main

November 28, 2024

CAB FARE PREDICTION

0.1 Import libraries and load dataset

[1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

[2]:

```
traindf = pd.read_csv("D:\Study\code\project\Cab-Fare-Prediction-master\cab.
 ↪csv", na_values={"pickup_datetime":"43"})
traindf.head()
```

[2]:

```
   fare_amount          pickup_datetime  pickup_longitude  pickup_latitude  \
0          4.5  2009-06-15 17:26:21 UTC        -73.844311        40.721319
1         16.9  2010-01-05 16:52:16 UTC        -74.016048        40.711303
2          5.7  2011-08-18 00:35:00 UTC        -73.982738        40.761270
3          7.7  2012-04-21 04:30:42 UTC        -73.987130        40.733143
4          5.3  2010-03-09 07:51:00 UTC        -73.968095        40.768008

   dropoff_longitude  dropoff_latitude  passenger_count
0         -73.841610         40.712278              1.0
1         -73.979268         40.782004              1.0
2         -73.991242         40.750562              2.0
3         -73.991567         40.758092              1.0
4         -73.956655         40.783762              1.0
```

0.2 Data clearning

[3]:

```
traindf.isna()
```

[3]:

```
        fare_amount  pickup_datetime  pickup_longitude  pickup_latitude  \
0             False            False             False            False
1             False            False             False            False
2             False            False             False            False
3             False            False             False            False
4             False            False             False            False
```

```
…            …                …               …              …
16062        False            False           False            False
16063        False            False           False            False
16064        False            False           False            False
16065        False            False           False            False
16066        False            False           False            False

       dropoff_longitude  dropoff_latitude  passenger_count
0                  False             False            False
1                  False             False            False
2                  False             False            False
3                  False             False            False
4                  False             False            False
…                  …                …              …
16062              False             False            False
16063              False             False            False
16064              False             False            False
16065              False             False            False
16066              False             False             True

[16067 rows x 7 columns]
```

[4]:
```
traindf.describe()
```

[4]:
```
       pickup_longitude  pickup_latitude  dropoff_longitude  dropoff_latitude  \
count      16067.000000     16067.000000       16067.000000      16067.000000
mean         -72.462787        39.914725         -72.462328         39.897906
std           10.578384         6.826587          10.575062          6.187087
min          -74.438233       -74.006893         -74.429332        -74.006377
25%          -73.992156        40.734927         -73.991182         40.734651
50%          -73.981698        40.752603         -73.980172         40.753567
75%          -73.966838        40.767381         -73.963642         40.768014
max           40.766125       401.083332          40.802437         41.366138

       passenger_count
count     16012.000000
mean          2.625070
std          60.844122
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max        5345.000000
```

0.2.1 Changing the dtype

```
[5]: traindf['fare_amount'] = pd.to_numeric(traindf['fare_amount'], errors='coerce')
```

```
[6]: traindf['pickup_datetime'] = pd.to_datetime(traindf['pickup_datetime'],␣
      ↪format='%Y-%m-%d %H:%M:%S UTC')
```

0.2.2 Extract Hour, Date, Day, Month, Year

```
[7]: traindf['Hour'] = traindf['pickup_datetime'].dt.hour
     traindf['minute'] = traindf['pickup_datetime'].dt.minute
     traindf['date'] = traindf['pickup_datetime'].dt.day
     traindf['day'] = traindf['pickup_datetime'].dt.dayofweek
     traindf['month'] = traindf['pickup_datetime'].dt.month
     traindf['year'] = traindf['pickup_datetime'].dt.year
```

```
[8]: traindf.head()
```

```
[8]:    fare_amount     pickup_datetime  pickup_longitude  pickup_latitude  \
     0          4.5 2009-06-15 17:26:21        -73.844311        40.721319
     1         16.9 2010-01-05 16:52:16        -74.016048        40.711303
     2          5.7 2011-08-18 00:35:00        -73.982738        40.761270
     3          7.7 2012-04-21 04:30:42        -73.987130        40.733143
     4          5.3 2010-03-09 07:51:00        -73.968095        40.768008

        dropoff_longitude  dropoff_latitude  passenger_count  Hour  minute  date  \
     0         -73.841610         40.712278              1.0  17.0    26.0  15.0
     1         -73.979268         40.782004              1.0  16.0    52.0   5.0
     2         -73.991242         40.750562              2.0   0.0    35.0  18.0
     3         -73.991567         40.758092              1.0   4.0    30.0  21.0
     4         -73.956655         40.783762              1.0   7.0    51.0   9.0

        day  month    year
     0  0.0    6.0  2009.0
     1  1.0    1.0  2010.0
     2  3.0    8.0  2011.0
     3  5.0    4.0  2012.0
     4  1.0    3.0  2010.0
```

0.3 Handle missing values

SimpleImputer

0.3.1 drop null row from pickup__datetime

```
[9]: traindf.drop(traindf[traindf['pickup_datetime'].isna()].index, inplace=True)
```

```
[10]: traindf.drop('pickup_datetime', axis=1, inplace=True)
```

[11]:

```python
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
#apply it to features

imputer.fit(traindf[['fare_amount','passenger_count']])
traindf[['fare_amount','passenger_count']] = imputer.
 ↪transform(traindf[['fare_amount','passenger_count']])
```

[12]:

```python
traindf.isna().sum()
```

[12]:

```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
Hour                 0
minute               0
date                 0
day                  0
month                0
year                 0
dtype: int64
```

[13]:

```python
traindf.shape
```

[13]:

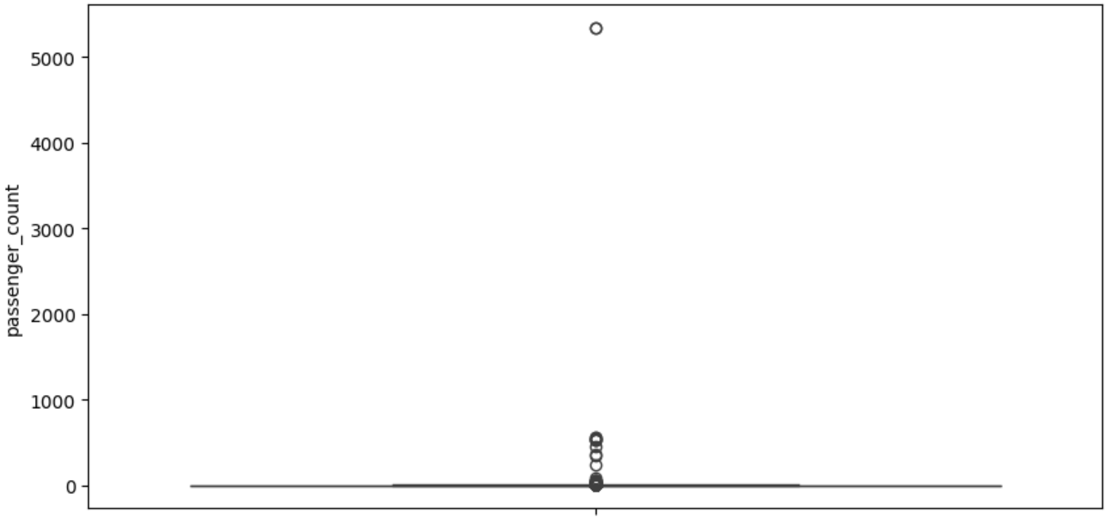
```
(16066, 12)
```

[14]:

```python
plt.figure(figsize=(10,5))
sns.boxplot(y=traindf['passenger_count'])
```

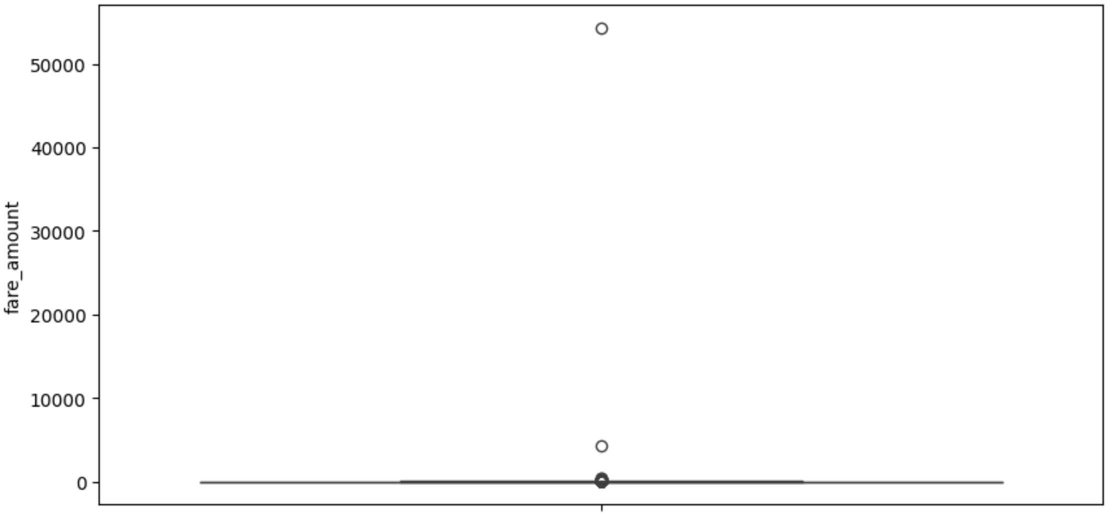
[14]:

```
<Axes: ylabel='passenger_count'>
```



[15]:
```
plt.figure(figsize=(10,5))
sns.boxplot(y=traindf['fare_amount'])
```

[15]:
```
<Axes: ylabel='fare_amount'>
```



0.4 Remove outlier

[16]:
```
Q1 = traindf['passenger_count'].quantile(0.25)
Q3 = traindf['passenger_count'].quantile(0.75)

IQR = Q3 - Q1
```

```

lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
print('Lower :',lower)
print('Upper :',upper)
```

```
Lower : -0.5
Upper : 3.5
```

1. Remove passenger_count less than zero and with more than 6

[17]:
```
traindf['passenger_count'].sort_values(ascending=True).head(10)
```

[17]:
```
13742    0.0
2425     0.0
6575     0.0
5150     0.0
4248     0.0
5058     0.0
9159     0.0
3413     0.0
4114     0.0
15514    0.0
Name: passenger_count, dtype: float64
```

[18]:
```
traindf.drop(traindf[traindf['passenger_count'] > 4 ].index, inplace = True)
traindf.drop(traindf[traindf['passenger_count'] < 0 ].index, inplace = True)
traindf.drop(traindf[traindf['passenger_count'] == 0.12 ].index, inplace = True)
traindf.drop(traindf[traindf['passenger_count'].isna() ].index, inplace = True)
```

2. Remove fare_amount less than zero and should be less than 454

[19]:
```
traindf['fare_amount'].sort_values(ascending=True).head(10)
```

[19]:
```
13032   -3.00
2039    -2.90
2486    -2.50
10002    0.00
2780     0.01
1427     1.14
8596     2.50
503      2.50
8711     2.50
6002     2.50
Name: fare_amount, dtype: float64
```

[20]:
```
traindf.drop(traindf[traindf['fare_amount'] > 150 ].index, inplace = True)
traindf.drop(traindf[traindf['fare_amount'] == 0 ].index, inplace = True)
traindf.drop(traindf[traindf['fare_amount'].isna() ].index, inplace = True)
```

3. Pickup and dropoff latitude should be (-90 to 90)

[21]:
```
traindf.drop(traindf[traindf['pickup_latitude'] > 90].index, inplace=True)
traindf.drop(traindf[traindf['pickup_latitude'] < -90].index, inplace=True)
traindf.drop(traindf[traindf['dropoff_latitude'] > 90].index, inplace=True)
traindf.drop(traindf[traindf['dropoff_latitude'] < -90].index, inplace=True)
```

4. Pickup and dropoff longtitude should be (-180 to 180)

[22]:
```
traindf.drop(traindf[traindf['pickup_longitude'] > 180].index, inplace=True)
traindf.drop(traindf[traindf['pickup_longitude'] < -180].index, inplace=True)
traindf.drop(traindf[traindf['dropoff_longitude'] > 180].index, inplace=True)
traindf.drop(traindf[traindf['dropoff_longitude'] < -180].index, inplace=True)
```

[23]:
```
traindf.shape
```

[23]:
```
(14690, 12)
```

[24]:
```
traindf.isna().sum()
```

[24]:
```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
Hour                 0
minute               0
date                 0
day                  0
month                0
year                 0
dtype: int64
```

0.4.1 Calculate distance using Haversion formulas

[25]:
```
from math import *

def haversine(a):
    lon1=a[0]
    lat1=a[1]
    lon2=a[2]
    lat2=a[3]
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c =  2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371* c
    return km
```

[26]:

```
traindf['distance'] =␣
  ↪traindf[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']].
  ↪apply(haversine,axis=1)
```

```
C:\Users\mukul\AppData\Local\Temp\ipykernel_20088\3890309117.py:4:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  lon1=a[0]
C:\Users\mukul\AppData\Local\Temp\ipykernel_20088\3890309117.py:5:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  lat1=a[1]
C:\Users\mukul\AppData\Local\Temp\ipykernel_20088\3890309117.py:6:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  lon2=a[2]
C:\Users\mukul\AppData\Local\Temp\ipykernel_20088\3890309117.py:7:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  lat2=a[3]
```

[27]:

```
traindf.
  ↪drop(['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude'],␣
  ↪axis=1, inplace=True)
```

[28]:

```
traindf['Hour'] = traindf['Hour'].astype('int64')
traindf['minute'] = traindf['minute'].astype('int64')
traindf['date'] = traindf['date'].astype('int64')
traindf['day'] = traindf['day'].astype('int64')
traindf['month'] = traindf['month'].astype('int64')
traindf['year'] = traindf['year'].astype('int64')
```

[29]:

```
traindf.head()
```

```
[29]:    fare_amount  passenger_count  Hour  minute  date  day  month  year  \
      0          4.5              1.0    17      26    15    0      6  2009
      1         16.9              1.0    16      52     5    1      1  2010
      2          5.7              2.0     0      35    18    3      8  2011
      3          7.7              1.0     4      30    21    5      4  2012
      4          5.3              1.0     7      51     9    1      3  2010

         distance
      0  1.030764
      1  8.450134
      2  1.389525
      3  2.799270
      4  1.999157
```

Distance should be positive and less than 130 Km.

```
[30]: traindf['distance'].sort_values(ascending=True).head(10)
```

```
[30]: 1542     0.0
      8135     0.0
      8130     0.0
      8123     0.0
      8109     0.0
      13446    0.0
      1397     0.0
      8068     0.0
      8063     0.0
      1419     0.0
      Name: distance, dtype: float64
```

```
[31]: traindf.drop(traindf[traindf['distance'] >= 130].index, inplace=True)
      traindf.drop(traindf[traindf['distance'] <= 0 ].index, inplace=True)
```

```
[32]: #traindf.drop(traindf[traindf['distance'].isna() ].index, inplace=True)
```

```
[33]: traindf.head()
```

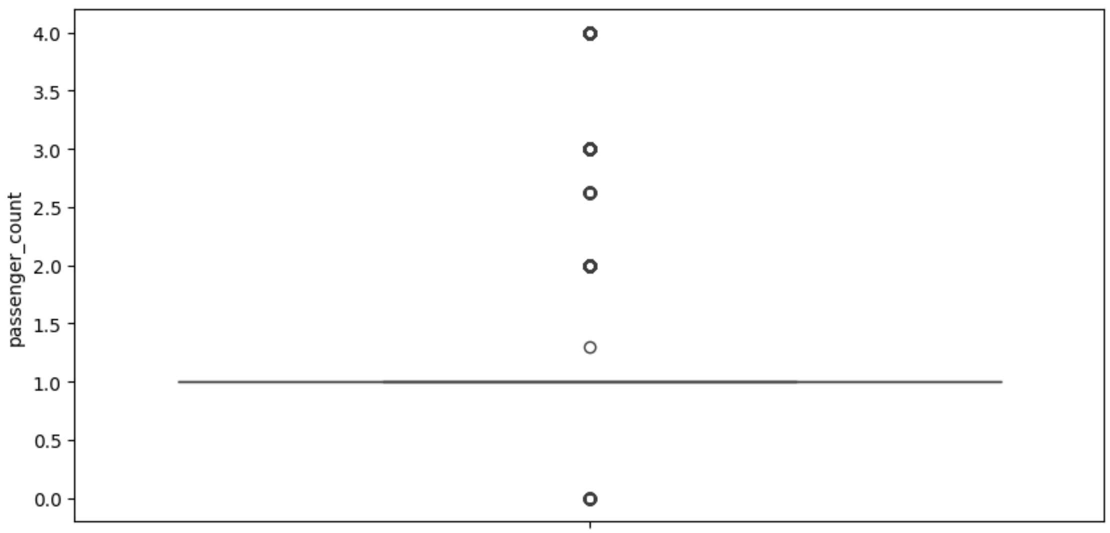
```
[33]:    fare_amount  passenger_count  Hour  minute  date  day  month  year  \
      0          4.5              1.0    17      26    15    0      6  2009
      1         16.9              1.0    16      52     5    1      1  2010
      2          5.7              2.0     0      35    18    3      8  2011
      3          7.7              1.0     4      30    21    5      4  2012
      4          5.3              1.0     7      51     9    1      3  2010

         distance
      0  1.030764
      1  8.450134
      2  1.389525
```

```
3   2.799270
4   1.999157
```

