Import numpy, pandas and datetime library

In [1]:
```
import numpy as np
import pandas as pd
import datetime as dt
```

Read data from an excel sheet

In [2]:
```
data = pd.read_excel('Energy.xlsx')
```

Convert DataDate column to datetime format

In [3]:
```
data['Data Date'] = pd.to_datetime(data['Data Date'], format='%Y%m%d')
```

Get column number of Accumulated Other Comprehensive Income (Loss) and Selling, General and Administrative Expenses

In [4]:
```
s = data.columns.get_loc("Accumulated Other Comprehensive Income (Loss)")
d = data.columns.get_loc("Selling, General and Administrative Expenses")
```

Create a new data Frame with only one column ie.Data Date

In [5]:
```
new_data = data.iloc[:,1]
```

Create a dataframe with the columns mentioned in the question

In [6]:
```
new_data_1 = data.iloc[:,16:376]
```

Combine the two dataframe

In [7]:
```
frames = [new_data,new_data_1]
final_data = pd.concat(frames,axis = 1)
print(final_data.shape)
```

```
(844, 361)
```

Display the final data Frame

In [8]:
```
final_data
```

Out[8]:

	Data Date	Accumulated Other Comprehensive Income (Loss)	Current Assets - Other - Total	Current Assets - Total	Other Long-term Assets	Non-Current Assets - Total	Assets Netting & Other Adjustments	Accum Other Comp Inc - Derivatives Unrealized Gain/Loss	Other Inc - Adjust
0	2010-03-31	-1749.0	1502.0	8780.0	2568.0	21169.0	-299.0	-1312.0	
1	2010-06-30	-1603.0	1434.0	8296.0	2587.0	21204.0	-254.0	-1058.0	
2	2010-09-30	-1377.0	865.0	8839.0	2742.0	24646.0	-228.0	-962.0	
3	2010-12-31	-1159.0	1002.0	8780.0	2638.0	26616.0	-517.0	-786.0	
4	2011-03-31	-873.0	759.0	9436.0	2602.0	27201.0	-789.0	-688.0	
...	...	...	...	...	...	...	...	...	
839	2015-12-31	-318.0	183.0	9471.0	119.0	33644.0	-62.0	4.0	
840	2016-03-31	-318.0	204.0	8097.0	886.0	33661.0	-67.0	4.0	
841	2016-06-30	-321.0	142.0	10304.0	875.0	33829.0	-146.0	4.0	
842	2016-09-30	-327.0	176.0	9545.0	848.0	33748.0	-278.0	4.0	
843	2016-12-31	-234.0	236.0	10401.0	107.0	34012.0	-688.0	4.0	

844 rows × 361 columns

Add another column year to the dataframe

In [9]:
```
final_data['year'] = final_data['Data Date'].dt.year
final_data
```

Out[9]:

	Data Date	Accumulated Other Comprehensive Income (Loss)	Current Assets - Other - Total	Current Assets - Total	Other Long-term Assets	Non-Current Assets - Total	Assets Netting & Other Adjustments	Accum Other Comp Inc - Derivatives Unrealized Gain/Loss	Other Inc - Adjust

	Data Date	Accumulated Other Comprehensive Income (Loss)	Current Assets - Other - Total	Current Assets - Total	Other Long-term Assets	Non-Current Assets - Total	Assets Netting & Other Adjustments	Accum Other Comp Inc - Derivatives Unrealized Gain/Loss	A Other Inc - Adjust
0	2010-03-31	-1749.0	1502.0	8780.0	2568.0	21169.0	-299.0	-1312.0	
1	2010-06-30	-1603.0	1434.0	8296.0	2587.0	21204.0	-254.0	-1058.0	
2	2010-09-30	-1377.0	865.0	8839.0	2742.0	24646.0	-228.0	-962.0	
3	2010-12-31	-1159.0	1002.0	8780.0	2638.0	26616.0	-517.0	-786.0	
4	2011-03-31	-873.0	759.0	9436.0	2602.0	27201.0	-789.0	-688.0	
...	...	...	...	...	...	...	...	...	
839	2015-12-31	-318.0	183.0	9471.0	119.0	33644.0	-62.0	4.0	
840	2016-03-31	-318.0	204.0	8097.0	886.0	33661.0	-67.0	4.0	
841	2016-06-30	-321.0	142.0	10304.0	875.0	33829.0	-146.0	4.0	
842	2016-09-30	-327.0	176.0	9545.0	848.0	33748.0	-278.0	4.0	
843	2016-12-31	-234.0	236.0	10401.0	107.0	34012.0	-688.0	4.0	

