Diamonds Project

Yumna Al Shalak

THE DATASET :DIAMONDS

- This classic dataset contains the prices and other attributes of almost 54,000 diamonds. It's a great dataset for beginners learning to work with data analysis and visualization.

- price price in US dollars (\$326--\$18,823)
- carat weight of the diamond (0.2--5.01)
- cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- color diamond colour, from J (worst) to D (best)
- clarity a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
- x length in mm (0--10.74)
- y width in mm (0--58.9)
- z depth in mm (0--31.8)
- depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)
- table width of top of diamond relative to widest point (43--95)

The problem is Regression

Type of machine learning system: Supervised - Batch Learning - Model Based

GET THE DATA:

```
# imports the main libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
diamonds_data = pd.read_csv('diamonds.csv')
diamonds_data.head()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
# drop Unnamed: 0 colunms
diamonds_data.drop(['Unnamed: 0'],axis=1,inplace=True)
```

DISCOVER AND VISUALIZE THE DATA

Data discovery:

from data information we don't have any null value in our data

```
diamonds_data.info()
```

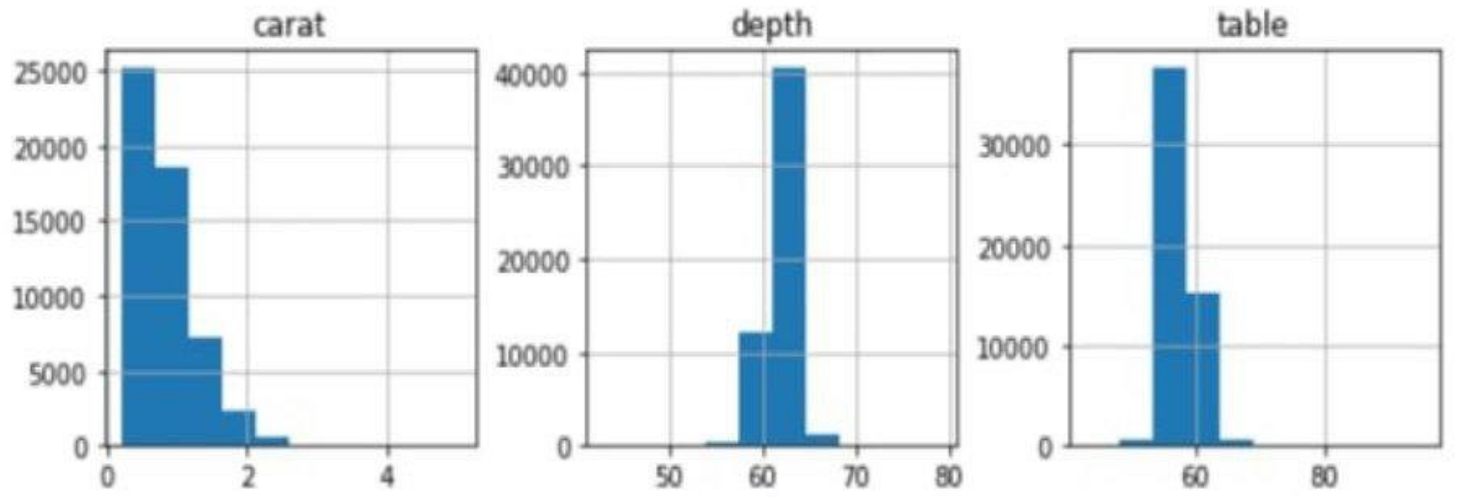
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17858 entries, 0 to 17857
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    17858 non-null  float64
 1   cut      17858 non-null  object
 2   color    17858 non-null  object
 3   clarity  17858 non-null  object
 4   depth    17858 non-null  float64
 5   table    17858 non-null  float64
 6   price    17858 non-null  int64
 7   x        17858 non-null  float64
 8   y        17858 non-null  float64
 9   z        17857 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 1.4+ MB
```

DISCOVER AND VISUALIZE THE DATA

Data visualization:

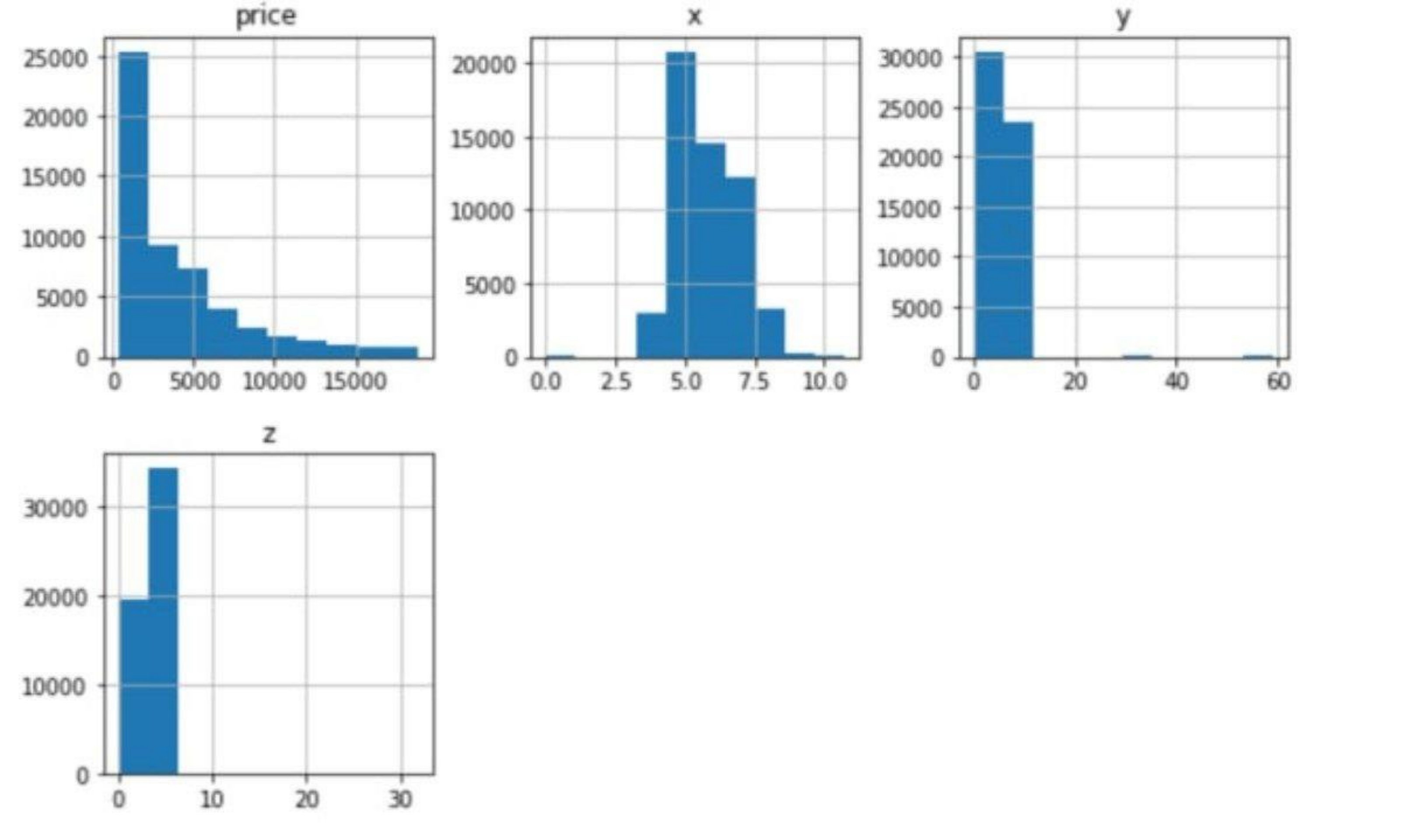
Create a hist plot for diamonds dataframe as shown down

```
[51] diamonds_data.hist(figsize=(10, 10))
     plt.show()
```



Discover and visualize the data

Data visualization:



DISCOVER AND VISUALIZE THE DATA

Data visualization:

1.Carat has a strong relationship to price

2.x, y and z have a very strong relationship with price but surprisingly depth (which comes from x, y and z) has little to do with price.

```
fig, ax = plt.subplots(figsize=(11, 9))
sns.heatmap(diamonds_data.corr(),annot=True)
plt.show()
```

TITLE PREPARE THE DATA FOR MACHINE LEARNING ALGORITHMS

The minimum values for x,y and z here are 0 but it is not possible because according to the data description they are the length, width and depth

```
diamonds_data.describe()
```

	carat	depth	table	price	x	y	z
count	17858.000000	17858.000000	17858.000000	17858.000000	17858.000000	17858.000000	17857.000000
mean	0.930882	61.833744	57.781504	4210.086628	6.171439	6.170635	3.813921
std	0.278732	1.578816	2.244017	1675.393211	0.736743	0.724990	0.462995
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.800000	61.000000	56.000000	3297.000000	5.940000	5.950000	3.670000
50%	1.000000	61.900000	58.000000	4318.000000	6.360000	6.360000	3.940000
75%	1.070000	62.700000	59.000000	5367.750000	6.600000	6.590000	4.070000
max	3.000000	71.800000	70.000000	7204.000000	9.230000	9.100000	5.770000

TITLE PREPARE THE DATA FOR MACHINE LEARNING ALGORITHMS

A zero value in these rows means that data is lost, so we can replace the zeros with nan after that we will drop nan values .

