UNIVERSITY OF WESTMINSTER

DEPARTMENT OF COMPUTER SCIENCE

MSc Big Data Technologies

MODULE CODE	: 7BUIS008W
MODULE TITLE	: Data Mining & Machine Learning
MODULE LEADER	: DR. Panagiotis Chountas

COURSEWORK - 2

Student ID: w1813148

STUDENT NAME: PATEL BHAUMIKKUMAR SHAILESH

INDEX

No. **Page No.**

A. Building an Ensemble

First of all, download the diabetes Datasets from the Blackboard and save on the Jupyter Notebook so we can easily use when needed for our process for building an Ensemble.

At the starting, I need to import all the useful libraries that required to run the scripts.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

After importing libraries, I need to add dataset that required to analyse the data. The name of the dataset is diabetes using .csv file format. I upload the dataset in the Jupyter Notebook so easy to locate the path then use head () command for display the five head rows from the dataset for visual understanding of data.

```
bk=pd.read_csv("diabetes.csv")
```

```
bk.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Using this script, I know the Range Index is 768 entries of the data from 0 to 767 and total 9 data columns used. I can also count if the any value is null from the dataset so there are no-null values. Only two datatypes used in this dataset, two times float64 and seven times int64. There are no missing values in this dataset.

```
bk.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
#identify missing values
bk.isnull().sum()
```

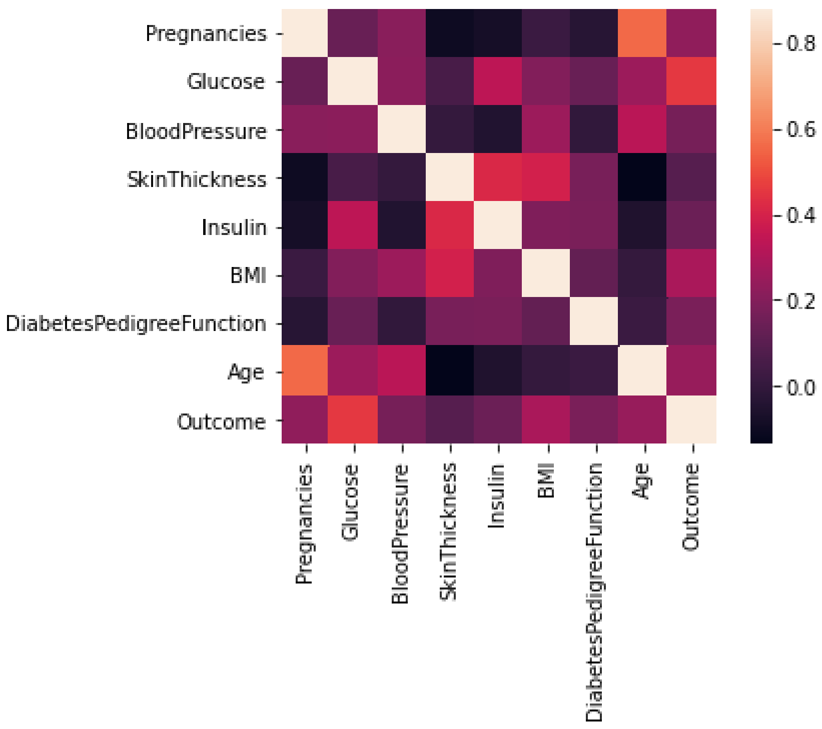
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

I found the one column name BloodPressure with the value 0 so I need to drop all the values that content 0 because BloodPressure have some values.

```
#dropping the Bloodpressure with 0
bk = bk[bk["BloodPressure"]!=0]
```

I can visualize the heatmap using the below script. Its looks stunning and easily understand the data columns correlation between each other.

```
#Check correlation of each values
corrmat = bk.corr()
sns.heatmap(corrmat, vmax=0.88, square=True)
```



Now, I can deal with the outliers. Outliers easily detected using three methods statistical approach, visualization and using mathematical formulas. I used “Box Plot” the visualization method to detect the outliers using the below codes. I also create the dataframe for this analysis.

```
#Dealing with Outliers
import sklearn
from sklearn.datasets import load_diabetes
```

```
# Load the dataset
bos_diabetes = load_diabetes()
```

```
# Create the dataframe
column_name = bos_diabetes.feature_names
bk_diabetes = pd.DataFrame(bos_diabetes.data)
bk_diabetes.columns = column_name
bk_diabetes.head()
```

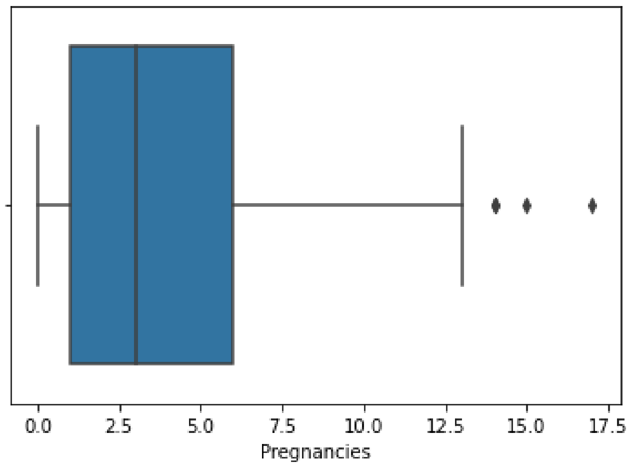
	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641

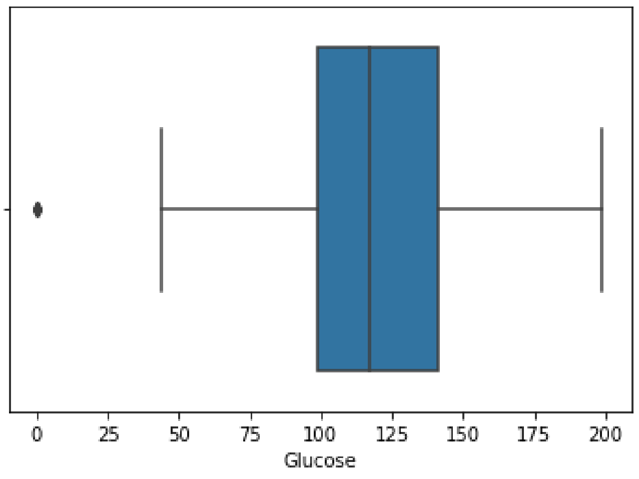
I used same below code for each column analysis and display the figures as outputs below as well as position of the outlier

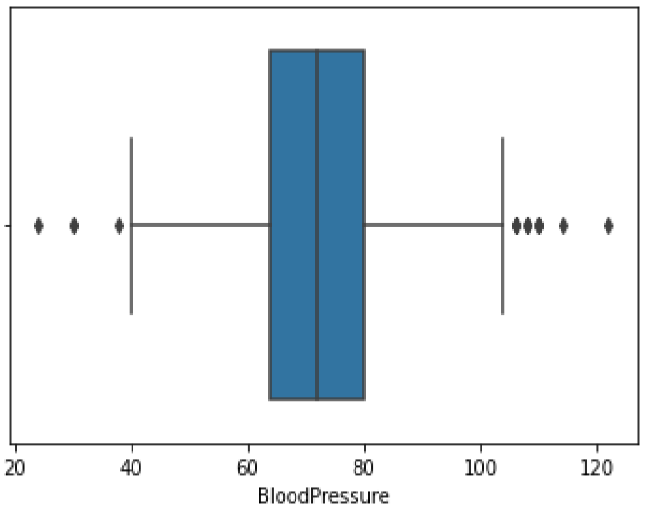
```
# Box Plot
import seaborn as sns

sns.boxplot(bk['Pregnancies'])
```

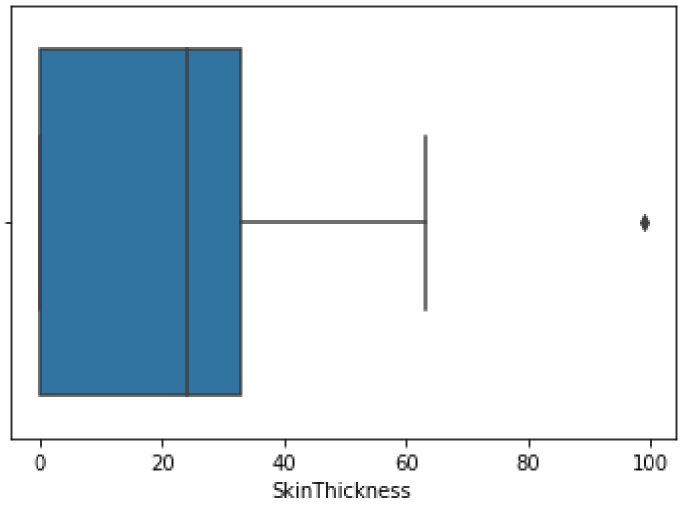
```
# Position of the Outlier

print(np.where(bk['Pregnancies']>10))
```

```
(array([ 22,  26,  34,  68,  80,  82, 153, 207, 245, 250, 262, 286, 310,
       319, 341, 358, 416, 434, 486, 494, 531, 532, 555, 562, 584, 604,
       616, 626, 659, 705, 709, 710], dtype=int64),)
```









Rescaling the real valued using the numeric attributes into 0 to 1 range refers by Normalization. We used Data Normalization in the machine learning for the scale of features for less sensitive data. This allow our dataset to a more accurate model. Below are the coding used for Normalization of my dataset and also display the output.

```
#Data normalization

from sklearn import preprocessing
import numpy as np

a = np.random.random((1, 4))
a = a*20
print("Data = ", a)

# normalize the data attributes
normalized = preprocessing.normalize(a)
print("Normalized Data = ", normalized)
```

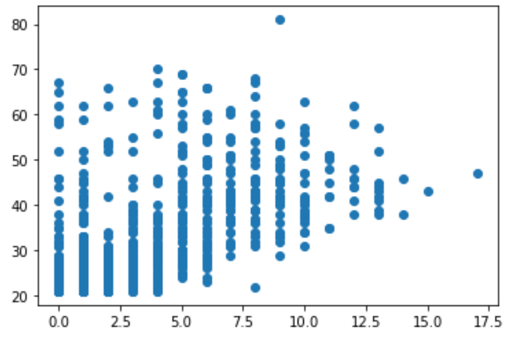
```
Data =  [[ 4.78942944 13.67435362  4.06206514 19.43677925]]
Normalized Data =  [[0.19484439 0.5563024  0.16525363 0.79073039]]
```

Using the below code, I visualise each and every column of this datasets with the Age so we can understand the health factors by age. I also know the healthiest age.

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#visualise Pregnancies by Age
plt.scatter(bk['Pregnancies'],bk['Age'])
```

Using the below output, I found that most of the girls pregnant between 20 to 35 ages. The rate almost null from the age 70 up

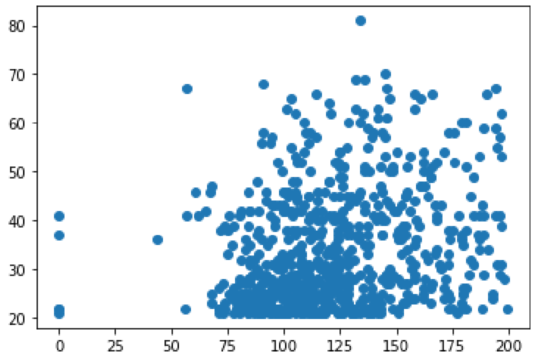


```
#visualise Glucose by Age
plt.scatter(bk['Glucose'],bk['Age'])
```