844 rows × 362 columns

Create a split function to split the data frame into training and testing based the conditions mentioned in the question

In [10]:

```
def split(start,end):
    if end == None:
        test = final_data[final_data['year'] == start]
        train = final_data[final_data['year'] != start]
        return train.to_numpy(),test.to_numpy()
    else:
        test = final_data[(final_data['year'] >= start) & (final_data['year'] <=
        train = final_data[(final_data['year'] < start) | (final_data['year'] >
        return train.to_numpy(),test.to_numpy()
```

Returning numpy array of the dataframe

In [11]:
```
print('Start Year = 2012, End year = None',split(2012,None))
print('Start Year = 2010, End year = 2013',split(2010,2013))
```

```
Start Year = 2012, End year = None (array([[Timestamp('2010-03-31 00:00:00'), -1
749.0, 1502.0, ..., nan,
         559.0, 2010],
        [Timestamp('2010-06-30 00:00:00'), -1603.0, 1434.0, ..., nan,
         576.0, 2010],
        [Timestamp('2010-09-30 00:00:00'), -1377.0, 865.0, ..., nan,
         608.0, 2010],
        ...,
        [Timestamp('2016-06-30 00:00:00'), -321.0, 142.0, ..., nan, 401.0,
         2016],
        [Timestamp('2016-09-30 00:00:00'), -327.0, 176.0, ..., nan, 420.0,
         2016],
        [Timestamp('2016-12-31 00:00:00'), -234.0, 236.0, ..., nan, 406.0,
         2016]], dtype=object), array([[Timestamp('2012-03-31 00:00:00'), -1057.
0, 1421.0, ..., nan,
         385.0, 2012],
        [Timestamp('2012-06-30 00:00:00'), -779.0, 1699.0, ..., nan,
         340.0, 2012],
        [Timestamp('2012-09-30 00:00:00'), -675.0, 2167.0, ..., nan,
         1011.0, 2012],
        ...,
        [Timestamp('2012-06-30 00:00:00'), -497.0, 73.0, ..., nan, 365.0,
         2012],
        [Timestamp('2012-09-30 00:00:00'), -489.0, 139.0, ..., nan, 293.0,
         2012],
        [Timestamp('2012-12-31 00:00:00'), -464.0, 110.0, ..., nan, 314.0,
         2012]], dtype=object))
Start Year = 2010, End year = 2013 (array([[Timestamp('2014-03-31 00:00:00'), -2
76.0, 1480.0, ..., nan,
         231.0, 2014],
        [Timestamp('2014-06-30 00:00:00'), -384.0, 3022.0, ..., nan,
         467.0, 2014],
        [Timestamp('2014-09-30 00:00:00'), -487.0, 869.0, ..., nan, 212.0,
         2014],
        ...,
        [Timestamp('2016-06-30 00:00:00'), -321.0, 142.0, ..., nan, 401.0,
         2016],
        [Timestamp('2016-09-30 00:00:00'), -327.0, 176.0, ..., nan, 420.0,
         2016],
        [Timestamp('2016-12-31 00:00:00'), -234.0, 236.0, ..., nan, 406.0,
         2016]], dtype=object), array([[Timestamp('2010-03-31 00:00:00'), -1749.
0, 1502.0, ..., nan,
         559.0, 2010],
        [Timestamp('2010-06-30 00:00:00'), -1603.0, 1434.0, ..., nan,
         576.0, 2010],
        [Timestamp('2010-09-30 00:00:00'), -1377.0, 865.0, ..., nan,
         608.0, 2010],
        ...,
        [Timestamp('2013-06-30 00:00:00'), -254.0, 190.0, ..., nan, 358.0,
         2013],
        [Timestamp('2013-09-30 00:00:00'), -205.0, 206.0, ..., nan, 304.0,
         2013],
        [Timestamp('2013-12-31 00:00:00'), -204.0, 197.0, ..., nan, 336.0,
         2013]], dtype=object))
```

