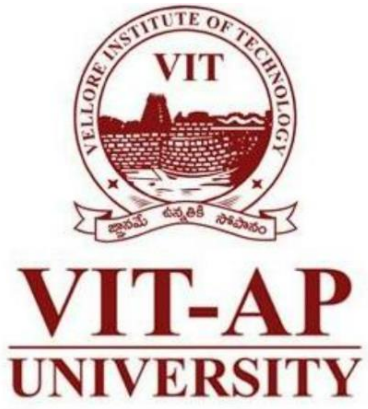
DATAMINING AND DATAWAREHOUSING

PROJECT REPORT ON

COMPARITIVE ANALYSIS AND VISUALIZATION OF STOCK MARKET PREDICTION MODELS



SUBMITTED BY:

B. Sahithi [22MSD7008]

P. Krishna Priya[22MSD7009]

N. Sri Vaishnavi [22MSD7050]

V. Jyothi Priya[22MSD7036]

SUBMITTED TO:

DR. AVADHESH KUMAR

ASSISTANT PROFESSOR

SCHOOL OF ADVANCED SCIENCES

CONTRIBUTION:

NAME	WORK ASSIGNED
1.B. SAHITHI	LSTM, STOCKMARKET, PREDICTION, ASSOCIATION RULE MINING
2.P. KRISHNA PRIYA	ANN, LINEAR REGRESSION, DECISION TREE
3.N. SRIVAISHNAVI	CNN, SKEWNESS ANALYSIS, CONFUSION MATRIX
5.V. JYOTHI PRIYA	CORRELATION, KMEANS CLUSTERING

INDEX

ABSTRACT

Accurate prediction of stock market trends is crucial for investors and financial institutions. This research presents a comprehensive analysis of different predictive models for stock market forecasting, along with the application of various data analysis techniques and visualizations.

The study begins by comparing the accuracy of linear regression, artificial neural networks (ANN), and convolutional neural networks (CNN) models in predicting stock market trends. Historical stock market data, including price, volume, and other relevant features, is used to train and evaluate these models. The comparative analysis identifies the most effective model for stock market prediction.

Additionally, the research incorporates time series analysis using the long short-term memory (LSTM) model, a type of recurrent neural network. The LSTM model is compared with ANN and CNN models in terms of their predictive accuracy for stock market forecasting. The analysis also includes statistical techniques such as correlation analysis, skewness analysis, k-means clustering, decision tree algorithms, and association rule mining.

To gain insights into the relationships between different stock market variables, correlation analysis is performed. Skewness analysis helps determine the distribution characteristics of the dataset. K-means clustering is applied to segment stocks based on similar features. Decision tree algorithms are employed to identify important features for prediction. Association rule mining is conducted to discover patterns and relationships among stock market variables.

Moreover, the study emphasizes the importance of visualizing stock market time series data. Line plots are utilized to visualize stock price trends over time, enabling the identification of upward or downward trends. Heat maps are generated to visualize the relationships and patterns among different stock market variables, providing a comprehensive overview of correlations and dependencies.

The results of this research contribute to the field of stock market prediction by providing insights into the accuracy of different models and the application of various data analysis techniques. The visualizations of line plots and heat maps offer intuitive representations of stock market trends and relationships among variables, facilitating informed decision-making processes for investors and financial institutions.

Keywords: stock market prediction, linear regression, artificial neural networks, CNN, LSTM, correlation analysis, skewness analysis, k-means clustering, decision tree, association rule mining, line plots, heat maps.

INTRODUCTION:

The stock market plays a vital role in the global economy, serving as a platform for companies to raise capital and for investors to participate in wealth creation. The ability to accurately predict stock market trends is of great interest to investors, financial institutions, and researchers alike. Accurate predictions can lead to informed investment decisions, risk mitigation, and potential financial gains.

Over the years, various predictive models have been developed and applied to stock market forecasting. Linear regression, artificial neural networks (ANN), and convolutional neural networks (CNN) are commonly used models in this domain. These models leverage historical stock market data and relevant features to make predictions about future trends. However, the choice of the most effective model depends on the specific characteristics of the dataset and the problem at hand.

In recent years, time series analysis has gained prominence in stock market prediction. Time series models, such as the long short-term memory (LSTM) model, are designed to capture temporal dependencies and patterns in sequential data. These models have shown promise in accurately forecasting stock market trends, taking into account the sequential nature of stock prices and other related variables.

In addition to predictive modelling, various data analysis techniques can provide further insights into stock market behaviour. Correlation analysis helps identify the relationships between different stock market variables, enabling the understanding of interdependencies and potential influencing factors. Skewness analysis provides information about the distribution characteristics of the dataset, aiding in understanding the underlying patterns. Clustering techniques, such as k-means clustering, can group stocks with similar characteristics, facilitating portfolio diversification and risk management. Decision tree algorithms can identify key features that contribute to stock market prediction, offering interpretability and insights into the driving factors. Association rule mining can uncover interesting patterns and relationships among stock market variables, revealing hidden dependencies and market dynamics.

Moreover, the evaluation of predictive models goes beyond accuracy metrics. The Receiver Operating Characteristic (ROC) curve and the confusion matrix are essential tools for assessing models that involve classification tasks. The ROC curve provides insights into the trade-off between true positive rate and false positive rate, allowing researchers to set an appropriate threshold for decision-making. The confusion matrix further analyzes the performance of the models by presenting the counts of true positives, true negatives, false positives, and false negatives.

Visualizations, such as line plots and heat maps, play a crucial role in understanding and communicating stock market trends. Line plots showcase the historical movements of stock prices, allowing for the identification of trends, seasonality, and potential turning points. Heat maps provide a graphical representation of the relationships and dependencies between different stock market variables, highlighting areas of strength or weakness.

This research aims to provide a comprehensive analysis of stock market prediction by comparing the accuracy of linear regression, ANN, CNN, and LSTM models. It also explores various data analysis techniques, including correlation analysis, skewness analysis, clustering, decision trees, and association rule mining, to gain deeper insights into stock market dynamics. Furthermore, the research emphasizes the importance of visualizations, employing line plots and heat maps to represent stock market trends and patterns.

By combining these approaches, this research aims to contribute to the understanding of stock market behavior and provide valuable tools and insights for investors, financial institutions, and researchers involved in stock market prediction and analysis.

CLASSIFICATION:

Stock Market Prediction:

Classification can be used to predict the direction of stock price movements, such as whether the stock will increase or decrease in value. Historical stock market data can be used as input features, and the target variable would be the binary classification of price movement (e.g., "up" or "down"). Algorithms like logistic regression, decision trees, random forest, SVM, or neural networks can be trained on historical data to classify stocks into these categories and make predictions for future instances.

Correlation Analysis:

Classification can be used to categorize the degree of correlation between different stock market variables. For example, stocks can be classified into categories such as strongly positively correlated, weakly negatively correlated, or not correlated at all. This classification can help identify relationships and dependencies among variables, assisting in portfolio diversification or risk management.

Skewness Analysis:

Skewness analysis typically involves assessing the distribution characteristics of a dataset. While classification may not be directly applicable to skewness analysis, it can be used in combination with other techniques. For example, stocks can be classified into categories based on their skewness values, such as positively skewed, negatively skewed, or approximately symmetric. This classification can aid in understanding the distribution patterns of stock market variables.

K-means Clustering:

K-means clustering is an unsupervised machine learning technique used to group similar data points together. In the context of stock market data, classification can be applied to cluster stocks into different groups based on their features or characteristics. This classification can assist in portfolio management and asset allocation by identifying stocks with similar behavior or properties.

Decision Tree:

Decision tree algorithms can be utilized for classification tasks within the mentioned topics. For instance, decision trees can be constructed to classify stock market variables or patterns based on different attributes, such as identifying specific market conditions associated with particular price movements or trends.

Association Rule Mining:

Association rule mining primarily focuses on discovering patterns or relationships among variables rather than classification. However, classification techniques can be used in combination with association rule mining. For example, classification can be applied to categorize stocks into groups based on their association rule patterns, such as stocks that frequently co-occur or exhibit specific relationships.

CNN (Convolutional Neural Network):

CNN is a deep learning model commonly used for image recognition and processing. However, in the context of stock market prediction, CNN can be employed to analyze and extract features from time series data. For instance, if the stock market data includes visual representations (e.g., candlestick charts), CNN can be used to learn patterns and features that contribute to predicting stock price movements.

ANN (Artificial Neural Network):

ANN is a versatile and widely used machine learning model. It consists of interconnected nodes (neurons) organized in layers, allowing it to learn complex patterns and relationships in data. In stock market prediction, ANN can be used to capture nonlinear dependencies between stock market variables and predict future stock prices or price movements based on historical data and relevant features.

LSTM (Long Short-Term Memory):

LSTM is a type of recurrent neural network (RNN) that excels in modelling sequential data, making it suitable for time series analysis. LSTM models have memory cells that can store information over time, enabling them to capture long-term dependencies. In stock market prediction, LSTM can leverage the sequential nature of stock market data to forecast future prices or predict trends based on past price movements, volume, and other relevant features.

Linear Regression:

Linear Regression is a traditional statistical modelling technique used to establish a linear relationship between dependent and independent variables. While it may not capture complex nonlinear relationships, Linear Regression can still be applied to stock market prediction. It can be used to model the relationship between stock prices and selected features, assuming a linear trend, and predict future stock prices based on historical data.

Confusion matrix:

The confusion matrix is typically represented as a table with rows and columns. Each row corresponds to the actual class labels, while each column represents the predicted class labels. The entries in the matrix indicate the counts or proportions of observations that fall into each category.