Its display the normal glucose level near 75 to 125 by age 20 to 30 but few

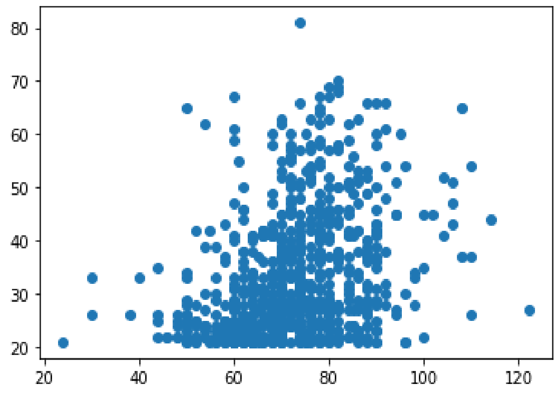
people also have high glucose in this range. That's

going increased for few of them when they older



```
#visualise BloodPressure by Age
plt.scatter(bk['BloodPressure'],bk['Age'])
```

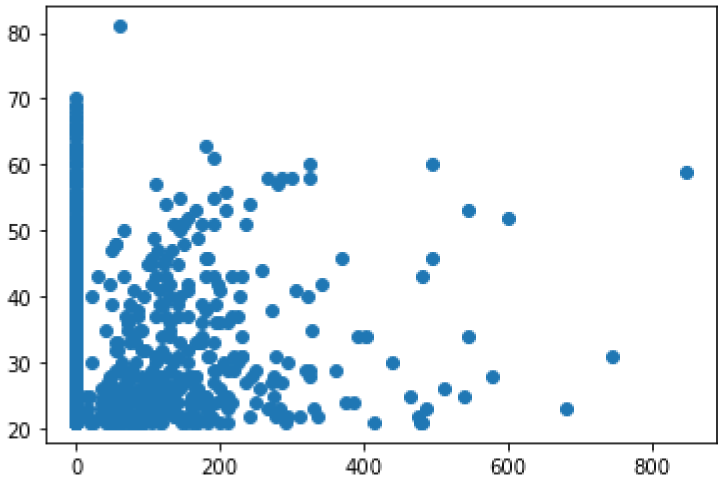
The Blood Pressure noted 60 to 85 for the youngest people and that least increase for others age group.

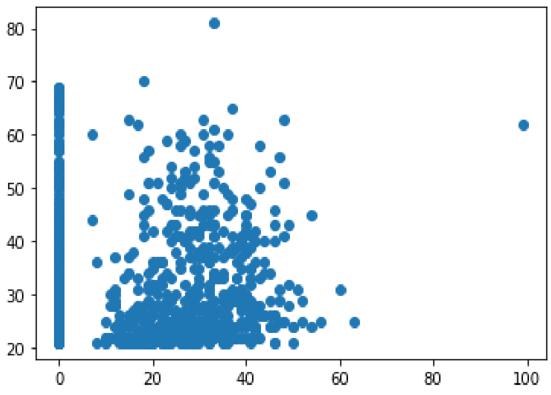


```
#visualise SkinThickness by Age
plt.scatter(bk['SkinThickness'],bk['Age'])
```

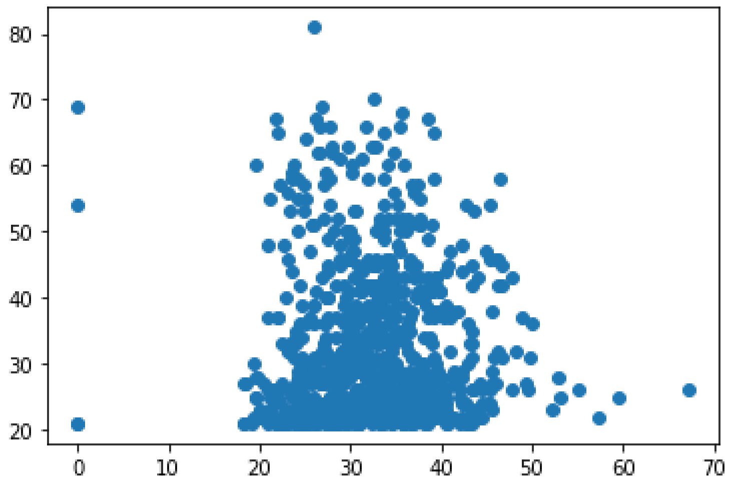
The figure illustrates the skin thickness of the people via the particular age groups. I easily understand that the age group from 20 to 40 have normal skin thickness and that's going down when they older.

```
#visualise Indulin by Age
plt.scatter(bk['Insulin'],bk['Age'])
```

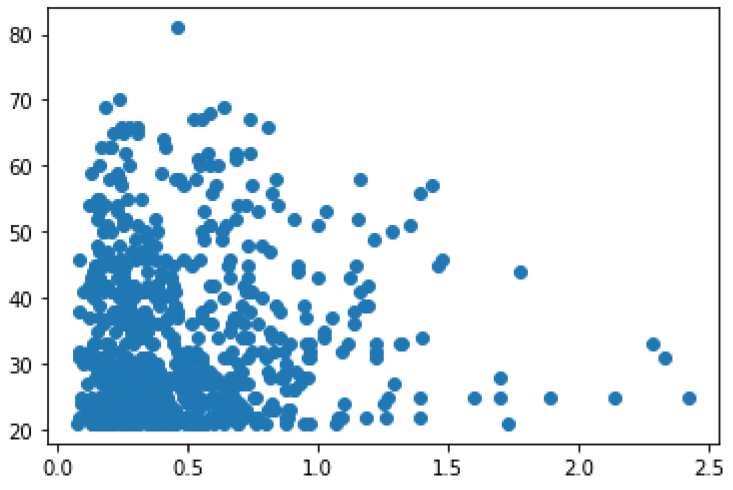




```
#visualise BMI by Age
plt.scatter(bk['BMI'],bk['Age'])
```



```
#visualise DiabetesPedigreeFunction vs Age
plt.scatter(bk['DiabetesPedigreeFunction'],bk['Age'])
```

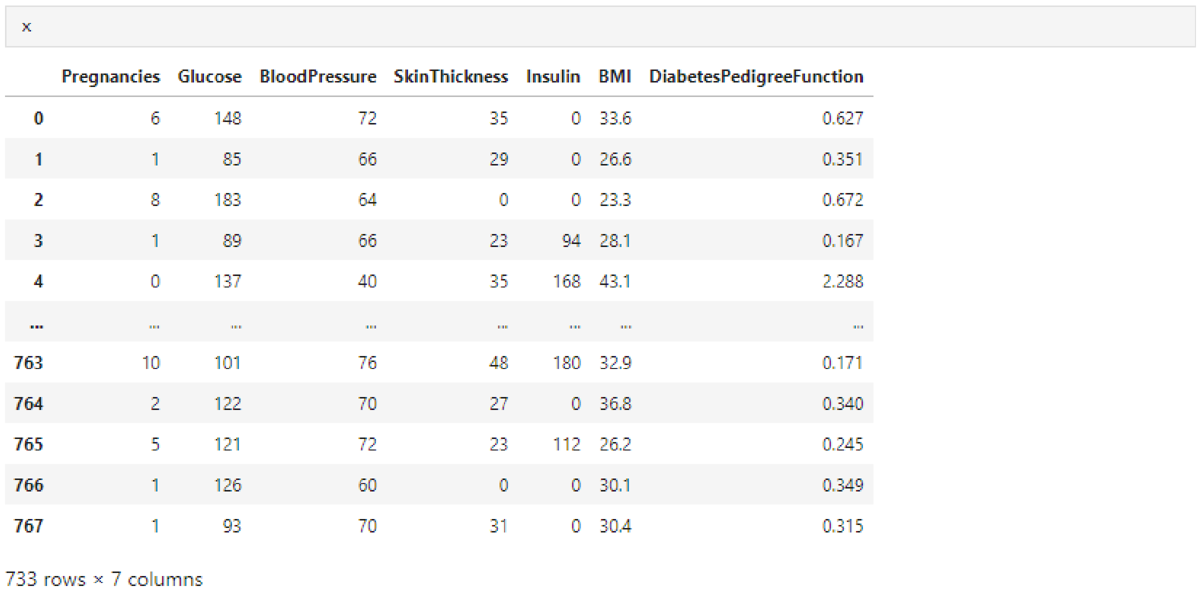


```
#split dataset in features and target variable

x = bk[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction']]
y = bk[['Outcome']]
```

Using the above code, we can split the data columns as datasets in two parts as features and target variables and after that we can split it into the data training and testing using the below script. After that need to use feature scaling technique for pre-processing the dataset. Finishing this, I display the dataset using the x command as well as y command for second set.

```
# split the dataset into training and testing sets

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.26,random_state=42)
```

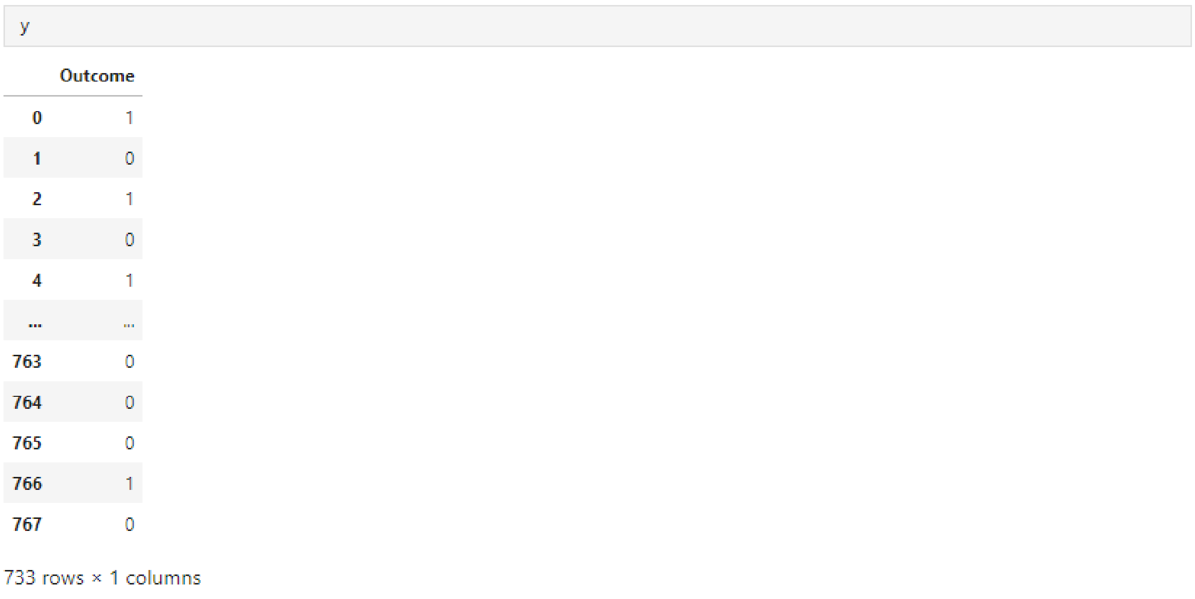
```
# feature scaling

from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```



```
x
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288
...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171
764	2	122	70	27	0	36.8	0.340
765	5	121	72	23	112	26.2	0.245
766	1	126	60	0	0	30.1	0.349
767	1	93	70	31	0	30.4	0.315