In [ ]:

Import Libraries

In [46]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
```

Reading the data using pd.read excel

In [47]:
```
data = pd.read_excel('ResearchDatasetV2.0.xlsx',parse_dates= True)
```

In [48]:
```
#data.head()
```

In [49]:
```
data['Date'] = pd.to_datetime(data['Date'], format='%Y%m%d')
```

In [50]:
```
data.describe()
```

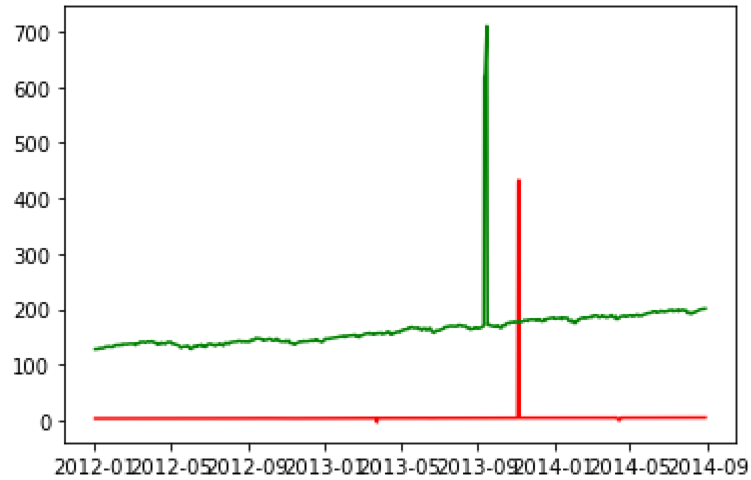
Out[50]:

	Signal	ClosePrice
count	667.000000	667.000000
mean	5.166802	163.169369
std	23.392809	39.210384
min	-3.802670	127.495000
25%	3.418083	140.880000
50%	3.893689	159.750000
75%	4.408313	181.500000
max	432.961165	710.310000

In [51]:
```
y = data['Signal'].tolist()
```

In [52]:
```
x = data['ClosePrice'].tolist()
```

In [53]:
```
plt.plot(data['Date'].tolist(), y, color='r', label='signal')
plt.plot(data['Date'].tolist(), x, color='g', label='closeprice')
plt.show()
```



Calculating signal returns and price returns

In [54]:

```
data['signal_return'] = data['Signal'].pct_change()
data['price_return'] = data['ClosePrice'].pct_change()
data.describe()
```

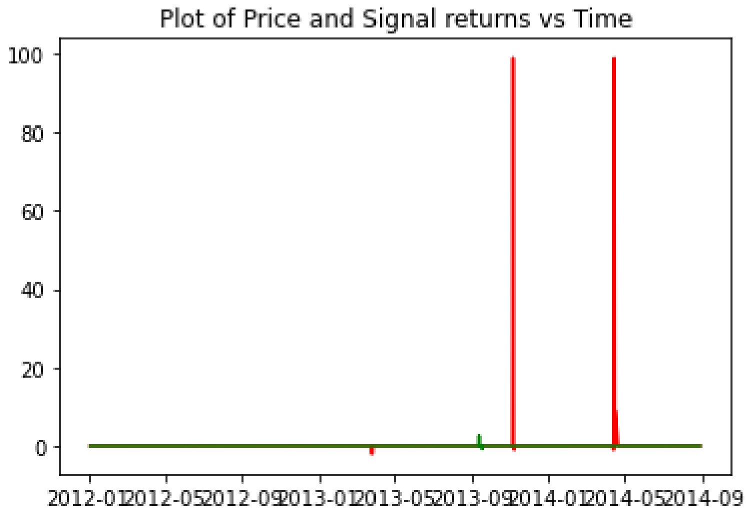
Out[54]:

	Signal	ClosePrice	signal_return	price_return
count	667.000000	667.000000	666.000000	666.000000
mean	5.166802	163.169369	0.301753	0.003760
std	23.392809	39.210384	5.421266	0.107373
min	-3.802670	127.495000	-2.003073	-0.759161
25%	3.418083	140.880000	-0.003357	-0.003199
50%	3.893689	159.750000	0.000111	0.000727
75%	4.408313	181.500000	0.005180	0.005070
max	432.961165	710.310000	98.796977	2.653778

Plotting signal return and price return against time

In [55]:

```
plt.plot(data['Date'].tolist(), data['signal_return'], color='r', label='signalr
plt.plot(data['Date'].tolist(), data['price_return'], color='g', label='closepri
plt.title('Plot of Price and Signal returns vs Time')
plt.show()
```



Define a function that caluclates cross correlation for a given lag

In [56]:
```
def cross_correlate(x, y, lag):
    return x.corr(y.shift(lag))
```

Find out cross correlation between price return and signal returns with given lags

In [57]:
```
cross_corrs = []
for lag in range(1, 11):
    cross_corrs.append(cross_correlate(data['price_return'],
                                      data['signal_return'], lag))
plt.plot(range(1,11), cross_corrs, marker='o')
plt.xlabel('Lags')
plt.ylabel('Correlation')
plt.title('Correlation between price returns and signal returns with lag k')
plt.show()
```

In [ ]:

The lag of 3 is significantly greater than the others. Therefore, we shall now use signals with a lag =3 for subsequent steps and trading strategies

Divide the signal returns into the required groups

In [58]:
```
grp1 = []
grp1_idx = []
grp2 = []
grp2_idx = []
grp3 = []
grp3_idx = []
grp4 = []
grp4_idx = []
for index in range(len(data['signal_return'])):
    if data['signal_return'][index] < -0.01:
        grp1.append(data['signal_return'][index])
        grp1_idx.append(index)
    elif data['signal_return'][index] >= -0.01 and data['signal_return'][index]
        grp2.append(data['signal_return'][index])
        grp2_idx.append(index)
    elif data['signal_return'][index] >= 0 and data['signal_return'][index] < 0.
        grp3.append(data['signal_return'][index])
        grp3_idx.append(index)
    elif data['signal_return'][index] >= 0.01:
        grp4.append(data['signal_return'][index])
        grp4_idx.append(index)
grp1_idx = [x+3 for x in grp1_idx if x+3 < len(data.index)]
grp2_idx = [x+3 for x in grp2_idx if x+3 < len(data.index)]
grp3_idx = [x+3 for x in grp3_idx if x+3 < len(data.index)]
grp4_idx = [x+3 for x in grp4_idx if x+3 < len(data.index)]
```

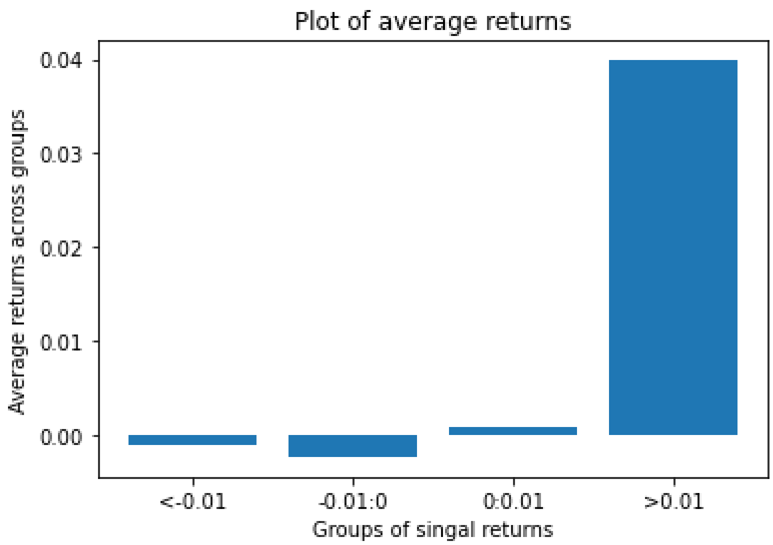
Calculate the average returns within the signal groups

In [59]:
```
average_returns = [np.mean(data['price_return'][grp1_idx]),
                   np.mean(data['price_return'][grp2_idx]),
                   np.mean(data['price_return'][grp3_idx]),
                   np.mean(data['price_return'][grp4_idx])]
grps = ['<-0.01', '-0.01:0', '0:0.01', '>0.01']
```

In [60]:
```
average_returns
```

Out[60]:
```
[-0.0011057692093343248,
 -0.0023702650368725526,
 0.0007304458878837299,
 0.03984880342589515]
```

In [61]:
```
plt.bar(grps, average_returns)
plt.title('Plot of average returns')
plt.xlabel('Groups of singal returns')
plt.ylabel('Average returns across groups')
plt.show()
```



Trading strategy

In [62]:
```
# Start with $100 and assuming the portfolio is self financing
value = [100]
investment_available = 100
data['buy_signal'] = pd.Series(np.zeros(len(data.index)))
data['sell_signal'] = pd.Series(np.zeros(len(data.index)))
invested = False
for i in range(len(data.index)):

