**STUDY ON**

**STOCK PRICE PREDICTION OF BANKS USING MACHINE LEARNING**

***Table of content***

1. **Introduction**

Executive Summary

Machine Learning Algorithm

Macro Economic Factors

1. **Analysis And Interpretation**

Multiple Linear Regression

Collection of Data

Cleaning of Data

Exploratory Data Analysis

Correlation

Splitting Data

Building the Model

Perform Model Diagnostics

Predicting

1. **Conclusion**

Executive Summary

The project titled “STOCK PRICE PREDICTION OF BANKS USING MACHINE LEARNING” the basic problem of investing on stock market is most of the investors are not clear about the fundamental and techniques of investments, they mainly concentrate on risk and return involved in investing and make decisions without the guidance of experts which are thus based on judgments made on their individual assessments.

Therefore, the objective here is to understand the basic techniques involved in investing in equity market. Learning the above said objective will give a clear idea of the concept involved in investment decisions.

Technical analysis is the study of financial market action. The technical analysts look at the price movement that occur on day to day or week to week or over other constant time period displayed in graphic form, called charts. Technical analysis is a process of analysis is a process of analyzing a security’s historical prices in an effort to determine probable future prices.

From the study it has been understand that the price movement of the shares can be determined by using technical analysis and accurate buy and sell decision can be made so that the investors can minimize their risk and maximize the profits also show that all technical analysis indicators can give accurate decisions. The Study show that the historical prices have an impact on future prices, however, even if you are unable to accurately forecast prices, technical analysis can be used to consistently reduce your risks and improve your profits.

Machine Learning Algorithm

Multiple Linear Regression

Multiple Linear Regression is a supervised learning algorithm for finding the existence of an association relationship between a dependent variable and several independent variables

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the [linear relationship](https://www.investopedia.com/terms/l/linearrelationship.asp) between the explanatory (independent) variables and response (dependent) variable.

In essence, multiple regression is the extension of ordinary least-squares (OLS) [regression](https://www.investopedia.com/terms/r/regression.asp) that involves more than one explanatory variable.

Explaining Multiple Linear Regression

A simple linear regression is a function that allows an analyst or statistician to make predictions about one variable based on the information that is known about another variable. Linear regression can only be used when one has two continuous variables an independent variable and a dependent variable. The independent variable is the parameter that is used to calculate the dependent variable or outcome. A multiple regression model extends to several explanatory variables.

The multiple regression model is based on the following assumptions:

* There is a [linear relationship](https://www.investopedia.com/terms/l/linearrelationship.asp) between the dependent variables and the independent variables.
* The independent variables are not too highly [correlated](https://www.investopedia.com/terms/c/correlation.asp) with each other.
* yi observations are selected independently and randomly from the population.
* Residuals should be [normally distributed](https://www.investopedia.com/terms/n/normaldistribution.asp) with a mean of 0 and [variance](https://www.investopedia.com/terms/v/variance.asp) *σ.*

The [coefficient of determination](https://www.investopedia.com/terms/c/coefficient-of-determination.asp) (R-squared) is a statistical metric that is used to measure how much of the variation in outcome can be explained by the variation in the independent variables. R2 always increases as more predictors are added to the MLR model even though the predictors may not be related to the outcome variable.

R2 by itself can't thus be used to identify which predictors should be included in a model and which should be excluded. R2 can only be between 0 and 1, where 0 indicates that the outcome cannot be predicted by any of the independent variables and 1 indicates that the outcome can be predicted without error from the independent variables.

Macro Economic Factors

GDP

Unemployment Rate

Inflation Rate

Consumer Confidence

Consumer Price Index

Gold price

Yield Value Of 10 Years bond

**COLLECTING DATA:**

We used eikon and quandl Sources for collecting the macro details and four banks stock prices of 20 years

We used API key and RIC’s to gets the data from eikon by the below codes and saved the data in the csv files.

import eikon as ek

import pandas as pd

ek.set\_app\_key('53f3b6e6a2af4ee987df42330c0fcbf40f75a204')

As we are unable to get the 20 years of data we merged the data by calling it in separate parts

bank = ek.get\_timeseries(["JPM"], start\_date = "1998-12-01", end\_date ="2007-12-21", interval="daily",corax="adjusted") # JP Morgan

bank\_1 = ek.get\_timeseries(["JPM"], start\_date = "2007-12-22", end\_date ="2019-11-12", interval="daily",corax="adjusted")

bank\_jpm\_1=pd.merge(bank,bank\_1,on=['Date','HIGH', 'CLOSE', 'LOW', 'OPEN', 'COUNT', 'VOLUME'],how="outer")

bank\_jpm\_1.to\_csv("JPM.csv")

Unemployment Rate :

US\_UNEMPLOY = ek.get\_timeseries(["USUNR=ECI"], start\_date = "1998-12-01", end\_date = "2019-11-12",interval="monthly")

US\_UNEMPLOY.to\_csv("unemployment.csv")

Inflation Rate:

US\_INFLATION = ek.get\_timeseries(["aUSWOCPIPR"], start\_date = "1998-12-01", end\_date = "2019-11-12",interval="yearly")

US\_INFLATION.to\_csv("inflation.csv")

Consumer Confidence Rate:

US\_CC = ek.get\_timeseries(["USCONC=ECI"], start\_date = "1998-12-01", end\_date = "2019-11-12",interval="monthly")

US\_CC.to\_csv("consumer\_confidence.csv")

USA GDP:

import quandl

USA\_GDP = quandl.get("FRED/GDP")

USA\_GDP.to\_csv("gdp.csv")

CPI:

US\_CPI = ek.get\_timeseries(["USCPSA=ECI"], start\_date = "1998-12-01", end\_date = "2019-11-12",interval="monthly")

US\_CPI.to\_csv("cpi.csv")

10Y Bond yield%:

Yield = ek.get\_timeseries(["aUSEBM10Y"], start\_date = "1998-12-01", end\_date = "2019-11-12",interval="monthly")

Yield.to\_csv("yield.csv")

Gold price world wise:

GOLD1 = ek.get\_timeseries(["XAU="], start\_date = "1998-12-01", end\_date = "2007-12-21",interval="daily")

GOLD2 = ek.get\_timeseries(["XAU="], start\_date = "2007-12-22", end\_date = "2019-11-12",interval="daily")

GOLD=pd.merge(GOLD1,GOLD2,on=['Date','HIGH', 'CLOSE', 'LOW', 'OPEN'],how="outer")

GOLD.to\_csv("gold.csv")

**CLEANING DATA:**

Our data has different granularity so we while merging the data we found lot of NaN values by using interpolate we are able to remove the NaN values and arranged them in equal granularity.

import pandas

def bank(filename):

bank=pandas.read\_csv(filename,parse\_dates=["Date"],index\_col="Date")

bank.drop(columns=['HIGH','LOW', 'OPEN', 'COUNT','VOLUME'],inplace=True)

dt=pandas.date\_range(start="1998-12-01",end="2019-11-11",freq="D")

idx=pandas.DatetimeIndex(dt)

bank=bank.reindex(idx)

bank.index.name='DATE'

return(bank)

BOA\_BANK=bank("BOA1.csv")

BOA\_BANK.rename(columns={"CLOSE":"BOA\_CLOSE"},inplace=True)

**USA GDP:**

usa\_gdp=pandas.read\_csv("gdp12.csv",parse\_dates=["Date"],index\_col="Date")

usa\_gdp.rename(columns={"Value":"GDP\_VALUE"},inplace=True)

dt=pandas.date\_range(start="1998-12-01",end="2019-11-11",freq="D")

idx=pandas.DatetimeIndex(dt)

usa\_gdp=usa\_gdp.reindex(idx)

usa\_gdp.interpolate(method="time",inplace=True)

**Usa Unemployement:**

usa\_unemp=pandas.read\_csv("unemployment.csv",parse\_dates=["Date"],index\_col="Date")

usa\_unemp.rename(columns={"VALUE":"UNEMPLO\_RATE"},inplace=True)

**Usa Inflation:**

usa\_inf=pandas.read\_csv("inflation.csv",parse\_dates=["Date"],index\_col="Date")

usa\_inf.rename(columns={"VALUE":"INFLA\_RATE"},inplace=True)

**Usa Consumer Confidence:**

usa\_CC=pandas.read\_csv("consumer\_confidence.csv",parse\_dates=["Date"],index\_col="Date")

usa\_CC.rename(columns={"VALUE":"CC\_VALUE"},inplace=True)

**USA CPI:**

usa\_CPI=pandas.read\_csv("CPI.csv",parse\_dates=["Date"],index\_col="Date")

usa\_CPI.rename(columns={"VALUE":"CPI\_VALUE"},inplace=True)

**GOLD:**

GOLD=pandas.read\_csv("gold.csv",parse\_dates=["Date"],index\_col="Date")

GOLD.drop(columns=['HIGH','LOW', 'OPEN'],inplace=True)

GOLD.rename(columns={"CLOSE":"GOLD\_CLOSE"},inplace=True)

**YIELD:**

YIELD=pandas.read\_csv("yield.csv",parse\_dates=["Date"],index\_col="Date")

YIELD.rename(columns={"VALUE":"YIELD\_VALUE"},inplace=True)

**Merging Data:**

data = BOA\_BANK.merge(CITY\_BANK, how='outer', left\_index=True, right\_index=True)

data=data.merge(WELLS\_BANK, how='outer', left\_index=True, right\_index=True)

data=data.merge(JPM\_BANK, how='outer', left\_index=True, right\_index=True)

data=data.merge(usa\_gdp, how='outer', left\_index=True, right\_index=True)

data=data.merge(usa\_unemp, how='outer', left\_index=True, right\_index=True)

data=data.merge(usa\_inf, how='outer', left\_index=True, right\_index=True)

data=data.merge(usa\_CC, how='outer', left\_index=True, right\_index=True)

data=data.merge(usa\_CPI, how='outer', left\_index=True, right\_index=True)

data=data.merge(GOLD, how='outer', left\_index=True, right\_index=True)

data=data.merge(YIELD, how='outer', left\_index=True, right\_index=True)

**For Filling The Nan Values:**

data.interpolate(method="time",inplace=True)

**Slicing Data:**

data=data['19990101':'20191111']

data.index.name='DATE'

**Checking For NaN values:**

data.info()

**OUTPUT:**

<class 'pandas.core.frame.DataFrame'>

DatetimeIndex: 7620 entries, 1999-01-01 to 2019-11-11

Data columns (total 11 columns):

BOA\_CLOSE 7620 non-null float64

JPM\_CLOSE 7620 non-null float64

CITY\_CLOSE 7620 non-null float64

WELLS\_CLOSE 7620 non-null float64

INFLA\_RATE 7620 non-null float64

GDP\_VALUE 7620 non-null float64

UNEMPLO\_RATE 7620 non-null float64

CC\_VALUE 7620 non-null float64

CPI\_VALUE 7620 non-null float64

GOLD\_CLOSE 7620 non-null float64

YIELD\_VALUE 7620 non-null float64

dtypes: float64(11)

