns_fig, call PairGrid on the DataFrame
returns_fig = sns.PairGrid(closingprice_df.dropna())
# 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, including the plot type (kde) & 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)
# Let's go ahead and use seaborn for a quick heatmap to get correlation for the daily return of the stocks.
sns.heatmap(tech_returns.corr(),annot=True,fmt=".3g",cmap='YlGnBu')
# Lets check out the correlation between closing prices of stocks
sns.heatmap(closingprice_df.corr(),annot=True,fmt=".3g",cmap='YlGnBu')
# Let's start by defining a new DataFrame as a clenaed version of the oriignal tech_returns DataFrame
rets = tech_returns.dropna()
rets.head()
# Defining the area for the circles of scatter plot to avoid tiny little points
area = np.pi*20
plt.scatter(rets.mean(),rets.std(),s=area)
# Set the x and y limits of the plot (optional, remove this if you don't see anything in your plot)
plt.xlim([-0.0025,0.0025])
plt.ylim([0.001,0.025])
#Set the plot axis titles
plt.xlabel('Expected returns')
```

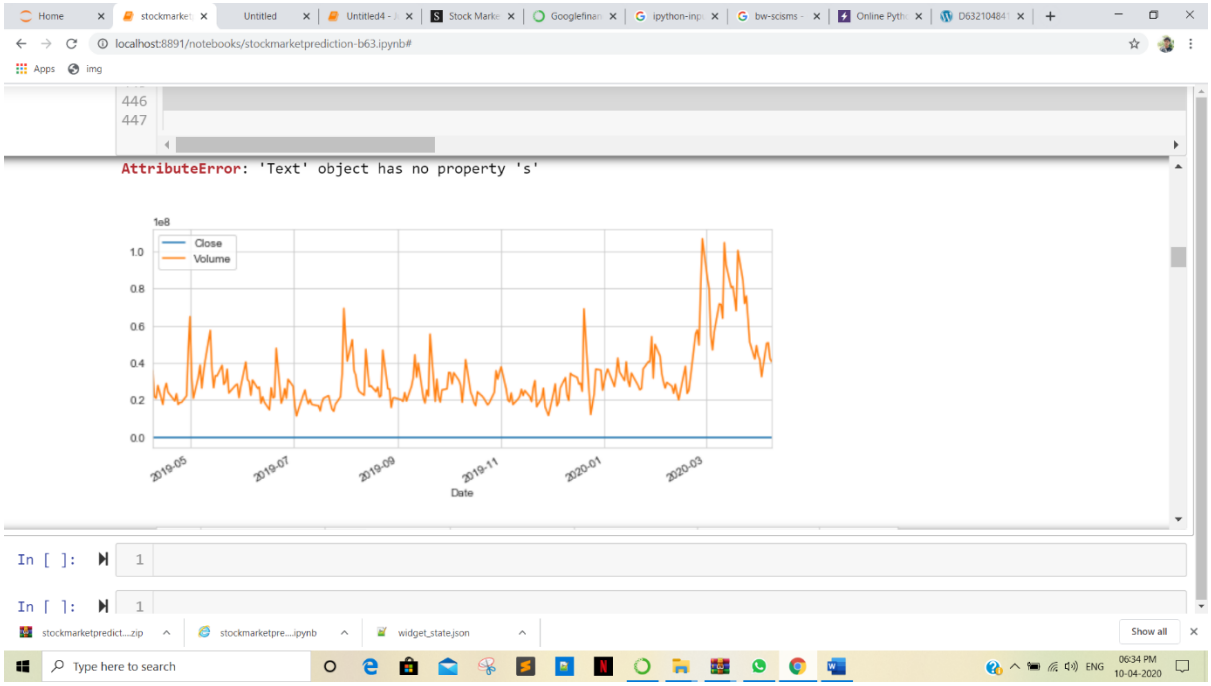
```
plt.ylabel('Risk')

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 = 'fancy', connectionstyle = 'arc3,rad=-0.3'))
# By looking at the scatter plot we can say these stocks have lower risk and positive expected returns.
# Let's go ahead and repeat the daily returns histogram for Apple stock.
# Note the use of dropna() here, otherwise the NaN values can't be read by seaborn
sns.distplot(AAPL['Daily Return'].dropna(),bins=100,color='purple')
# For APPL stocks
rets["AAPL"].quantile(0.05)
# For AMZN stocks
rets["AMZN"].quantile(0.05)
# For GOOGL stocks
rets["GOOGL"].quantile(0.05)
# For MSFT stocks
rets["MSFT"].quantile(0.05)
rets.head()
# Set up our time horizon
days = 365
# Now our delta
dt = 1/days
# Now let's grab our mu (drift) from the expected return data we got for GOOGL
mu = rets.mean()['GOOGL']
# Now let's grab the volatility of the stock from the std() of the average return for GOOGL
sigma = rets.std()['GOOGL']
# Next, we will create a function that takes in the starting price and number of days, and uses the sigma and mu we already
calculated form our daily returns.