    if data['buy_signal'][i] == 1:
        shares = investment_available/data['ClosePrice'][i]
        invested = True

    if data['sell_signal'][i] == 1:
        investment_available = shares * data['ClosePrice'][i]

    if data['signal_return'][i] < 0 and invested == False:
        data['buy_signal'][i+3] = 1


    if invested == False:
        value.append(value[i-1])

    if invested == True:
        value.append(shares*data['ClosePrice'][i])

    if data['sell_signal'][i] == 1:
        invested = False

    if data['signal_return'][i] > 0.01 and invested == True:
        data['sell_signal'][i+3] = 1
```

```
<ipython-input-62-a821866b31e3>:17: SettingWithCopyWarning:
```

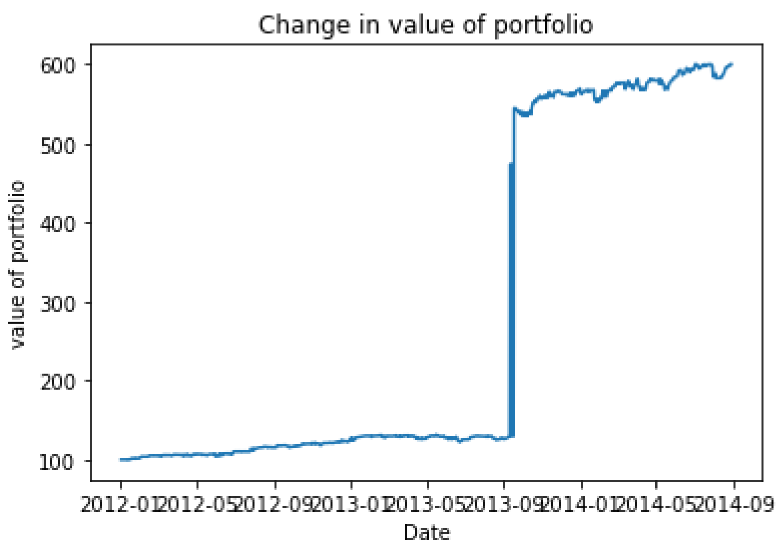
```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['buy_signal'][i+3] = 1
<ipython-input-62-a821866b31e3>:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['sell_signal'][i+3] = 1
```

In [63]:

```
# Plotting the value of the investment on every date of trading
plt.plot(data['Date'], value[1:])
plt.xlabel('Date')
plt.ylabel('value of portfolio')
plt.title('Change in value of portfolio')
plt.show()
```



In [ ]:

In [64]:

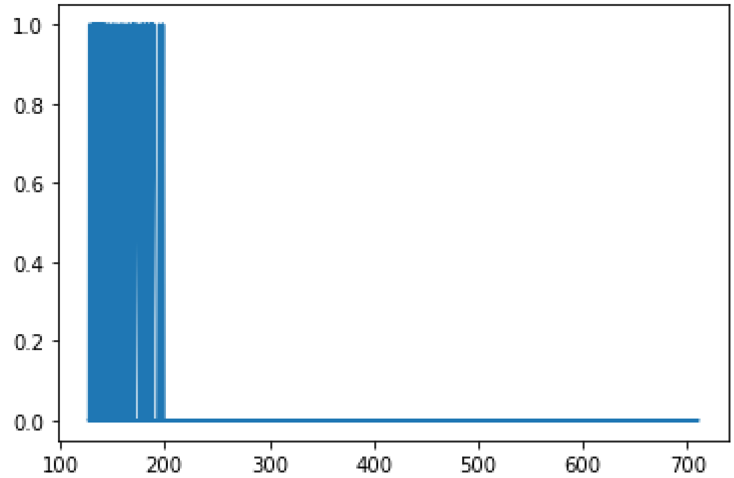
```
# Total return of the trading strat
total_return = value[-1]/value[0] -1
total_return
```

Out[64]: 4.997003279990955

In [65]:

```
# Plotting buy signals vs close prices
plt.plot(data['ClosePrice'], data['buy_signal'])
```

Out[65]: [<matplotlib.lines.Line2D at 0x7facc0a67490>]



The buy signal vs close price chart shows us that most of the buy signals are generated when the prices are low and thus allows the strategy to sell when prices are high and thus generating profits

In [66]:
```
# Calculating sharpe ratio
risk_free = 0.01 #1% rate of return on risk free asset
portfolio_return = total_return # Calculated above
std = np.std(pd.Series(value).pct_change())
```

In [67]:
```
sharpe = (total_return - risk_free) / std
```

In [68]:
```
sharpe
```

Out[68]: 25.612455172694524

In [ ]:

Import Numpy,Pandas and sklearn Libaries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import datasets,linear_model
```

Load the diabetes dataset from sklearn,datasets and store it in data variable

In [2]:
```
data = datasets.load_diabetes()
```

Store all features in X variable and target variable in y

In [3]:
```
X = data.data
y = data.target
print(X.shape,y.shape)
```

```
(442, 10) (442,)
```

Import train test split from sklearn

In [4]:
```
from sklearn.model_selection import train_test_split
```

Split the data into 80% training and 20% testing

In [5]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.20)
```

Display the shape of training and testing data

In [6]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(353, 10)
(353,)
(89, 10)
(89,)
```

Import Linear Regression Model from sklearn

In [7]:
```
model = linear_model.LinearRegression()
```

Fit the training data on the model created

In [8]:
```
model.fit(X_train,y_train)
```

Out[8]:
```
LinearRegression()
```

Predict the new value using X_test

In [9]:
```
pred = model.predict(X_test)
```

In [10]:
```
pred.shape
```

Out[10]:
```
(89,)
```

In [11]:
```
from sklearn.metrics import r2_score
```

Printing coefficiant of the Linear Model

In [12]:
```
coefficient = model.coef_
print(coefficient)
```

```
[ 1.55643605e+00 -2.25740184e+02  5.33913754e+02  3.52296498e+02
 -1.70562366e+03  1.23712616e+03  4.98728652e+02  2.27539505e+02
  1.11320829e+03  8.54283237e+01]
```

Printing r2 score of the model

In [13]:
```
r2_score = r2_score(y_test,pred)
print(r2_score)
```

```
0.34561086021089027
```