METHODOLOGY:

IMPORTING ALL THE REQUIRED LIBRARIES

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

IMPORTING DATASET

```
data = pd.read_csv("/content/prices-split-adjusted.csv")
df = pd.DataFrame(data)
df
```

OUTPUT:

	date	symbol	open	close	low	high	volume
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0
...	...	...	...	...	...	...	...
34252	2010-04-19	NFX	52.310001	52.610001	51.299999	52.730000	1285400.0
34253	2010-04-19	NI	6.392928	6.440079	6.357564	6.447937	9150500.0
34254	2010-04-19	NKE	18.737499	18.895000	18.562500	18.950001	8790400.0
34255	2010-04-19	NOC	59.342048	60.010640	59.269766	60.082917	1492300.0

DISPLAYING THE 5 ROWS OF THE DATASET

```
df.head()
```

	date	symbol	open	close	low	high	volume
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

DESCRIPTIVE STATISTICS FOR THE DATASET

```
df.describe()
```

	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	64.993618	65.011913	64.336541	65.639748	5.415113e+06
std	75.203893	75.201216	74.459518	75.906861	1.249468e+07
min	1.660000	1.590000	1.500000	1.810000	0.000000e+00
25%	31.270000	31.292776	30.940001	31.620001	1.221500e+06
50%	48.459999	48.480000	47.970001	48.959999	2.476250e+06
75%	75.120003	75.139999	74.400002	75.849998	5.222500e+06
max	1584.439941	1578.130005	1549.939941	1600.930054	8.596434e+08

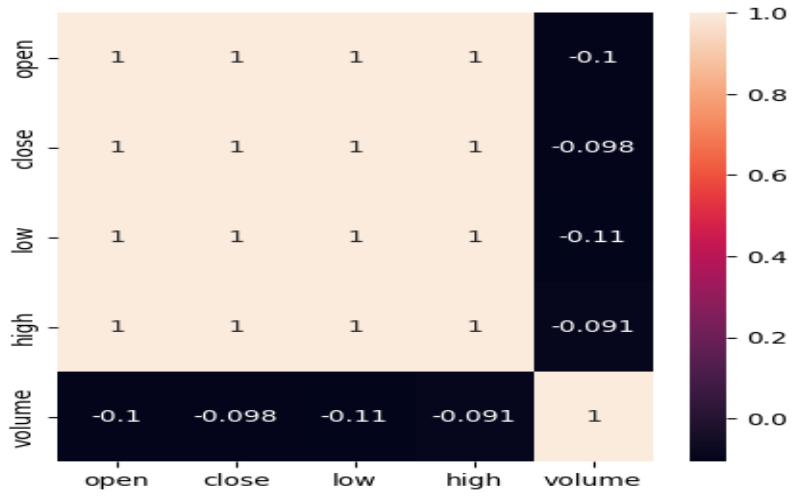
COLUMN WISE NULL VALUES

```
# showing column wise %ge of NaN values they contains
for i in df.columns:
  print(i,"\t-\t", df[i].isna().mean()*100)
```

```
date    -       0.0
symbol  -       0.0
open    -       0.0
close   -       0.0
low     -       0.0
high    -       0.0
volume  -       0.0
```

APPLYING THE CORRELATION FOR THE DATASET

```
df = df[df['symbol']=='AAP']   # Choosin stock values for any company
cormap = df.corr()
fig, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cormap, annot = True)
```




```
def get_corelated_col(cor_dat, threshold):
  # Cor_data to be column along which corelation to be measured
  #Threshold be the value above which of corelation to considered
  feature=[]
  value=[]
 for i ,index in enumerate(cor_dat.index):
    if abs(cor_dat[index]) > threshold:
      feature.append(index)
      value.append(cor_dat[index])
 df = pd.DataFrame(data = value, index = feature, columns=['corr value'])
  return df
```

CORRELATION VALUES OF THE COLUMNS

```
top_corelated_values = get_corelated_col(cormap['close'], 0.60)
top_corelated_values
```

	corr value
open	0.999382
close	1.000000
low	0.999615
high	0.999737

INDEXING THE TOP-CORRELATED VALUES

```
df = df[top_corelated_values.index]
df
```

	open	close	low	high
253	40.700001	40.380001	40.360001	41.040001
720	40.299999	40.139999	39.720001	40.310001
1188	40.049999	40.490002	40.049999	40.779999
1656	39.549999	40.480000	39.549999	40.540001
2124	40.250000	40.639999	40.110001	40.820000
...	...	...	...	...
848766	170.690002	170.889999	170.000000	172.080002

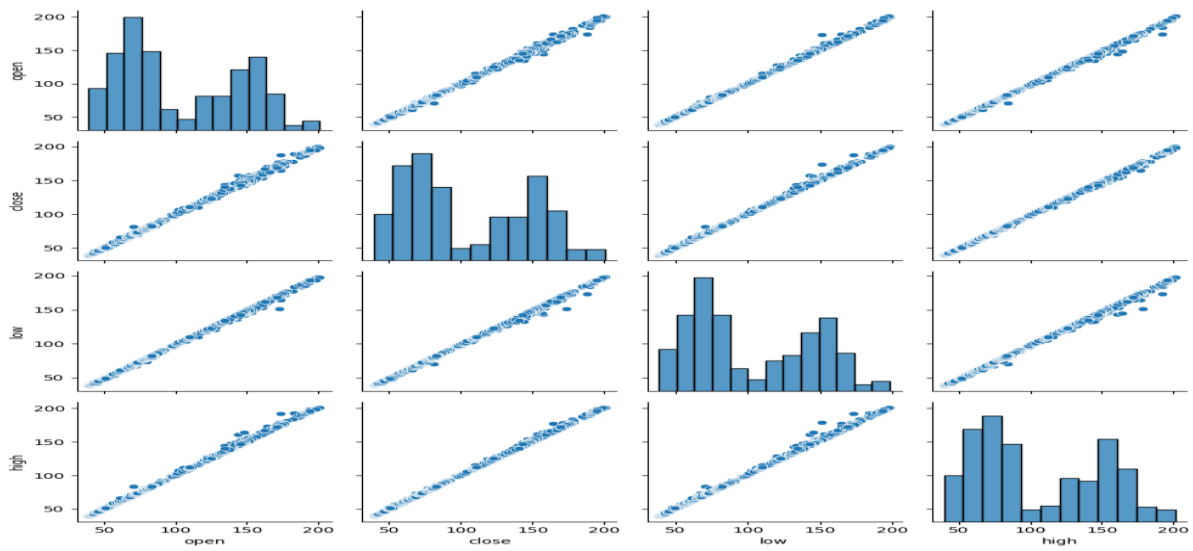
NOW WE TAKEN THE TOP CORRELATED VALUES AND SHOWING HOW MANY ROWS AND COLUMNS ARE THERE

```
df.shape
```

(1762, 4)

PAIRPLOT

```
sns.pairplot(df)
plt.tight_layout()
```



DROP THE CLOSE COLUMN:

```
X = df.drop(['close'], axis=1)
```

```
y = df['close']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

	open	low	high
0	0.012001	0.012392	0.010256
1	0.009539	0.008387	0.005746
2	0.008000	0.010452	0.008649
3	0.004923	0.007323	0.007167
4	0.009231	0.010827	0.008897

THE CODE SHOWS THAT WE ARE SPLITTING THE DATA 20PER IS FOR TESTING AND 80PER FOR TRAINING

```
#now lets split data in test train pairs
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle=False)
Acc = []
```

LINEAR REGRESSION MODEL

```
from sklearn.linear_model import LinearRegression
# model training
model_1 = LinearRegression()
model_1.fit(X_train, y_train)
```

☑ LinearRegression

```
LinearRegression()
```

APPLYING THE LINEAR REGRESSION MODEL AND PREDICTING VALUES

```
# prediction
y_pred_1 = model_1.predict(X_test)
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_1})
pred_df.head()
```

ACCURACY OF APPLYING THE LINEAR REGRESSION MODEL

```
# Measure the Accuracy Score
from sklearn.metrics import r2_score
```

	Actual	Predicted
675111	173.660004	173.682489
675608	171.919998	172.593759
676105	172.000000	171.182789
676602	187.789993	187.980305
677099	187.029999	188.440838

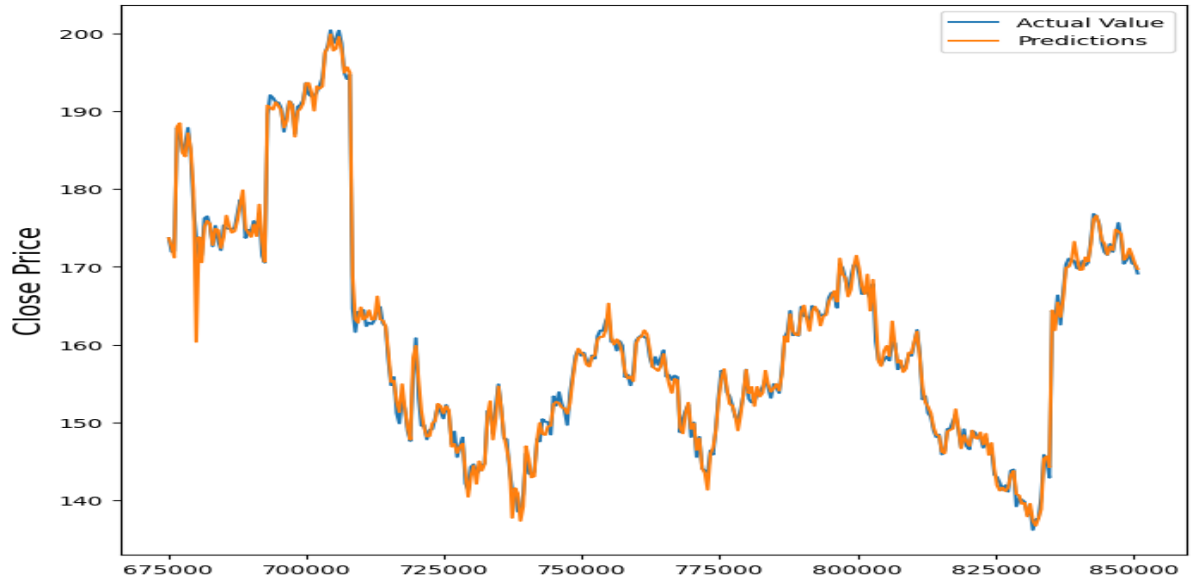
```
print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_1)))
Acc.append(r2_score(y_test, y_pred_1))
```

OUTPUT:

Accuracy score of the predictions: 0.9931342019332019

LINEPLOT-SEEING THE DIFFERENCE OF ACTUAL VALUE AND PREDICTED VALUE IN THE GRAPH

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



ANN-ARTIFICIAL NEURAL NETWORKS

```
# Model Creation
from keras.models import Sequential
from keras.layers import Dense
def regressor(inp_dim):
  model = Sequential()
   model.add(Dense(20, input_dim=inp_dim, kernel_initializer='normal', activation='relu'))
   model.add(Dense(25, kernel_initializer='normal', activation='relu'))
   model.add(Dense(10, kernel_initializer='normal', activation='relu'))
   model.add(Dense(1, kernel_initializer='normal'))
   model.compile(loss='mean_squared_error', optimizer='adam')
   return model
```

TRAINING THE MODEL