memory usage: 1.0 MB

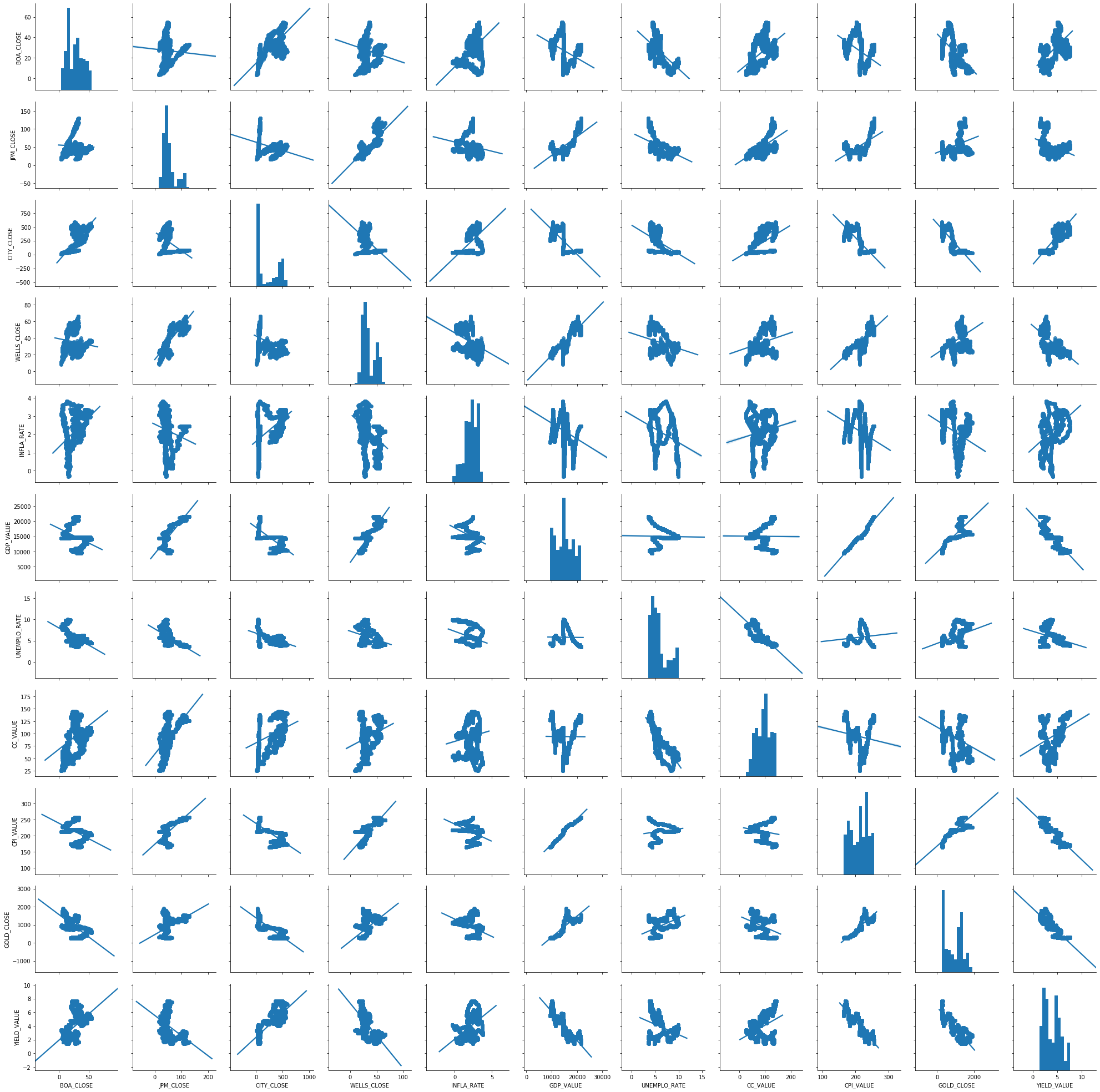
**VISUALIZING THE DATA:**

We used pair plot, histograms and scatter plots to visualize the data to know the insights of data.

Below are the codes and outputs:

sns.pairplot(data,kind='reg')

plt.show()



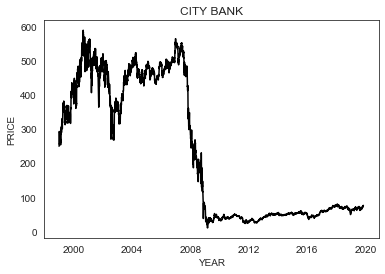
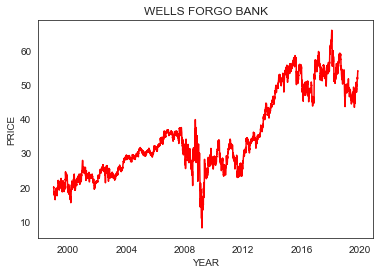
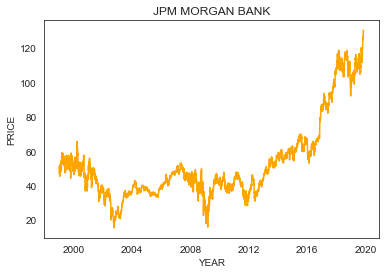
**SCATTER PLOTS**: Below is the code for

plt.plot(data["JPM\_CLOSE"],color='orange')

plt.xlabel("YEAR")

plt.ylabel("PRICE")

plt.title("JPM MORGAN BANK")



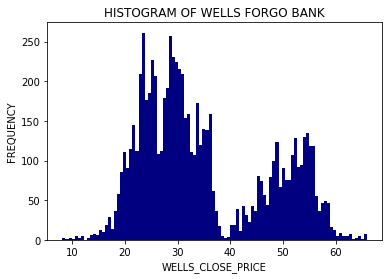
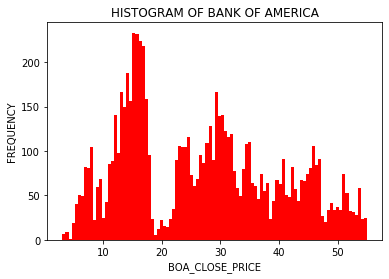
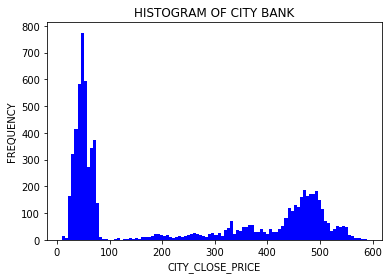
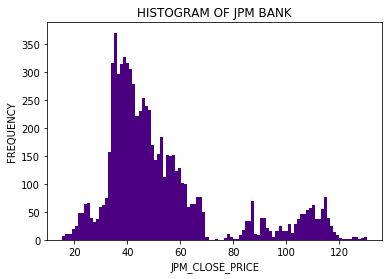
**HISTOGRAM PLOTS**: Below is the code for

plt.hist(data["JPM\_CLOSE"],bins=100,color='indigo')

plt.xlabel("JPM\_CLOSE\_PRICE")

plt.ylabel("FREQUENCY")

plt.title("HISTOGRAM OF JPM BANK")



**CORRELATION PLOT**: Below is the code for correlation

mask=np.zeros\_like(data.corr())

traingle=np.triu\_indices\_from(mask)

mask[traingle]=True

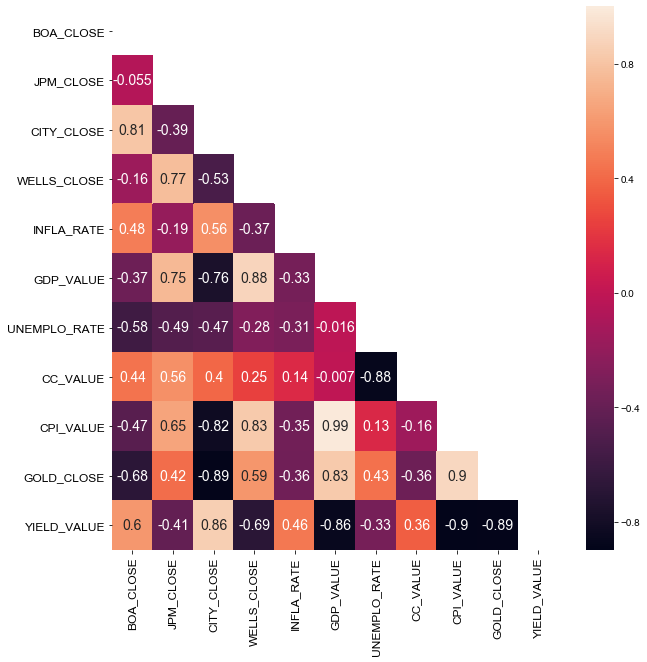
plt.figure(figsize=(10,10))

sns.heatmap(data.corr(),mask=mask,annot=True,annot\_kws={"size":14})

plt.xticks(fontsize=12)

plt.yticks(fontsize=12)

sns.set\_style("white")



**SPLITTING DATA:**

Below is the code for splitting data.in the below code x is the target value y is the features.used sklearn module for splitting the data into training and testing data in the ratio 80:20.

from sklearn.model\_selection import train\_test\_split

x2=data[['JPM\_CLOSE']]

y2=data.drop(['BOA\_CLOSE', 'JPM\_CLOSE', 'CITY\_CLOSE','WELLS\_CLOSE'],axis=1)

x\_train2,x\_test2,y\_train2,y\_test2=train\_test\_split(y2,x2,test\_size=0.2,random\_state=0)

**BUILDING MODEL:**

Used multi linear regression in order to predict the model we considered JP Morgan bank Close price as target variable and features as all the macro factors

**MULTI LINEAR REGRESSION MODEL CREATION OF JPM BANK:**

regression2=LinearRegression()

regression2.fit(x\_train2,y\_train2)

print(pandas.DataFrame({"index":x\_train2.columns,"coeff":regression2.coef\_.tolist()[0]}))

pandas.DataFrame(data=regression2.coef\_,index=['coefficient'],columns=x\_train2.columns)

print("intercept:",regression2.intercept\_)

print('R^2 train dataset:',regression2.score(x\_train2,y\_train2))

print('R^2 test dataset:',regression2.score(x\_test2,y\_test2))

**OUTPUT:**

**index** **coeff**

0 INFLA\_RATE - 0.941970

1 GDP\_VALUE 0.013303

2 UNEMPLO\_RATE 1.701889

3 CC\_VALUE 0.381072

4 CPI\_VALUE -0.871155

5 GOLD\_CLOSE 0.004967

6 YIELD\_VALUE 4.657317

**intercept:** [-28.65914051]

**R^2 train dataset:** 0.9002138646376936

**R^2 test dataset:** 0.9067647947946865

**EVALUATING THE MODEL:** In this we done some statistical test in order to find out the significance of each independent variable to the target variable so we performed p statistics by using OLS (ordinary Least Square ) method of statsmodels.

**Code:**

import statsmodels.api as sm

x2\_include\_constant=sm.add\_constant(x\_train2)

model2=sm.OLS(y\_train2,x2\_include\_constant)

results2=model2.fit()

results2.params

results2.pvalues

print(pandas.DataFrame({"Coeffiencients":results2.params,"P-Values":round(results2.pvalues,2)}))

**output:**

**Coeffiencients P-Values**

**const** -28.659141 0.0

**INFLA\_RATE** -0.941970 0.0

**GDP\_VALUE** 0.013303 0.0

**UNEMPLO\_RATE** 1.701889 0.0

**CC\_VALUE** 0.381072 0.0

**CPI\_VALUE -**0.871155 0.0

**GOLD\_CLOSE** 0.004967 0.0

**YIELD\_VALUE** 4.657317 0.0

By above results we found all the variables are significant to the target value.

**CHECKING FOR MULTI COLLINEARITY**:

Variance inflation Factor (VIF) is used to check the multi co-linearity this can be done by importing variance\_inflation\_factor from statsmodels in python.

**CODE:**

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

VIF=[]

for i in range(0, len(x\_include\_constant.columns)):

VIF.append(variance\_inflation\_factor(exog=x\_include\_constant.values,exog\_idx=i))

print(VIF)

print(pandas.DataFrame({"coeffients":x\_include\_constant.columns,"VIF":np.around(VIF,3)}))

**output:**

**coeffients VIF**

**0 const** 6185.503

**1 INFLA\_RATE** 1.968

**2 GDP\_VALUE** 295.629

**3 UNEMPLO\_RATE** 10.599

**4 CC\_VALUE** 12.833

**5 CPI\_VALUE** 350.044

**6 GOLD\_CLOSE** 22.206

**7 YIELD\_VALUE** 11.811

For the above output table we observed the VIF values are high for some variable so we will try to drop those variables in the table. So we will drop CPI\_VALUE and then run the model below are the results.

**index coeff**

**0 INFLA\_RATE** -0.698328

**1 GDP\_VALUE** 0.007073

**2 UNEMPLO\_RATE** 1.957227

**3 CC\_VALUE** 0.490040

**4 GOLD\_CLOSE** -0.003325

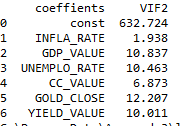
**5 YIELD\_VALUE** 3.654057

**intercept:** [-121.04221419]

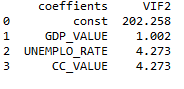
**R^2 train dataset:** 0.8973329178860255

**R^2 test dataset**: 0.9034611975914223

**VIF RESULTS AFTER DROPING CPI\_VALUE:**



Still we have multi co-linearity as the GOLD \_CLOSE is high so we will drop that, and similarly we checked for the rest once also and find and below are the results.



**index coeff**

**0 GDP\_VALUE** 0.005197

**1 UNEMPLO\_RATE** 1.319710

**2 CC\_VALUE** 0.548225

**intercept:** [-84.52143208]

**R^2 train dataset:** 0.8871161790732169

**R^2 test dataset:** 0.8955482922548127

So we will continue the model with the above parameters as we considered the VIF limit is less than 5

**RESIDUALS**: These are the difference between predicted value and target value.

Below code will be used to find the residuals

**CODE:**

results2.resid

Co-relation Between y\_Train (actual target value) And Predicted y\_Train(predicted value)

**CODE:**

act\_pred2=pandas.DataFrame({"actual":y\_train2["JPM\_CLOSE"],"predicted":results2.fittedvalues})

corr2=round(act\_pred2["actual"].corr(act\_pred2["predicted"]),2)

print(corr2)

**output:**

0.94

So we can say by above result the relation holds well.

**GRAPH OF ACTUAL VS PREDICTION:**

SCATTER PLOT

plt.figure(figsize=(7,7))

plt.scatter(x=act\_pred2["actual"],y=act\_pred2["predicted"],color="red",alpha=0.6)

plt.plot(act\_pred2["actual"],act\_pred2["actual"],color="black")

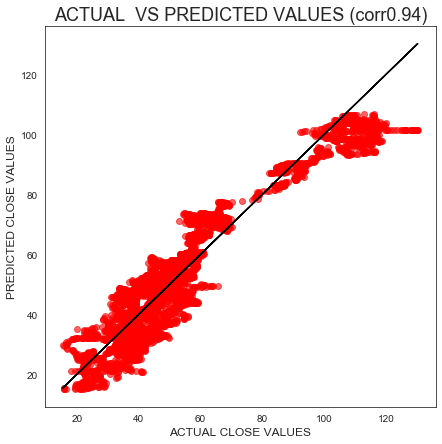
plt.xlabel("ACTUAL CLOSE VALUES",fontsize=12)

plt.ylabel("PREDICTED CLOSE VALUES",fontsize=12)

plt.title(f'ACTUAL VS PREDICTED VALUES (corr{corr})',fontsize=18 )

plt.show()

**output:**



**RESIDUAL VS PREDICTED VALUES**: In general we should have irregularity between these two parameters

**CODE:**

plt.figure(figsize=(7,7))

plt.scatter(x=results2.fittedvalues,y=results2.resid,color="navy",alpha=0.6)

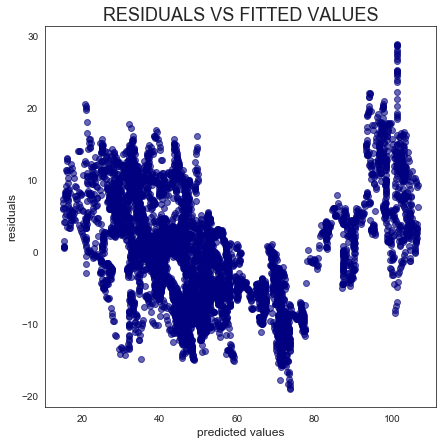
plt.xlabel("predicted values",fontsize=12)

plt.ylabel("residuals",fontsize=12)

plt.title("RESIDUALS VS FITTED VALUES",fontsize=18 )

plt.show()

**output:**



**DISTRIBUTION OF RESIDUALS:**

**CODE:**

residual\_mean2=round(results2.resid.mean(),3)

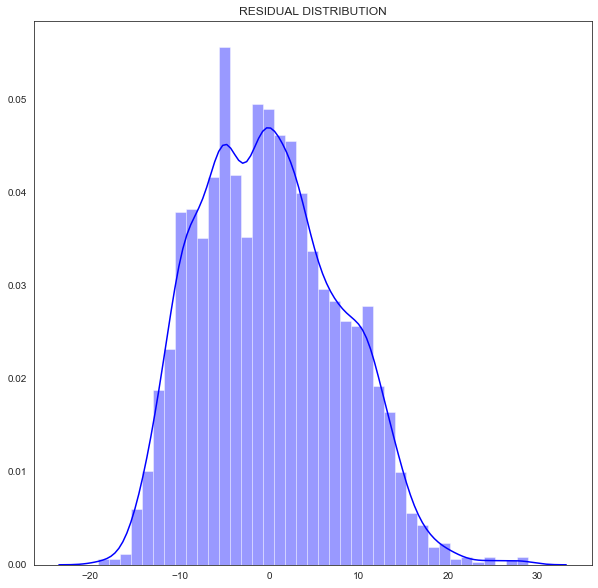
residual\_skew2=round(results2.resid.skew(),3)

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

sns.distplot(results2.resid,color="blue")

plt.title("RESIDUAL DISTRIBUTION")

**OUTPUT:**



**MEAN SQUARED ERROR:**

**CODE:**

print(results2.mse\_resid)

**OUTPUT:**

60.263730672821495

**ROOT MEAN SQUARED ERROR:**

**CODE:**

RMSE2=np.sqrt(results2.mse\_resid)

pandas.DataFrame({"R-Squared":[results2.rsquared],"Mean Square Error":[results2.mse\_resid],"Root Mean Square":np.sqrt(results2.mse\_resid)},index=["JPM\_CLOSE"])

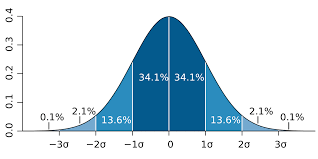
**OUTPUT:**

R-Squared Mean Square Error Root Mean Square

JPM\_CLOSE 0.887116 60.263731 7.762972

**NORMAL DISTRIBUTION GRAPH:**

Below is the figure representing normal distribution graph with standard deviations (1 sigma =68%; 2sigma=95%,3 sigma=100%)



**CODE:**

print("one standard deviation:",np.sqrt(results2.mse\_resid)\*1)

print("Two standard deviation:",np.sqrt(results2.mse\_resid)\*2)

print("Three standard deviation:",np.sqrt(results2.mse\_resid)\*3)

**OUTPUT:**

one standard deviation: 7.762971768132452

Two standard deviation: 15.525943536264904

Three standard deviation: 23.288915304397356

**PREDICTING THE VALUES WITH ONE STANDARD DEVIATION:**

In order to do this we defined a function that will plot the graph with predicted, upper, lower limit and actual graph.

**CODE:**

def predict\_Bank\_price(dataset):

a=regression2.predict(dataset)

upper=a+1\*RMSE2

lower=a-1\*RMSE2

predict=pandas.DataFrame.from\_records(a,columns=["PREDICTED PRICE"])

upper=pandas.DataFrame.from\_records(upper,columns=["UPPER PRICE"])

lower=pandas.DataFrame.from\_records(lower,columns=["LOWER PRICE"])

ind=pandas.DataFrame(dataset.index)

frames=[ind,predict,upper,lower]

df=pandas.concat(frames,axis=1)

df.set\_index(["DATE"],inplace=True)

print(df)

plt.plot(df)

plt.legend(loc='best')

plt.show(block=False)

return df

Calling function and plotting:

b2=predict\_Bank\_price(y2)

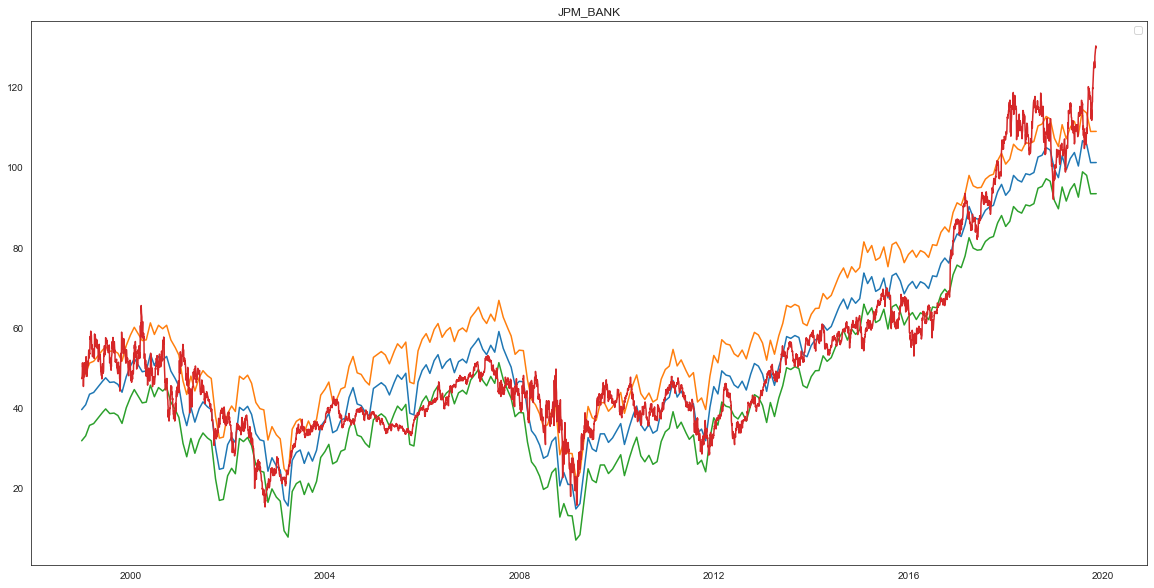
plt.figure(figsize=(20,10))

plt.plot(b2)

plt.plot(x2)

plt.title("JPM\_BANK")

**OUTPUT:**



**CONCLUSION:**

The stock price of 4 listed companies in US stock market has been predicted using multi linear regression. Based on the stock of company either of the models can be used to predict the stock values the accuracy of stock models can be further improved by optimizing the model for relevant indicators and current market trends.