Import cross validation score

In [14]:
```
from sklearn.model_selection import cross_val_score
```

Number of fold is 10

In [15]:
```
kFold = 10
```

In [16]:
```
linearRegresssion = model.fit(X_train,y_train)
kRange = [1,2,3,4,5,6,7,8,9,10]
```

Fit the 10fold cross validation on the linear regression model

In [17]:
```
for kValue in kRange:
```

```
    value = cross_val_score(linearRegresssion,X,y,cv=10)
    print(value)
```

```
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
[0.55614411 0.23056092 0.35357777 0.62190498 0.26587602 0.61819338
 0.41815916 0.43515232 0.43436983 0.68568514]
```

In [18]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score
```

Create a Random Forest Regressor Model

In [19]:
```
regressor = RandomForestRegressor(max_depth = 7,random_state = 0)
```

Fit the model on the training data

In [20]:
```
regressor.fit(X_train,y_train)
```

Out[20]:
```
RandomForestRegressor(max_depth=7, random_state=0)
```

Predict the number of trees and its branches using the model created

In [21]:
```
pred = regressor.predict(X_test)
print(pred)
```

```
[185.27052066 143.55627812 115.50511006 235.2617613  176.40174057
 243.02450658 173.0402188  197.64982973  70.49279092 174.66402638
 204.97894727 182.8067521  122.67628262 166.43157852 224.40280386
 103.11521096  86.75757135  91.7552188  244.21037092  90.93747649
 178.9316034  138.24667567 106.16906969  93.34365891 136.45721918
 240.79274491  88.0581426  135.81957903  85.8576934  147.24348623
 222.4703159  276.54191401 117.8973772  115.31708581 182.04496887
 168.86479253  87.14606552 131.32618896 143.21240312 130.7515442
 141.78198554 105.21174504 259.65207935 222.05477651 114.68500617
 188.98276304  90.12445883 133.95228267 103.60737568 181.28533248
 273.08637968 168.24118344 125.83004161  90.49766364  76.73054682
 192.22963958 206.78790113  81.16709992 192.74866257  83.58999001
 215.71182135  96.08704253  95.3189499  176.97706097  86.59310825
```

```
138.40999992 238.96166803  86.50094618 154.36246705 179.10361932
 89.04511863 168.19406183 194.62913506 147.49948428 155.04514502
181.55549004 186.62280677  78.16786887 175.2520078  227.29163723
222.3635764   85.69971009 135.94517352  87.03427407 280.21921838
 84.59956975 109.05804095  76.40642567 107.94726216]
```

Print Random Forest Regressor score

In [22]:
```
regressor.score(X_train,y_train)
```

Out[22]:
```
0.8491454735874503
```

In [23]:
```
from sklearn.model_selection import GridSearchCV
```

Create parameters as mentioned in the question

In [24]:
```
parameters = {'max_depth': [None,7,4],'min_samples_split':[2,10,20]}
```

Create a model using grid search to esitmate the number of parameters

In [25]:
```
grid_GBR = GridSearchCV(estimator = regressor,param_grid = parameters,cv = 2,n_j
```

Fit the model on the train data

In [26]:
```
grid_GBR.fit(X_train,y_train)
```

Out[26]:
```
GridSearchCV(cv=2, estimator=RandomForestRegressor(max_depth=7, random_state=0),
             n_jobs=1,
             param_grid={'max_depth': [None, 7, 4],
                         'min_samples_split': [2, 10, 20]})
```

In [27]:
```
print('Results from Grid Search')
print('\nThe best estimator across All params is:',grid_GBR.best_estimator_)
print('\nThe best score across All params is:',grid_GBR.best_score_)
print('\nThe best parameter across All params is:',grid_GBR.best_params_)
```

```
Results from Grid Search

The best estimator across All params is: RandomForestRegressor(max_depth=4, rand
om_state=0)

The best score across All params is: 0.398724581536767

The best parameter across All params is: {'max_depth': 4, 'min_samples_split':
2}
```

In [ ]: