import pandas as pd

import matplotlib.pyplot as plt

from pylab import rcParams

import numpy as np

import seaborn as sns

import os

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import cross\_val\_score, train\_test\_split, GridSearchCV

from sklearn.feature\_selection import RFECV, SelectFromModel, SelectKBest

from sklearn.preprocessing import StandardScaler

from sklearn import metrics

%matplotlib inline

Stock = pd.read\_csv("C:/Users/x/Desktop/Desktop/ML/AAPL.csv", index\_col=0)

df\_Stock = Stock

df\_Stock = df\_Stock.rename(columns={'Close(t)':'Close'})

df\_Stock.head()

df\_Stock.tail(5)

df\_Stock.shape

df\_Stock.columns

df\_Stock['Close'].plot(figsize=(10, 7))

plt.title("Stock Price", fontsize=17)

plt.ylabel('Price', fontsize=14)

plt.xlabel('Time', fontsize=14)

plt.grid(which="major", color='k', linestyle='-.', linewidth=0.5)

plt.show()

df\_Stock = df\_Stock.drop(columns='Date\_col')

def create\_train\_test\_set(df\_Stock):

features = df\_Stock.drop(columns=['Close\_forcast'], axis=1)

target = df\_Stock['Close\_forcast']

data\_len = df\_Stock.shape[0]

print('Historical Stock Data length is - ', str(data\_len))

#create a chronological split for train and testing

train\_split = int(data\_len \* 0.88)

print('Training Set length - ', str(train\_split))

val\_split = train\_split + int(data\_len \* 0.1)

print('Validation Set length - ', str(int(data\_len \* 0.1)))

print('Test Set length - ', str(int(data\_len \* 0.02)))

# Splitting features and target into train, validation and test samples

X\_train, X\_val, X\_test = features[:train\_split], features[train\_split:val\_split], features[val\_split:]

Y\_train, Y\_val, Y\_test = target[:train\_split], target[train\_split:val\_split], target[val\_split:]

#print shape of samples

print(X\_train.shape, X\_val.shape, X\_test.shape)

print(Y\_train.shape, Y\_val.shape, Y\_test.shape)

return X\_train, X\_val, X\_test, Y\_train, Y\_val, Y\_test

X\_train, X\_val, X\_test, Y\_train, Y\_val, Y\_test = create\_train\_test\_set(df\_Stock)

# Prediction using Linear Regression

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

lr.fit(X\_train, Y\_train)

print('LR Coefficients: \n', lr.coef\_)

print('LR Intercept: \n', lr.intercept\_)

print("Performance (R^2): ", lr.score(X\_train, Y\_train))

def get\_mape(y\_true, y\_pred):

"""

Compute mean absolute percentage error (MAPE)

"""

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

return np.mean(np.abs((y\_true - y\_pred) / y\_true)) \* 100

#Predict for the test dataset

Y\_train\_pred = lr.predict(X\_train)

Y\_val\_pred = lr.predict(X\_val)

Y\_test\_pred = lr.predict(X\_test)

print("Training R-squared: ",round(metrics.r2\_score(Y\_train,Y\_train\_pred),2))

print("Training Explained Variation: ",round(metrics.explained\_variance\_score(Y\_train,Y\_train\_pred),2))

print('Training MAPE:', round(get\_mape(Y\_train,Y\_train\_pred), 2))

print('Training Mean Squared Error:', round(metrics.mean\_squared\_error(Y\_train,Y\_train\_pred), 2))

print("Training RMSE: ",round(np.sqrt(metrics.mean\_squared\_error(Y\_train,Y\_train\_pred)),2))

print("Training MAE: ",round(metrics.mean\_absolute\_error(Y\_train,Y\_train\_pred),2))

print(' ')

print("Validation R-squared: ",round(metrics.r2\_score(Y\_val,Y\_val\_pred),2))

print("Validation Explained Variation: ",round(metrics.explained\_variance\_score(Y\_val,Y\_val\_pred),2))

print('Validation MAPE:', round(get\_mape(Y\_val,Y\_val\_pred), 2))

print('Validation Mean Squared Error:', round(metrics.mean\_squared\_error(Y\_train,Y\_train\_pred), 2))

print("Validation RMSE: ",round(np.sqrt(metrics.mean\_squared\_error(Y\_val,Y\_val\_pred)),2))

print("Validation MAE: ",round(metrics.mean\_absolute\_error(Y\_val,Y\_val\_pred),2))

print(' ')

print("Test R-squared: ",round(metrics.r2\_score(Y\_test,Y\_test\_pred),2))

print("Test Explained Variation: ",round(metrics.explained\_variance\_score(Y\_test,Y\_test\_pred),2))

print('Test MAPE:', round(get\_mape(Y\_test,Y\_test\_pred), 2))

print('Test Mean Squared Error:', round(metrics.mean\_squared\_error(Y\_test,Y\_test\_pred), 2))

print("Test RMSE: ",round(np.sqrt(metrics.mean\_squared\_error(Y\_test,Y\_test\_pred)),2))

print("Test MAE: ",round(metrics.mean\_absolute\_error(Y\_test,Y\_test\_pred),2))

df\_pred = pd.DataFrame(Y\_val.values, columns=['Actual'], index=Y\_val.index)

df\_pred['Predicted'] = Y\_val\_pred

df\_pred = df\_pred.reset\_index()

df\_pred.loc[:, 'Date'] = pd.to\_datetime(df\_pred['Date'],format='%Y-%m-%d')

df\_pred

df\_pred[['Actual', 'Predicted']].plot()