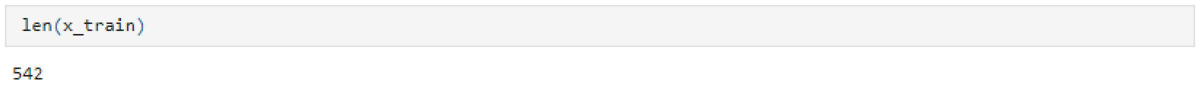
733 rows × 7 columns



```
y
```

	Outcome
0	1
1	0
2	1
3	0
4	1
...	...
763	0
764	0
765	0
766	1
767	0

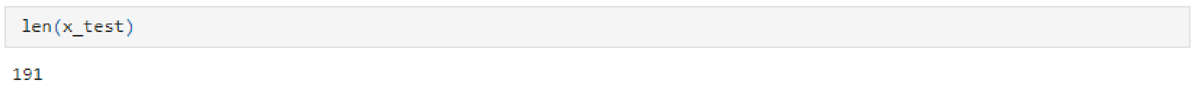
733 rows × 1 columns



```
len(x_train)
```

```
542
```

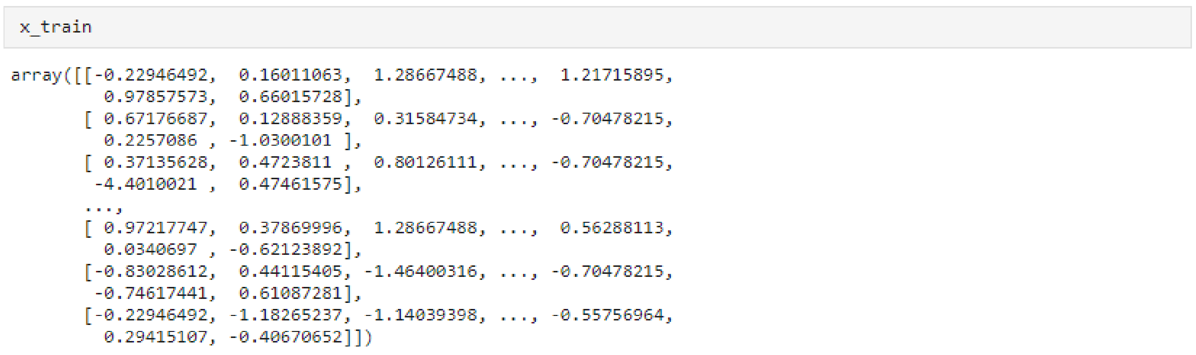
Using the above script, I can figure out the total set of the x_train dataset and below script represent total the x_test dataset



```
len(x_test)
```

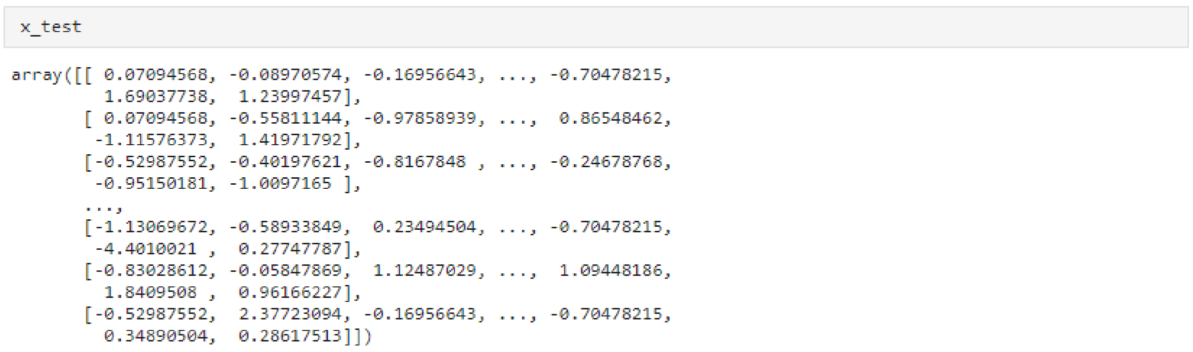
```
191
```

Using the below method, I get the data that given output using the array () that based on the numerical range data. The both split inputs and outputs are below using a single function call.



```
x_train
```

```
array([[-0.22946492,  0.16011063,  1.28667488, ...,  1.21715895,
         0.97857573,  0.66015728],
       [ 0.67176687,  0.12888359,  0.31584734, ..., -0.70478215,
         0.2257086 , -1.0300101 ],
       [ 0.37135628,  0.4723811 ,  0.80126111, ..., -0.70478215,
        -4.4010021 ,  0.47461575],
       ...,
       [ 0.97217747,  0.37869996,  1.28667488, ...,  0.56288113,
         0.0340697 , -0.62123892],
       [-0.83028612,  0.44115405, -1.46400316, ..., -0.70478215,
        -0.74617441,  0.61087281],
       [-0.22946492, -1.18265237, -1.14039398, ..., -0.55756964,
         0.29415107, -0.40670652]])
```



```
x_test
```

```
array([[ 0.07094568, -0.08970574, -0.16956643, ..., -0.70478215,
         1.69037738,  1.23997457],
       [ 0.07094568, -0.55811144, -0.97858939, ...,  0.86548462,
        -1.11576373,  1.41971792],
       [-0.52987552, -0.40197621, -0.8167848 , ..., -0.24678768,
        -0.95150181, -1.0097165 ],
       ...,
       [-1.13069672, -0.58933849,  0.23494504, ..., -0.70478215,
        -4.4010021 ,  0.27747787],
       [-0.83028612, -0.05847869,  1.12487029, ...,  1.09448186,
         1.8409508 ,  0.96166227],
       [-0.52987552,  2.37723094, -0.16956643, ..., -0.70478215,
         0.34890504,  0.28617513]])
```

Now, I can use the training dataset to fit the model and the test dataset for evaluate the model. Below are the classifiers we used using the four process steps

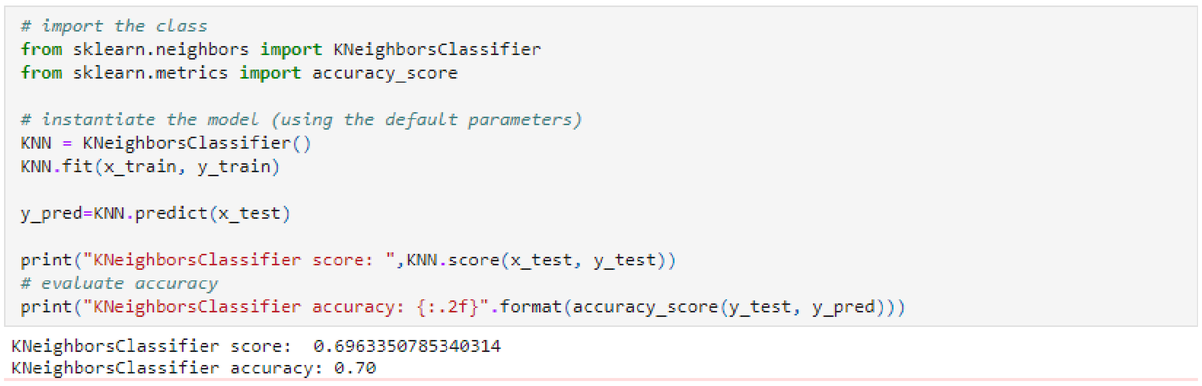
(1) Import the class we need

(2) Now, using this class I create the model instances

(3) using fit( ) function in the training set

(4) Evaluate the classifier score using test set.

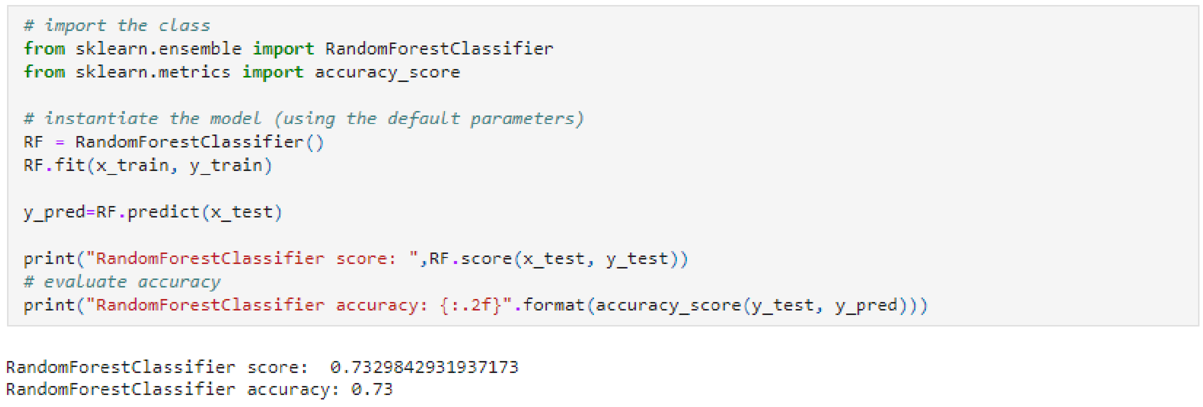
Using below code, I got the KNN Classifier score 0.6963350785340314 and accuracy score 0.70



```
# import the class
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# instantiate the model (using the default parameters)
KNN = KNeighborsClassifier()
KNN.fit(x_train, y_train)

y_pred=KNN.predict(x_test)

print("KNeighborsClassifier score: ",KNN.score(x_test, y_test))
# evaluate accuracy
print("KNeighborsClassifier accuracy: {:.2f}".format(accuracy_score(y_test, y_pred)))
```

```
KNeighborsClassifier score:  0.6963350785340314
KNeighborsClassifier accuracy: 0.70
```

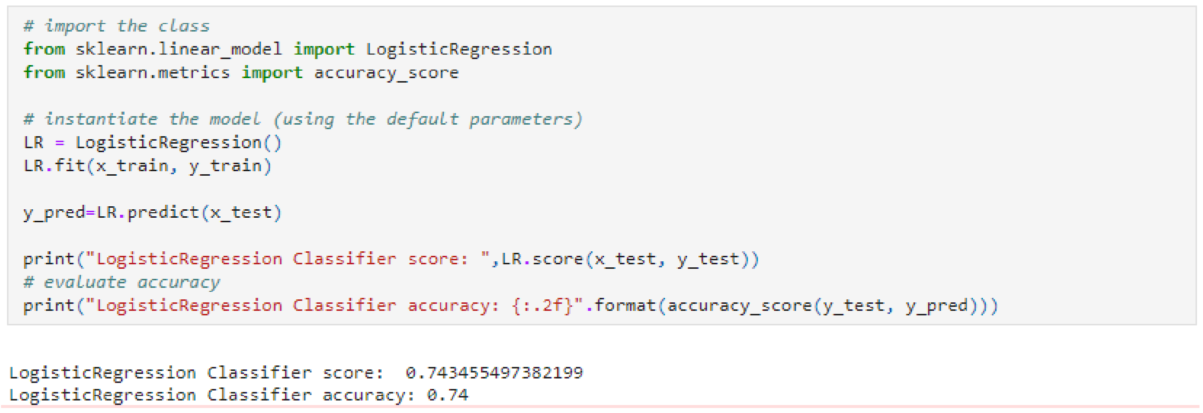
Using below code, I got the Random Forest Classifier score 0.7329842931937173 and accuracy score 0.73



```
# import the class
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# instantiate the model (using the default parameters)
RF = RandomForestClassifier()
RF.fit(x_train, y_train)

y_pred=RF.predict(x_test)

print("RandomForestClassifier score: ",RF.score(x_test, y_test))
# evaluate accuracy
print("RandomForestClassifier accuracy: {:.2f}".format(accuracy_score(y_test, y_pred)))
```

```
RandomForestClassifier score:  0.7329842931937173
RandomForestClassifier accuracy: 0.73
```

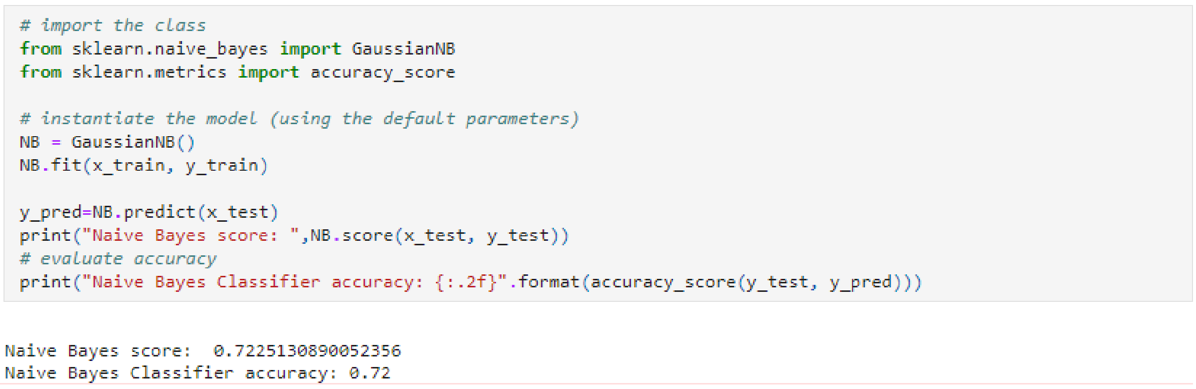
Using below code, I got the Logistic Regression Classifier score 0.743455497382199 and accuracy score 0.74



```
# import the class
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# instantiate the model (using the default parameters)
LR = LogisticRegression()
LR.fit(x_train, y_train)

y_pred=LR.predict(x_test)

print("LogisticRegression Classifier score: ",LR.score(x_test, y_test))
# evaluate accuracy
print("LogisticRegression Classifier accuracy: {:.2f}".format(accuracy_score(y_test, y_pred)))
```

```
LogisticRegression Classifier score:  0.743455497382199
LogisticRegression Classifier accuracy: 0.74
```

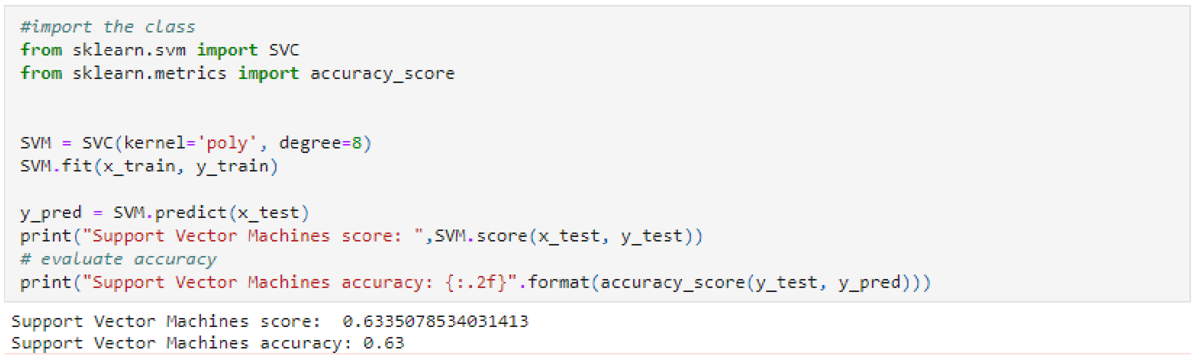
Using below code, I got the Naïve Bayes score 0.7225130890052356 and accuracy score 0.72



```python
# import the class
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# instantiate the model (using the default parameters)
NB = GaussianNB()
NB.fit(x_train, y_train)

y_pred=NB.predict(x_test)
print("Naive Bayes score: ",NB.score(x_test, y_test))
# evaluate accuracy
print("Naive Bayes Classifier accuracy: {:.2f}".format(accuracy_score(y_test, y_pred)))
```

```
Naive Bayes score:  0.7225130890052356
Naive Bayes Classifier accuracy: 0.72
```

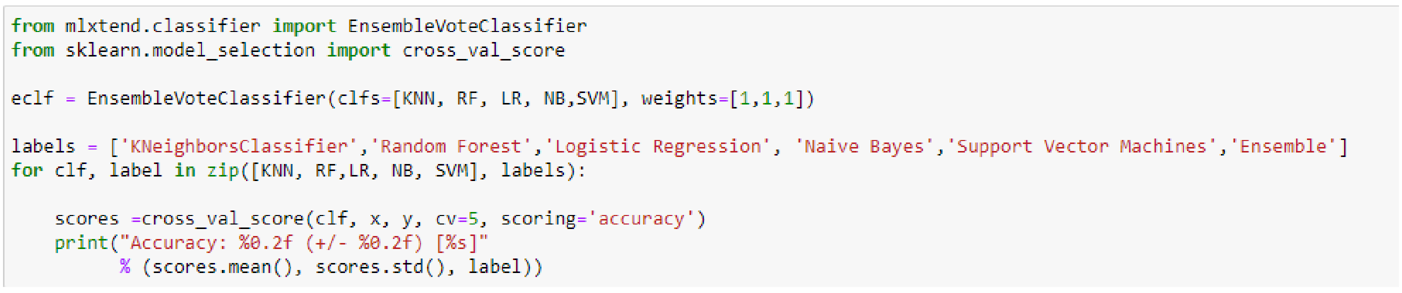
Using below code, I got the Support Vector Machines score 0.6335078534031413 and accuracy score 0.63



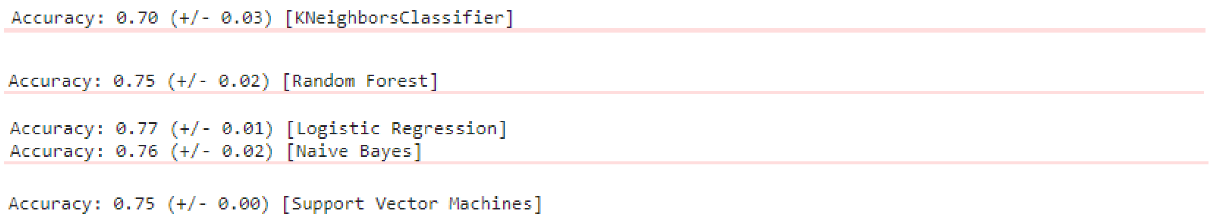
```python
#import the class
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score


SVM = SVC(kernel='poly', degree=8)
SVM.fit(x_train, y_train)

y_pred = SVM.predict(x_test)
print("Support Vector Machines score: ",SVM.score(x_test, y_test))
# evaluate accuracy
print("Support Vector Machines accuracy: {:.2f}".format(accuracy_score(y_test, y_pred)))
```

```
Support Vector Machines score:  0.6335078534031413
Support Vector Machines accuracy: 0.63
```

We can use below script for the Ensemble vote classifier using the KNN, RF, LR, NB and SVM. Using this code, we also get the prediction of the cross validation and the accuracy of each classifier as below display as output of the script.



```python
from mlxtend.classifier import EnsembleVoteClassifier
from sklearn.model_selection import cross_val_score

eclf = EnsembleVoteClassifier(clfs=[KNN, RF, LR, NB,SVM], weights=[1,1,1])

labels = ['KNeighborsClassifier','Random Forest','Logistic Regression', 'Naive Bayes','Support Vector Machines','Ensemble']
for clf, label in zip([KNN, RF,LR, NB, SVM], labels):

    scores =cross_val_score(clf, x, y, cv=5, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```



```
Accuracy: 0.70 (+/- 0.03) [KNeighborsClassifier]

Accuracy: 0.75 (+/- 0.02) [Random Forest]

Accuracy: 0.77 (+/- 0.01) [Logistic Regression]
Accuracy: 0.76 (+/- 0.02) [Naive Bayes]

Accuracy: 0.75 (+/- 0.00) [Support Vector Machines]
```

Now, I use the Grid Search method to performing the top n_neighbors values. Using this script, we needs to create new KNN model as KNN2 after that create a directory as param_grid after that required to use grid search for test all values for n_neighbors for fit model to data.

```
from sklearn.model_selection import GridSearchCV
#create new a knn model
KNN2 = KNeighborsClassifier()
#create a dictionary of all values we want to test for n_neighbors
param_grid = {'n_neighbors': np.arange(1, 25)}
#use gridsearch to test all values for n_neighbors
KNN_gscv = GridSearchCV(KNN2, param_grid, cv=5)
#fit model to data
KNN_gscv.fit(x, y)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24])})
```

```
#check top performing n_neighbors value
KNN_gscv.best_params_
```

```
{'n_neighbors': 14}
```

By performing the Top performing n_neighbors value script we get the 14 as n_neighbors value and for the best score we performing below script and got 0.7476190476190476. As per the last KNN score we got 4% increased because of the Grid search CV method.

```
#check mean score for the top performing value of n_neighbors
KNN_gscv.best_score_
```

```
0.7476190476190476
```

I assign the KNN_model as below and other classifier models as follows

```
KNN_model = KNeighborsClassifier(n_neighbors=3)
KNN_model.fit(x_train, y_train)
```

```
RF_model = RandomForestClassifier(random_state=42)
RF_model.fit(x_train, y_train)
```

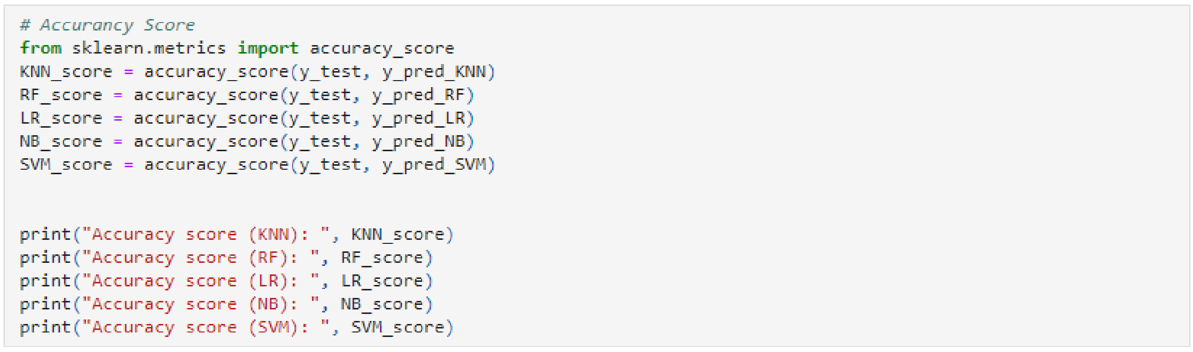
```
LR_model = LogisticRegression()
LR_model.fit(x_train, y_train)
```

```
NB_model = GaussianNB()
NB_model.fit(x_train, y_train)
```

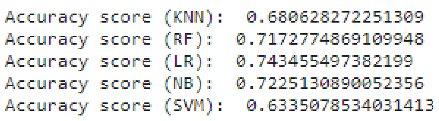
```
SVM_model = SVC(kernel='poly', degree=8)
SVM_model.fit(x_train, y_train)
```