```
# Model Training
model_2 = regressor(inp_dim=3)
model_2.fit(X_train, y_train, epochs=70, validation_split=0.2)
```

OUTPUT:

```
Epoch 61/70
36/36 [==============================] - 0s 7ms/step - loss: 0.4619 - val_loss: 2.0303
Epoch 62/70
36/36 [==============================] - 0s 7ms/step - loss: 0.4546 - val_loss: 1.9088
Epoch 63/70
36/36 [==============================] - 0s 7ms/step - loss: 0.4519 - val_loss: 1.8724
Epoch 64/70
36/36 [==============================] - 0s 6ms/step - loss: 0.4510 - val_loss: 1.9235
Epoch 65/70
36/36 [==============================] - 0s 6ms/step - loss: 0.4500 - val_loss: 1.8282
Epoch 66/70
36/36 [==============================] - 0s 6ms/step - loss: 0.4483 - val_loss: 1.8761
Epoch 67/70
36/36 [==============================] - 0s 7ms/step - loss: 0.4466 - val_loss: 1.7902
Epoch 68/70
36/36 [==============================] - 0s 8ms/step - loss: 0.4473 - val_loss: 1.8306
Epoch 69/70
36/36 [==============================] - 0s 7ms/step - loss: 0.4459 - val_loss: 1.8832
Epoch 70/70
36/36 [==============================] - 0s 6ms/step - loss: 0.4487 - val_loss: 1.7329
<keras.callbacks.History at 0x7f64e1a4bd00>
```

```
# Prediction
y_pred_3 = model_3.predict(X_test)
```

OUTPUT:

```
12/12 [==============================] - 0s 3ms/step
```

USING THE ANN MODEL SEE THE DIFFERENCE BETWEEN THE ACTUAL AND PREDICTED VALUES

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_3.flatten()})
pred_df.head()
```

	Actual	Predicted
675111	173.660004	174.225983
675608	171.919998	172.585266
676105	172.000000	170.722504
676602	187.789993	178.878448
677099	187.029999	188.398682

ACCURACY OF ANN MODEL

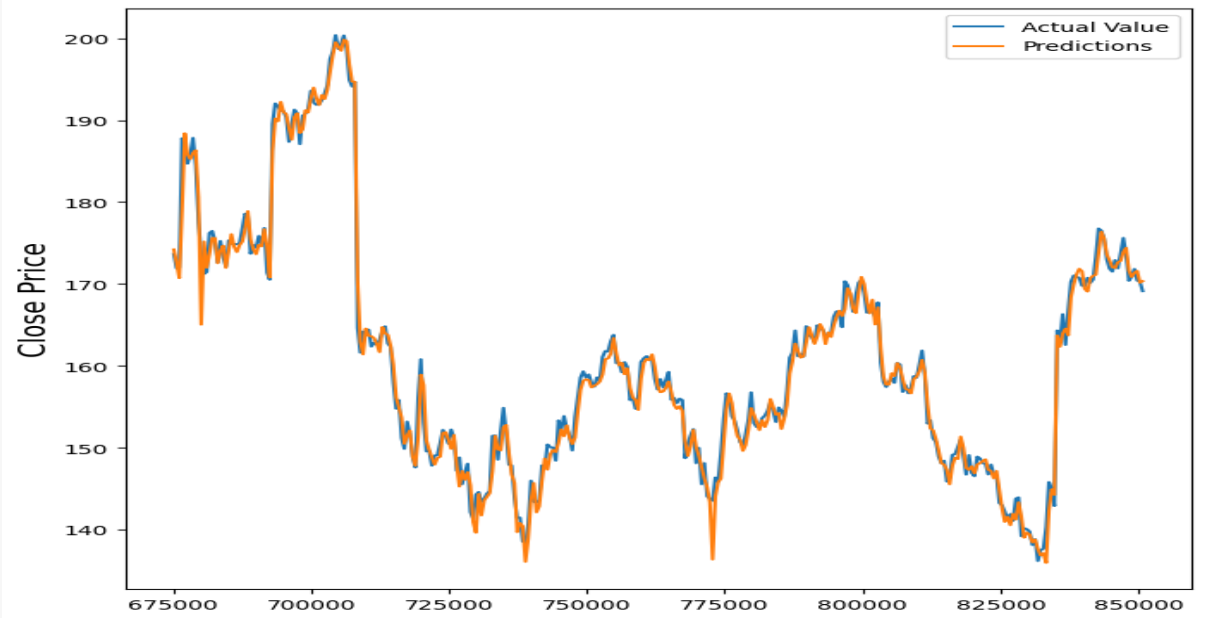
```
# Measure the Accuracy Score
from sklearn.metrics import r2_score
 print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_3)))
Acc.append(r2_score(y_test, y_pred_3))
```

OUTPUT:

```
Accuracy score of the predictions: 0.9874831939742366
```

LINEPLOT-SEEING THE DIFFERENCE OF ACTUAL VALUE AND PREDICTED VALUE IN THE GRAPH

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



CNN-CONVOLUTIONAL NEURAL NETWORKS

```
X_train = np.array(X_train).reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = np.array(X_test).reshape(X_test.shape[0], X_test.shape[1], 1)
from tensorflow.keras import Sequential,utils
from tensorflow.keras.layers import Flatten, Dense, Conv1D, MaxPool1D, Dropout
def reg():
    model = Sequential()
    model.add(Conv1D(32, kernel_size=(3,), padding='same', activation='relu', input_shape =
(X_train.shape[1],1)))
    model.add(Conv1D(64, kernel_size=(3,), padding='same', activation='relu'))
    model.add(Conv1D(128, kernel_size=(5,), padding='same', activation='relu'))
     model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(units = 1))
    model.compile(loss='mean_squared_error', optimizer='adam')
     return model
```

TRAINING THE MODEL

```
# Model Training
model_3 = reg()
model_3.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

OUTPUT:

```
Epoch 91/100
36/36 [==============================] - 0s 6ms/step - loss: 0.4594 - val_loss: 2.9394
Epoch 92/100
36/36 [==============================] - 0s 6ms/step - loss: 0.6381 - val_loss: 1.7132
Epoch 93/100
36/36 [==============================] - 0s 6ms/step - loss: 0.5624 - val_loss: 2.1760
Epoch 94/100
36/36 [==============================] - 0s 7ms/step - loss: 0.5152 - val_loss: 1.7497
Epoch 95/100
36/36 [==============================] - 0s 7ms/step - loss: 0.5365 - val_loss: 2.0970
Epoch 96/100
36/36 [==============================] - 0s 6ms/step - loss: 0.4975 - val_loss: 1.9343
Epoch 97/100
36/36 [==============================] - 0s 5ms/step - loss: 0.4640 - val_loss: 2.5313
Epoch 98/100
36/36 [==============================] - 0s 6ms/step - loss: 0.4970 - val_loss: 2.1054
Epoch 99/100
36/36 [==============================] - 0s 6ms/step - loss: 0.5051 - val_loss: 1.8325
Epoch 100/100
36/36 [==============================] - 0s 7ms/step - loss: 0.6213 - val_loss: 2.3513
<keras.callbacks.History at 0x7f397fd69780>
```

PREDICTING THE TARGET VALUES

```
# Prediction
y_pred_3 = model_3.predict(X_test)
```

OUTPUT:

```
12/12 [==============================] - 0s 2ms/step
```

USING THE CNN MODEL SEE THE DIFFERENCE BETWEEN THE ACTUAL AND PREDICTED VALUES

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_3.flatten()})
pred_df.head()
```

	Actual	Predicted
675111	173.660004	173.501434
675608	171.919998	171.860184
676105	172.000000	170.040131
676602	187.789993	178.271545
677099	187.029999	187.505493

ACCURACY OF CNN MODEL

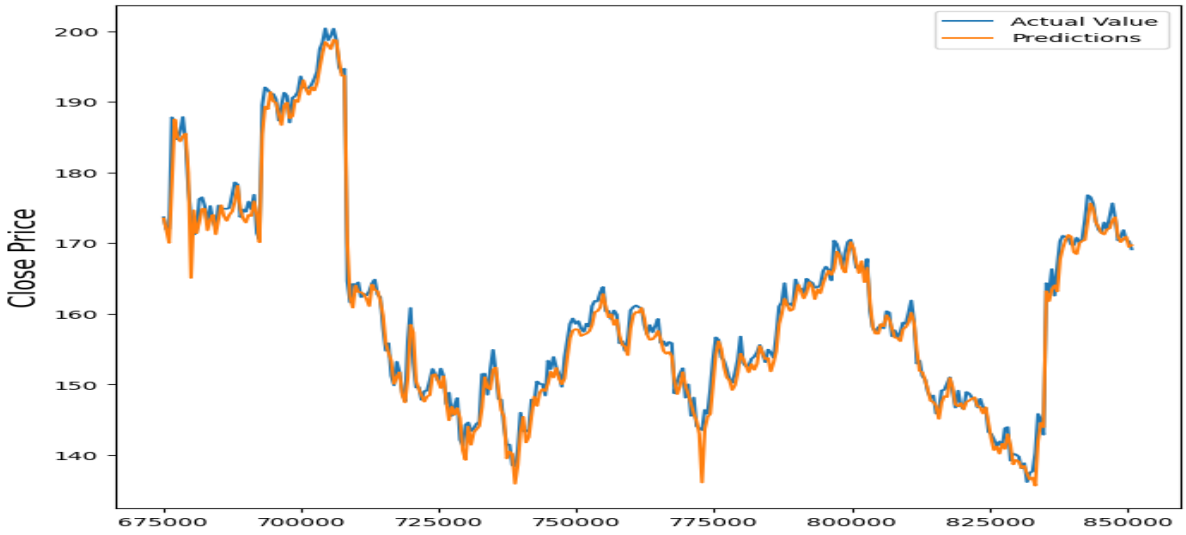
```
# Measure the Accuracy Score
from sklearn.metrics import r2_score
 print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_3)))
Acc.append(r2_score(y_test, y_pred_3))
```

