In [1]:
```
# For the final project I have decided to use Yfin as a source insteac
# the original dataset being used only gave the prices of the stocks a
# the aim is to predict the performance of the benchmark as a whole ir
# prices of stocks only. Secondly, Yfinance is more user friendly and s
# of charts and visuals as well and it is easier to visualise their da
# the results in this project. This is the link to my github:
#https://github.com/zahracodes123/Final-Capstone-Project/tree/main
import pandas as pd
import numpy as np
import yfinance as yf
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn import neighbors
from math import sqrt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score

data=yf.download("^GSPC", start="2019-11-01", end="2022-11-01")
data_financials=pd.DataFrame(data)
data_financials.to_csv("S&P500.csv")

d=pd.read_csv('S&P500.csv')
df=d.dropna()
df.set_index("Date",inplace=True)
column=df.pop("Volume")
df.insert(0,"Volume",column)

df=pd.get_dummies(df)
x=df.iloc[:,0:5].values
y=df.iloc[:,5]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,randc

#In order to use the data for analysis , it is important to clean the
#had all of its Na values removed and the date has beeen set as the ir
#and use it more efficiently. The stock price from the past three year
#cover the period from before COVID-19 to now. The model can also be a
#time period can be used by exporting data from more years. The data h
#feature applied to it so that in case there is any categorical variab
#The features that will be used to predict the dependant variable whic
#'Open', 'Close' and 'Volume'. These variables are basically showing s
#trading activity which is volume. The adjusted close price is the pri
#been performed and this is our y variable and will be used as the var
#a train test ratio which is 70-30.


model=RandomForestRegressor(n_estimators=10, random_state=0)
```

```
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print(y_pred)
print(y_pred.shape)
RMSE1=float(format(np.sqrt(mean_squared_error(y_test, y_pred)),'.3f'))
RFR2=r2_score(y_test, y_pred)

#The random forest regression model is used. In this case the hyper pa
#adjusted as this is a skeletal code and will be improved in the final
#are adjusted in order to improve the accuracy and fitting of the mode


regressor=LinearRegression()
regressor.fit(x_train,y_train)
predict_y=regressor.predict(x_test)
RMSE2=mean_squared_error(y_test, predict_y)
LRR2=r2_score(y_test,predict_y)

#The second regression model being used is Multiple Linear Regression.
#variable is 'Adjusted Close' and the independant variables are 'High'
#'volume'. Since we are using multiple variables to predict one indepe
#Regression is being used.


scaler=MinMaxScaler(feature_range=(0,1))
x_train_scaled=scaler.fit_transform(x_train)
x_train=pd.DataFrame(x_train_scaled)
x_test_scaled=scaler.fit_transform(x_test)
x_test=pd.DataFrame(x_test_scaled)


model=neighbors.KNeighborsRegressor(n_neighbors=5)
model.fit(x_train,y_train)
predd=model.predict(x_test)
RMSE3=sqrt(mean_squared_error(y_test,predd))
KNR2=r2_score(y_test, predd)
#The third algorithm that will be used is the KN Regressor. The number
#used is 5 which is also the default. The data is first scaled using t
#the values of the data to a number between 0 and 1 without changing t
#fit on the data to predict the stock prices.

print(RMSE1)
print(RFR2)
print(RMSE2)
print(LRR2)
print(RMSE3)
print(KNR2)

#In order to test which regression algorithm is more accurate , two me
#other is the R^2 values. When we run the results, we see that KN regr
#Higher RMSE values indicate that the model is not very good at predic
```

```