```
y_pred_KNN = KNN_model.predict(x_test)
y_pred_RF = RF_model.predict(x_test)
y_pred_LR = LR_model.predict(x_test)
y_pred_NB = NB_model.predict(x_test)
y_pred_SVM = SVM_model.predict(x_test)
```

Now, the Accuracy score means total number of the positive predictions score. Classification models evaluated by the accuracy. Using the below script, I can perform the accuracy score for the KNN, RF, LR, NB and SVM. the highest score noted for NB and lowest for SVM

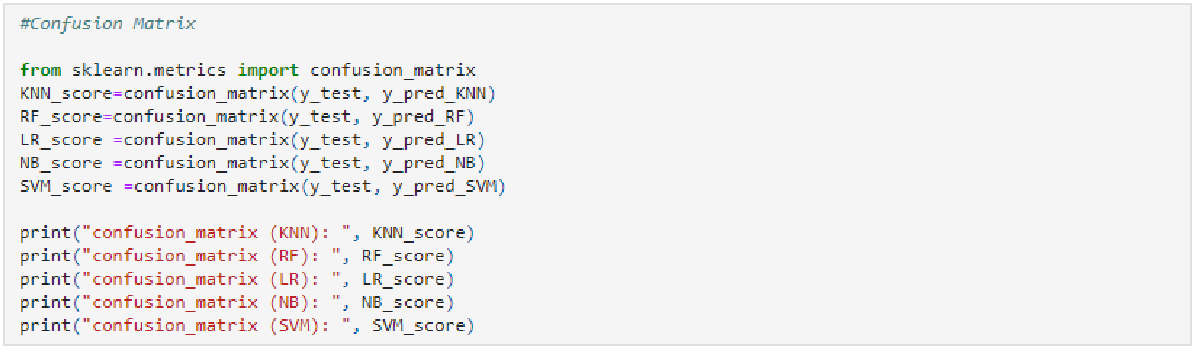


```
# Accurancy Score
from sklearn.metrics import accuracy_score
KNN_score = accuracy_score(y_test, y_pred_KNN)
RF_score = accuracy_score(y_test, y_pred_RF)
LR_score = accuracy_score(y_test, y_pred_LR)
NB_score = accuracy_score(y_test, y_pred_NB)
SVM_score = accuracy_score(y_test, y_pred_SVM)


print("Accuracy score (KNN): ", KNN_score)
print("Accuracy score (RF): ", RF_score)
print("Accuracy score (LR): ", LR_score)
print("Accuracy score (NB): ", NB_score)
print("Accuracy score (SVM): ", SVM_score)
```

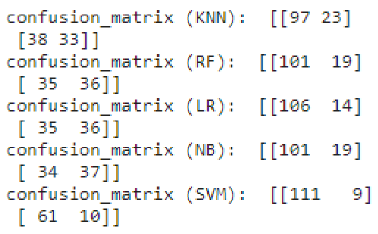


```
Accuracy score (KNN):  0.680628272251309
Accuracy score (RF):  0.7172774869109948
Accuracy score (LR):  0.743455497382199
Accuracy score (NB):  0.7225130890052356
Accuracy score (SVM):  0.6335078534031413
```

Whenever we need to extract more information from the performance model then we can use the confusion matrix. That’s illustrate the model which is confused discriminating in between 2 classes. This labels always have 2x2 matrix means 2 rows and 2 columns. Now, rows represent the truth label and the column represents the predicted labels. This might be changed. Using below code I perform the confusion matrix for all of the classifiers.



```
#Confusion Matrix

from sklearn.metrics import confusion_matrix
KNN_score=confusion_matrix(y_test, y_pred_KNN)
RF_score=confusion_matrix(y_test, y_pred_RF)
LR_score =confusion_matrix(y_test, y_pred_LR)
NB_score =confusion_matrix(y_test, y_pred_NB)
SVM_score =confusion_matrix(y_test, y_pred_SVM)

print("confusion_matrix (KNN): ", KNN_score)
print("confusion_matrix (RF): ", RF_score)
print("confusion_matrix (LR): ", LR_score)
print("confusion_matrix (NB): ", NB_score)
print("confusion_matrix (SVM): ", SVM_score)
```



```
confusion_matrix (KNN):  [[97 23]
 [38 33]]
confusion_matrix (RF):  [[101  19]
 [ 35  36]]
confusion_matrix (LR):  [[106  14]
 [ 35  36]]
confusion_matrix (NB):  [[101  19]
 [ 34  37]]
confusion_matrix (SVM):  [[111   9]
 [ 61  10]]
```

$$\text{Precision} = \text{True}_{\text{positive}} / (\text{True}_{\text{positive}} + \text{False}_{\text{positive}})$$

$$\text{Total Predicted Positive} = \text{True}_{\text{positive}} + \text{False}_{\text{positive}}$$

$$\text{Precision} = \text{True}_{\text{positive}} / \text{Total Predicted Positive}$$

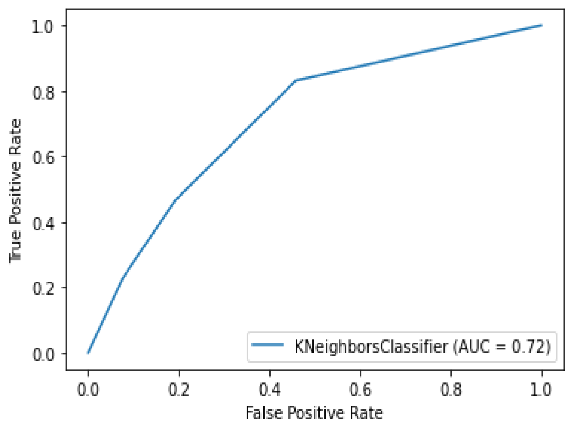
$$\text{Recall} = \text{True}_{\text{positive}} / (\text{True}_{\text{positive}} + \text{False}_{\text{negative}})$$

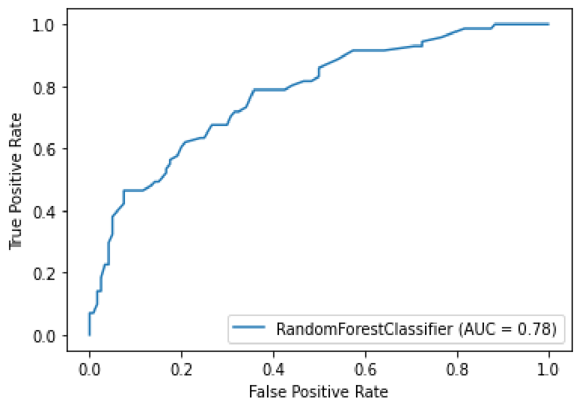
```
#PRECISION, RECALL, F1 SCORE, SUPPORT

import sklearn.metrics
KNN_score=sklearn.metrics.classification_report(y_test, y_pred_KNN)
RF_score=sklearn.metrics.classification_report(y_test, y_pred_RF)
LR_score =sklearn.metrics.classification_report(y_test, y_pred_LR)
NB_score =sklearn.metrics.classification_report(y_test, y_pred_NB)
SVM_score =sklearn.metrics.classification_report(y_test, y_pred_SVM)

print("KNN: ", KNN_score)
print("RF: ", RF_score)
print("LR: ", LR_score)
print("NB: ", NB_score)
print("SVM: ", SVM_score)
```

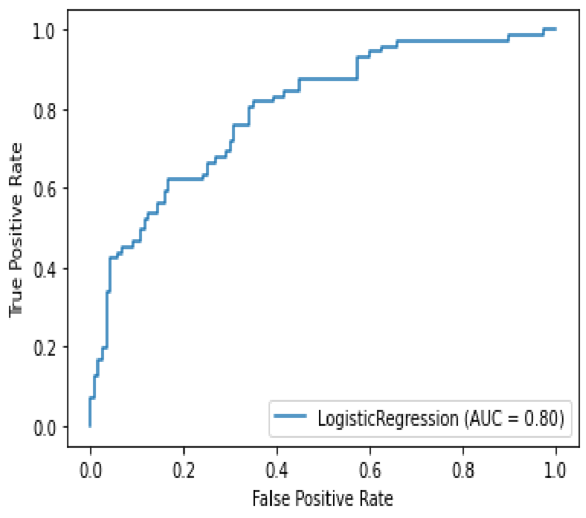
```
KNN:                precision    recall  f1-score   support

           0       0.72      0.81      0.76       120
           1       0.59      0.46      0.52        71

    accuracy                           0.68       191
   macro avg       0.65      0.64      0.64       191
weighted avg       0.67      0.68      0.67       191

RF:               precision    recall  f1-score   support

           0       0.74      0.84      0.79       120
           1       0.65      0.51      0.57        71

    accuracy                           0.72       191
   macro avg       0.70      0.67      0.68       191
weighted avg       0.71      0.72      0.71       191

LR:               precision    recall  f1-score   support

           0       0.75      0.88      0.81       120
           1       0.72      0.51      0.60        71

    accuracy                           0.74       191
   macro avg       0.74      0.70      0.70       191
weighted avg       0.74      0.74      0.73       191

NB:               precision    recall  f1-score   support

           0       0.75      0.84      0.79       120
           1       0.66      0.52      0.58        71

    accuracy                           0.72       191
   macro avg       0.70      0.68      0.69       191
weighted avg       0.72      0.72      0.71       191

SVM:                precision    recall  f1-score   support

           0       0.65      0.93      0.76       120
           1       0.53      0.14      0.22        71

    accuracy                           0.63       191
   macro avg       0.59      0.53      0.49       191
weighted avg       0.60      0.63      0.56       191
```

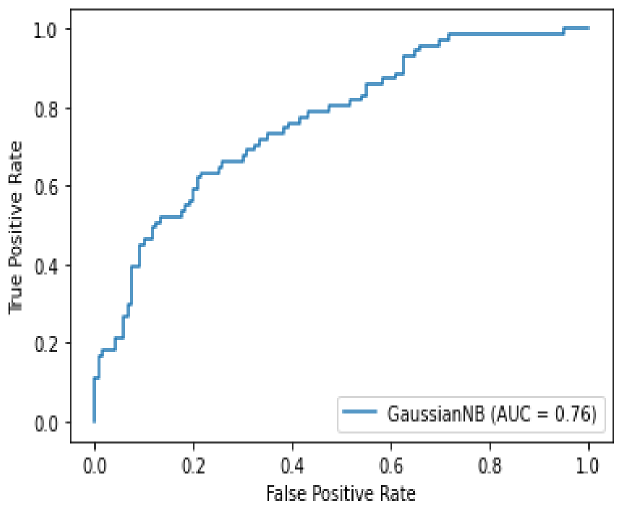
Using below code, I get the ROC Curve and AUROC for all classifier

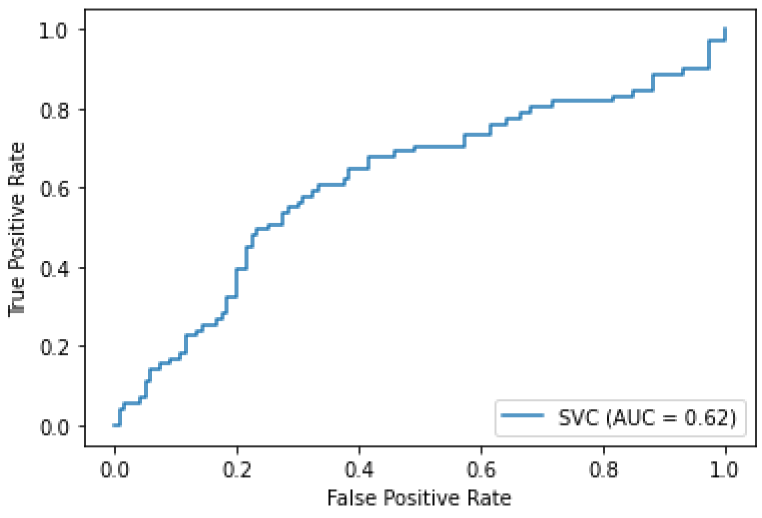
```
#ROC Curve and AUROC

from sklearn.metrics import plot_roc_curve
plot_roc_curve(KNN_model, x_test, y_test)
plot_roc_curve(RF_model, x_test, y_test)
plot_roc_curve(LR_model, x_test, y_test)
plot_roc_curve(NB_model, x_test, y_test)
plot_roc_curve(SVM_model, x_test, y_test)
```





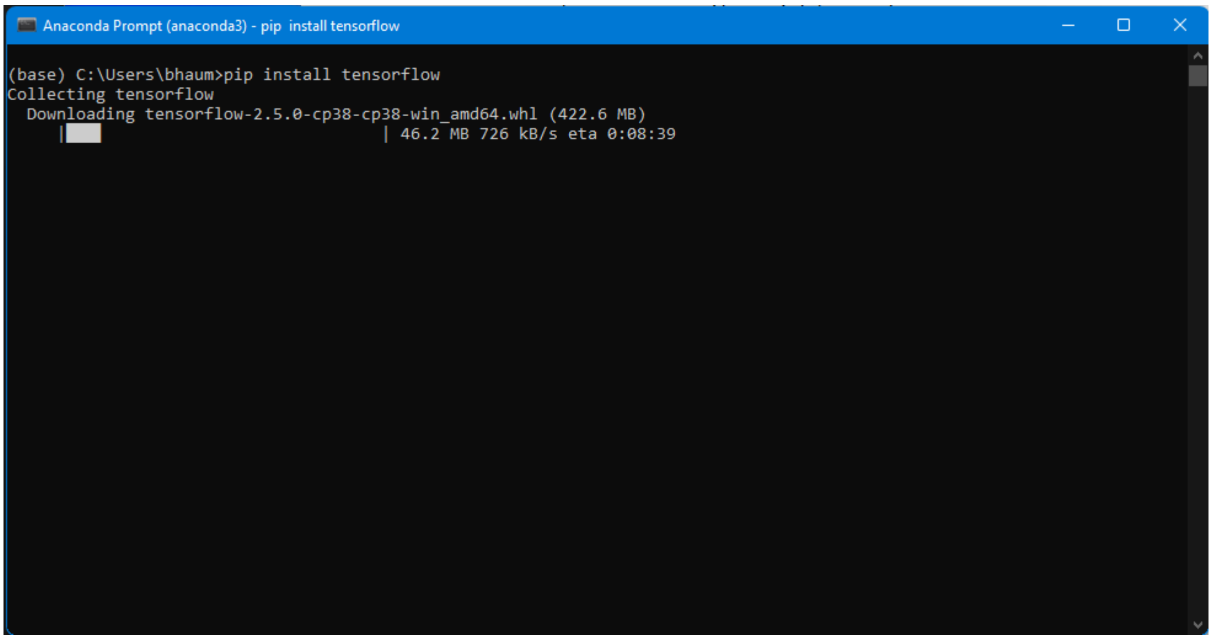




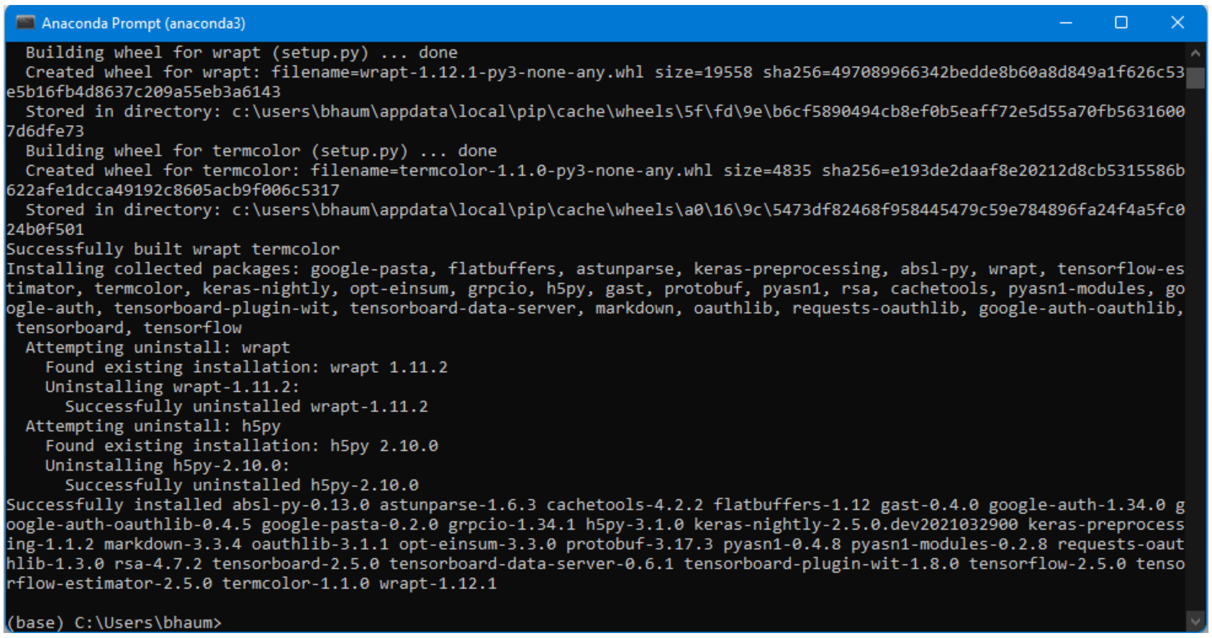


B. Predictign the Price of Apple Stock with LSTM Neural Networks

First of all, I need to install the tensorflow library using Anaconda Prompt. Below are the screenshots of the installation process. Collecting tensorflow data from internet approx. 422.6MB using pip install tensorflow command.



```
(base) C:\Users\bhaum>pip install tensorflow
Collecting tensorflow
  Downloading tensorflow-2.5.0-cp38-cp38-win_amd64.whl (422.6 MB)
     |██                              | 46.2 MB 726 kB/s eta 0:08:39
```



```
  Building wheel for wrapt (setup.py) ... done
  Created wheel for wrapt: filename=wrapt-1.12.1-py3-none-any.whl size=19558 sha256=497089966342bedde8b60a8d849a1f626c53
e5b16fb4d8637c209a55eb3a6143
  Stored in directory: c:\users\bhaum\appdata\local\pip\cache\wheels\5f\fd\9e\b6cf5890494cb8ef0b5eaff72e5d55a70fb5631600
7d6dfe73
  Building wheel for termcolor (setup.py) ... done
  Created wheel for termcolor: filename=termcolor-1.1.0-py3-none-any.whl size=4835 sha256=e193de2daaf8e20212d8cb5315586b
622afe1dcca49192c8605acb9f006c5317
  Stored in directory: c:\users\bhaum\appdata\local\pip\cache\wheels\a0\16\9c\5473df82468f958445479c59e784896fa24f4a5fc0
24b0f501
Successfully built wrapt termcolor
Installing collected packages: google-pasta, flatbuffers, astunparse, keras-preprocessing, absl-py, wrapt, tensorflow-es
timator, termcolor, keras-nightly, opt-einsum, grpcio, h5py, gast, protobuf, pyasn1, rsa, cachetools, pyasn1-modules, go
ogle-auth, tensorboard-plugin-wit, tensorboard-data-server, markdown, oauthlib, requests-oauthlib, google-auth-oauthlib,
 tensorboard, tensorflow
  Attempting uninstall: wrapt
    Found existing installation: wrapt 1.11.2
    Uninstalling wrapt-1.11.2:
      Successfully uninstalled wrapt-1.11.2
  Attempting uninstall: h5py
    Found existing installation: h5py 2.10.0
    Uninstalling h5py-2.10.0:
      Successfully uninstalled h5py-2.10.0
Successfully installed absl-py-0.13.0 astunparse-1.6.3 cachetools-4.2.2 flatbuffers-1.12 gast-0.4.0 google-auth-1.34.0 g
oogle-auth-oauthlib-0.4.5 google-pasta-0.2.0 grpcio-1.34.1 h5py-3.1.0 keras-nightly-2.5.0.dev2021032900 keras-preprocess
ing-1.1.2 markdown-3.3.4 oauthlib-3.1.1 opt-einsum-3.3.0 protobuf-3.17.3 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-oaut
hlib-1.3.0 rsa-4.7.2 tensorboard-2.5.0 tensorboard-data-server-0.6.1 tensorboard-plugin-wit-1.8.0 tensorflow-2.5.0 tenso
rflow-estimator-2.5.0 termcolor-1.1.0 wrapt-1.12.1

(base) C:\Users\bhaum>
```

First of all, import all the useful libraries that required to perform various operations related to the dataset.

```
#Importing Libraries

import numpy as np
import pandas as pd
import tensorflow as tf
import keras
import matplotlib as plt
import math

from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt

from keras import layers
from keras import models
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
```

Using the provided link from coursework, I download the full 1year of the Apple stock data for analysis. Below script used for importing dataset into the coding for analysis.

```
bk=pd.read_csv("HistoricalData_1628101863466.csv")
```

I assign the data set to my own nick name bk. Performing bk command, displayed full dataset 252 rows x 6 columns

```
bk
```

	Date	Close/Last	Volume	Open	High	Low
0	08/03/2021	$147.36	64786620	$145.81	$148.045	$145.18
1	08/02/2021	$145.52	62879960	$146.36	$146.95	$145.25
2	07/30/2021	$145.86	70440630	$144.38	$146.33	$144.11
3	07/29/2021	$145.64	56699480	$144.685	$146.55	$144.58
4	07/28/2021	$144.98	118931200	$144.81	$146.97	$142.54
...	...	...	...	...	...	...
247	08/10/2020	$112.7275	212403440	$112.6	$113.775	$110
248	08/07/2020	$111.1125	198045600	$113.205	$113.675	$110.2925
249	08/06/2020	$113.9025	202428920	$110.405	$114.4125	$109.7975
250	08/05/2020	$110.0625	121991960	$109.3775	$110.3925	$108.8975
251	08/04/2020	$109.665	172792360	$109.1325	$110.79	$108.3875

252 rows × 6 columns

Performing below command, checking the null values of the dataset. So, there are no null values from the datasets. We can see that from the output of the command.

```
#Checking null values
print(bk.isnull().sum())
total_null = bk.isnull().sum()
print("The total number of null values are: " + str(total_null))
```

```
Date          0
Close/Last    0
Volume        0
Open          0
High          0
Low           0
dtype: int64
The total number of null values are: Date          0
Close/Last    0
Volume        0
Open          0
High          0
Low           0
dtype: int64
```

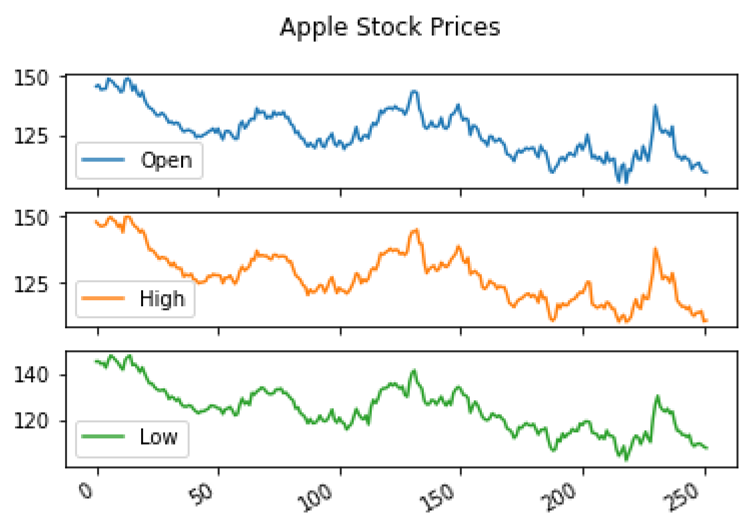
Now, I would like to drop some of the columns from the dataset like Date, Close/Last and Volume. Using below code our intention performed. After that I use script for drop the values from the column except open, high and low

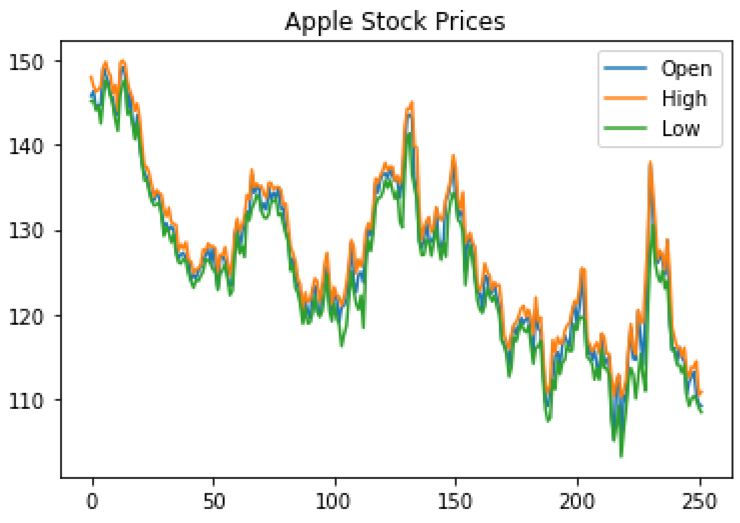
```
#Drop every column except open, high and low columns
bk.drop(['Date', 'Close/Last', 'Volume'], axis = 1, inplace = True)
```

```
bk['Open'] = bk['Open'].str.replace('$', '').astype(float)
bk['High'] = bk['High'].str.replace('$', '').astype(float)
bk['Low'] = bk['Low'].str.replace('$', '').astype(float)
```

After that performing the visual effect of the stock price via open, high and low using below command lines.