def stock_monte_carlo(start_price,days,mu,sigma):
    ''' This function takes in starting stock price, days of simulation,mu,sigma, and returns simulated price array'''
    # Define a price array
    price = np.zeros(days)
    price[0] = start_price
     # Schok and Drift
    shock = np.zeros(days)
    drift = np.zeros(days)
    # Run price array for number of days
    for x in range(1,days):
         # Calculate Schock
        shock[x] = np.random.normal(loc=mu * dt, scale=sigma * np.sqrt(dt))
        # Calculate Drift
        drift[x] = mu * dt
        # Calculate Price
        price[x] = price[x-1] + (price[x-1] * (drift[x] + shock[x]))
       return price
# For Google Stock - GOOGL
GOOGL.head()
start_price = 830.09
for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))
    plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Google')
# For Amazon Stock - AMZN
AMZN.head()
start_price = 824.95
```

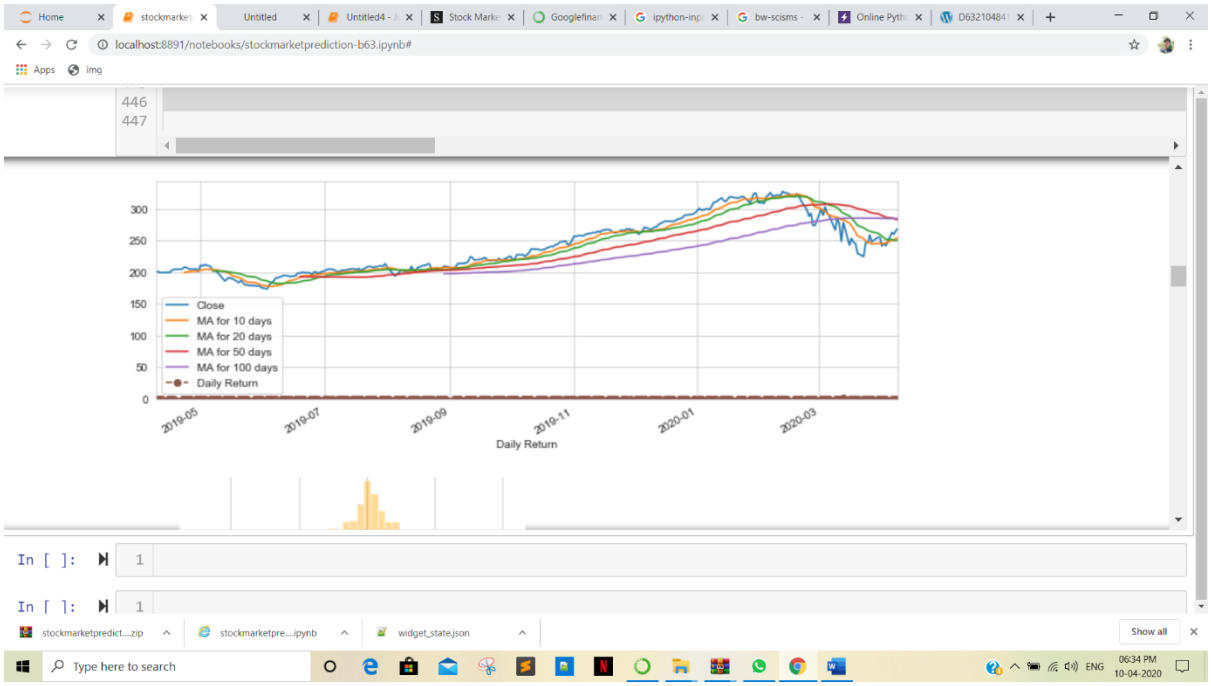
```
for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))
    plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Amazon')
# For Apple Stock - AAPL
AAPL.head()
start_price = 117.10
for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))
    plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Apple')
# For Microsoft Stock - MSFT
MSFT.head()
start_price = 59.94
for run in range(100):
    plt.plot(stock_monte_carlo(start_price, days, mu, sigma))
    plt.xlabel("Days")
plt.ylabel("Price")
plt.title('Monte Carlo Analysis for Microsoft')
# Lets start with Google stock price
start_price = 830.09
# Set a large numebr of runs
runs = 10000
# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)
for run in range(runs):
    # Set the simulation data point as the last stock price for that run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)[days-1]
# Now we'll define q as the 1% empirical quantile, this basically means that 99% of the values
#should fall between here
q = np.percentile(simulations,1)
# Now let's plot the distribution of the end prices
plt.hist(simulations, bins=200)
# Using plt.figtext to fill in some additional information onto the plot
# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)
# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())
# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))
# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)
# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')
# For plot title
plt.title(s="Final price distribution for Google Stock(GOOGL) after %s days" % days, weight='bold', color='Y', label=x)
# For Amazon Stock Price
start_price = 824.95
# Set a large numebr of runs
runs = 10000
# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)
for run in range(runs):
    # Set the simulation data point as the last stock price for that run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)[days-1]
# Now we'll define q as the 1% empirical quantile, this basically means that 99% of the values should fall between here
q = np.percentile(simulations,1)
# Now let's plot the distribution of the end prices
```

```
plt.hist(simulations, bins=200)
# Using plt.figtext to fill in some additional information onto the plot
# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)
# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())
# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))
# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)
# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')
# For plot title
plt.title(s="Final price distribution for Amazon Stock(AMZN) after %s days" % days, weight='bold', color='G')
# For Apple Stock Price
start_price = 117.10
# Set a large numebr of runs
runs = 10000
# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)
for run in range(runs):
    # Set the simulation data point as the last stock price for that run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)[days-1]
# Now we'll define q as the 1% empirical quantile, this basically means that 99% of the values should fall between here
q = np.percentile(simulations,1)
# Now let's plot the distribution of the end prices
plt.hist(simulations, bins=200)
# Using plt.figtext to fill in some additional information onto the plot
# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)
# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())
# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))
# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)
# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')
# For plot title
plt.title(s="Final price distribution for Apple Stock(AAPL) after %s days" % days, weight='bold', color='B')
# Great! This basically means for every initial AAPL stock you purchase you're putting about $2.48 at risk 99% of the time from
our Monte Carlo Simulation.