OUTPUT:

```
Accuracy score of the predictions: 0.9844685276424856
```

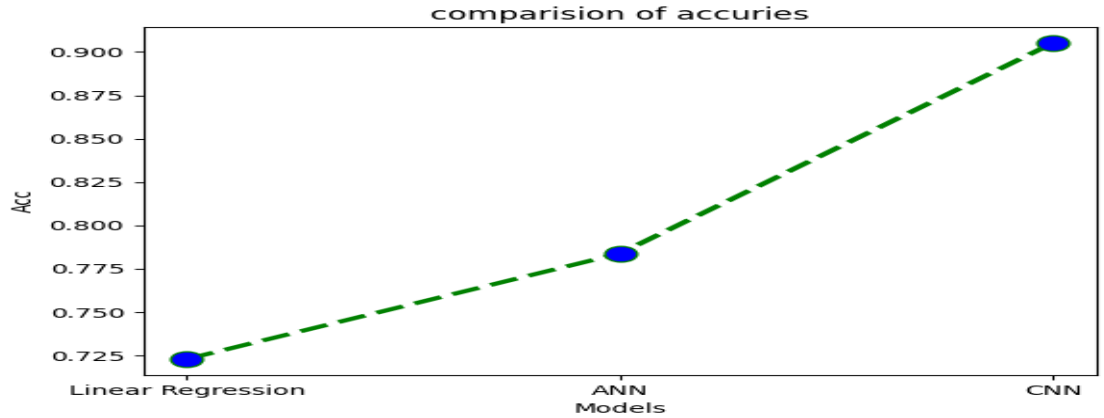
LINEPLOT-SEEING THE DIFFERENCE OF ACTUAL VALUE AND PREDICTED VALUE IN THE GRAPH

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



COMPARISION OF ACCURACIES

```
plt.plot(range(3), Acc, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)
plt.ylabel('Acc')
plt.xlabel('Models')
plt.title("comparision of accuries")
plt.xticks(range(3), ['Linear Regression', 'ANN', 'CNN'])
plt.show()
```



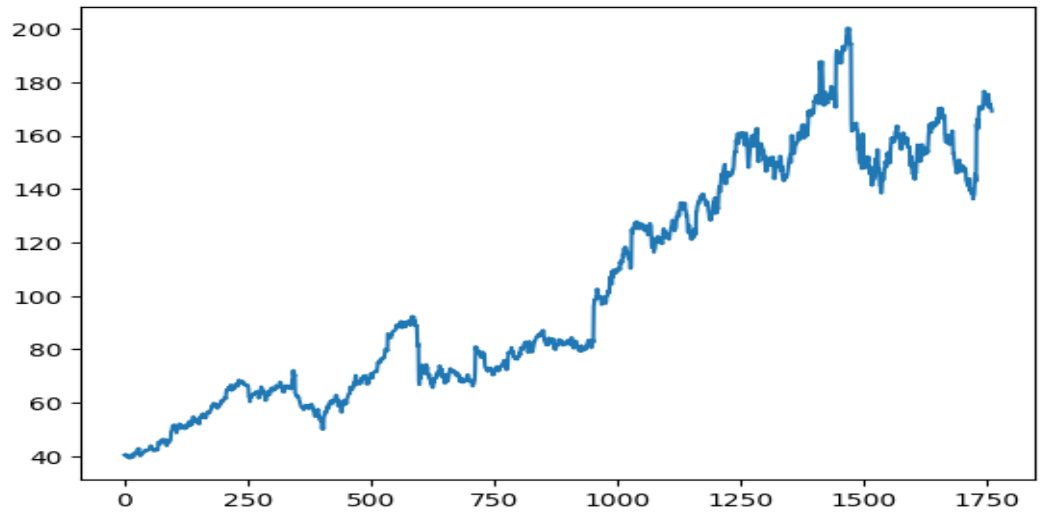
NOW CONVERTING THE DATA INTO NORMAL FOR APPLYING THE TIMESERIES MODELS

```
close = df.reset_index()['close']
close.head()
```

```
0    40.380001
1    40.139999
2    40.490002
3    40.480000
4    40.639999
Name: close, dtype: float64
```

PLOTTING THE CLOSE COLUMN

```
plt.plot(close)
plt.show()
```



CODE IMPLEMENTS A SLIDING WINDOW APPROACH TO CREATE INPUT-OUTPUT PAIRS FOR A TIME SERIES FORECASTING TASK

```
time_step = 30
X, y = [], []
for i in range(len(close)-time_step-1):
    X.append(close[i:(i+time_step)])
    y.append(close[(i+time_step)])
X = np.array(X)
y = np.array(y)
```

THE FIRST FIVE ELEMENTS OF THE NUMPY ARRAY X,WHICH CONTAINS THE INPUT SEQUENCES CREATED USING THE SLIDING WINDOW APPROACH.

```
X[:5]
```

```
array([[40.380001, 40.139999, 40.490002, 40.48    , 40.639999, 40.240002,
        39.540001, 40.09    , 39.560001, 39.310001, 39.5     , 39.16    ,
        39.23    , 39.740002, 40.5     , 40.549999, 40.59    , 39.77    ,
        39.450001, 40.490002, 41.189999, 41.189999, 40.93    , 40.720001,
        40.810001, 41.57    , 42.330002, 42.549999, 42.810001, 42.630001],
       [40.139999, 40.490002, 40.48    , 40.639999, 40.240002, 39.540001,
        40.09    , 39.560001, 39.310001, 39.5     , 39.16    , 39.23    ,
        39.740002, 40.5     , 40.549999, 40.59    , 39.77    , 39.450001,
        40.490002, 41.189999, 41.189999, 40.93    , 40.720001, 40.810001,
        41.57    , 42.330002, 42.549999, 42.810001, 42.630001, 42.880001],
       [40.490002, 40.48    , 40.639999, 40.240002, 39.540001, 40.09    ,
        39.560001, 39.310001, 39.5     , 39.16    , 39.23    , 39.740002,
        40.5     , 40.549999, 40.59    , 39.77    , 39.450001, 40.490002,
        41.189999, 41.189999, 40.93    , 40.720001, 40.810001, 41.57    ,
        42.330002, 42.549999, 42.810001, 42.630001, 42.880001, 40.150002],
       [40.48    , 40.639999, 40.240002, 39.540001, 40.09    , 39.560001,
        39.310001, 39.5     , 39.16    , 39.23    , 39.740002, 40.5     ,
        40.549999, 40.59    , 39.77    , 39.450001, 40.490002, 41.189999,
        41.189999, 40.93    , 40.720001, 40.810001, 41.57    , 42.330002,
        42.549999, 42.810001, 42.630001, 42.880001, 40.150002, 40.      ],
       [40.639999, 40.240002, 39.540001, 40.09    , 39.560001, 39.310001,
        39.5     , 39.16    , 39.23    , 39.740002, 40.5     , 40.549999,
        40.59    , 39.77    , 39.450001, 40.490002, 41.189999, 41.189999,
        40.93    , 40.720001, 40.810001, 41.57    , 42.330002, 42.549999,
        42.810001, 42.630001, 42.880001, 40.150002, 40.      , 40.240002]])
```

```
y[:5]
```

OUTPUT:

array([42.880001, 40.150002, 40. , 40.240002, 40.220001])

THE RESULTING OUTPUT IS A DATAFRAME CONTAINING THE NORMALIZED VALUES OF X,WHERE EACH COLUMN REPRESENTS A FEATURE OR VARIABLE

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X).head()
```

	0	1	2	3	4	5	6	7	8	9	...	20	
0	0.007567	0.006079	0.008250	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	...	0.007420	0.007
1	0.006079	0.008250	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	...	0.007420	0.005
2	0.008250	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	0.000000	...	0.005799	0.004
3	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	0.000000	0.000434	...	0.004489	0.005
4	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	0.000000	0.000434	0.003598	...	0.005051	0.009

5 rows × 30 columns

SPLITTING THE DATA TO TRAINING AND TESTING

```
#now lets split data in test train pairs
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle=False)
Acc = []
X_train_ = X_train.reshape(X_train.shape[0],X_train.shape[1],1)
X_test_ = X_test.reshape(X_test.shape[0],X_test.shape[1],1)
```

LSTM-LONG SHORT TERM MEMORY

```
from tensorflow.keras.layers import LSTM
def Reg():
    model = Sequential()
    model.add(LSTM(70, return_sequences=True, input_shape=(30,1)))
    model.add(LSTM(70, return_sequences=True))
    model.add(LSTM(70))
```

```
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

TRAINING THE MODELS

```
# Model Training
model_1 = reg()
model_1.fit(X_train_, y_train, epochs=100, validation_split=0.2)
```

OUTPUT:

```
Epoch 90/100
35/35 [==============================] - 1s 19ms/step - loss: 3.3475 - val_loss: 10.7295
Epoch 91/100
35/35 [==============================] - 1s 18ms/step - loss: 3.6378 - val_loss: 11.5784
Epoch 92/100
35/35 [==============================] - 1s 18ms/step - loss: 3.1080 - val_loss: 7.0888
Epoch 93/100
35/35 [==============================] - 1s 18ms/step - loss: 2.6003 - val_loss: 16.4153
Epoch 94/100
35/35 [==============================] - 1s 18ms/step - loss: 4.2621 - val_loss: 7.2827
Epoch 95/100
35/35 [==============================] - 1s 22ms/step - loss: 3.1074 - val_loss: 9.4735
Epoch 96/100
35/35 [==============================] - 1s 33ms/step - loss: 3.1837 - val_loss: 8.6071
Epoch 97/100
35/35 [==============================] - 1s 32ms/step - loss: 3.2264 - val_loss: 6.8003
Epoch 98/100
35/35 [==============================] - 1s 23ms/step - loss: 2.9291 - val_loss: 7.9528
Epoch 99/100
35/35 [==============================] - 1s 18ms/step - loss: 2.8059 - val_loss: 8.0432
Epoch 100/100
35/35 [==============================] - 1s 18ms/step - loss: 2.5645 - val_loss: 13.4775
<keras.callbacks.History at 0x7f64c3b85a80>
```

PREDICTING THE TARGET VARIABLE

```
# Prediction
y_pred_1 = model_1.predict(X_test_)
```

OUTPUT:

```
11/11 [==============================] - 0s 9ms/step
```

USING THE LSTM MODEL SEE THE DIFFERENCE BETWEEN THE ACTUAL AND PREDICTED VALUES

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_1.flatten()})
pred_df.head()
```

	Actual	Predicted
0	184.690002	191.162582
1	185.770004	192.843933
2	187.839996	192.525314
3	184.449997	193.812912
4	177.539993	194.093613

ACCURACY OF LSTM MODEL

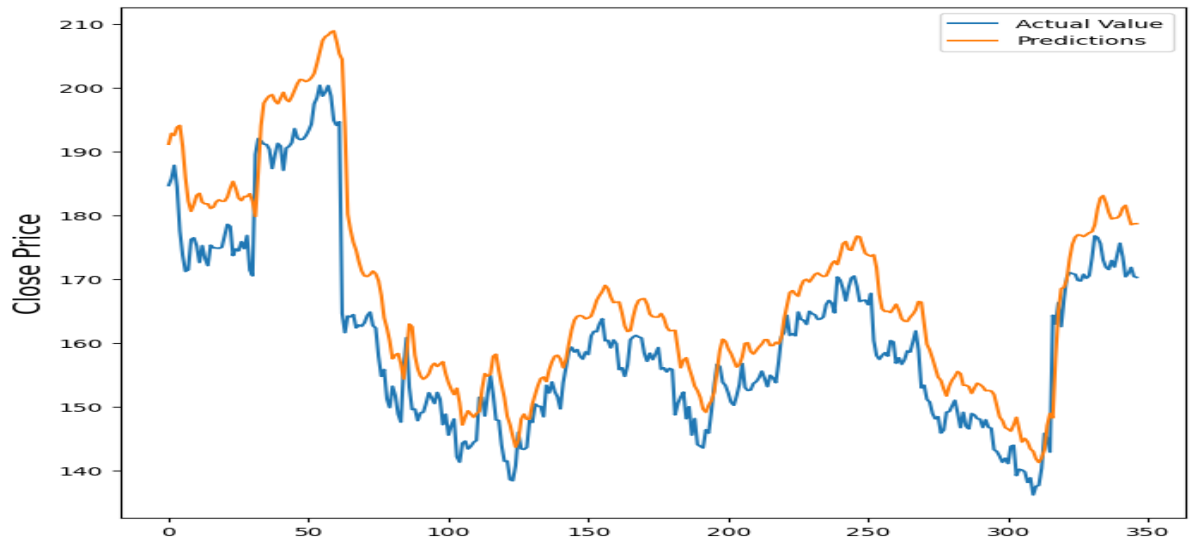
```
# Measure the Accuracy Score
from sklearn.metrics import r2_score
 print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_1)))
Acc.append(r2_score(y_test, y_pred_1))
```

OUTPUT:

Accuracy score of the predictions: 0.7228019774793342

LINEPLOT-SEEING THE DIFFERENCE OF ACTUAL VALUE AND PREDICTED VALUE IN THE GRAPH

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```




```
# Model Training
model_2 = regressor(inp_dim=30)
model_2.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 91/100
35/35 [==============================] - 0s 3ms/step - loss: 7.8026 - val_loss: 17.2894
Epoch 92/100
35/35 [==============================] - 0s 3ms/step - loss: 7.6841 - val_loss: 18.8580
Epoch 93/100
```

```
35/35 [==============================] - 0s 3ms/step - loss: 7.6776 - val_loss: 17.2357
Epoch 94/100
35/35 [==============================] - 0s 3ms/step - loss: 7.4983 - val_loss: 16.6812
Epoch 95/100
35/35 [==============================] - 0s 3ms/step - loss: 7.4026 - val_loss: 16.7703
Epoch 96/100
35/35 [==============================] - 0s 3ms/step - loss: 7.3434 - val_loss: 17.5881
Epoch 97/100
35/35 [==============================] - 0s 3ms/step - loss: 7.2822 - val_loss: 16.1491
Epoch 98/100
35/35 [==============================] - 0s 3ms/step - loss: 7.2085 - val_loss: 15.9987
Epoch 99/100
35/35 [==============================] - 0s 3ms/step - loss: 7.1127 - val_loss: 16.0978
Epoch 100/100
35/35 [==============================] - 0s 3ms/step - loss: 7.0268 - val_loss: 15.7359
<keras.callbacks.History at 0x7f3987f02ec0>
```

PREDICTING THE TARGET VALUE

```
# Prediction
y_pred_2 = model_2.predict(X_test)
```

OUTPUT:

```
11/11 [==============================] - 0s 3ms/step
```

ACCURACY OF ANN MODEL

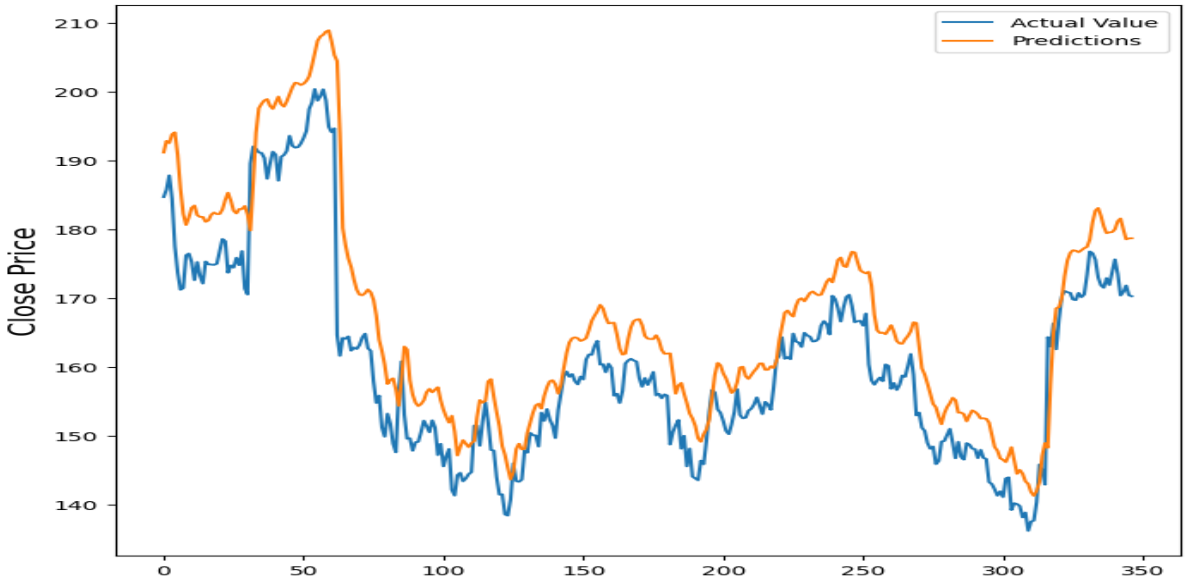
```
# Measure the Accuracy Score

from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_2)))
Acc.append(r2_score(y_test, y_pred_2))
```

OUTPUT:

```
Accuracy score of the predictions: 0.7837310259299168
```

LINEPLOT-SEEING THE DIFFERENCE OF ACTUAL VALUE AND PREDICTED VALUE IN THE GRAPH

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



CNN MODEL

```
# Model Training
model_3 = reg()
model_3.fit(X_train_, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 91/100
35/35 [==============================] - 1s 15ms/step - loss: 4.4238 - val_loss: 8.7975
Epoch 92/100
35/35 [==============================] - 1s 16ms/step - loss: 4.0307 - val_loss: 32.4325
Epoch 93/100
35/35 [==============================] - 1s 15ms/step - loss: 5.5199 - val_loss: 8.5408
Epoch 94/100
35/35 [==============================] - 1s 16ms/step - loss: 3.3338 - val_loss: 9.1608
Epoch 95/100
35/35 [==============================] - 1s 15ms/step - loss: 3.1096 - val_loss: 10.8817
Epoch 96/100
35/35 [==============================] - 1s 15ms/step - loss: 3.5590 - val_loss: 18.1243
Epoch 97/100
35/35 [==============================] - 1s 15ms/step - loss: 4.2190 - val_loss: 8.6936
Epoch 98/100
35/35 [==============================] - 1s 15ms/step - loss: 2.9081 - val_loss: 15.8735
Epoch 99/100
35/35 [==============================] - 1s 16ms/step - loss: 3.4238 - val_loss: 10.1643
Epoch 100/100
35/35 [==============================] - 1s 15ms/step - loss: 3.9303 - val_loss: 10.9768
<keras.callbacks.History at 0x7f3987d087c0>
```

PREDICTING THE TARGET VARIABLE

```
# Prediction
y_pred_3 = model_3.predict(X_test_)
```

OUTPUT:

```
11/11 [==============================] - 0s 4ms/step
```

USING THE CNN MODEL SEE THE DIFFERENCE BETWEEN THE ACTUAL AND PREDICTED VALUES

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_3.flatten()})
pred_df.head()
```

	Actual	Predicted
0	184.690002	178.864548
1	185.770004	180.684158
2	187.839996	182.158020
3	184.449997	183.637985
4	177.539993	183.687012

ACCURACY OF CNN MODEL

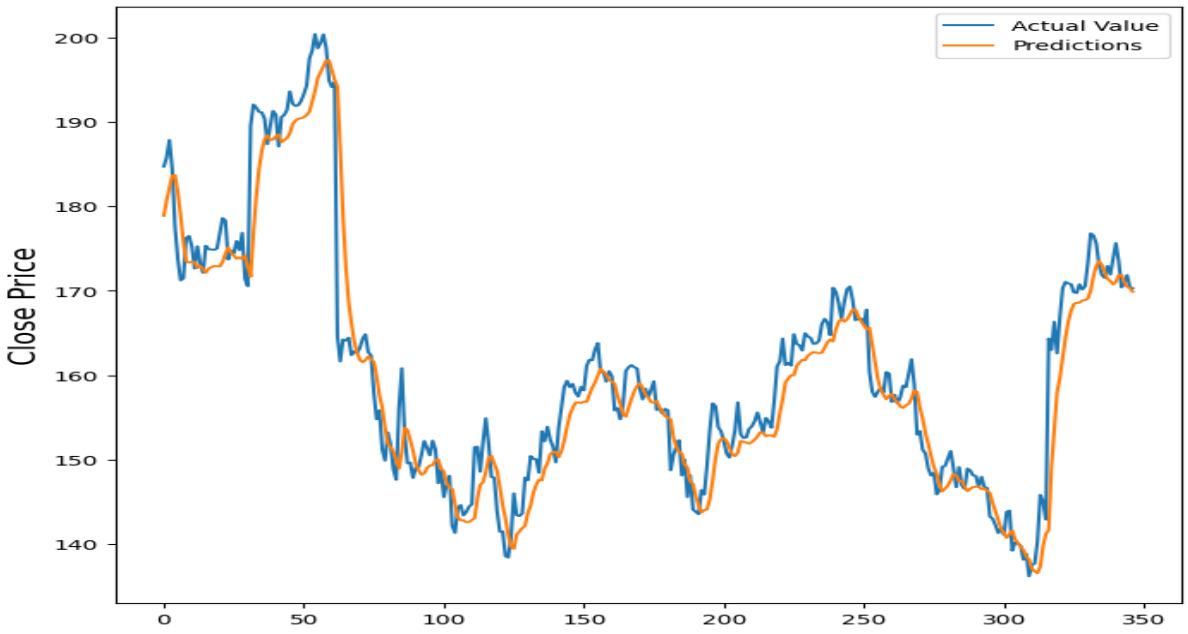
```
from sklearn.metrics import r2_score
 print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_3)))
Acc.append(r2_score(y_test, y_pred_3))
```

OUTPUT:

Accuracy score of the predictions: 0.9051730933662152

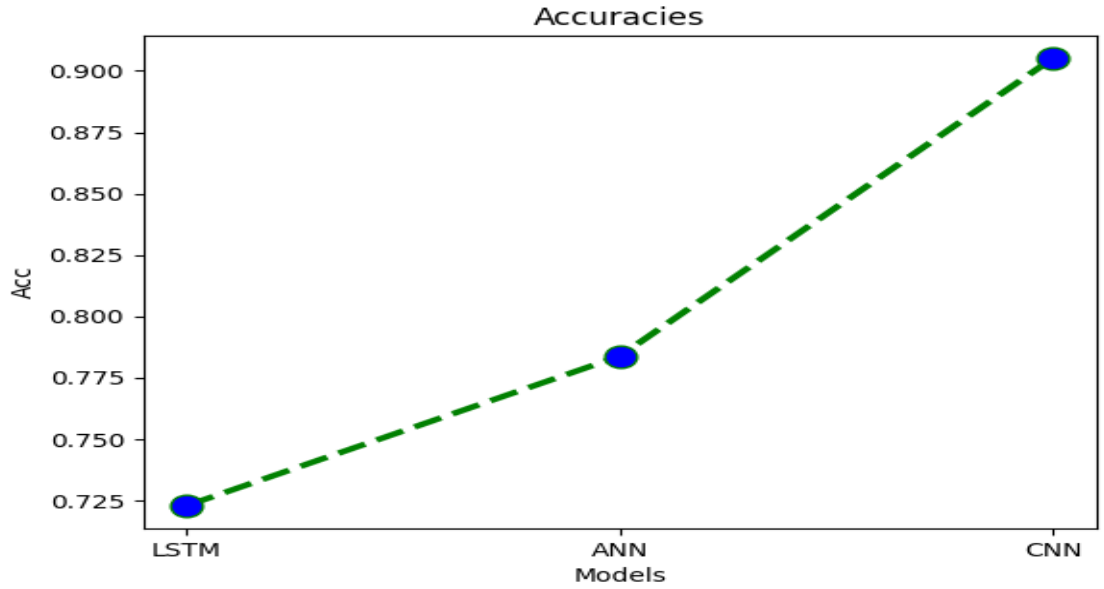
LINEPLOT-SEEING THE DIFFERENCE OF ACTUAL VALUE AND PREDICTED VALUE IN THE GRAPH

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



COMPARISION OF ACCURACIES

```
plt.plot(range(3), Acc, color='green', linestyle='dashed', linewidth = 3,
        marker='o', markerfacecolor='blue', markersize=12)
plt.ylabel('Acc')
plt.xlabel('Models')
plt.title("Accuracies")
plt.xticks(range(3), ['LSTM', 'ANN', 'CNN'])
plt.show()
```



COMPUTING CORRELATIONS

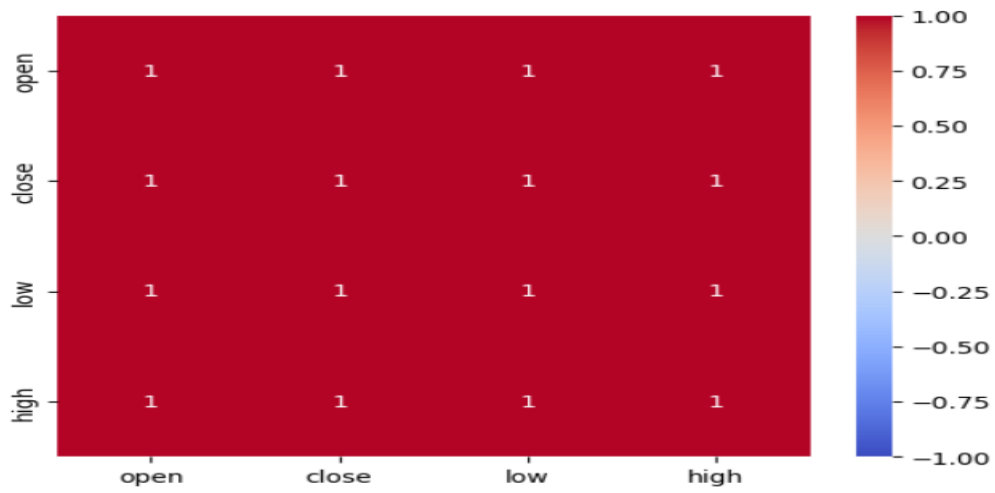
corr_matrix = df.corr()

corr_matrix

	open	close	low	high
open	1.000000	0.999382	0.999691	0.999628
close	0.999382	1.000000	0.999615	0.999737
low	0.999691	0.999615	1.000000	0.999475
high	0.999628	0.999737	0.999475	1.000000

plot correlation matrix as heatmap(VISUALIZING CORRELATIONS)

sns.heatmap(corr_matrix, cmap='coolwarm', annot=True, vmin=-1, vmax=1)



CORRELATION FOR THE OPEN AND CLOSE COLUMN:

print(corr_matrix.loc['open', 'close'])

0.9993817292161483

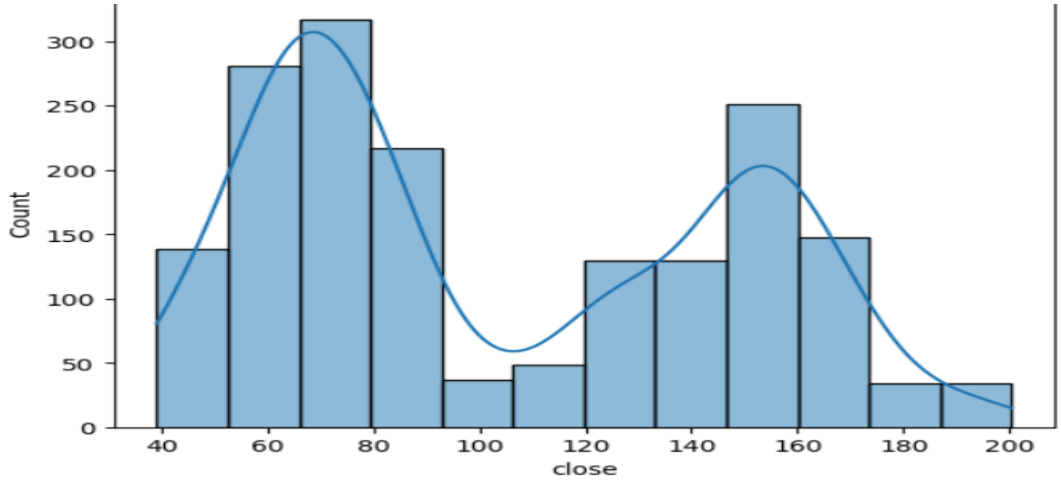
SKEWNESS

close_skew = df['close'].skew()
open_skew = df['open'].skew()
low_skew = df['low'].skew()
high_skew = df['high'].skew()
print('Close Skewness:', close_skew)
print('Open Skewness:', open_skew)
print('Low Skewness:', low_skew)
print('High Skewness:', high_skew)

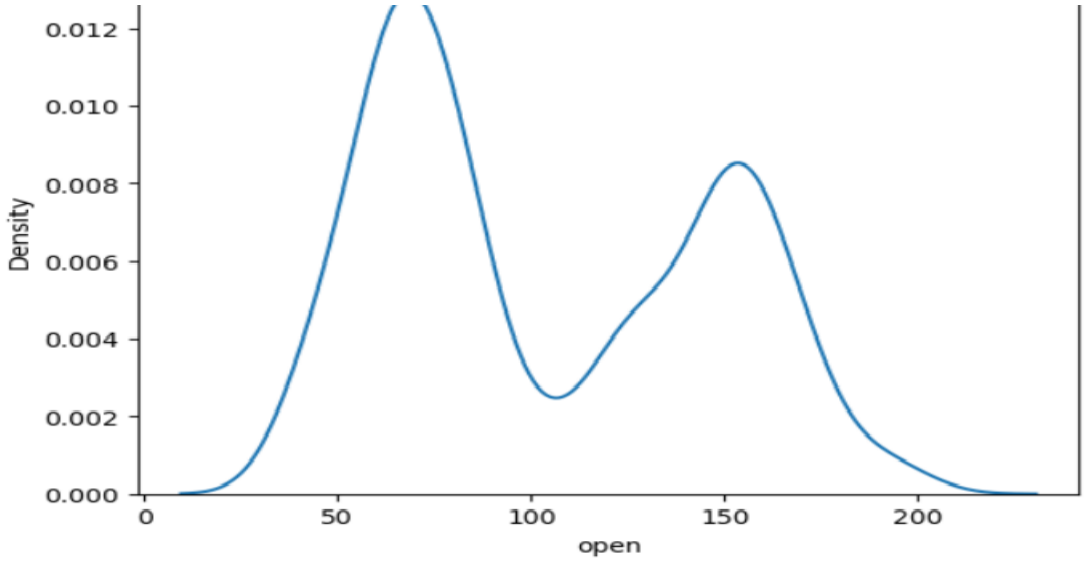
OUTPUT:

Close Skewness: 0.3219049667575204
Open Skewness: 0.3214634068842909
Low Skewness: 0.32028839822816646
High Skewness: 0.32183524318806644

sns.histplot(data=df, x='close', kde=True)



```
sns.kdeplot(data=df, x='open')
```



INTERPRET SKEWNESS

```
if close_skew > 0:
 print('The close variable is positively skewed.')
elif close_skew < 0:
 print('The close variable is negatively skewed.')
else:
 print('The close variable is normally distributed.')
```

OUTPUT:

The close variable is positively skewed.

SPLIT THE DATA INTO FEATURES AND TARGET VARIABLE (DECISION TREE) (CLOSE-TARGET; OPEN CATEGORICAL)

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
X = df.drop('close', axis=1)
y = df['close']
# encode categorical variables
encoder = LabelEncoder()
X['open'] = encoder.fit_transform(X['open'])
# fill missing values
X.fillna(X.mean(), inplace=True)
# scale the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
# split the data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# create the decision tree classifier object
```

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
# hyperparameters to tune
param_grid = {'max_depth': [3, 5, 7],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4]}
# grid search cross-validation
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(dtc, param_grid, cv=5)
from sklearn.tree import DecisionTreeRegressor
# Create a decision tree regressor object
tree_regressor = DecisionTreeRegressor()
# Fit the decision tree regressor to the training data
tree_regressor.fit(X_train, y_train)
```

OUTPUT:

```
DecisionTreeRegressor()
```

```
# Make predictions on test data
y_pred = tree_regressor.predict(X_test)
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
# Create a decision tree regressor object
tree_regressor = DecisionTreeRegressor()
# Define the parameter grid for grid search
param_grid = {
 'max_depth': [3, 5, 7],
 'min_samples_split': [2, 4, 6],
 'min_samples_leaf': [1, 2, 3]
}
# Create the grid search object
grid_search = GridSearchCV(tree_regressor, param_grid, cv=5)
# Fit the grid search object to the training data
grid_search.fit(X_train, y_train)
# Retrieve the best parameters
best_params = grid_search.best_params_
# Create the final decision tree model with the best hyperparameters
final_model = DecisionTreeRegressor(
 max_depth=best_params['max_depth'],
 min_samples_split=best_params['min_samples_split'],
 min_samples_leaf=best_params['min_samples_leaf']
)

# Fit the final model to the training data
final_model.fit(X_train, y_train)
```

OUTPUT:

DecisionTreeRegressor(max_depth=7, min_samples_leaf=2, min_samples_split=4)

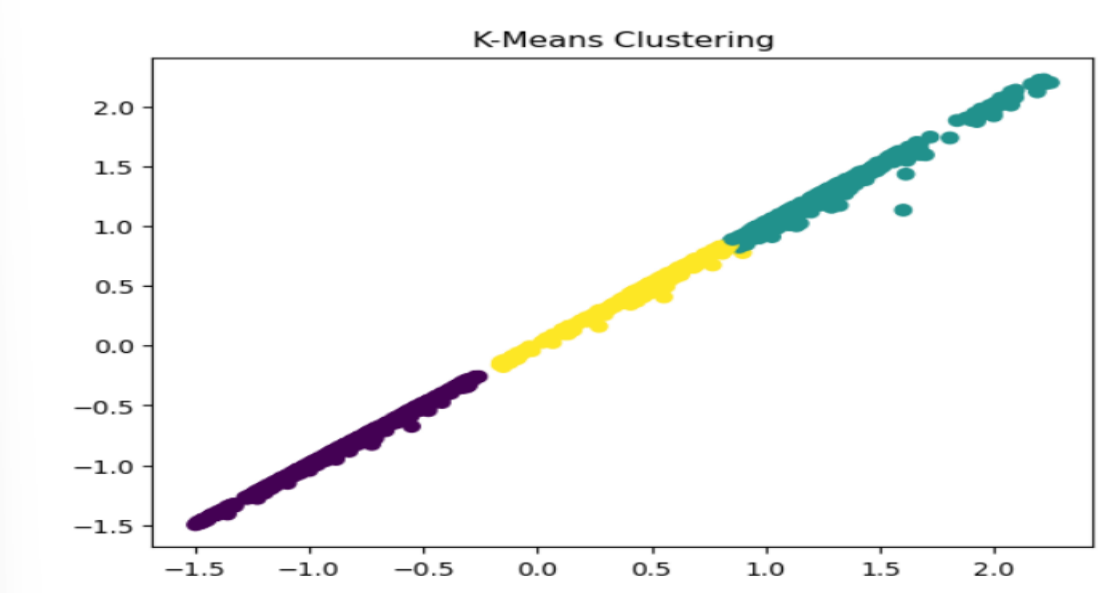
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# Make predictions on test data
y_pred = final_model.predict(X_test)
# Compute regression evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("Mean absolute error is: ",mae)
print("Mean squared error is: ",mse)
print("R-squared is: ",r2)
```

OUTPUT:

Mean absolute error is: 0.7171359521452576
Mean squared error is: 1.1151255161598734
R-squared is: 0.9993997218908506

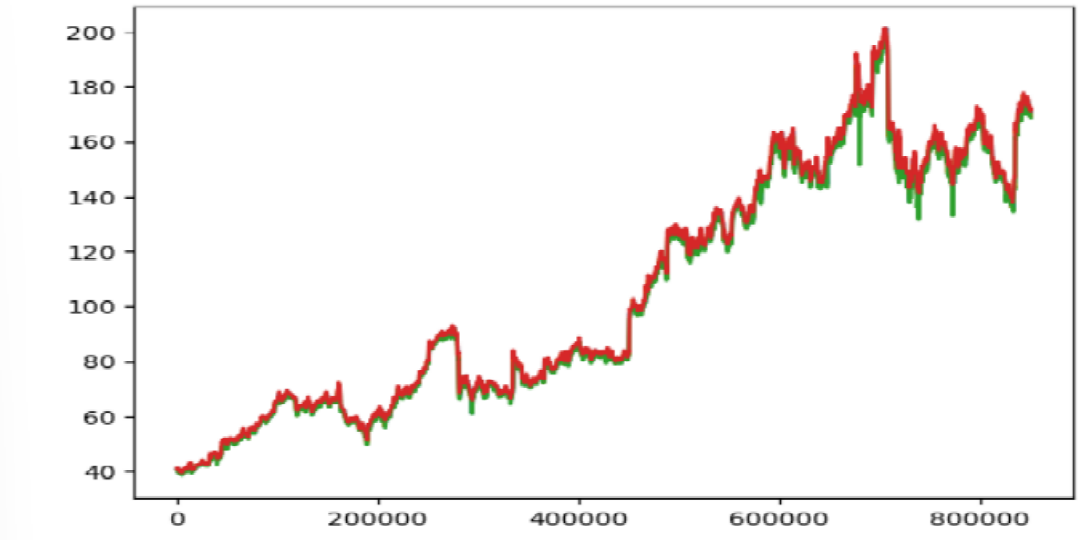
K-MEANS

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
X = df.drop('close', axis=1)
# preprocess the data by scaling the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# choose the number of clusters
k = 3
# create the KMeans clustering object
kmeans = KMeans(n_clusters=k, random_state=42)
# fit the algorithm to the data and obtain the cluster labels
labels = kmeans.fit_predict(X_scaled)
# evaluate the performance of the clustering using silhouette score
score = silhouette_score(X_scaled, labels)
print('Silhouette score:', score)
# visualize the results
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels)
plt.title('K-Means Clustering')
plt.show()
```



TIMESERIES

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
plt.plot(df)
plt.show()
```



ASSOCIATION RULE MINING

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

```
dataset = [['open', 'close', 'low'],
 ['open', 'volume', 'high'],
 ['open', 'close', 'low', 'high', 'volume'],
 ['open', 'volume', 'low', 'close', 'high'],
 ['close', 'low', 'high', 'volume']]

te = TransactionEncoder()
te_ary = te.fit_transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
print(df)
# Step 2: Support Calculation
min_support = 0.4
freq_items = apriori(df, min_support=min_support, use_colnames=True)
print(freq_items)
# Step 3: Frequent Itemset Generation
min_threshold = 0.7
assoc_rules = association_rules(freq_items, metric="confidence",
min_threshold=min_threshold)
print(assoc_rules)
# Step 4: Rule Generation
min_confidence = 0.8
rules = association_rules(freq_items, metric="confidence", min_threshold=min_confidence)
print(rules)
# Step 5: Rule Evaluation
rules["lift"] = association_rules(freq_items, metric="lift",
min_threshold=min_confidence)["lift"
rules["conviction"] = association_rules(freq_items, metric="conviction",
min_threshold=min_confid
print(rules)
```

OUTPUT:

```
close high low open volume
0 True False True True False
1 False True False True True
2 True True True True True
3 True True True True True
4 True True True False True
 support itemsets
0 0.8 (close)
1 0.8 (high)
2 0.8 (low)
3 0.8 (open)
4 0.8 (volume)
5 0.6 (high, close)
6 0.8 (close, low)
7 0.6 (close, open)
8 0.6 (close, volume)
```

9 0.6 (high, low)
10 0.6 (high, open)
11 0.8 (high, volume)

LOCALITY SENSITIVITY HASHING

```
X = np.random.rand(100, 10)
from sklearn.random_projection import SparseRandomProjection
rp = SparseRandomProjection(n_components=5)
# Transform the data using the random projection
X_rp = rp.fit_transform(X)
X_rp
```

OUTPUT:

```
array([[-0.00893743, 0.02802295, 0.05954404, -0.25215123, -0.97969252],
 [-0.10157833, 0.15425237, -0.23710551, -0.58155089, -0.50146989],
 [-0.26888246, 0.7466086 , -0.25253755, -0.54278773, -0.95001067],
 [-0.76547213, 0.92453109, -0.30168417, -0.63579927, -0.71176313],
 [-0.2132047 , 0.72373173, -0.01468971, -1.15824927, -0.87445205],
 [-0.63688853, 0.69153872, -0.57086425, -0.32678388, -0.81327409],
 [-0.38089537, 0.95459264, -0.69370915, 0.05562309, -1.59281984],
 [-0.2516888 , 0.46232538, 0.05245795, -0.85781803, -0.15839716],
 [-0.01170683, 0.6990972 , 0.46028483, -0.415864 , -0.94354021],
 [-0.09304659, 0.23264894, -0.6165143 , -0.03651614, -0.14970285],
 [-0.44664074, 0.49817116, -0.23350027, 0.20335505, -1.16599803],
 [-0.11009092, 0.57995656, -0.11633803, -0.92340283, -0.92206884],
 [-0.7351686 , 1.02740042, 0.20106924, -0.54125348, 0.24316932],
 [-0.36591537, 1.10601017, 0.63846698, -0.78556162, -0.5587942 ],
 [-0.5852557 , 1.30028745, -0.15152852, -0.69957649, -0.59883068],
 [-0.43208351, 0.65968718, -0.23788166, -0.40299102, -0.41058953],
 [-0.71267887, 1.25719934, -0.07280254, -0.52598227, -0.61312189],
 [-0.73310829, 0.77368495, -0.2584726 , -0.22036931, -0.54510241],
 [-0.11848165, 0.62852424, -0.43795221, -0.33605381, -0.49261325],
 [-0.10228279, 0.1601532 , 0.13291006, -0.28851429, -1.04501296],
 [-0.42207975, 0.93113427, 0.14509041, -0.92744452, -0.26654873],
 [-0.10340316, 0.80994474, 0.39487168, -0.58801493, -0.20023699],
 [-0.67632262, 1.01382612, -0.18693943, 0.44195837, -0.94379002],
 [-0.40534182, 0.51154394, -0.01923345, -0.88123629, 0.05345514],
 [-0.29254982, 0.59796368, 0.05782815, -0.28971646, -0.4364229 ],
 [-0.17763021, 0.65424729, 0.03832217, 0.19705265, -1.01084553],
 [-0.43671935, 1.11781488, -0.38320147, -0.60525362, -1.31338568],
 [-0.30857834, 1.02278991, -0.43480447, 0.10991007, -1.23021052],
 [-0.04822639, 0.58555942, -0.14838622, -0.73491405, -1.35384302],
 [-0.39158882, 1.13343331, -0.04094888, -0.32332416, -0.93214079],
 [-0.53344268, 1.11156919, 0.23768846, -0.42751308, -0.17945805],
 [-0.29345757, 0.75368309, 0.17983582, -0.52402287, -0.91021569],
 [-0.38084899, 0.99957132, 0.16389361, -0.49232353, -0.76267865],
 [-0.5880033 , 0.76738081, 0.16791112, -0.30030429, -0.83789066],
```

```
[-0.14456713, 0.60380907, 0.58990912, -0.1285485 , -0.78651661],
[-0.40116456, 0.43821497, 0.44518472, -0.63001889, -0.64825967],
[-0.36182359, 0.50036189, -0.4800447 , -0.20436374, -1.3146845 ],
[-0.46110568, 0.5675078 , 0.15739276, -0.64338177, -1.21045739],
[-0.38165364, 0.81747117, 0.30151752, -0.9059856 , 0.00585634],
[-0.37690131, 1.02380944, 0.11162654, -1.10498992, -0.42320301],
[-0.08928064, 0.3247933 , 0.21588843, -0.64410766, -0.93464591],
[-0.11369346, 0.41393274, 0.14608968, -0.23953399, -1.52100354],
[-0.42949419, 1.07668656, -0.47394007, -0.05715454, -0.27738104],
[-0.70674952, 1.07994897, 0.0398614 , -0.67198275, -0.70502129]
```

CONFUSION MATRIX

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
# Assuming you have predicted labels and true labels as numpy arrays or pandas Series
predicted_labels = np.array([0, 1, 1, 0, 1, 0])
true_labels = np.array([0, 0, 1, 0, 1, 1])
# Compute confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)
# Create a pandas DataFrame for better visualizationConfusion Matrix:

2labels = ['Negative', 'Positive']  # Assuming 0 represents Negative and 1 represents Positive
```

OUTPUT:

```
  Negative  Positive

Negative        2        1

Positive        1        2
```

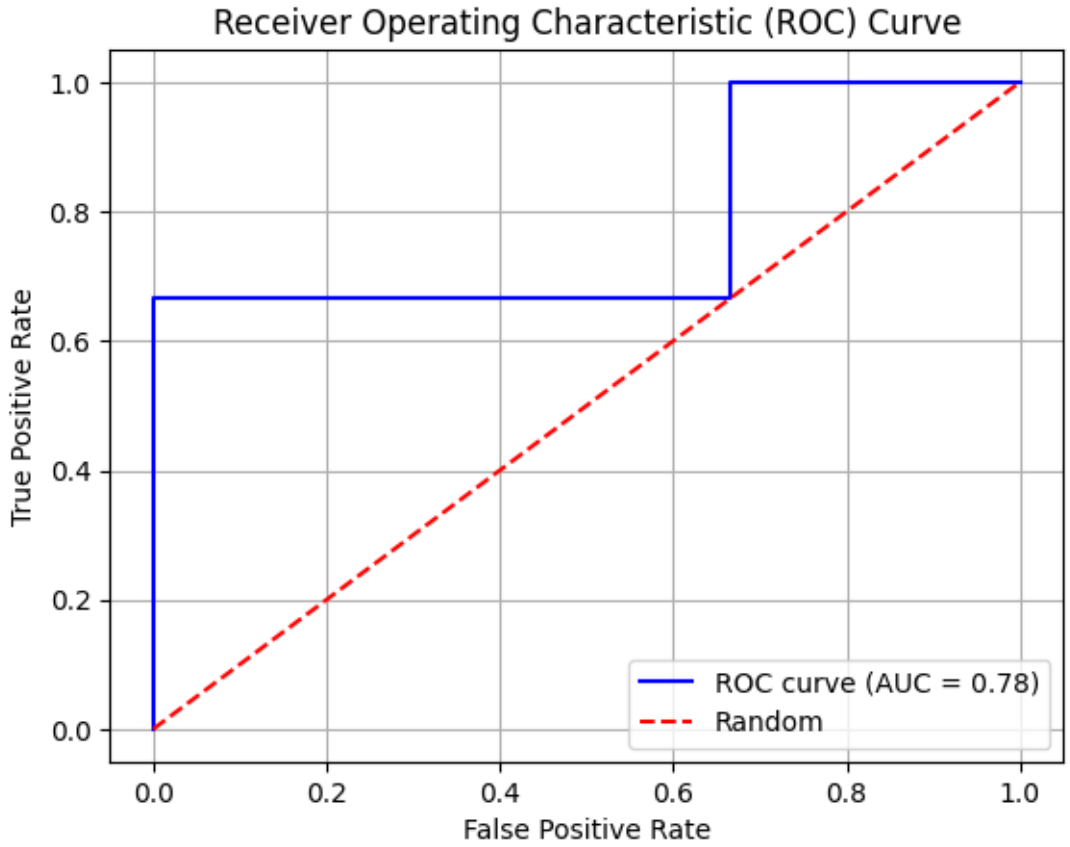
```
df_cm = pd.DataFrame(cm, index=labels, columns=labels)
# Print the confusion matrix
print("Confusion Matrix:")
print(df_cm)
```

```
Confusion Matrix:
          Negative  Positive
Negative        2        1
Positive        1        2
```

PLOTTING ROC CURVE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
# Assuming you have predicted probabilities and true labels as numpy arrays or pandas Series
```

```
predicted_probabilities = np.array([0.2, 0.6, 0.7, 0.4, 0.8, 0.3])
true_labels = np.array([0, 0, 1, 0, 1, 1])
# Compute false positive rate (FPR), true positive rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(true_labels, predicted_probabilities)
# Compute area under the ROC curve (AUC)
roc_auc = auc(fpr, tpr)
# Plot ROC curve
plt.plot(fpr, tpr, color='b', label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='r', linestyle='--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```



CONCLUSION:

In this study, we explored the application of various models, including Linear Regression, Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM), for stock market prediction. Additionally, we performed data analysis techniques such as correlation analysis, skewness analysis, k-means clustering, decision tree analysis, and association rule mining.

By employing these models, we aimed to predict stock market trends and movements, analyze relationships between variables, and uncover meaningful patterns within the data. The combination of these models allowed us to assess the performance and accuracy of different prediction approaches and data analysis techniques.

Throughout the analysis, we leveraged the strengths of each model. Linear Regression provided insights into the linear relationship between stock market variables, ANN captured complex nonlinear patterns, CNN excelled at extracting features from visual representations, and LSTM effectively handled time series data with long-term dependencies.

Furthermore, we utilized visualization techniques such as line plots and heat maps to represent time series trends and understand the distribution and relationships within the data. These visualizations enhanced our ability to interpret and communicate the findings effectively.

The results of our study contribute to the understanding of stock market prediction by providing insights into the performance and accuracy of different models. The comparison of the models allowed us to determine their strengths and limitations in the context of stock market forecasting. Additionally, the application of data analysis techniques provided valuable insights into the correlations, skewness, clustering, and association patterns within the stock market data.

Overall, this study sheds light on the potential of using linear regression, ANN, CNN, and LSTM models for stock market prediction and demonstrates the importance of data analysis techniques in uncovering valuable insights. The findings can be utilized by investors, financial institutions, and researchers to make informed decisions, develop trading strategies, and gain a deeper understanding of the dynamics within the stock market.

REFERENCES:

1. Stock Market Prediction using a Linear Regression; International Conference on Electronics, Communication and Aerospace Technology (ICECA 2017).

2. Stock Market Prediction; 2015 19th International Conference on System Theory, Control and Computing (ICSTCC), October 14-16, Cheile Gradistei, Romania.

3. Developing a Prediction Model for Stock Analysis; 2017 International Conference on Technical Advancements in Computers and Communications.

4. The Analysis and Prediction of Stock Price; 2013 IEEE International Conference on Granular Computing (GrC).

5. Stock Market Prediction using a Hybrid Approach; International Conference on Computing, Communication and Automation (ICCCA2016).

6. Stock Market Prediction Using Machine Learning; 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC).

7. Stock Transaction Prediction Modelling and Analysis Based on LSTM; 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA).

8. Real-Time Stock Prediction using Neural Network; 2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence).

9. A Stock Market Prediction Model using Artificial Neural Network; ICCCNT'12 26th _28th July 2012, Coimbatore, India.

10. Apply Multiple Linear Regression Model to Predict the Audit Opinion; 2009 ISECS International Colloquium on Computing, Communication, Control, and Management.