```
import pandas as pd

import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

import pylab
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing

df = pd.read_csv('uber.csv')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         200000 non-null  int64  
 1   key                200000 non-null  object 
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object 
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

```
df.head()
```

```
   Unnamed: 0                            key  fare_amount  \
0    24238194    2015-05-07 19:52:06.0000003          7.5   
1    27835199    2009-07-17 20:04:56.0000002          7.7   
2    44984355   2009-08-24 21:45:00.00000061         12.9   
3    25894730    2009-06-26 08:22:21.0000001          5.3   
4    17610152  2014-08-28 17:47:00.000000188         16.0   

           pickup_datetime  pickup_longitude  pickup_latitude  \
0  2015-05-07 19:52:06 UTC        -73.999817        40.738354   
1  2009-07-17 20:04:56 UTC        -73.994355        40.728225   
2  2009-08-24 21:45:00 UTC        -74.005043        40.740770   
3  2009-06-26 08:22:21 UTC        -73.976124        40.790844   
4  2014-08-28 17:47:00 UTC        -73.925023        40.744085   
```

```
   dropoff_longitude  dropoff_latitude  passenger_count
0         -73.999512         40.723217                1
1         -73.994710         40.750325                1
2         -73.962565         40.772647                1
3         -73.965316         40.803349                3
4         -73.973082         40.761247                5
```

```
df.describe()
```

```
         Unnamed: 0    fare_amount  pickup_longitude  pickup_latitude
\
count  2.000000e+05  200000.000000     200000.000000    200000.000000

mean   2.771250e+07      11.359955        -72.527638        39.935885

std    1.601382e+07       9.901776         11.437787         7.720539

min    1.000000e+00     -52.000000      -1340.648410       -74.015515

25%    1.382535e+07       6.000000        -73.992065        40.734796

50%    2.774550e+07       8.500000        -73.981823        40.752592

75%    4.155530e+07      12.500000        -73.967154        40.767158

max    5.542357e+07     499.000000         57.418457      1644.421482
```

```
       dropoff_longitude  dropoff_latitude  passenger_count
count      199999.000000     199999.000000    200000.000000
mean          -72.525292         39.923890         1.684535
std            13.117408          6.794829         1.385997
min         -3356.666300       -881.985513         0.000000
25%           -73.991407         40.733823         1.000000
50%           -73.980093         40.753042         1.000000
75%           -73.963658         40.768001         2.000000
max          1153.572603        872.697628       208.000000
```

```
df = df.drop(['Unnamed: 0', 'key'], axis=1)
```

```
df.isna().sum()
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

```
df.dropna(axis=0,inplace=True)

df.dtypes
```

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime,
errors='coerce')

df= df.assign(
    second = df.pickup_datetime.dt.second,
    minute = df.pickup_datetime.dt.minute,
    hour = df.pickup_datetime.dt.hour,
    day= df.pickup_datetime.dt.day,
    month = df.pickup_datetime.dt.month,
    year = df.pickup_datetime.dt.year,
    dayofweek = df.pickup_datetime.dt.dayofweek
)
df = df.drop('pickup_datetime',axis=1)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 199999 entries, 0 to 199999
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   fare_amount        199999 non-null  float64
 1   pickup_longitude   199999 non-null  float64
 2   pickup_latitude    199999 non-null  float64
 3   dropoff_longitude  199999 non-null  float64
 4   dropoff_latitude   199999 non-null  float64
 5   passenger_count    199999 non-null  int64
 6   second             199999 non-null  int32
 7   minute             199999 non-null  int32
 8   hour               199999 non-null  int32
 9   day                199999 non-null  int32
 10  month              199999 non-null  int32
 11  year               199999 non-null  int32
 12  dayofweek          199999 non-null  int32
dtypes: float64(5), int32(7), int64(1)
memory usage: 16.0 MB
```

```
df.head()
```

```
   fare_amount  pickup_longitude  pickup_latitude
dropoff_longitude  \
0          7.5        -73.999817        40.738354         -73.999512