```
#Visualize the stock price data
bk.plot.line(subplots = True, title='Apple Stock Prices')
bk.plot(title='Apple Stock Prices')
```



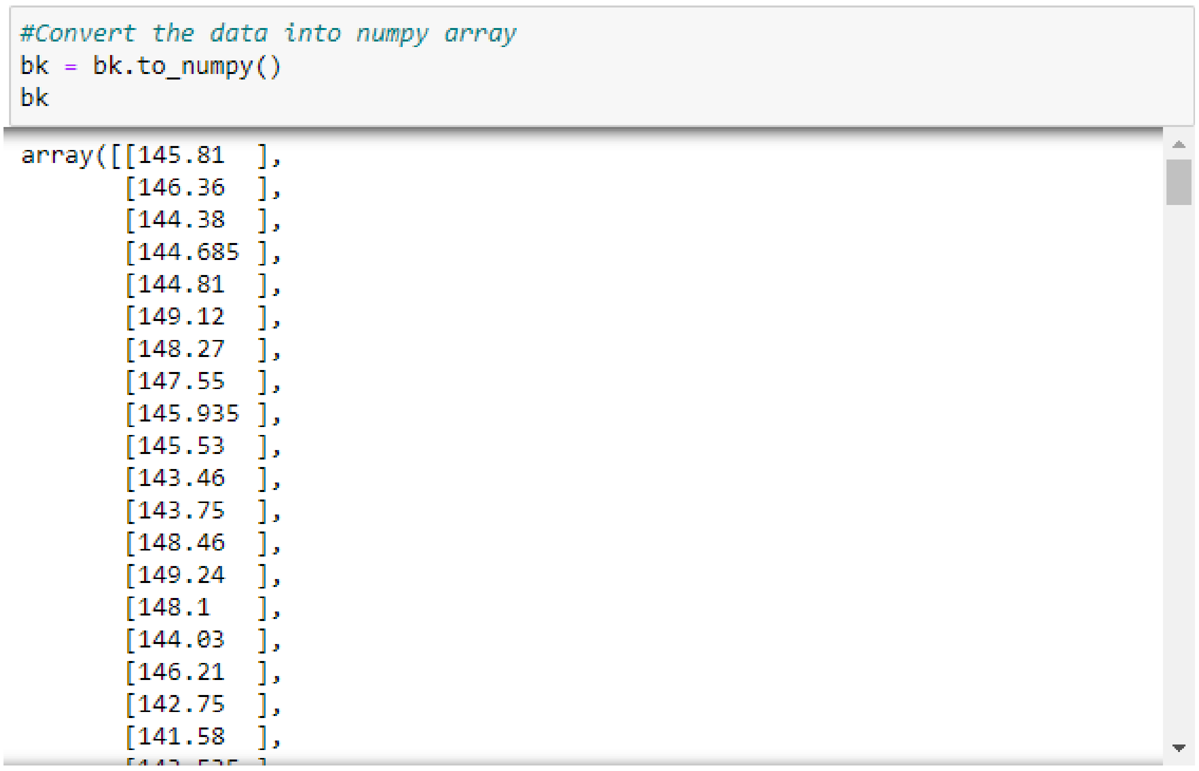


The above figures displayed the Apple stock market price as separate and together. Visually, I understand the Apple stock market Price very fluctuate and decreased from last year due to pandemic.

```
#Drop the high and low columns
bk.drop(['High', 'Low'], axis = 1, inplace = True)
#Drop the bottom 2 columns
bk.drop(bk.tail(2).index,inplace=True)
print(bk)
print(bk.shape)
```

```
         Open
0    145.8100
1    146.3600
2    144.3800
3    144.6850
4    144.8100
..        ...
245  110.4975
246  111.9688
247  112.6000
248  113.2050
249  110.4050

[250 rows x 1 columns]
(250, 1)
```

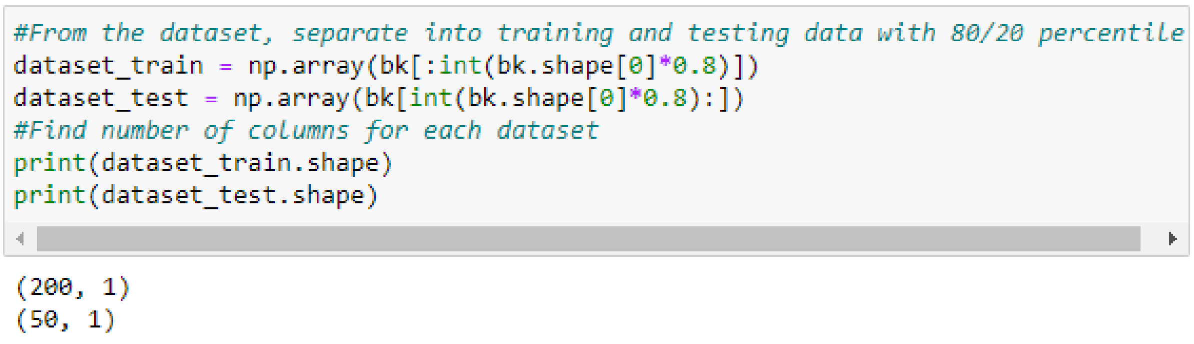
For convert the dataset into the numpy array I use below script code.



```
#Convert the data into numpy array
bk = bk.to_numpy()
bk
```

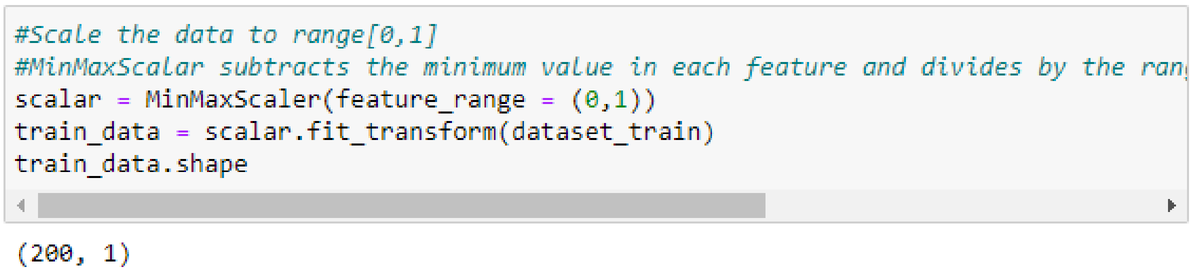
```
array([[145.81  ],
       [146.36  ],
       [144.38  ],
       [144.685 ],
       [144.81  ],
       [149.12  ],
       [148.27  ],
       [147.55  ],
       [145.935 ],
       [145.53  ],
       [143.46  ],
       [143.75  ],
       [148.46  ],
       [149.24  ],
       [148.1   ],
       [144.03  ],
       [146.21  ],
       [142.75  ],
       [141.58  ],
```

Now, I separate the train and test data using below code and displayed the output as well as identify the number of columns for each dataset



```
#From the dataset, separate into training and testing data with 80/20 percentile
dataset_train = np.array(bk[:int(bk.shape[0]*0.8)])
dataset_test = np.array(bk[int(bk.shape[0]*0.8):])
#Find number of columns for each dataset
print(dataset_train.shape)
print(dataset_test.shape)
```

```
(200, 1)
(50, 1)
```



```
#Scale the data to range[0,1]
#MinMaxScalar subtracts the minimum value in each feature and divides by the ran
scalar = MinMaxScaler(feature_range = (0,1))
train_data = scalar.fit_transform(dataset_train)
train_data.shape
```

```
(200, 1)
```

Now I scale the data to range [0,1]. Min scalar subtracts the minimum values in each frame and divides by range and after that declares the x_train and y_train on below code

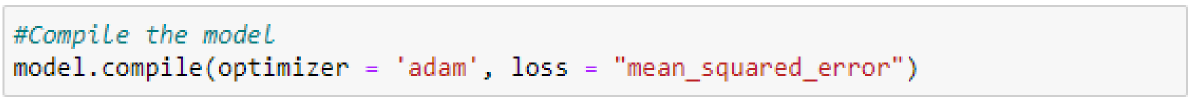
```
#Declare the x_train and y_train
x_train = train_data[0:98]
y_train = train_data[1:99]

x_train = np.reshape(x_train, (98,1,1))
x_train.shape
```

```
(98, 1, 1)
```

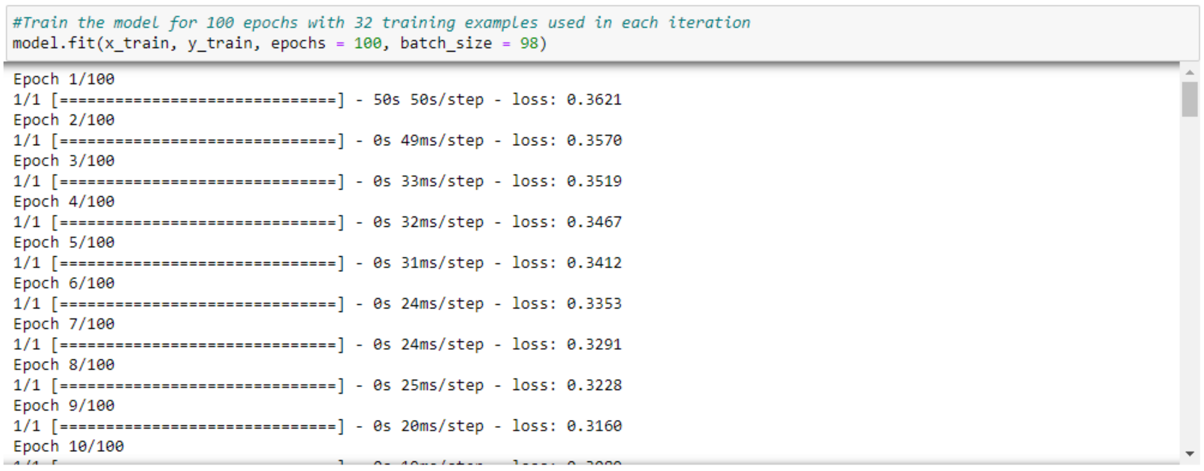
In this script, instantiate the sequential model class with the set the number of neurons/nodes with the units parameter as well as reflects and adding more layers

