In [ ]:

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

## Loading the Dataset

#### First we load the dataset and ﬁnd out the number of columns, rows, NULL values, etc.

In [ ]:

df = pd.read\_csv('uber.csv')

In [ ]:

df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 200000 entries, 0 to 199999 Data columns (total 9 columns):

# Column Non-Null Count Dtype

1. Unnamed: 0 200000 non-null int64
2. key 200000 non-null object
3. fare\_amount 200000 non-null float64
4. pickup\_datetime 200000 non-null object
5. pickup\_longitude 200000 non-null float64
6. pickup\_latitude 200000 non-null float64
7. dropoff\_longitude 199999 non-null float64
8. dropoff\_latitude 199999 non-null float64
9. passenger\_count 200000 non-null int64 dtypes: float64(5), int64(2), object(2)

memory usage: 13.7+ MB

df.head()

In [ ]:

Out[ ]:

Unnamed:

0

key fare\_amount pickup\_datetime pickup\_longitude pickup\_latitude dropoﬀ\_longitude dropoﬀ\_latitude pa

0 24238194 2015-05-07

19:52:06.0000003

7.5 2015-05-07

19:52:06 UTC

-73.999817 40.738354 -73.999512 40.723217

1 27835199

2009-07-17

20:04:56.0000002

7.7

2009-07-17

20:04:56 UTC

-73.994355

40.728225

-73.994710

40.750325

2 44984355 2009-08-24

21:45:00.00000061

12.9 2009-08-24

21:45:00 UTC

-74.005043 40.740770 -73.962565 40.772647

3 25894730

2009-06-26

08:22:21.0000001

5.3

2009-06-26

08:22:21 UTC

-73.976124

40.790844

-73.965316

40.803349

4 17610152 2014-08-28

17:47:00.000000188

16.0 2014-08-28

17:47:00 UTC

-73.925023 40.744085 -73.973082 40.761247

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In [ ]: | df.describe() |  | | | | | | |
| Out[ ]: | Unnamed: 0 | fare\_amount | pickup\_longitude | pickup\_latitude | dropoﬀ\_longitude | dropoﬀ\_latitude | passenger\_count |  |
|  | count 2.000000e+05 | 200000.000000 | 200000.000000 | 200000.000000 | 199999.000000 | 199999.000000 | 200000.000000 |  |
|  | mean 2.771250e+07 | 11.359955 | -72.527638 | 39.935885 | -72.525292 | 39.923890 | 1.684535 |  |
|  | std 1.601382e+07 | 9.901776 | 11.437787 | 7.720539 | 13.117408 | 6.794829 | 1.385997 |  |
|  | min 1.000000e+00 | -52.000000 | -1340.648410 | -74.015515 | -3356.666300 | -881.985513 | 0.000000 |  |
|  | 25% 1.382535e+07 | 6.000000 | -73.992065 | 40.734796 | -73.991407 | 40.733823 | 1.000000 |  |
|  | 50% 2.774550e+07 | 8.500000 | -73.981823 | 40.752592 | -73.980093 | 40.753042 | 1.000000 |  |
|  | 75% 4.155530e+07 | 12.500000 | -73.967154 | 40.767158 | -73.963658 | 40.768001 | 2.000000 |  |
|  | max 5.542357e+07 | 499.000000 | 57.418457 | 1644.421482 | 1153.572603 | 872.697628 | 208.000000 |  |

## Cleaning

In [ ]:

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

In [ ]:

|  |  |
| --- | --- |
| df.isna().sum() |  |
| fare\_amount | 0 |
| pickup\_datetime | 0 |
| pickup\_longitude | 0 |
| pickup\_latitude | 0 |
| dropoff\_longitude | 1 |
| dropoff\_latitude | 1 |
| passenger\_count dtype: int64 | 0 |
| Remove null rows |  |

Out[ ]:

In [ ]:

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

In [ ]:

Out[ ]:

In [ ]:

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

### Fix data type of pickup\_datetime from Object to DateTime

df.pickup\_datetime = pd.to\_datetime(df.pickup\_datetime, errors='coerce')

Separating the date and time into separate columns for more usability.

In [ ]:

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)

In [ ]:

df.info()

<class 'pandas.core.frame.DataFrame'> Int64Index: 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 | int64 |
| 7 |  | minute | 199999 non-null | int64 |
| 8 |  | hour | 199999 non-null | int64 |
| 9 |  | day | 199999 non-null | int64 |
| 10 |  | month | 199999 non-null | int64 |
| 11 |  | year | 199999 non-null | int64 |
| 12 |  | dayofweek | 199999 non-null | int64 |

dtypes: float64(5), int64(8) memory usage: 21.4 MB

In [ ]:

df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[ ]: | fare\_amount | pickup\_longitude | pickup\_latitude | dropoﬀ\_longitude | dropoﬀ\_latitude | passenger\_count | second | minute | hour | day | mon |
|  | 0 7.5 | -73.999817 | 40.738354 | -73.999512 | 40.723217 | 1 | 6 | 52 | 19 | 7 |  |
|  | 1 7.7 | -73.994355 | 40.728225 | -73.994710 | 40.750325 | 1 | 56 | 4 | 20 | 17 |  |
|  | 2 12.9 | -74.005043 | 40.740770 | -73.962565 | 40.772647 | 1 | 0 | 45 | 21 | 24 |  |
|  | 3 5.3 | -73.976124 | 40.790844 | -73.965316 | 40.803349 | 3 | 21 | 22 | 8 | 26 |  |
|  | 4 16.0 | -73.925023 | 40.744085 | -73.973082 | 40.761247 | 5 | 0 | 47 | 17 | 28 |  |

## Haversine Formula

#### Calculatin the distance between the pickup and drop co-ordinates using the Haversine formual for accuracy.



In [ ]:

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')

In [ ]:

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

In [ ]:

df['Distance'] = distance\_transform( df['pickup\_longitude'], df['pickup\_latitude'], df['dropoff\_longitude'], df['dropoff\_latitude']

)

In [ ]:

df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[ ]: | fare\_amount | pickup\_longitude | pickup\_latitude | dropoﬀ\_longitude | dropoﬀ\_latitude | passenger\_count | second | minute | hour | day | mon |
|  | 0 7.5 | -73.999817 | 40.738354 | -73.999512 | 40.723217 | 1 | 6 | 52 | 19 | 7 |  |
|  | 1 7.7 | -73.994355 | 40.728225 | -73.994710 | 40.750325 | 1 | 56 | 4 | 20 | 17 |  |
|  | 2 12.9 | -74.005043 | 40.740770 | -73.962565 | 40.772647 | 1 | 0 | 45 | 21 | 24 |  |
|  | 3 5.3 | -73.976124 | 40.790844 | -73.965316 | 40.803349 | 3 | 21 | 22 | 8 | 26 |  |
|  | 4 16.0 | -73.925023 | 40.744085 | -73.973082 | 40.761247 | 5 | 0 | 47 | 17 | 28 |  |

## Outliers

#### We can get rid of the trips with very large distances that are outliers as well as trips with 0 distance.

In [ ]:

plt.scatter(df['Distance'], df['fare\_amount']) plt.xlabel("Distance") plt.ylabel("fare\_amount")

Out[ ]:

Text(0, 0.5, 'fare\_amount')



In [ ]:

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

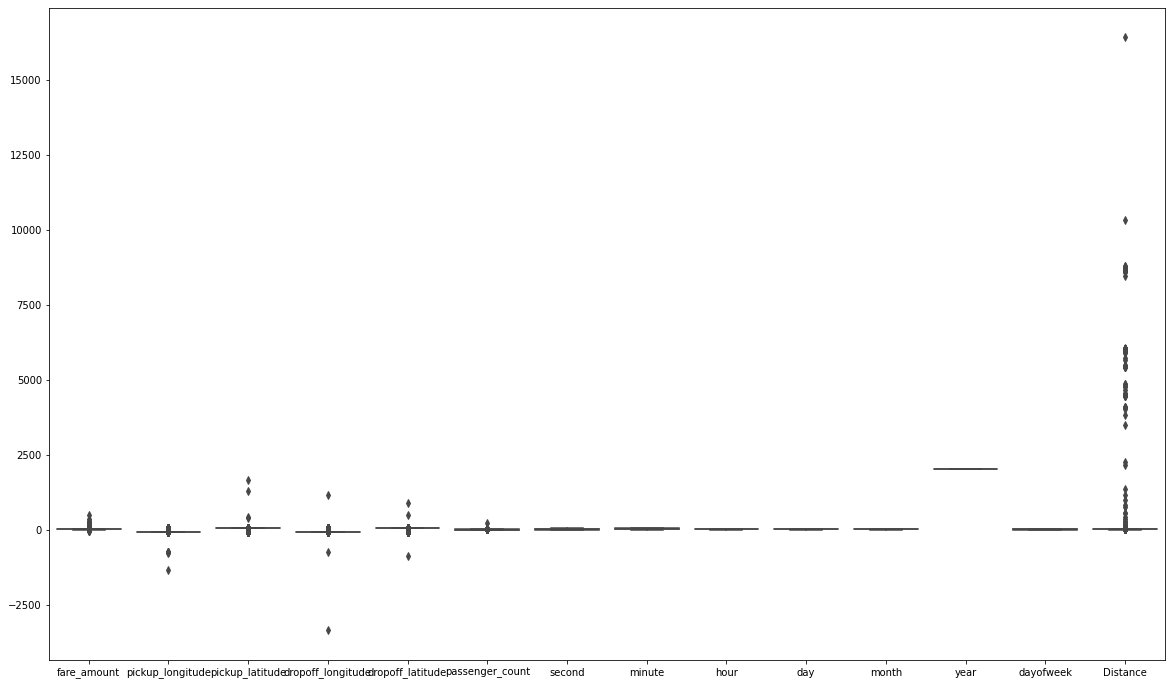
Out[ ]:

In [ ]:

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 )

<AxesSubplot:>



In [ ]:

plt.scatter(df['Distance'], df['fare\_amount']) plt.xlabel("Distance") plt.ylabel("fare\_amount")

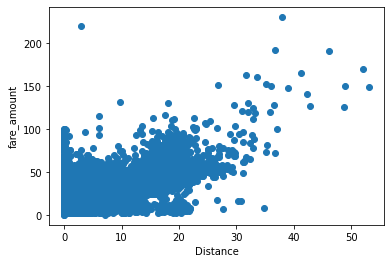
Out[ ]:

In [ ]:

corr = df.corr()

corr.style.background\_gradient(cmap='BuGn')

Text(0, 0.5, 'fare\_amount')



## Coorelation Matrix

To ﬁnd the two variables that have the most inter-dependence

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[ ]: |  | fare\_amount | pickup\_longitude | pickup\_latitude | dropoﬀ\_longitude | dropoﬀ\_latitude | passenger\_count | second | min |
|  | fare\_amount | 1.000000 | 0.005885 | -0.006253 | 0.005501 | -0.006142 | 0.011693 | -0.000995 | -0.007 |
|  | pickup\_longitude | 0.005885 | 1.000000 | -0.973204 | 0.999992 | -0.981941 | -0.000649 | -0.014677 | 0.002 |
|  | pickup\_latitude | -0.006253 | -0.973204 | 1.000000 | -0.973206 | 0.991076 | -0.001190 | 0.016809 | -0.002 |
|  | dropoﬀ\_longitude | 0.005501 | 0.999992 | -0.973206 | 1.000000 | -0.981942 | -0.000650 | -0.014638 | 0.002 |
|  | dropoﬀ\_latitude | -0.006142 | -0.981941 | 0.991076 | -0.981942 | 1.000000 | -0.001035 | 0.017202 | -0.002 |
|  | passenger\_count | 0.011693 | -0.000649 | -0.001190 | -0.000650 | -0.001035 | 1.000000 | -0.202987 | 0.000 |
|  | second | -0.000995 | -0.014677 | 0.016809 | -0.014638 | 0.017202 | -0.202987 | 1.000000 | 0.001 |
|  | minute | -0.007795 | 0.002796 | -0.002295 | 0.002803 | -0.002593 | 0.000733 | 0.001893 | 1.000 |
|  | hour | -0.020692 | 0.001547 | -0.001823 | 0.001316 | -0.001460 | 0.013226 | -0.013419 | 0.001 |
|  | day | 0.001059 | 0.005300 | -0.008901 | 0.005307 | -0.008900 | 0.003146 | -0.002100 | -0.001 |
|  | month | 0.023759 | -0.002667 | 0.004098 | -0.002656 | 0.004143 | 0.009921 | -0.049734 | -0.001 |
|  | year | 0.121195 | 0.005907 | -0.008466 | 0.005878 | -0.008553 | 0.004841 | 0.083106 | -0.002 |
|  | dayofweek | 0.006181 | 0.003006 | -0.004787 | 0.003082 | -0.004648 | 0.033360 | -0.000113 | -0.002 |
|  | Distance | 0.857729 | -0.117044 | 0.110843 | -0.117282 | 0.109486 | 0.007784 | -0.000350 | -0.007 |