1          7.7        -73.994355        40.728225         -73.994710

2         12.9        -74.005043        40.740770         -73.962565

3          5.3        -73.976124        40.790844         -73.965316

4         16.0        -73.925023        40.744085         -73.973082


   dropoff_latitude  passenger_count  second  minute  hour  day  month
year  \
0         40.723217                1       6      52    19    7      5
2015
1         40.750325                1      56       4    20   17      7
2009
2         40.772647                1       0      45    21   24      8
2009
3         40.803349                3      21      22     8   26      6
2009
4         40.761247                5       0      47    17   28      8
2014

   dayofweek
0          3
1          4
2          0
3          4
4          3
```

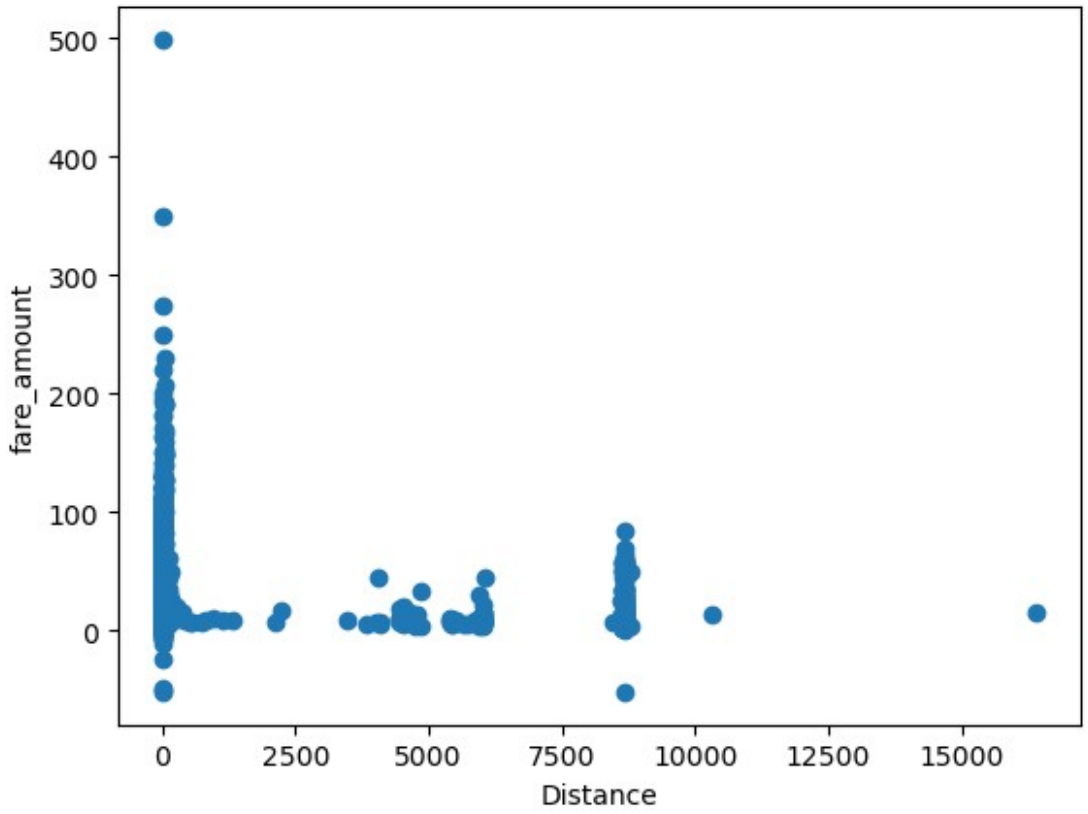
```
incorrect_coordinates = df.loc[
    (df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
    (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
    (df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
    (df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')

def distance_transform(longitude1, latitude1, longitude2, latitude2):
    long1, lati1, long2, lati2 = map(np.radians, [longitude1,
latitude1, longitude2, latitude2])
    dist_long = long2 - long1
    dist_lati = lati2 - lati1
    a = np.sin(dist_lati/2)**2 + np.cos(lati1) * np.cos(lati2) *
np.sin(dist_long/2)**2
    c = 2 * np.arcsin(np.sqrt(a)) * 6371
```

```
    # long1,lati1,long2,lati2 =
longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]
    # c = sqrt((long2 - long1) ** 2 + (lati2 - lati1) ** 2)asin

    return c

df['Distance'] = distance_transform(
    df['pickup_longitude'],
    df['pickup_latitude'],
    df['dropoff_longitude'],
    df['dropoff_latitude']
)

df.head()
```

```
   fare_amount  pickup_longitude  pickup_latitude
dropoff_longitude  \
0          7.5        -73.999817        40.738354         -73.999512

1          7.7        -73.994355        40.728225         -73.994710

2         12.9        -74.005043        40.740770         -73.962565

3          5.3        -73.976124        40.790844         -73.965316

4         16.0        -73.925023        40.744085         -73.973082


   dropoff_latitude  passenger_count  second  minute  hour  day  month
year  \
0         40.723217                1       6      52    19    7      5
2015
1         40.750325                1      56       4    20   17      7
2009
2         40.772647                1       0      45    21   24      8
2009
3         40.803349                3      21      22     8   26      6
2009
4         40.761247                5       0      47    17   28      8
2014

   dayofweek  Distance
0          3  1.683323
1          4  2.457590
2          0  5.036377
3          4  1.661683
4          3  4.475450
```

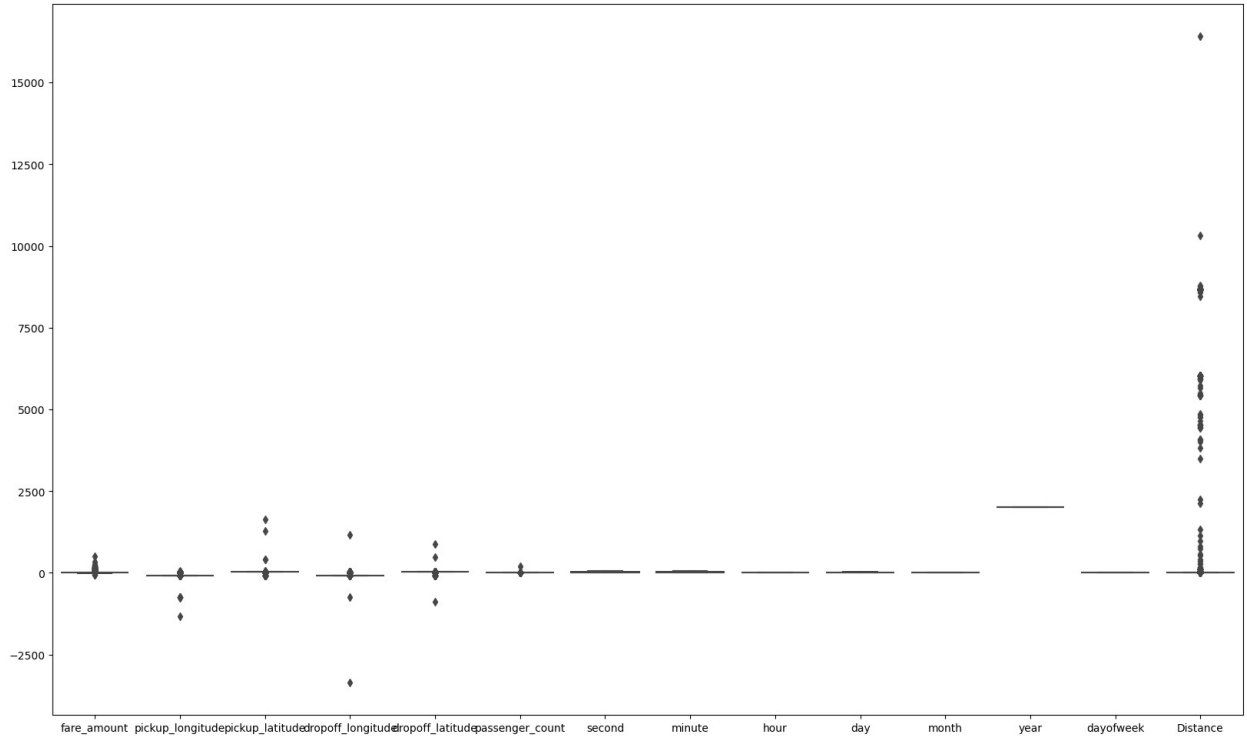
```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

```
Text(0, 0.5, 'fare_amount')
```