0.4.2 EDA

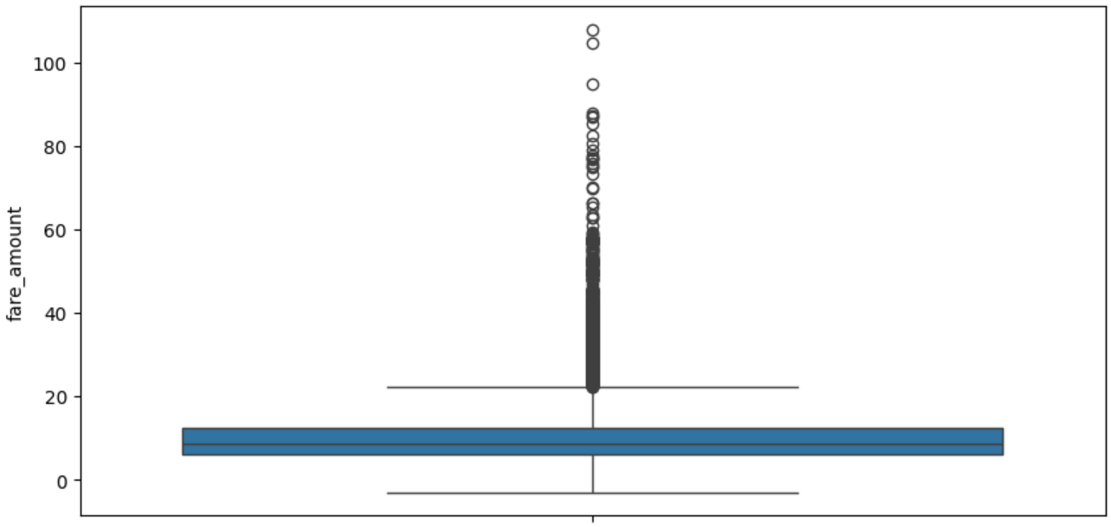
```
[34]: plt.figure(figsize=(10,5))
      sns.boxplot(y=traindf['passenger_count'])
```

```
[34]: <Axes: ylabel='passenger_count'>
```



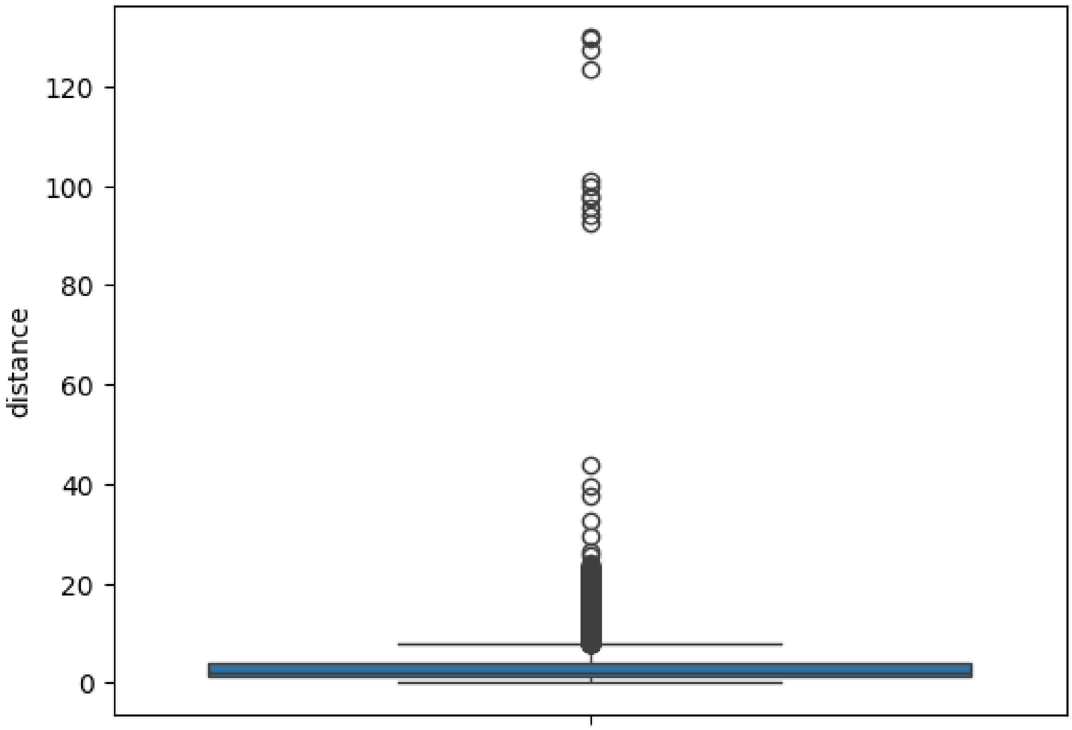
```
[35]: plt.figure(figsize=(10,5))
      sns.boxplot(y=traindf['fare_amount'])
```

```
[35]: <Axes: ylabel='fare_amount'>
```



```
[36]: sns.boxplot(y=traindf['distance'])
```

```
[36]: <Axes: ylabel='distance'>
```



```
[37]: traindf.describe()
```

[37]:

```
        fare_amount  passenger_count          Hour        minute  \
count  14244.000000     14244.000000  14244.000000  14244.000000
mean      11.310145         1.320604     13.490312     29.746069
std        9.434057         0.675145      6.514905     17.288493
min       -3.000000         0.000000      0.000000      0.000000
25%        6.000000         1.000000      9.000000     15.000000
50%        8.500000         1.000000     14.000000     30.000000
75%       12.500000         1.000000     19.000000     45.000000
max      108.000000         4.000000     23.000000     59.000000

               date           day         month          year      distance
count  14244.000000  14244.000000  14244.000000  14244.000000  14244.000000
mean      15.668211      3.030820      6.264462   2011.739890      3.451184
std        8.685281      1.971422      3.444883      1.870919      4.674488
min        1.000000      0.000000      1.000000   2009.000000      0.000111
25%        8.000000      1.000000      3.000000   2010.000000      1.277104
50%       16.000000      3.000000      6.000000   2012.000000      2.196772
75%       23.000000      5.000000      9.000000   2013.000000      3.935514
max       31.000000      6.000000     12.000000   2015.000000    129.950482
```

[38]:

```
traindf['passenger_count'] = traindf['passenger_count'].astype('int64')
```

[39]:

```
traindf
```

[39]:

```
       fare_amount  passenger_count  Hour  minute  date  day  month  year  \
0              4.5                1    17      26    15    0      6  2009
1             16.9                1    16      52     5    1      1  2010
2              5.7                2     0      35    18    3      8  2011
3              7.7                1     4      30    21    5      4  2012
4              5.3                1     7      51     9    1      3  2010
…                …                …     …       …     …    …      …     …
16062          6.5                1     7      41    12    4     12  2014
16063         16.1                2     7      58    13    0      7  2009
16064          8.5                1    11      19    11    2     11  2009
16065          8.1                1    23      53    11    1      5  2010
16066          8.5                2     6      24    14    2     12  2011

       distance
0      1.030764
1      8.450134
2      1.389525
3      2.799270
4      1.999157
…             …
16062  0.850044
16063  7.867638
16064  1.469105
```

```
16065  2.590036
16066  3.898113

[14244 rows x 9 columns]
```

[40]:
```
def passenger(no,cost):
    if(no==1):
        cost=cost*20
        return cost
    if(no==2):
        cost=(cost*20)+10
        return cost
    if(no==3):
        cost=(cost*20)+20
        return cost
    elif(no==4 and no==5):
        cost=(cost*20)+25
        return cost
    elif(no>5):
        return("no service")
```

[41]:
```
temp =traindf['passenger_count'].count()
x=0
z=[]
for x in range(temp):
    g=traindf['passenger_count'].iloc[x]
    j=traindf['fare_amount'].iloc[x]
    z.append(passenger(g,j))
traindf['Changed_fare']=z
```

[42]:
```
traindf
```

[42]:
```
       fare_amount  passenger_count  Hour  minute  date  day  month  year  \
0              4.5                1    17      26    15    0      6  2009
1             16.9                1    16      52     5    1      1  2010
2              5.7                2     0      35    18    3      8  2011
3              7.7                1     4      30    21    5      4  2012
4              5.3                1     7      51     9    1      3  2010
…            …              …   …     …   …  …    …   …
16062          6.5                1     7      41    12    4     12  2014
16063         16.1                2     7      58    13    0      7  2009
16064          8.5                1    11      19    11    2     11  2009
16065          8.1                1    23      53    11    1      5  2010
16066          8.5                2     6      24    14    2     12  2011

       distance  Changed_fare
0      1.030764          90.0
```

```
1      8.450134         338.0
2      1.389525         124.0
3      2.799270         154.0
4      1.999157         106.0
…         …           …
16062  0.850044         130.0
16063  7.867638         332.0
16064  1.469105         170.0
16065  2.590036         162.0
16066  3.898113         180.0

[14244 rows x 10 columns]
```

[43]:
```
traindf.isna().sum()
```

[43]:
```
fare_amount          0
passenger_count      0
Hour                 0
minute               0
date                 0
day                  0
month                0
year                 0
distance             0
Changed_fare       373
dtype: int64
```

0.4.3 Train Test Split

[44]:
```
traindf=traindf.dropna()
```

[45]:
```
traindf.head()
```

[45]:
```
   fare_amount  passenger_count  Hour  minute  date  day  month  year  \
0          4.5                1    17      26    15    0      6  2009
1         16.9                1    16      52     5    1      1  2010
2          5.7                2     0      35    18    3      8  2011
3          7.7                1     4      30    21    5      4  2012
4          5.3                1     7      51     9    1      3  2010

   distance  Changed_fare
0  1.030764          90.0
1  8.450134         338.0
2  1.389525         124.0
3  2.799270         154.0
4  1.999157         106.0
```

[46]:
```
X=traindf.drop(['fare_amount','Changed_fare'],axis=1)
y=traindf['Changed_fare'].astype('int64')
```

[47]:
```
X
```

[47]:
```
       passenger_count  Hour  minute  date  day  month  year  distance
0                    1    17      26    15    0      6  2009  1.030764
1                    1    16      52     5    1      1  2010  8.450134
2                    2     0      35    18    3      8  2011  1.389525
3                    1     4      30    21    5      4  2012  2.799270
4                    1     7      51     9    1      3  2010  1.999157
…                …   …     …   …  …    …   …       …
16062                1     7      41    12    4     12  2014  0.850044
16063                2     7      58    13    0      7  2009  7.867638
16064                1    11      19    11    2     11  2009  1.469105
16065                1    23      53    11    1      5  2010  2.590036
16066                2     6      24    14    2     12  2011  3.898113

[13871 rows x 8 columns]
```

[48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
 ↪random_state = 42)
```

0.4.4 Model selection

[49]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=150,␣
 ↪criterion='absolute_error',max_depth=11)
```

[50]:
```
model.fit(X_train, y_train)
```

[50]:
```
RandomForestRegressor(criterion='absolute_error', max_depth=11,
                      n_estimators=150)
```

[51]:
```
predicts = model.predict(X_test)
```

[52]:
```
model.score(X_train, y_train)
```

[52]:
```
0.903482640567547
```

0.4.5 Model Evaluation

[53]:
```
from sklearn.metrics import r2_score

score=r2_score(y_test,predicts)
score
```

[53]: 0.8294424351455236

[54]:
```
from sklearn.metrics import mean_squared_error

RMSE = np.sqrt(mean_squared_error(y_test, predicts))
RMSE
```

[54]: 77.85002005591717

[55]:
```
import pickle
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

[56]:
```
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

```
0.8294424351455236
```

[57]:
```
pred=loaded_model.predict(np.array([2,17,26,15,0,11,2022,1.030764]).
  ↪reshape(1,-1))
```

```
c:\Users\mukul\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

[58]:
```
pred
```

[58]: array([152.11333333])