# Standardization

#### For more accurate results on our linear regression model

In [ ]:

X = df['Distance'].values.reshape(-1, 1) *#Independent Variable*

y = df['fare\_amount'].values.reshape(-1, 1) *#Dependent Variable*

In [ ]:

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]]

# Splitting the Dataset

#### Training and Test Set

In [ ]:

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)

# Simple Linear Regression

#### Training the simple linear regression model on the training set

In [ ]:

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

In [ ]:

y\_pred = l\_reg.predict(X\_test)

result = pd.DataFrame() result[['Actual']] = y\_test result[['Predicted']] = y\_pred

result.sample(10)

|  |  |  |  |
| --- | --- | --- | --- |
| Out[ ]: |  | Actual | Predicted |
|  | 6607 | 0.278535 | 0.256442 |
|  | 4656 | 0.747050 | 0.539006 |
|  | 38102 | -0.294094 | -0.360193 |
|  | 33546 | -0.137922 | -0.618219 |
|  | 26106 | -0.335740 | -0.255066 |
|  | 10178 | -0.377386 | -0.257612 |
|  | 4447 | 0.372238 | 0.351430 |
|  | 30142 | -0.335740 | -0.253503 |
|  | 37765 | 0.330592 | 0.189662 |
|  | 12811 | 0.642935 | 0.832901 |

In [ ]:

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: 0.26621298757938955 Mean Absolute % Error: 1.983074763340738 Mean Squared Error: 0.2705243510778542

Root Mean Squared Error: 0.5201195546005305 R Squared (R²): 0.8567653080822022

Visualization

In [ ]:

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")

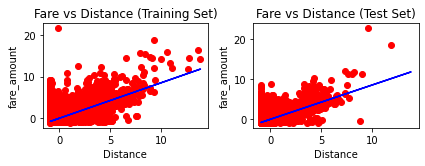
plt.subplot(2, 2, 2)

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()



In [ ]:

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

Out[ ]:

In [ ]:

Model RMSE R-Squared

0 Linear Regresion model 0.52012 0.856765

# RandomForestRegressor

#### Training the RandomForestRegressor model on the training set

Out[ ]:

RandomForestRegressor(random\_state=10)

RandomForestRegressor

▾

rf\_reg = RandomForestRegressor(n\_estimators=100, random\_state=10)

*# fit the regressor with training dataset*

rf\_reg.fit(X\_train, y\_train)

In [ ]:

*# 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)

|  |  |  |  |
| --- | --- | --- | --- |
| Out[ ]: |  | Actual | Predicted |
|  | 36840 | -0.502323 | -0.461350 |
|  | 20708 | -0.419031 | 0.031472 |
|  | 14255 | -0.814666 | -0.783328 |
|  | 15882 | -0.460677 | -0.280559 |
|  | 4628 | 0.747050 | 1.350747 |
|  | 14809 | 0.226478 | -0.074751 |
|  | 25913 | -0.658494 | -0.661410 |
|  | 30875 | -0.460677 | -0.408620 |
|  | 28673 | -0.502323 | -0.307421 |
|  | 32829 | -0.554380 | -0.447350 |

In [ ]:

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

Visualization

In [ ]:

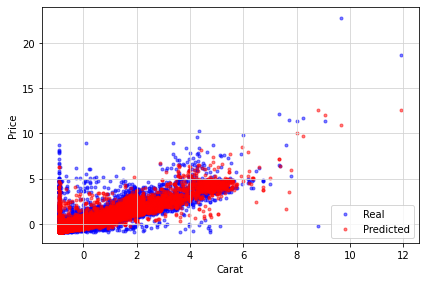
*# Build scatterplot*

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()



In [ ]:

*# 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

Out[ ]:

Model RMSE R-Squared

0 Linear Regresion model 0.520120 0.856765

1 Random Forest Regressor model 0.577042 0.820152