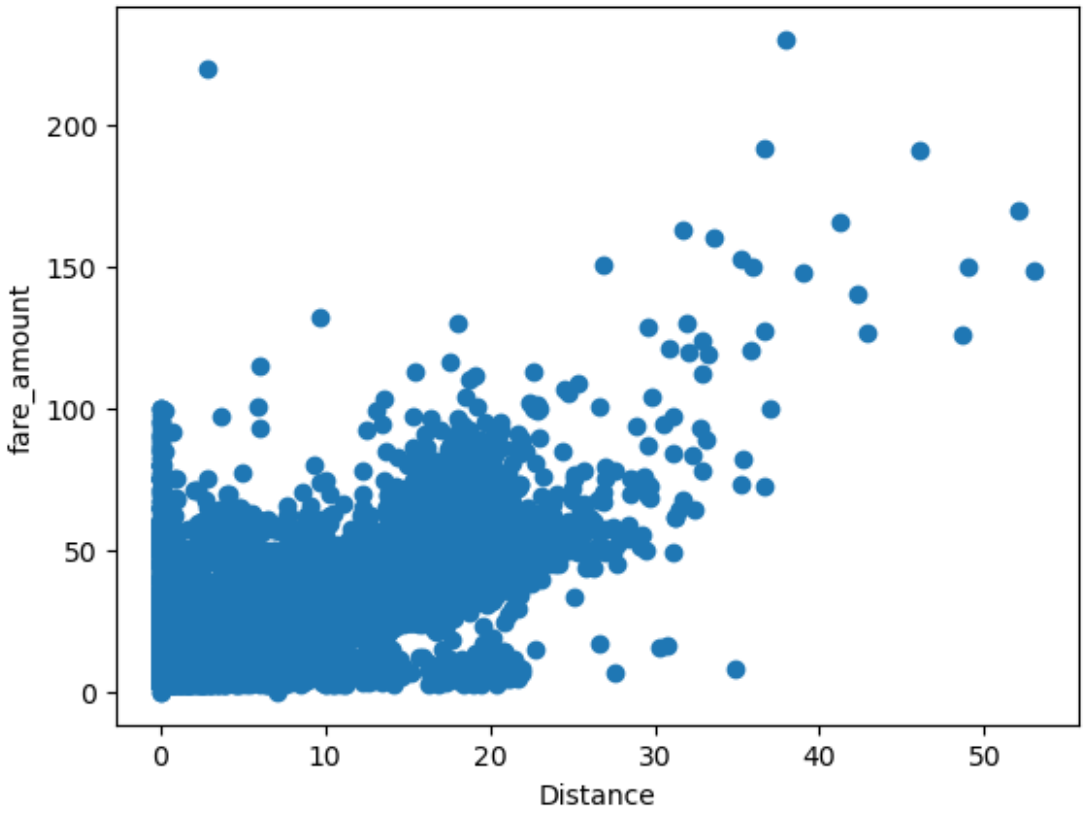



```
plt.figure(figsize=(20,12))
sns.boxplot(data = df)
```

```
<Axes: >
```




```
df.drop(df[df['Distance'] >= 60].index, inplace = True)
df.drop(df[df['fare_amount'] <= 0].index, inplace = True)

df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index,
inplace = True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index,
inplace = True )

plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")

Text(0, 0.5, 'fare_amount')
```




```
corr = df.corr()

corr.style.background_gradient(cmap='BuGn')
```

```
<pandas.io.formats.style.Styler at 0x1638b9d5010>
```

```
X = df['Distance'].values.reshape(-1, 1)        #Independent Variable
y = df['fare_amount'].values.reshape(-1, 1)     #Dependent Variable

from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(y)
print(y_std)

x_std = std.fit_transform(X)
print(x_std)
```

```
[[-0.39820843]
 [-0.37738556]
 [ 0.1640092 ]
 ...
 [ 2.03806797]
 [ 0.3305922 ]
 [ 0.28894645]]
[[-0.43819769]
```

```
 [-0.22258873]
 [ 0.49552213]
 ...
 [ 2.67145829]
 [ 0.07874908]
 [ 0.60173174]]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std,
test_size=0.2, random_state=0)

from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train,
y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

```
Training set score: 0.74
Test set score: 0.7340468
```

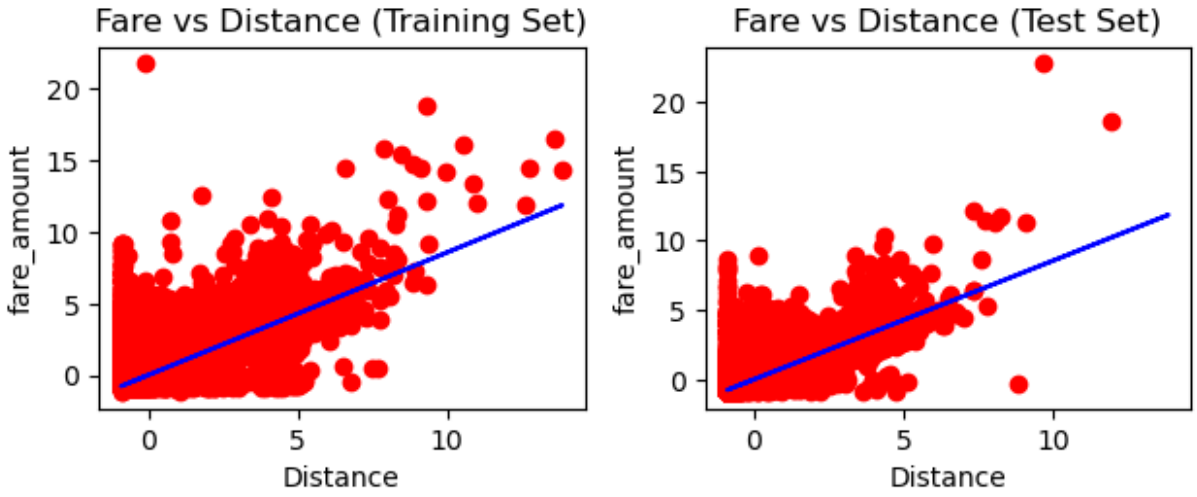
```
y_pred = l_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result[['Predicted']] = y_pred

result.sample(10)
```

```
         Actual  Predicted
2281   1.267622   0.333371
9361  -0.450266  -0.292862
7857  -0.242037  -0.141543
5043   0.622112   0.655647
23082 -0.543969  -0.264608
254   -0.502323  -0.491023
35286 -0.419031  -0.013399
2239  -0.210803  -0.327257
9881  -0.866723  -0.745827
35264 -0.085865  -0.310878
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Absolute % Error:',
metrics.mean_absolute_percentage_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred)))
```

```
Mean Absolute Error: 0.26621298757938944
Mean Absolute % Error: 1.98307476334074
Mean Squared Error: 0.27052435107785416
Root Mean Squared Error: 0.5201195546005304
R Squared (R²): 0.8567653080822022
```

```
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")

plt.tight_layout()
plt.show()
```




```
cols = ['Model', 'RMSE', 'R-Squared']

# create a empty dataframe of the colums
# columns: specifies the columns to be selected
result_tabulation = pd.DataFrame(columns = cols)

# compile the required information
linreg_metrics = pd.DataFrame([[
     "Linear Regresion model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
```

```
     np.sqrt(metrics.r2_score(y_test, y_pred))
]], columns = cols)

result_tabulation = pd.concat([result_tabulation, linreg_metrics],
ignore_index=True)

result_tabulation
```

```
C:\Users\vaish\AppData\Local\Temp\ipykernel_8816\1464293630.py:14:
FutureWarning: The behavior of DataFrame concatenation with empty or
all-NA entries is deprecated. In a future version, this will no longer
exclude empty or all-NA columns when determining the result dtypes. To
retain the old behavior, exclude the relevant entries before the
concat operation.
  result_tabulation = pd.concat([result_tabulation, linreg_metrics],
ignore_index=True)
```

	Model	RMSE	R-Squared
0	Linear Regresion model	0.52012	0.856765

```
rf_reg = RandomForestRegressor(n_estimators=100, random_state=10)

# fit the regressor with training dataset
rf_reg.fit(X_train, y_train)
```

```
C:\Users\vaish\AppData\Local\Temp\ipykernel_8816\2264728154.py:4:
DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples,), for
example using ravel().
  rf_reg.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=10)

# predict the values on test dataset using predict()
y_pred_RF = rf_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result['Predicted'] = y_pred_RF

result.sample(10)
```

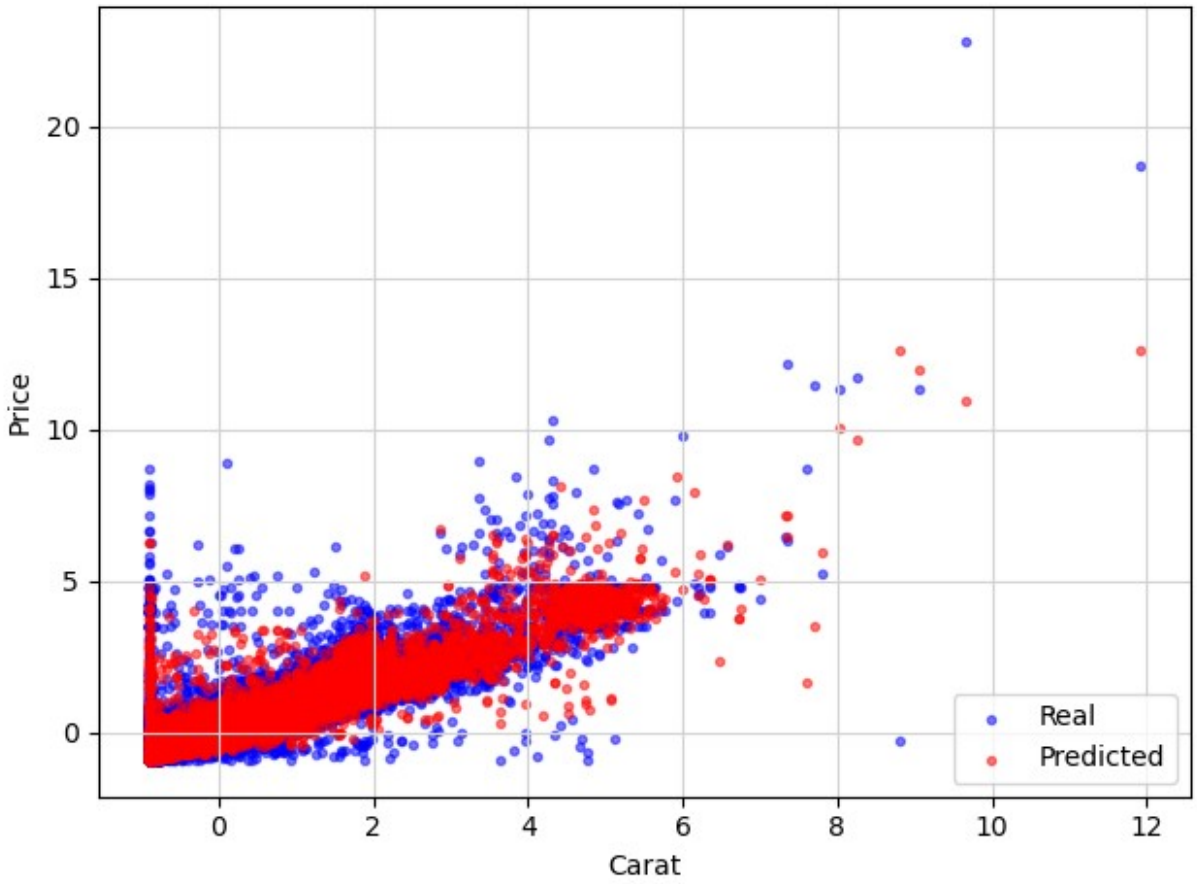
	Actual	Predicted
17623	-0.460677	-0.483062
1816	-0.294094	0.562663
4063	-0.627260	-0.466091
5690	-0.346151	-0.465570
35811	-0.242037	0.195035
9945	-0.835489	-0.776456
25814	-0.033808	0.569535
2651	1.663256	1.952590

```
35914 -0.627260  -0.317416
4641  -0.658494  -0.712217
```

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred_RF))
print('Mean Absolute % Error:',
metrics.mean_absolute_percentage_error(y_test, y_pred_RF))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred_RF))
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred_RF)))
```

```
Mean Absolute Error: 0.3077087698385678
Mean Absolute % Error: 2.161623761570947
Mean Squared Error: 0.33297733033643484
Root Mean Squared Error: 0.5770418791876677
R Squared (R²): 0.8201518783882692
```

```
# Build scatterplot
plt.scatter(X_test, y_test, c = 'b', alpha = 0.5, marker = '.', label
= 'Real')
plt.scatter(X_test, y_pred_RF, c = 'r', alpha = 0.5, marker = '.',
label = 'Predicted')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.grid(color = '#D3D3D3', linestyle = 'solid')
plt.legend(loc = 'lower right')


plt.tight_layout()
plt.show()
```




```
# compile the required information
random_forest_metrics = pd.DataFrame([[
     "Random Forest Regressor model",
     np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)),
     np.sqrt(metrics.r2_score(y_test, y_pred_RF))
]], columns = cols)

result_tabulation = pd.concat([result_tabulation,
random_forest_metrics], ignore_index=True)

result_tabulation
```

	Model	RMSE	R-Squared
0	Linear Regresion model	0.520120	0.856765
1	Random Forest Regressor model	0.577042	0.820152