# For Microsoft Stock Price
start_price = 59.94
# Set a large numebr of runs
runs = 10000
# Create an empty matrix to hold the end price data
simulations = np.zeros(runs)
for run in range(runs):
    # Set the simulation data point as the last stock price for that run
    simulations[run] = stock_monte_carlo(start_price,days,mu,sigma)[days-1]
# Now we'll define q as the 1% empirical quantile, this basically means that 99% of the values should fall between here
q = np.percentile(simulations,1)
# Now let's plot the distribution of the end prices
plt.hist(simulations, bins=200)
# Using plt.figtext to fill in some additional information onto the plot
# starting price
plt.figtext(0.6,0.8, s='Start Price: $%.2f' % start_price)
# mean ending price
plt.figtext(0.6,0.7, s='Mean Final Price: $%.2f' % simulations.mean())
```

```
# Variance of the price (within 99% confidence interval)
plt.figtext(0.6,0.6, s='VaR(0.99): $%.2f' % (start_price - q))
# To display 1% quantile
plt.figtext(0.15, 0.6, s="q(0.99): $%.2f" % q)
# Plot a line at the 1% quantile result
plt.axvline(x=q, linewidth=4, color='r')
# For plot title
plt.title(s="Final price distribution for Microsoft Stock(MSFT) after %s days" % days, weight='bold', color='M')
# By using the above methods to get Value at Risk.
# List of NYSE_stocks for analytics
NYSE_list = ['JNJ','NKE','WMT']
# set up Start and End time for data grab
end = datetime.now()
start = datetime(end.year-1,end.month,end.day)
#For-loop for grabing google finance data and setting as a dataframe
# Set DataFrame as the Stock Ticker
for stock in NYSE_list:
    globals()[stock] = DataReader(stock,'google',start,end)
# Let's go ahead and play aorund with the JNJ(Johnson & Johnson) Stock DataFrame to get a feel for the data.
JNJ.head()
JNJ.describe()
JNJ.info()
# Now that we've seen the DataFrame, let's go ahead and plot out the closing prices of NYSE stocks.
# Let's see a historical view of the closing price for JNJ(Johnson & Johnson)
JNJ['Close'].plot(title='Closing Price - JNJ',legend=True, figsize=(10,4))
# Let's see a historical view of the closing price for NKE(Nike Inc.)
NKE['Close'].plot(title='Closing Price - NKE',legend=True, figsize=(10,4))
# Let's see a historical view of the closing price for WMT(Wal-Mart Stores Inc.)
WMT['Close'].plot(title='Closing Price - WMT',legend=True, figsize=(10,4))
# Value at risk using the "Bootstrap" method
# We'll use pct_change to find the percent change for each day
#For JNJ stocks
JNJ['Daily Return'] = JNJ['Close'].pct_change()
# Note the use of dropna() here, otherwise the NaN values can't be read by seaborn
sns.distplot(JNJ['Daily Return'].dropna(),bins=100,color='R')
(JNJ['Daily Return'].dropna()).quantile(0.05)
# For WMT stocks
WMT['Daily Return'] = WMT['Close'].pct_change()
sns.distplot(WMT['Daily Return'].dropna(),bins=100,color='G')
(WMT['Daily Return'].dropna()).quantile(0.05)
# For NKE stocks
NKE['Daily Return'] = NKE['Close'].pct_change()
sns.distplot(NKE['Daily Return'].dropna(),bins=100,color='B')
(NKE['Daily Return'].dropna()).quantile(0.05)
```

## Output:




**Conclusion:** Trading in stock markets is one of the most popular channels of financial investments. But it is always hard to decide the best time to buy or sell due to the highly fluctuating and dynamic behavior of stock market. The key to realize high profits in stock trading is to determine the suitable trading time when the risk of trading should be minimum. As stock trading is a very risky business, it is necessary to predict the stock indices, its movements and to evaluate the risks and benefits before entering into any trading. In this thesis three important aspects of financial time series analysis such as, stock price prediction, modeling and forecasting volatility and stock trading following stock price index movement classification have been demonstrated using machine intelligence and evolutionary computing techniques.