```
#Instantiate the Sequential model class
model = Sequential()
#Set the number of neurons/nodes with the units parameter
#Return_sequences=True reflects adding more layers
model.add(LSTM(units=98, return_sequences=True, input_shape=(None, 1)))
#Add dropout to reduce over-fitting
model.add(Dropout(0.2))
model.add(LSTM(units=98, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=98, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=98))
model.add(Dropout(0.2))
#Add dense layer to reduce spacial parameters of the vector
model.add(Dense(units=1))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, None, 98)          39200
_________________________________________________________________
dropout (Dropout)            (None, None, 98)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 98)          77224
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 98)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 98)          77224
_________________________________________________________________
dropout_2 (Dropout)          (None, None, 98)          0
_________________________________________________________________
lstm_3 (LSTM)                (None, 98)                77224
_________________________________________________________________
dropout_3 (Dropout)          (None, 98)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 99
=================================================================
Total params: 270,971
Trainable params: 270,971
Non-trainable params: 0
```

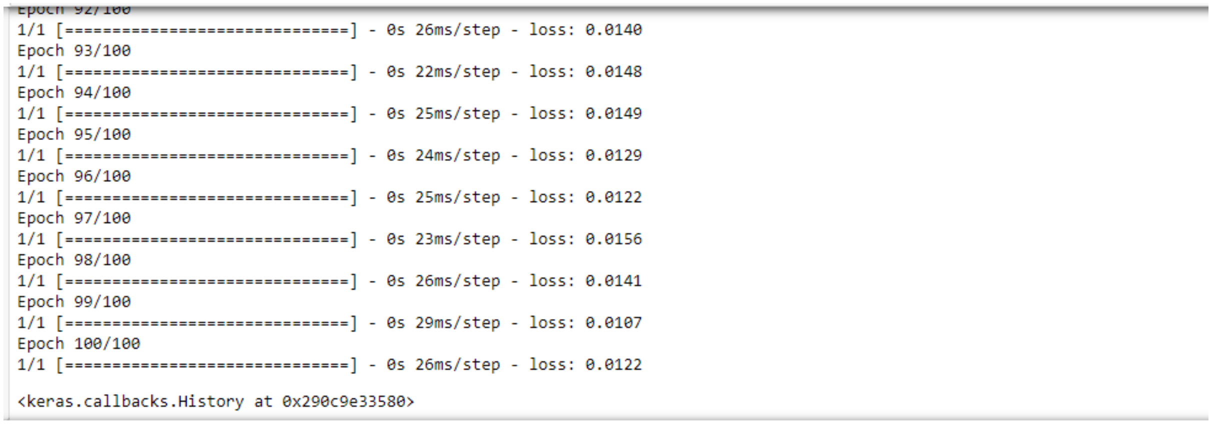


```
#Compile the model
model.compile(optimizer = 'adam', loss = "mean_squared_error")
```

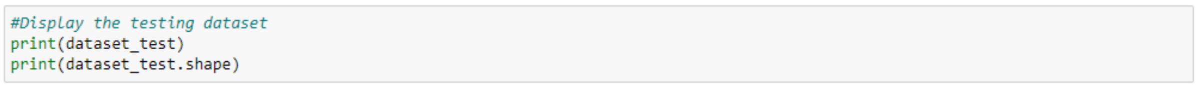


```
#Train the model for 100 epochs with 32 training examples used in each iteration
model.fit(x_train, y_train, epochs = 100, batch_size = 98)
```

```
Epoch 1/100
1/1 [==============================] - 50s 50s/step - loss: 0.3621
Epoch 2/100
1/1 [==============================] - 0s 49ms/step - loss: 0.3570
Epoch 3/100
1/1 [==============================] - 0s 33ms/step - loss: 0.3519
Epoch 4/100
1/1 [==============================] - 0s 32ms/step - loss: 0.3467
Epoch 5/100
1/1 [==============================] - 0s 31ms/step - loss: 0.3412
Epoch 6/100
1/1 [==============================] - 0s 24ms/step - loss: 0.3353
Epoch 7/100
1/1 [==============================] - 0s 24ms/step - loss: 0.3291
Epoch 8/100
1/1 [==============================] - 0s 25ms/step - loss: 0.3228
Epoch 9/100
1/1 [==============================] - 0s 20ms/step - loss: 0.3160
Epoch 10/100
```



```
Epoch 92/100
1/1 [==============================] - 0s 26ms/step - loss: 0.0140
Epoch 93/100
1/1 [==============================] - 0s 22ms/step - loss: 0.0148
Epoch 94/100
1/1 [==============================] - 0s 25ms/step - loss: 0.0149
Epoch 95/100
1/1 [==============================] - 0s 24ms/step - loss: 0.0129
Epoch 96/100
1/1 [==============================] - 0s 25ms/step - loss: 0.0122
Epoch 97/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0156
Epoch 98/100
1/1 [==============================] - 0s 26ms/step - loss: 0.0141
Epoch 99/100
1/1 [==============================] - 0s 29ms/step - loss: 0.0107
Epoch 100/100
1/1 [==============================] - 0s 26ms/step - loss: 0.0122

<keras.callbacks.History at 0x290c9e33580>
```



```
#Display the testing dataset
print(dataset_test)
print(dataset_test.shape)
```

```
[[118.72  ]
 [121.    ]
 [125.27  ]
 [120.06  ]
 [115.28  ]
 [116.25  ]
 [114.62  ]
 [115.7   ]
 [113.91  ]
 [112.89  ]
 [117.64  ]
 [113.79  ]
 [114.55  ]
 [115.01  ]
 [108.43  ]
 [105.17  ]
 [111.62  ]
 [112.68  ]
 [104.54  ]
 [110.4   ]
 [109.72  ]
 [115.23  ]
 [118.33  ]
 [114.72  ]
 [114.57  ]
 [120.36  ]
 [117.26  ]
 [113.95  ]
 [120.07  ]
 [126.91  ]
 [137.59  ]
 [132.76  ]
 [127.58  ]
 [126.0125]
 [127.1425]
 [126.1793]
 [124.6975]
 [128.6975]
 [119.2625]
 [115.75  ]
 [115.9833]
 [114.3525]
 [116.0625]
 [114.8288]
 [114.43  ]
 [110.4975]
 [111.9688]
 [112.6   ]
 [113.205 ]
 [110.405 ]]
(50, 1)
```

```
#Reshape the training data and scale
inputs = np.reshape(scalar.transform(dataset_test), (50,1,1))
#Reverse the scaled predictions to their original values
stock_prediction = scalar.inverse_transform(model.predict(inputs))
stock_prediction
```

Above code illustrate the reshape the training data and scale as well as reverse the scaled predictions to their original values. Below the outputs of the array.

```
array([[125.418274],
       [126.47971 ],
       [128.55153 ],
       [126.03836 ],
       [123.874664],
       [124.302956],
       [123.58635 ],
       [124.05945 ],
       [123.27897 ],
       [122.842415],
       [124.926216],
       [123.2273  ],
       [123.555916],
       [123.75642 ],
       [121.001945],
       [119.725365],
       [122.30706 ],
       [122.75327 ],
       [119.48519 ],
```

Now, we can use stock prediction using below code as well as the last figure displayed the current rate and predicted rate of the Apple stock using the dataset. As per the last figure, its clear that apple stock are fluctuated as well as after pick in the middle of the year again fall day by day sharply.

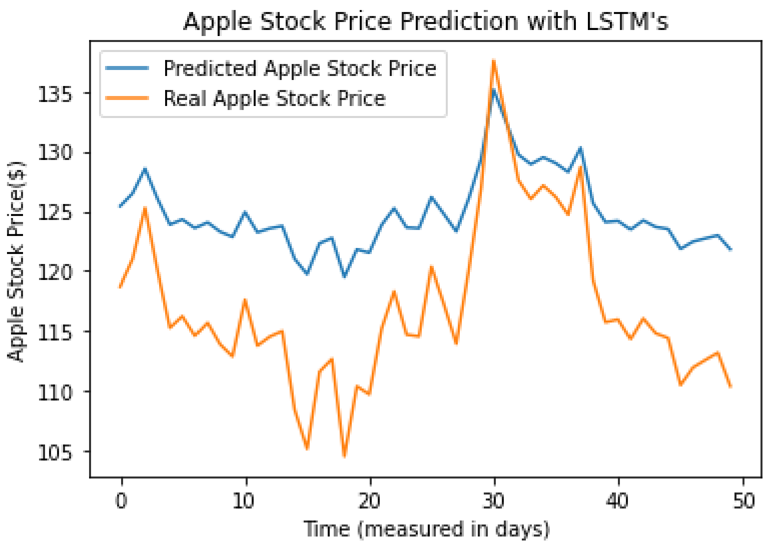
```
stock_prediction = np.squeeze(stock_prediction)
stock_prediction
```

```
array([125.418274, 126.47971 , 128.55153 , 126.03836 , 123.874664,
       124.302956, 123.58635 , 124.05945 , 123.27897 , 122.842415,
       124.926216, 123.2273  , 123.555916, 123.75642 , 121.001945,
       119.725365, 122.30706 , 122.75327 , 119.48519 , 121.80129 ,
       121.522964, 123.85274 , 125.2398  , 123.629875, 123.564606,
       126.17864 , 124.754715, 123.29621 , 126.04302 , 129.3768  ,
       135.16115 , 132.45644 , 129.71872 , 128.92313 , 129.49513 ,
       129.00706 , 128.26733 , 130.29518 , 125.66803 , 124.08151 ,
       124.18466 , 123.4702  , 124.21974 , 123.67729 , 123.50381 ,
       121.84141 , 122.45319 , 122.71936 , 122.9766  , 121.803345],
      dtype=float32)
```

```
stock_prediction = stock_prediction.reshape(-1,1)
stock_prediction
```

```
array([[125.418274],
       [126.47971 ],
       [128.55153 ],
       [126.03836 ],
       [123.874664],
       [124.302956],
       [123.58635 ],
       [124.05945 ],
       [123.27897 ],
       [122.842415],
       [124.926216],
       [123.2273  ],
       [123.555916],
       [123.75642 ],
       [121.001945],
       [119.725365],
       [122.30706 ],
       [122.75327 ],
       [119.48519 ],
       [121.80129 ],
       [121.522964],
       [123.85274 ],
       [125.2398  ],
       [123.629875],
       [123.564606],
       [126.17864 ],
       [124.754715],
       [123.29621 ],
       [126.04302 ],
       [129.3768  ],
       [135.16115 ],
       [132.45644 ],
       [129.71872 ],
       [128.92313 ],
       [129.49513 ],
       [129.00706 ],
       [128.26733 ],
       [130.29518 ],
       [125.66803 ],
       [124.08151 ],
       [124.18466 ],
       [123.4702  ],
       [124.21974 ],
       [123.67729 ],
       [123.50381 ],
       [121.84141 ],
       [122.45319 ],
       [122.71936 ],
       [122.9766  ],
       [121.803345]], dtype=float32)
```

```
#Graph the real stock prices against the model's prediction
plt.plot(stock_prediction, label = 'Predicted Apple Stock Price', linewidth = 1.5)
plt.plot(dataset_test, label = 'Real Apple Stock Price', linewidth = 1.5)
plt.title("Apple Stock Price Prediction with LSTM's")
plt.xlabel('Time (measured in days)')
plt.ylabel('Apple Stock Price($)')
plt.legend()
plt.show()
```



REFERENCE

- https://www.educative.io/edpresso/data-normalization-in-python
- Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/
- Blackboard: Course work material
- CW1 DATA MINING AND MACHINE LEARNING: BY BHAUMIKKUMAR PATEL