```
print(f"Number of rows with x == 0: {sum(diamonds_data.x==0)} ")
print(f"Number of rows with y == 0: {sum(diamonds_data.y==0)} ")
print(f"Number of rows with z == 0: {sum(diamonds_data.z==0)} ")
print(f"Number of rows with depth == 0: {sum(diamonds_data.depth==0)} ")
```

```
Number of rows with x == 0: 3
Number of rows with y == 0: 2
Number of rows with z == 0: 9
Number of rows with depth == 0: 0
```

TITLE PREPARE THE DATA FOR MACHINE LEARNING ALGORITHMS

```
diamonds_data[['x','y','z']] = diamonds_data[['x','y','z']].replace(0,np.NaN)
```

```
# Treatment of data for missing values
diamonds_data.dropna(inplace=True)
diamonds_data.isnull().sum()
```

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

```
Q1 = diamonds_data.price.quantile(0.25) #detecting outliers
Q3 = diamonds_data.price.quantile(0.75)
print(Q1,Q3)
IQR = Q3 - Q1
IQR
```

```
3297.0 5367.25
2070.25
```

```
lower_limit = Q1 - 1.5*IQR
upper_limit = Q3 + 1.5*IQR
lower_limit, upper_limit
```

```
(191.625, 8472.625)
```

TITLE PREPARE THE DATA FOR MACHINE LEARNING ALGORITHMS

```
diamonds_data_no_outlier = diamonds_data[(diamonds_data.price>lower_limit)&(diamonds_data.price<upper_limit)]  #removing outliers
diamonds_data_no_outlier.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

ONE HOT ENCODING

```
from sklearn.preprocessing import OneHotEncoder
```

```
ohe = OneHotEncoder()
new_diamonds = diamonds_data_no_outlier.copy()
data_encoder = ohe.fit_transform(new_diamonds[['cut','color','clarity']])
data = pd.DataFrame(data_encoder.toarray())
```

```
new_diamonds.drop(['cut','color','clarity'],axis=1,inplace=True)
```

```
#To use join method we have to have the same index in the both data frames.
print(new_diamonds.shape)
new_diamonds.index = np.arange(0,17848)
```

```
(17848, 7)
```

```
diamonds = new_diamonds.join(data)
diamonds.head()
```

	carat	depth	table	price	x	y	z	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0.23	61.5	55.0	326	3.95	3.98	2.43	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.21	59.8	61.0	326	3.89	3.84	2.31	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.23	56.9	65.0	327	4.05	4.07	2.31	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
3	0.29	62.4	58.0	334	4.20	4.23	2.63	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	0.31	63.3	58.0	335	4.34	4.35	2.75	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

DATA SCALING

```
from sklearn.preprocessing import StandardScaler
```

```
scale = StandardScaler()
data_scale = scale.fit_transform(diamonds[['carat','depth','table','x','y','z']])
```

```
scaling =  pd.DataFrame(data_scale,columns=['carat','depth','table','x','y','z'],index=diamonds.index)
```

```
scaling.head()
```

	carat	depth	table	x	y	z
0	-2.514084	-0.211919	-1.239890	-3.033936	-3.034227	-3.045101
1	-2.585836	-1.289165	1.435287	-3.115851	-3.228099	-3.308775
2	-2.514084	-3.126820	3.218739	-2.897410	-2.909595	-3.308775
3	-2.298828	0.358387	0.097699	-2.692622	-2.688027	-2.605643
4	-2.227076	0.928694	0.097699	-2.501486	-2.521851	-2.341968

```
scale_diamond = diamonds.copy()
scale_diamond.drop(['carat','depth','table','x','y','z'],axis=1,inplace = True)
```

```
#To use join method we have to have the same index in the both data frames.
print(scale_diamond.shape)
scale_diamond.index = np.arange(0,17848)
```

```
(17848, 21)
```

DATA SCALING

```
diamonds =  scaling.join(scale_diamond)
diamonds.head()
```

	carat	depth	table	x	y	z	price	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	-2.514084	-0.211919	-1.239890	-3.033936	-3.034227	-3.045101	326	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	-2.585836	-1.289165	1.435287	-3.115851	-3.228099	-3.308775	326	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	-2.514084	-3.126820	3.218739	-2.897410	-2.909595	-3.308775	327	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	-2.298828	0.358387	0.097699	-2.692622	-2.688027	-2.605643	334	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
4	-2.227076	0.928694	0.097699	-2.501486	-2.521851	-2.341968	335	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0

MODEL TRAINING

Split diamond data into train and test set

```
from sklearn.model_selection import train_test_split

train_set ,test_set = train_test_split(diamonds,test_size = 0.2,random_state = 42 )
```

MODEL TRAINING

Split diamond data into data & labels

```
train_diamonds = train_set.drop(['price'],axis=1)
train_labels = train_set['price']

test_diamonds = test_set.drop(['price'],axis=1)
test_labels = test_set['price']
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

def MAE_MSE_RMSE(model,diam_data,labels):
  y_pred = model.predict(diam_data)

  MAE = mean_absolute_error(labels, y_pred)
  print(f'Mean Absolute Error MAE = {MAE}')

  MSE = mean_squared_error(labels, y_pred)
  print(f'Mean Squared Error MSE = {MSE}')

  RMSE = np.sqrt(MSE)
  print(f'Root Mean Squared Error RMSE = {RMSE}')
```

MODEL TRAINING

Regression model

```
from sklearn.linear_model import LinearRegression
```

```
lin_reg_model = LinearRegression()
lin_reg_model.fit(train_diamonds,train_labels)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

MODEL TRAINING

Evaluate model:
MAE & MSE & RMSE:

```
MAE_MSE_RMSE(lin_reg_model,test_diamonds,test_labels)
```

```
Mean Absolute Error MAE = 431.5976857965423
Mean Squared Error MSE = 310780.5499288162
Root Mean Squared Error RMSE = 557.4769501323048
```

MODEL TRAINING

Random forest model:

```python
from sklearn.ensemble import RandomForestRegressor
```

```python
rand_reg_model = RandomForestRegressor()
rand_reg_model.fit(train_diamonds,train_labels)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

MODEL TRAINING

Evaluate model:
MAE & MSE & RMSE:

```
MAE_MSE_RMSE(rand_reg_model,test_diamonds,test_labels)
```

```
Mean Absolute Error MAE = 271.35722178538083
Mean Squared Error MSE = 151037.7730677531
Root Mean Squared Error RMSE = 388.63578459497666
```

MODEL TRAINING

DecisionTree Regressor model:

```
from sklearn.tree import DecisionTreeRegressor
```

```
dec_reg_model = DecisionTreeRegressor()
dec_reg_model.fit(train_diamonds,train_labels)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

MODEL TRAINING

Evaluate model
MAE & MSE & RMSE

```
MAE_MSE_RMSE(dec_reg_model,test_diamonds,test_labels)

Mean Absolute Error MAE = 366.30322128851543
Mean Squared Error MSE = 280292.79586834734
Root Mean Squared Error RMSE = 529.4268560135076
```

MODEL TRAINING

Fine Tuning The model

Fine Tuning The model

will fine tune for Random forest model beacuse there have best RMSE

```
[47] from sklearn.model_selection import GridSearchCV
```

```
[48] param_grid ={'n_estimators':[600,750, 1000], 'max_features':[10,15,20]}
```

```
[49] gridsearch_rand_reg_model = RandomForestRegressor()
     grid_search = GridSearchCV(gridsearch_rand_reg_model,param_grid,cv=5,scoring='neg_mean_squared_error',return_train_score=True)
```

```
grid_search.fit(train_diamonds,train_labels)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
```

MODEL TRAINING

Fine Tuning The model

```
[51] grid_search.best_params_

     {'max_features': 15, 'n_estimators': 750}
```

```
[52] grid_search.best_estimator_

     RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                           max_depth=None, max_features=15, max_leaf_nodes=None,
                           max_samples=None, min_impurity_decrease=0.0,
                           min_impurity_split=None, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=750, n_jobs=None, oob_score=False,
                           random_state=None, verbose=0, warm_start=False)
```

```
[54] grid_search_random_forest_model = RandomForestRegressor(n_estimators=750,max_features=15)
     grid_search_random_forest_model.fit(train_diamonds,train_labels)

     RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                           max_depth=None, max_features=15, max_leaf_nodes=None,
                           max_samples=None, min_impurity_decrease=0.0,
                           min_impurity_split=None, min_samples_leaf=1,
                           min_samples_split=2, min_weight_fraction_leaf=0.0,
                           n_estimators=750, n_jobs=None, oob_score=False,
                           random_state=None, verbose=0, warm_start=False)
```

MODEL TRAINING

Evaluate model
MAE & MSE & RMSE

Evaluate model

MAE & MSE & RMSE

```
[56] MAE_MSE_RMSE(grid_search_random_forest_model,test_diamonds,test_labels)

Mean Absolute Error MAE = 204.5345131194146
Mean Squared Error MSE = 138078.0322142499
Root Mean Squared Error RMSE = 371.58852540713616
```