#is a prediction model, higher RMSE is not ideal. R^2 value of Multipl
#perfectly. Second best score is Random Forest Regression's. The worst
#initial result, it can be concluded that Multiple Linear Regression i
#accurate.
```

```
[*********************100%***********************]  1 of 1 completed
[3927.20698242 4658.821875   3933.22199707 4160.67304688 4247.1899902
3
 3909.62697754 4004.55405273 3754.04689941 4683.46293945 4456.6220214
8
 4364.90380859 4203.20385742 4243.16801758 3143.57192383 3275.3459472
7
 3790.4609375  4062.6279541  3274.42897949 4172.85795898 4352.5580566
4
 4359.61083984 4623.60498047 2791.96203613 3776.31501465 4013.3480468
7
 3900.57507324 4557.22695313 4061.00600586 4298.00185547 2958.4330566
4
 3038.12502441 4522.49306641 3305.26308594 3941.15705566 4289.4389160
2
 4570.25996094 4697.8199707  3824.81201172 2859.17495117 4209.6250976
6
 2842.8869873  2871.15397949 3625.96096191 3091.26000977 4539.5989746
1
 4697.67998047 4461.66206055 3408.89099121 3275.28498535 2612.0000244
1
 4143.78295898 4534.23398437 4115.11210937 3925.59697266 3505.2689941
4
 3057.59501953 3132.17202148 3071.4170166  3761.39804688 3822.5990478
5
 4287.34394531 4468.68500977 2588.6590332  3985.6        3299.2830078
1
 3113.97104492 3900.92102051 3393.24299316 4187.64111328 3115.7500732
4
 3385.33103027 4373.42089844 4723.12109375 3676.83300781 4525.6800781
3
 4178.92509766 4656.77792969 4162.89897461 3933.90402832 4518.6060058
6
 4070.69797363 2940.175      4461.61708984 3910.56799316 3289.0909668
 4594.66098633 4464.94106445 3383.78203125 3276.89997559 3847.0360839
8
 4183.14707031 3596.45300293 4154.7440918  4109.11201172 4138.6199707
 4786.79208984 4400.87504883 3228.53293457 3222.0109375  4656.0599609
4
 3971.19997559 4591.22099609 3078.01105957 4366.54580078 3665.6790283
2
 4538.31601562 3700.1840332  3353.55495605 4188.23310547 4641.9429687
5
 2487.83295898 3054.56601563 4530.63188477 3247.00898438 4354.3189941
4
```

```
 4597.61494141 3251.84404297 4095.08596191 4169.64799805 4346.4810058
6
 4148.23708496 3567.33500977 3222.17192383 4159.14799805 3443.4560546
9
 4400.85097656 3244.18505859 3915.01994629 3249.24799805 3827.8140380
9
 3717.72102051 4403.06791992 2743.2579834  3809.52607422 4126.7200195
3
 3066.43000488 4667.97192383 3370.63798828 3800.71899414 3329.7570800
8
 4461.61606445 3843.68203125 3739.75197754 3876.84599609 3196.1659668
 3312.36005859 4398.11513672 3791.21196289 4353.95292969 3447.2440429
7
 3109.56303711 4666.09199219 4128.79296875 3623.16594238 2469.6830078
1
 3643.95197754 4536.3421875  3794.54997559 3578.37006836 4781.9280761
7
 3391.29396973 4435.44897461 3925.83601074 2832.84294434 4292.9399414
1
 3389.5380127  4393.27192383 4156.2440918  4330.18491211 3710.2979980
5
 3233.84897461 4278.52900391 3360.09799805 3221.24692383 4358.0199707
 3272.87597656 2939.5369873  3334.99602051 3483.25100098 2806.7440185
5
 4359.9019043  4701.50507813 4307.76499023 3222.71394043 3900.3050293
 4115.6640625  3260.35500488 3737.91599121 3131.83701172 4281.9090820
3
 4173.63095703 4624.32602539 2836.22495117 3005.0770752  4076.4589355
5
 4107.04499512 3336.59797363 2720.70698242 3143.80495605 4227.6739257
8
 3185.20197754 4660.79589844 2489.05395508 4352.70200195 3410.3760009
8
 4199.87407227 4375.78388672 4484.10390625 3249.03798828 3590.2280029
3
 4202.00791016 3369.64399414 4423.73891602 4081.42294922 3121.0989990
2
 3396.45305176 2748.21499023 4436.52495117 4770.03911133 3112.1100341
8
 4525.06508789 3966.41098633 4202.11293945 4015.63706055 3754.0030273
4
 4510.69189453 4291.28388672]
(227,)
12.85
0.9995135997154627
2.2571510588651662e-20
1.0
81.7826843727659
0.9802994937173006
```

In [ ]:

In [ ]:

In [ ]: