

Import the necessary libraries and read data

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

```
In [2]: from warnings import filterwarnings
filterwarnings("ignore")
```

```
In [3]: df1=pd.read_parquet('yellow_tripdata_2022-01.parquet')
df2=pd.read_parquet('yellow_tripdata_2022-02.parquet')
df3=pd.read_parquet('yellow_tripdata_2022-03.parquet')
df4=pd.read_parquet('yellow_tripdata_2022-04.parquet')
df5=pd.read_parquet('yellow_tripdata_2022-05.parquet')
df6=pd.read_parquet('yellow_tripdata_2022-06.parquet')

data=pd.concat([df1.sample(11553) ,df2.sample(11553) ,df3.sample(11553) ,df4.sample(11553) ,df5.sample(11553) ,df6.sample(11553)])
```

Read the data randomly as parquet using pandas

```
In [4]: data.head()
```

```
Out[4]:
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	
	2356663	2	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0	1.18
	1419101	1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0	0.20
	2271387	2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0	2.24
	2034114	1	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0	5.40
	20678	2	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0	2.29

```
In [5]: data = data.reset_index()
```

```
In [6]: data.head()
```

```
Out[6]:
```

	index	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	2356663	2	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0	1.18
1	1419101	1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0	0.20
2	2271387	2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0	2.24
3	2034114	1	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0	5.40
4	20678	2	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0	2.29

When We import the data randomly we can see our data has index in improper order.

Although index column is not important in determining our result so we drop it along with different columns but it is important to know whether our data is random or not.

```
In [7]: long_lat=pd.read_csv("nyc_taxi_trip_duration.csv")
```

```
In [8]: long_lat.columns
```

```
Out[8]: Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',  
            'passenger_count', 'pickup_longitude', 'pickup_latitude',  
            'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',  
            'trip_duration'],  
          dtype='object')
```

Import another dataset which has latitude and longitude columns.

```
In [9]: data['pickup_longitude']=long_lat['pickup_longitude']  
data['pickup_latitude']= long_lat['pickup_latitude']  
data['dropoff_longitude']= long_lat['dropoff_longitude']  
data['dropoff_latitude']=long_lat['dropoff_latitude']  
data.head()
```

```
Out[9]:
```

	index	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distan
0	2356663	2	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0	1.
1	1419101	1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0	0.
2	2271387	2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0	2.
3	2034114	1	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0	5.
4	20678	2	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0	2.

5 rows × 24 columns

We add 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude' columns to our dataset named 'data'.

```
In [10]: dk_time = pd.to_datetime(data["tpep_dropoff_datetime"]);  
pk_time = pd.to_datetime(data["tpep_pickup_datetime"]);  
D=round(abs(dk_time - pk_time)/np.timedelta64(1,"s") / 60)  
  
data["trip_duration"]=D  
data.head()
```

```
Out[10]:
```

	index	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distan
0	2356663	2	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0	1.
1	1419101	1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0	0.
2	2271387	2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0	2.
3	2034114	1	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0	5.
4	20678	2	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0	2.

5 rows × 25 columns

Create new column called 'trip_duration' calculated as 'tpep_dropoff_datetime' subtracted to 'tpep_pickup_datetime'.

```
In [11]: data.to_csv("new_yellow_taxi.csv")
```

Convert parquet format to csv format data with name 'new_yellow_taxi.csv'

```
In [12]: df=pd.read_csv("new_yellow_taxi.csv",nrows=30000)
```

Read data with less columns so that model running process is too long without compromising results

```
In [13]: df.head(10)
```

```
Out[13]:
```

	Unnamed: 0	index	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count
0	0	2356663	2	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0
1	1	1419101	1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0
2	2	2271387	2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0
3	3	2034114	1	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0
4	4	20678	2	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0
5	5	1188903	2	2022-01-17 07:05:09	2022-01-17 07:35:12	1.0
6	6	583066	1	2022-01-09 11:38:35	2022-01-09 11:45:53	1.0
7	7	2135020	2	2022-01-28 08:05:09	2022-01-28 08:20:16	1.0
8	8	2104869	1	2022-01-27 20:41:56	2022-01-27 20:45:31	1.0
9	9	2034793	2	2022-01-27 08:55:19	2022-01-27 09:04:04	1.0

10 rows × 26 columns

Drop statistical unimportant columns

```
In [14]: df=df.drop(labels=['Unnamed: 0','index','VendorID','RatecodeID','store_and_fwd_flag'])
```

These columns do not contribute in prediction hence we are dropping them

Data preprocessing/Munging

```
In [15]: df.columns #columns overview
```

```
Out[15]: Index(['tpep_pickup_datetime', 'tpep_dropoff_datetime', 'passenger_count',  
              'trip_distance', 'PULocationID', 'DOLocationID', 'payment_type',  
              'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',  
              'improvement_surcharge', 'total_amount', 'congestion_surcharge',  
              'airport_fee', 'pickup_longitude', 'pickup_latitude',  
              'dropoff_longitude', 'dropoff_latitude', 'trip_duration'],  
              dtype='object')
```

```
In [16]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
---  -
0   tpep_pickup_datetime  30000 non-null  object
1   tpep_dropoff_datetime 30000 non-null  object
2   passenger_count       29051 non-null  float64
3   trip_distance         30000 non-null  float64
4   PULocationID          30000 non-null  int64
5   DOLocationID          30000 non-null  int64
6   payment_type          30000 non-null  int64
7   fare_amount           30000 non-null  float64
8   extra                 30000 non-null  float64
9   mta_tax               30000 non-null  float64
10  tip_amount            30000 non-null  float64
11  tolls_amount          30000 non-null  float64
12  improvement_surcharge 30000 non-null  float64
13  total_amount          30000 non-null  float64
14  congestion_surcharge  29051 non-null  float64
15  airport_fee           29051 non-null  float64
16  pickup_longitude      30000 non-null  float64
17  pickup_latitude       30000 non-null  float64
18  dropoff_longitude     30000 non-null  float64
19  dropoff_latitude      30000 non-null  float64
20  trip_duration         30000 non-null  float64
dtypes: float64(16), int64(3), object(2)
memory usage: 4.8+ MB

```

```
In [17]: df.isna().sum()
```

```

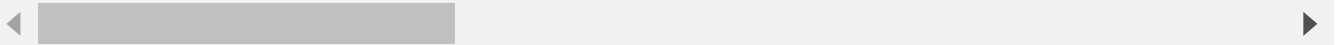
Out[17]: tpep_pickup_datetime    0
tpep_dropoff_datetime      0
passenger_count            949
trip_distance              0
PULocationID              0
DOLocationID              0
payment_type              0
fare_amount               0
extra                     0
mta_tax                   0
tip_amount                0
tolls_amount              0
improvement_surcharge     0
total_amount              0
congestion_surcharge      949
airport_fee               949
pickup_longitude          0
pickup_latitude           0
dropoff_longitude         0
dropoff_latitude          0
trip_duration             0
dtype: int64

```

```
In [18]: df.describe()    #stataistical summary
```

Out[18]:

	passenger_count	trip_distance	PULocationID	DOLocationID	payment_type	fare_amount
count	29051.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	1.384290	9.352918	165.513567	163.410700	1.181567	13.154855
std	0.974155	689.127829	65.676377	70.350156	0.493094	12.452520
min	0.000000	0.000000	1.000000	1.000000	0.000000	-99.700000
25%	1.000000	1.070000	132.000000	113.000000	1.000000	6.500000
50%	1.000000	1.790000	162.000000	162.000000	1.000000	9.500000
75%	1.000000	3.180000	234.000000	234.000000	1.000000	14.500000
max	6.000000	92455.610000	265.000000	265.000000	4.000000	357.000000



Exploratory Data Analysis

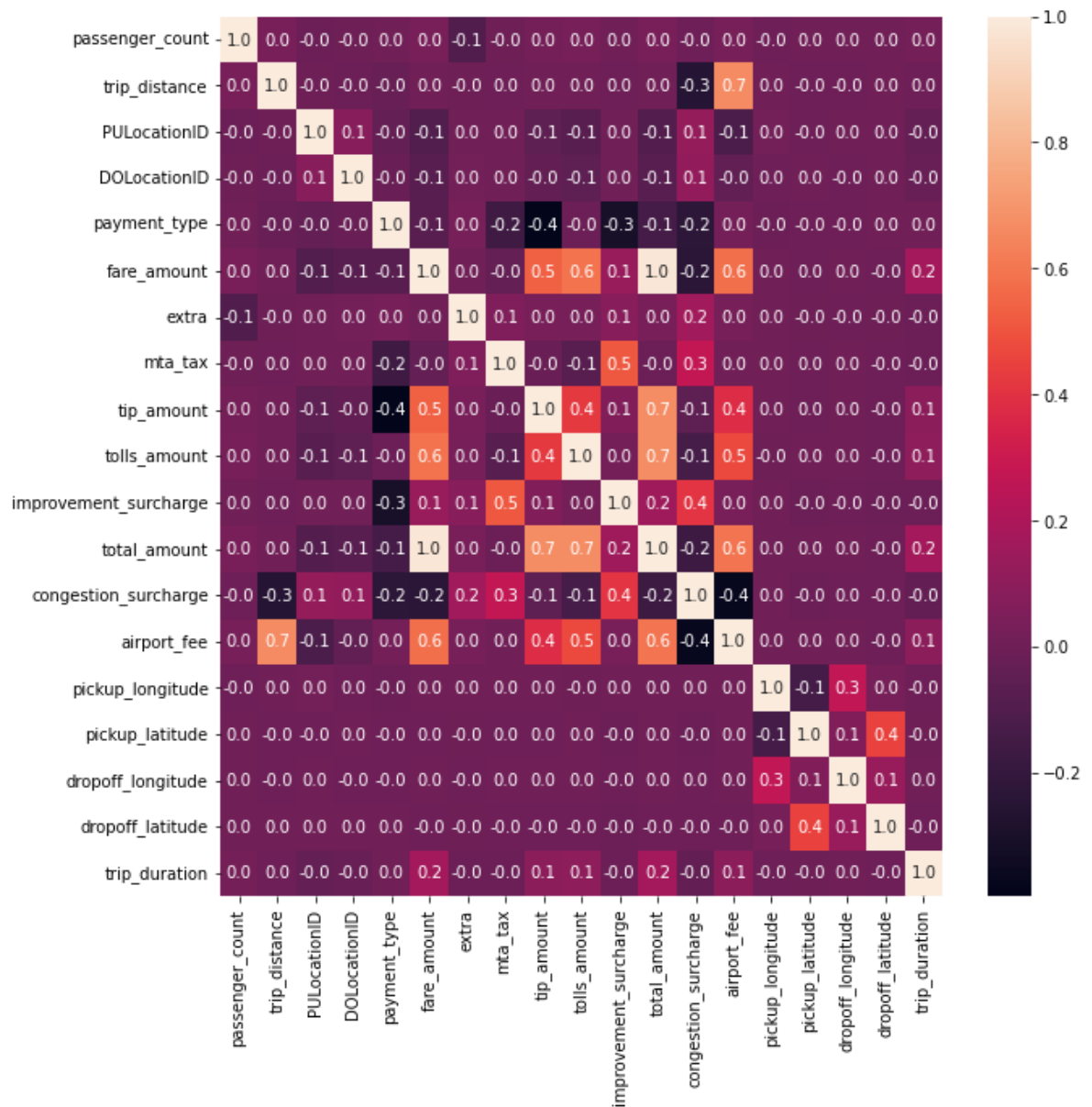
Importing Visualization Libraries

In [19]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [20]:

```
plt.figure(figsize=(10,10)) #Analysis using heatmap
sns.heatmap(df.corr(),annot=True,fmt='.1f')
plt.show()
```



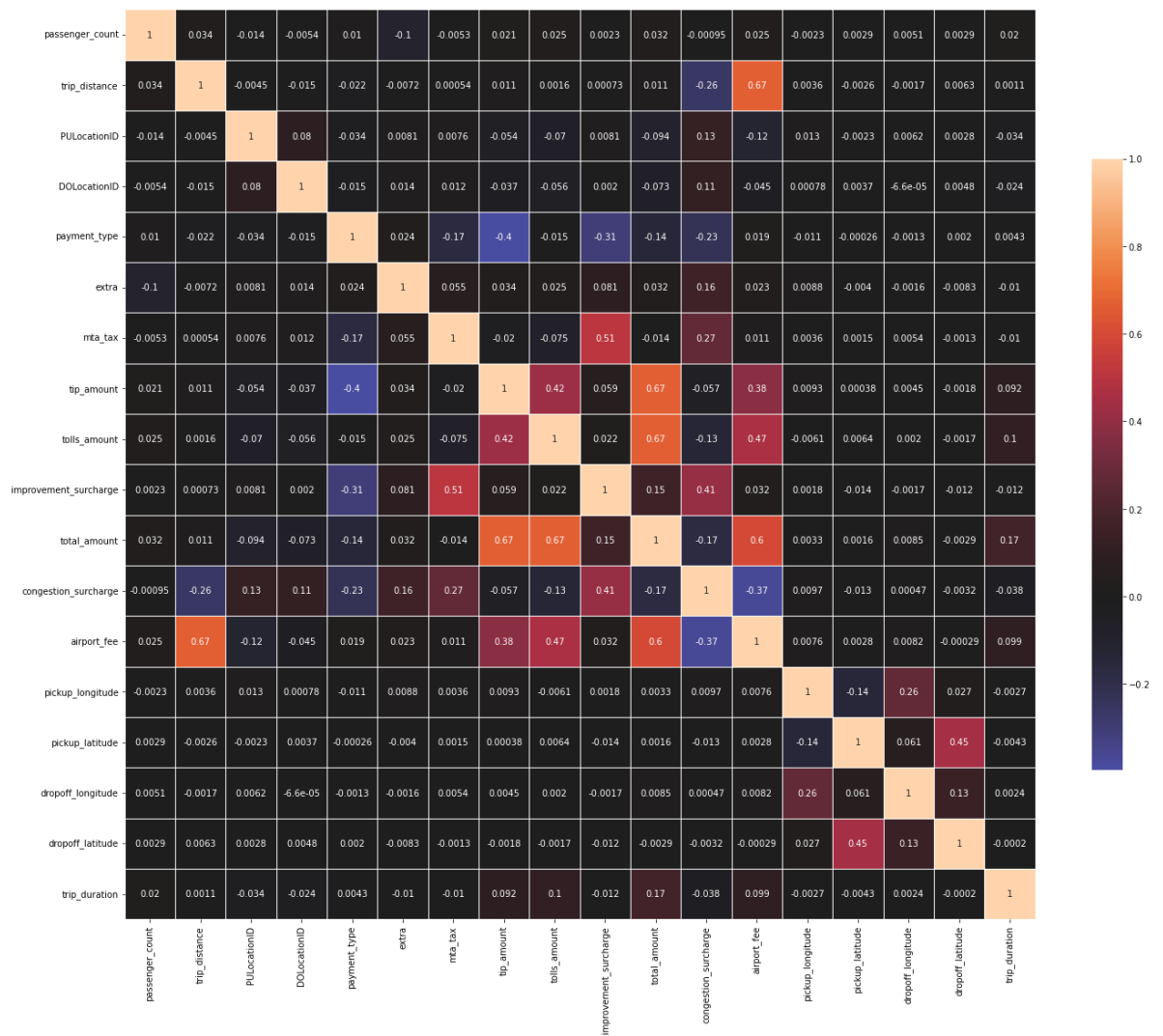
Plot heat map to observe the relationship between the variables.

```
In [21]: df=df.drop(labels=["fare_amount"],axis=1)
```

Removed the column 'fare_amount' due to high correlation with 'total_amount' to avoid the Multicollinearity

Multicollinearity occurs when two or more independent variables are highly correlated with each other, change in one variable would cause change to another variable and so the model results vary significantly.

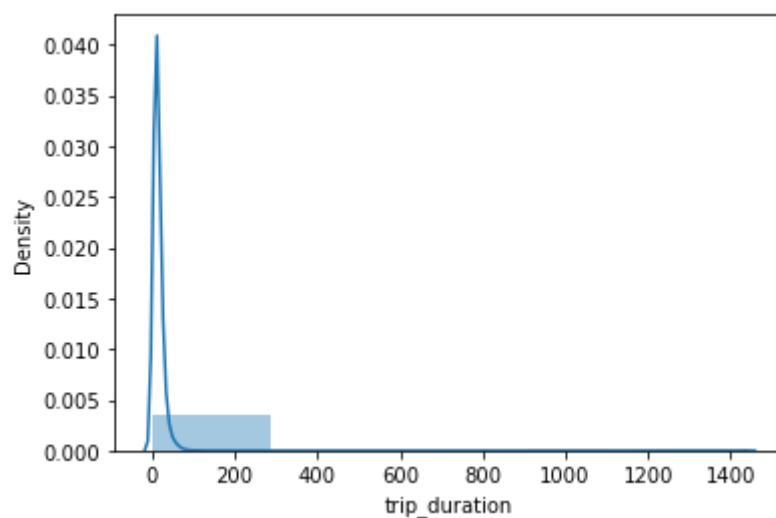
```
In [22]: plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},annot=True)
plt.tight_layout()
plt.show()
```



Heat map after removing multicollinearity

```
In [23]: sns.distplot(df['trip_duration'], kde = True, bins = 5)
```

```
Out[23]: <AxesSubplot:xlabel='trip_duration', ylabel='Density'>
```



Analysis of target columnn

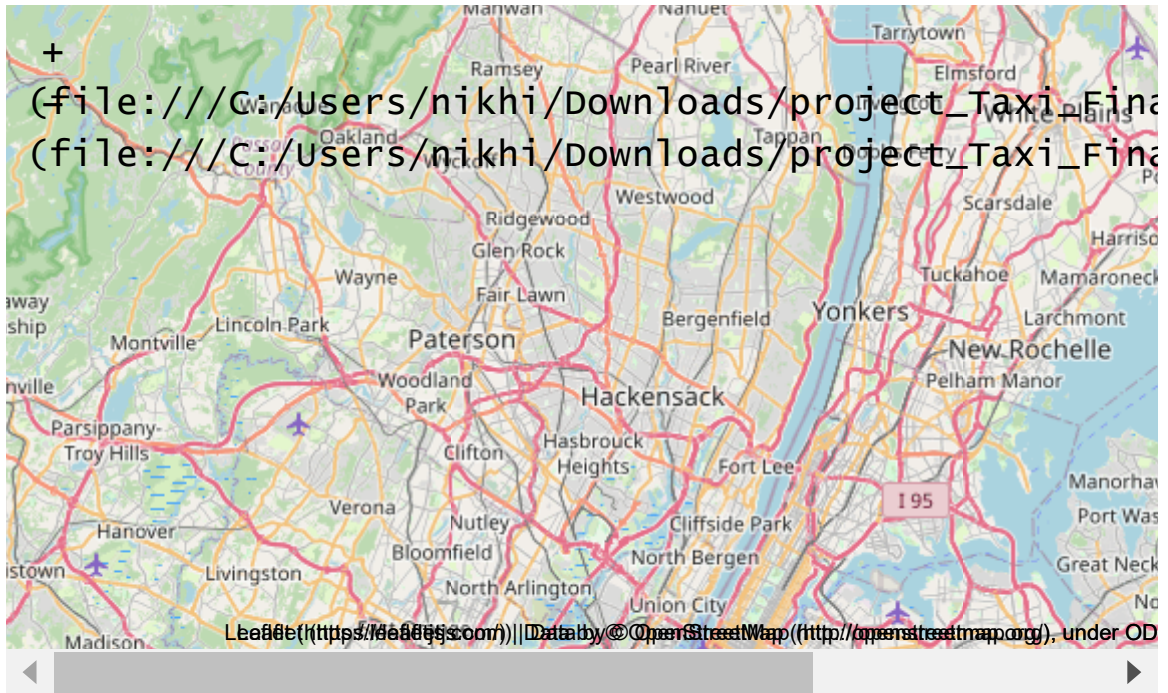
Map view for Pickup Points

```
In [24]: import folium # folium is python library for geospatial analysis

location = [40.730610, -73.935242]
location2=[40.708469, -74.017120]

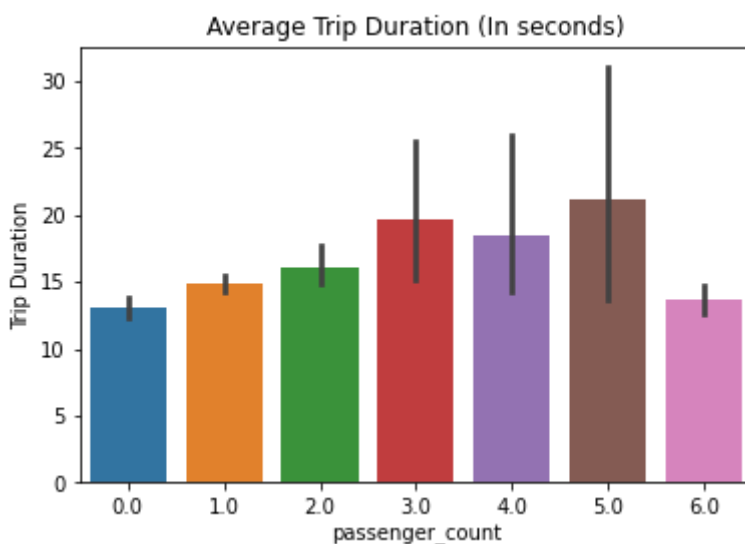
map=folium.Map(location=location ,width=800,height=400,zoom_start = 10,)
map
```

Out[24]:



Map for one pickup and dropoff location based on the latitude and longitude of New york

```
In [25]: sns.barplot(x="passenger_count", y="trip_duration",data=df); #Bivariate Analysis
plt.title("Average Trip Duration (In seconds)");
plt.xlabel("passenger_count");
plt.ylabel("Trip Duration");
```



Above plot shows the graph between Trip Duration and passenger count. From the graph we can interpret as the passengers count increases trip duration also increases.

Map view for pickup point

```
In [26]: pickup =df[["pickup_longitude",'pickup_latitude']]
pickup
```



```
data = list(zip(pickup.pickup_latitude.values,
                pickup.pickup_longitude.values,
                ))

type(data[0][0])
```

Out[26]: numpy.float64

```
In [27]: pickup = pd.DataFrame(data)
pickup.head()
```

```
Out[27]:
```

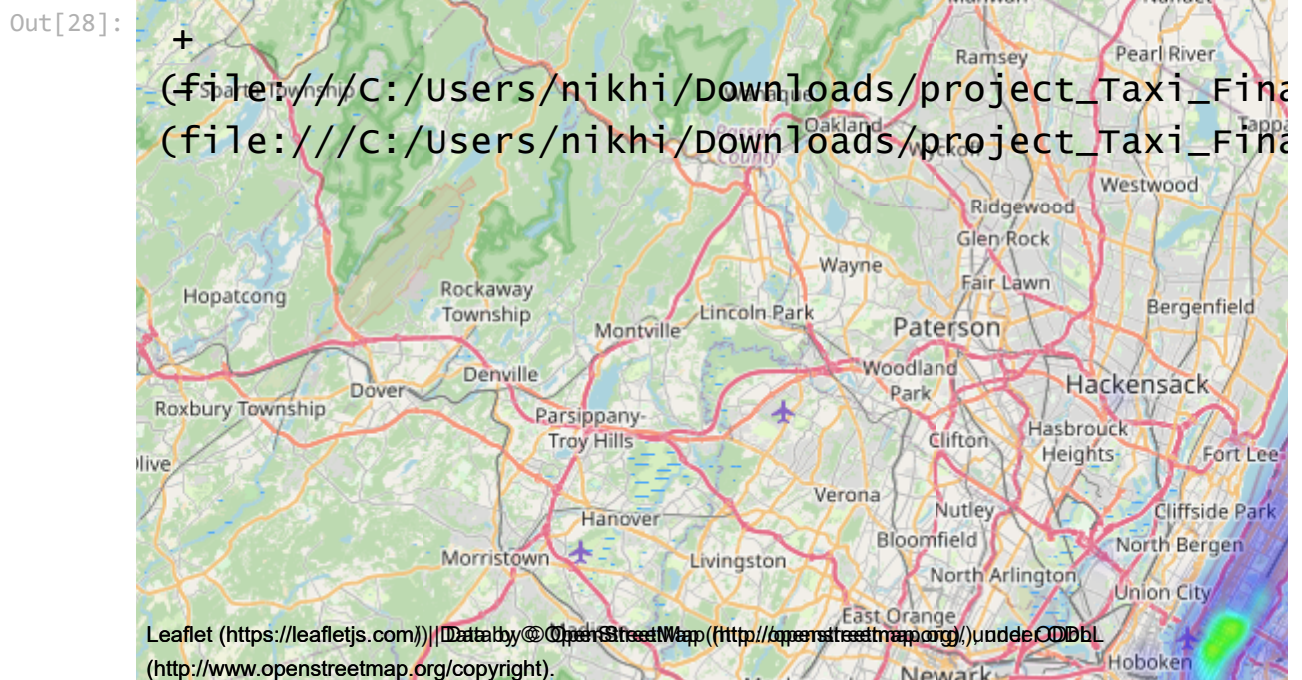
	0	1
0	40.778873	-73.953918
1	40.731743	-73.988312
2	40.721458	-73.997314
3	40.759720	-73.961670
4	40.708469	-74.017120

```
In [28]: from folium.plugins import HeatMap # import library heatmap
pickup_map = folium.Map(location = location, zoom_start = 10,)

hm_wide = HeatMap( pickup.values,
                    min_opacity= 0.2,
                    radius= 5, blur= 8,
                    max_zoom= 1
                    )

pickup_map.add_child(hm_wide)

pickup_map
```



Map view for Dropoff Point

```
In [29]: dropoff = pd.DataFrame(data)
dropoff.head()
```

```
Out[29]:
```

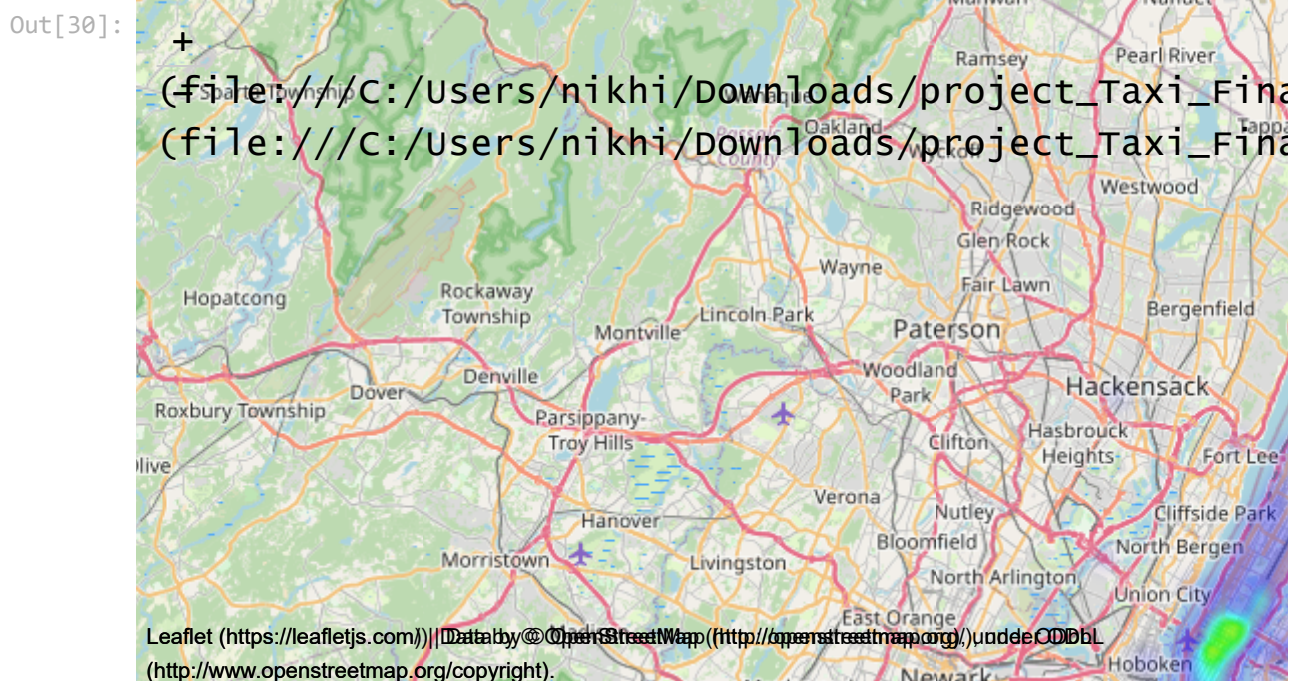
	0	1
0	40.778873	-73.953918
1	40.731743	-73.988312
2	40.721458	-73.997314
3	40.759720	-73.961670
4	40.708469	-74.017120

```
In [30]: dropoff_map = folium.Map(location = location, zoom_start = 10,)

hm_wide = HeatMap( dropoff.values,
                    min_opacity= 0.2,
                    radius= 5, blur= 8,
                    max_zoom= 1
                )

dropoff_map.add_child(hm_wide)

dropoff_map
```



The above seen green coloured area signify large number of taxi pickup and dropoff points, which is situated in Manhattan area of New York. Manhattan is among major commercial, financial and cultural centre in the world. People tend to travel more to manhattan from other parts of the city for work and tourism, so the map indicates more saturation on the areas of Manhattan.

```
In [31]: df.head()
```

```
Out[31]:
```

	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	PULocationID	D
0	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0	1.18	237	
1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0	0.20	161	
2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0	2.24	43	
3	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0	5.40	48	
4	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0	2.29	186	

```
In [32]: df['tpep_pickup_datetime']=pd.to_datetime(df["tpep_pickup_datetime"])
df['tpep_dropoff_datetime']=pd.to_datetime(df["tpep_dropoff_datetime"])
```

```
In [33]: print(type(df['tpep_pickup_datetime']))
print(type(df['tpep_dropoff_datetime']))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

Convert 'tpep_pickup_datetime' and 'tpep_dropoff_datetime' from object column to datetime column

```
In [34]: #Function to convert date-time features into date & time
def convert_to_date_dtype(Dataframe,col):

    Dataframe[col] = pd.to_datetime(Dataframe[col], format= '%d-%m-%Y %H:%M')
    Dataframe[col+'_day'] = Dataframe[col].dt.dayofweek
    Dataframe[col+'_month'] = Dataframe[col].dt.month
    Dataframe[col+'_hour'] = Dataframe[col].dt.hour
    Dataframe[col+'_minute'] = Dataframe[col].dt.minute
```

Function for converting 'tpep_pickup_datetime' and 'tpep_dropoff_datetime' from object column to datetime column. We can confirm it in the below cell.

```
In [35]: convert_to_date_dtype(df, 'tpep_pickup_datetime')
convert_to_date_dtype(df, 'tpep_dropoff_datetime')

df[['tpep_pickup_datetime', 'tpep_dropoff_datetime']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 2 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   tpep_pickup_datetime    30000 non-null  datetime64[ns]
1   tpep_dropoff_datetime    30000 non-null  datetime64[ns]
dtypes: datetime64[ns](2)
memory usage: 468.9 KB
```

'tpep_pickup_datetime' and 'tpep_dropoff_datetime' columns are further divided into respective tpep_pickup_datetime_day, tpep_pickup_datetime_month, tpep_pickup_datetime_hour, tpep_pickup_datetime_minute columns and tpep_dropoff_datetime_day, tpep_dropoff_datetime_month, tpep_dropoff_datetime_hour, tpep_dropoff_datetime_minute columns for accurate prediction.

```
In [36]: df.head()
```

```
Out[36]:
```

	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	PULocationID	DOLocationID
0	2022-01-31 16:45:42	2022-01-31 16:53:20	1.0	1.18	237	186
1	2022-01-20 07:07:09	2022-01-20 07:08:37	1.0	0.20	161	161
2	2022-01-30 14:00:06	2022-01-30 14:08:52	1.0	2.24	43	43
3	2022-01-27 08:32:15	2022-01-27 09:04:09	1.0	5.40	48	48
4	2022-01-01 09:05:42	2022-01-01 09:12:44	1.0	2.29	186	186

5 rows × 7 columns

```
In [37]: df=df.drop(labels=["tpep_pickup_datetime","tpep_dropoff_datetime"],axis=1) #Dropi
```

```
In [ ]:
```

```
In [38]: df.info() #Overview/Summary of Dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 26 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   passenger_count                       29051 non-null  float64
1   trip_distance                         30000 non-null  float64
2   PULocationID                          30000 non-null  int64
3   DOLocationID                          30000 non-null  int64
4   payment_type                          30000 non-null  int64
5   extra                                30000 non-null  float64
6   mta_tax                              30000 non-null  float64
7   tip_amount                           30000 non-null  float64
8   tolls_amount                         30000 non-null  float64
9   improvement_surcharge                 30000 non-null  float64
10  total_amount                          30000 non-null  float64
11  congestion_surcharge                   29051 non-null  float64
12  airport_fee                           29051 non-null  float64
13  pickup_longitude                      30000 non-null  float64
14  pickup_latitude                       30000 non-null  float64
15  dropoff_longitude                     30000 non-null  float64
16  dropoff_latitude                      30000 non-null  float64
17  trip_duration                         30000 non-null  float64
18  tpep_pickup_datetime_day              30000 non-null  int64
19  tpep_pickup_datetime_month            30000 non-null  int64
20  tpep_pickup_datetime_hour             30000 non-null  int64
21  tpep_pickup_datetime_minute           30000 non-null  int64
22  tpep_dropoff_datetime_day              30000 non-null  int64
23  tpep_dropoff_datetime_month            30000 non-null  int64
24  tpep_dropoff_datetime_hour             30000 non-null  int64
25  tpep_dropoff_datetime_minute           30000 non-null  int64
dtypes: float64(15), int64(11)
memory usage: 6.0 MB
```

```
In [39]: df.head()
```

```
Out[39]:
```

	passenger_count	trip_distance	PULocationID	DOLocationID	payment_type	extra	mta_tax	tip
0	1.0	1.18	237	236	1	1.0	0.5	
1	1.0	0.20	161	161	1	2.5	0.5	
2	1.0	2.24	43	142	1	0.0	0.5	
3	1.0	5.40	48	88	1	2.5	0.5	
4	1.0	2.29	186	162	1	0.0	0.5	

5 rows × 26 columns

```
In [40]: df['passenger_count'].value_counts()
```

```
Out[40]:
```

1.0	21816
2.0	4150
3.0	1035
0.0	639
5.0	570
4.0	451
6.0	390

Name: passenger_count, dtype: int64

```
In [41]: df['passenger_count'] = df['passenger_count'].fillna(1)
```

```
In [42]: df['passenger_count']
```

```
Out[42]:
```

0	1.0
1	1.0
2	1.0
3	1.0
4	1.0
...	
29995	1.0
29996	2.0
29997	1.0
29998	6.0
29999	1.0

Name: passenger_count, Length: 30000, dtype: float64

```
In [43]: df.isna().sum()
```



```
Out[43]: passenger_count      0
trip_distance      0
PULocationID      0
DOLocationID      0
payment_type      0
extra      0
mta_tax      0
tip_amount      0
tolls_amount      0
improvement_surcharge      0
total_amount      0
congestion_surcharge      949
airport_fee      949
pickup_longitude      0
pickup_latitude      0
dropoff_longitude      0
dropoff_latitude      0
trip_duration      0
tpep_pickup_datetime_day      0
tpep_pickup_datetime_month      0
tpep_pickup_datetime_hour      0
tpep_pickup_datetime_minute      0
tpep_dropoff_datetime_day      0
tpep_dropoff_datetime_month      0
tpep_dropoff_datetime_hour      0
tpep_dropoff_datetime_minute      0
dtype: int64
```

```
In [44]: df["passenger_count"].unique() #checking data element in the columns
```

```
Out[44]: array([1., 6., 2., 0., 3., 4., 5.]
```

```
In [45]: # sb.barplot(x="passenger_count", y="trip_duration",data=df); #Bivariate Analysis
# plt.title("Average Trip Duration (In seconds)");
# plt.xlabel("passenger_count");
# plt.ylabel("Trip Duration");
```

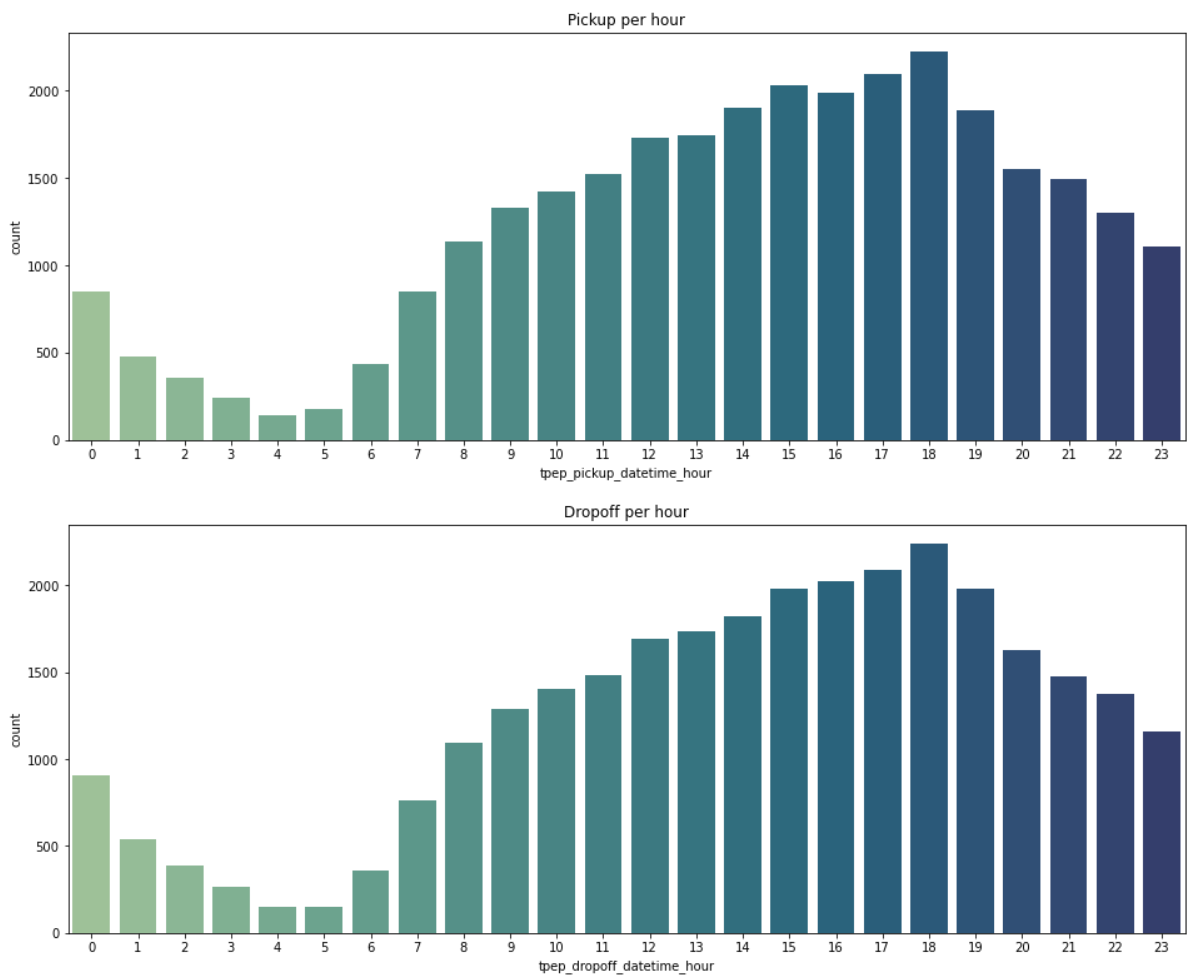
```
In [46]: plt.figure(figsize=(16, 6))
plt.title('Pickup per hour')

sns.countplot(x='tpep_pickup_datetime_hour', data=df, palette=("crest"))

plt.figure(figsize=(16, 6))
plt.title('Dropoff per hour')

sns.countplot(x='tpep_dropoff_datetime_hour', data=df, palette=("crest"))
```

```
Out[46]: <AxesSubplot:title={'center':'Dropoff per hour'}, xlabel='tpep_dropoff_datetime_ho
ur', ylabel='count'>
```



The graph shows count of passengers using the taxi at each hour of the day. It is clear that the city has high traffic from 8 am to 11 pm as most people tend to go for work and traffic peaks at evening 5pm to 6 pm.

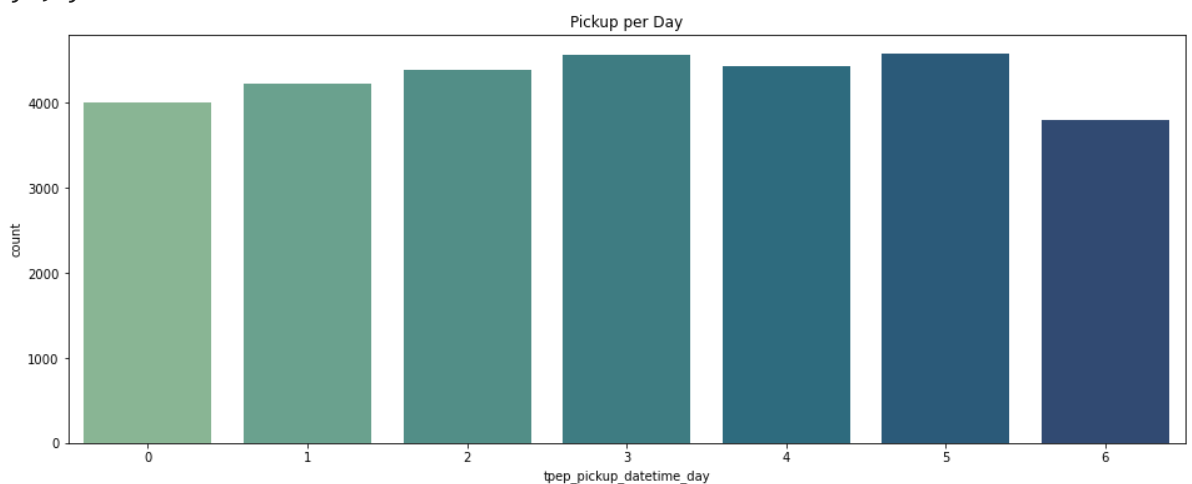
```
In [47]: plt.figure(figsize=(16, 6))
plt.title('Pickup per Day')

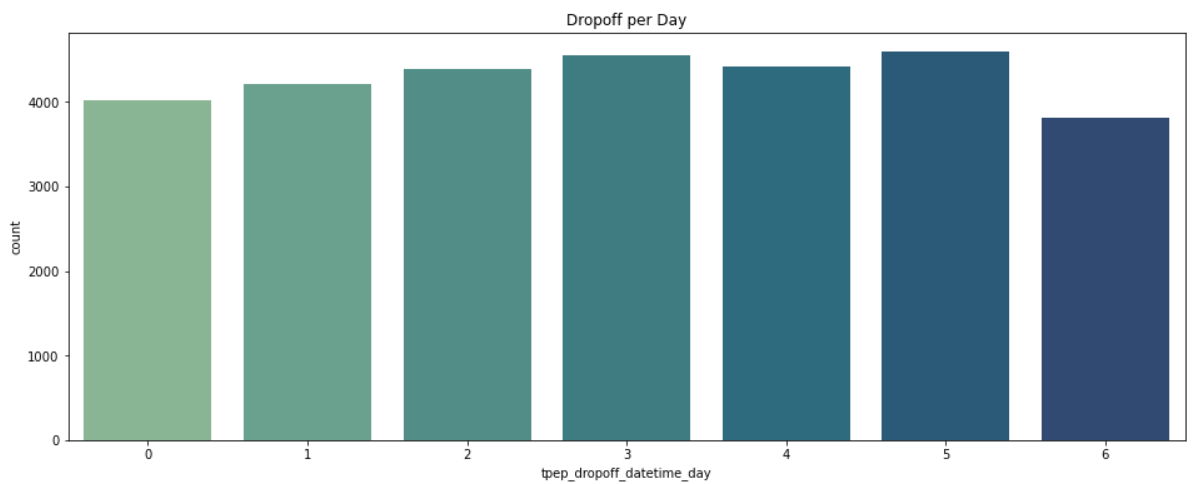
sns.countplot(x='tpep_pickup_datetime_day', data=df, palette=("crest"))

plt.figure(figsize=(16, 6))
plt.title('Dropoff per Day')

sns.countplot(x='tpep_dropoff_datetime_day', data=df, palette=("crest"))
```

```
Out[47]: <AxesSubplot:title={'center':'Dropoff per Day'}, xlabel='tpep_dropoff_datetime_da
y', ylabel='count'>
```





The graph determines less traffic on weekends compared to weekdays because of holiday to offices on weekends

Checking outliers

```
In [48]: df=df.drop(labels=['trip_distance', 'PULocationID', 'DOLocationID',
    'payment_type', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'total_amount',
    'improvement_surcharge', 'congestion_surcharge',
    'airport_fee'],axis=1)
```

df

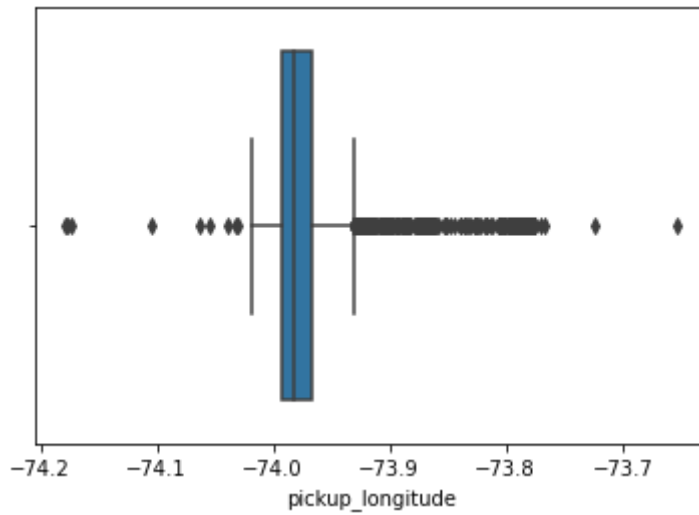
```
Out[48]:
```

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_distance
0	1.0	-73.953918	40.778873	-73.963875	40.771164	0.0
1	1.0	-73.988312	40.731743	-73.994751	40.694931	0.0
2	1.0	-73.997314	40.721458	-73.948029	40.774918	0.0
3	1.0	-73.961670	40.759720	-73.956779	40.780628	0.0
4	1.0	-74.017120	40.708469	-73.988182	40.740631	0.0
...
29995	1.0	-73.984726	40.759773	-73.865250	40.770683	0.0
29996	2.0	-73.980057	40.754837	-73.974434	40.759300	0.0
29997	1.0	-73.976624	40.788319	-73.970436	40.785755	0.0
29998	6.0	-74.005669	40.714779	-73.992653	40.687550	0.0
29999	1.0	-73.963661	40.777065	-73.965836	40.754364	0.0

30000 rows × 14 columns

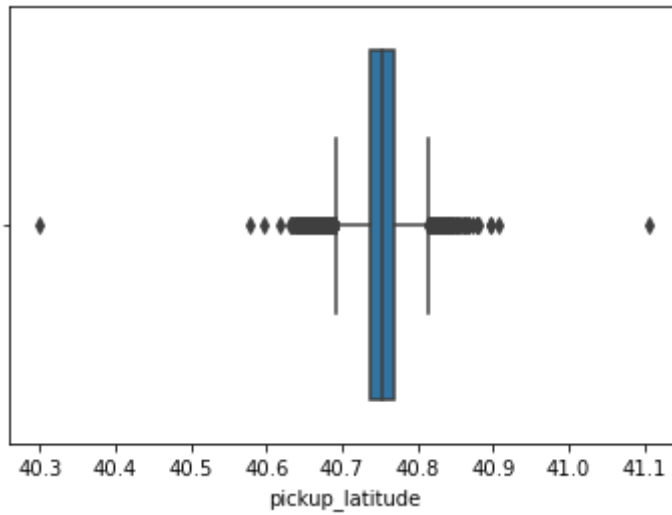
```
In [49]: sns.boxplot(x=df["pickup_longitude"])
```

```
Out[49]: <AxesSubplot:xlabel='pickup_longitude'>
```

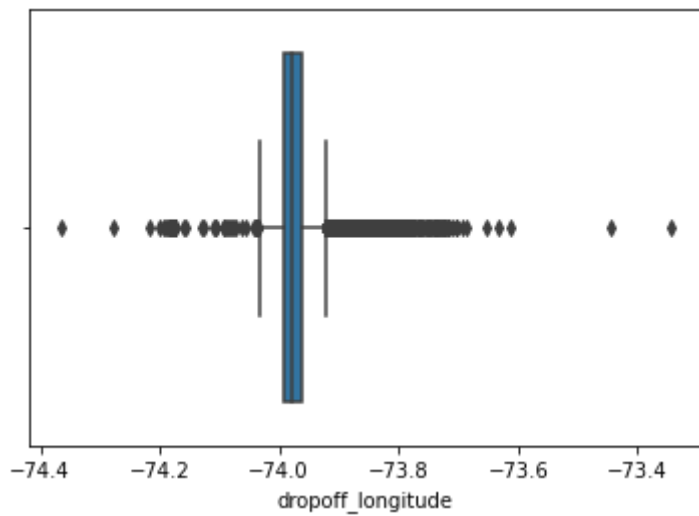
```
In [50]: sns.boxplot(x=df["pickup_latitude"])
```

```
Out[50]: <AxesSubplot:xlabel='pickup_latitude'>
```



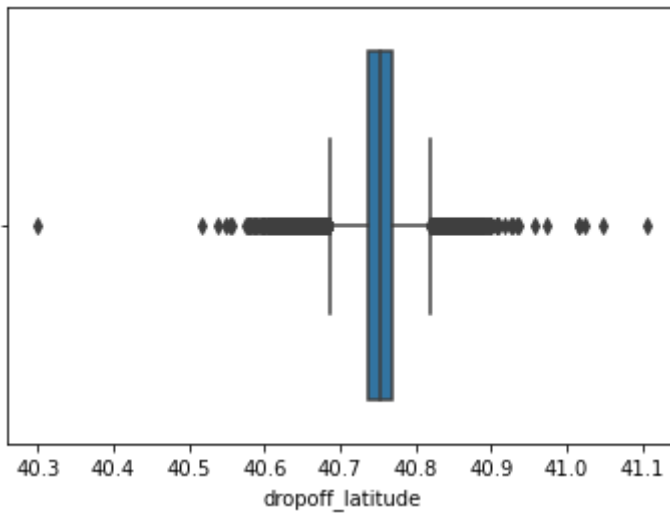
```
In [51]: sns.boxplot(x=df["dropoff_longitude"])
```

```
Out[51]: <AxesSubplot:xlabel='dropoff_longitude'>
```



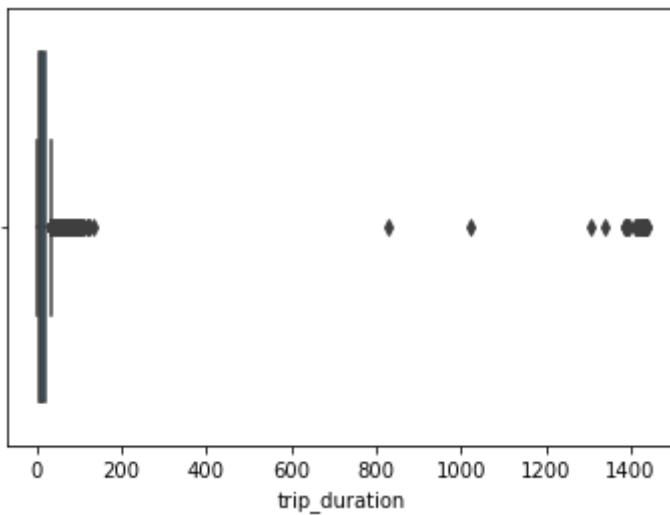
```
In [52]: sns.boxplot(x=df["dropoff_latitude"])
```

```
Out[52]: <AxesSubplot:xlabel='dropoff_latitude'>
```



```
In [53]: sns.boxplot(x=df["trip_duration"])
```

```
Out[53]: <AxesSubplot:xlabel='trip_duration'>
```



As we can see there are outliers in the dataset. We have to remove outliers so that our model error is reduced and predicts better

```
In [54]: df["trip_duration"].value_counts()
```

```
Out[54]: 6.0      1928
7.0      1908
8.0      1888
10.0     1780
9.0      1731
...
1438.0      1
95.0        1
109.0       1
122.0       1
826.0       1
Name: trip_duration, Length: 133, dtype: int64
```

```
In [55]: df.skew()
```

```
Out[55]: passenger_count      2.882108
pickup_longitude      3.260655
pickup_latitude      -1.135050
dropoff_longitude      2.512302
dropoff_latitude      -0.456944
trip_duration      27.005770
tpep_pickup_datetime_day      -0.019162
tpep_pickup_datetime_month      0.272407
tpep_pickup_datetime_hour      -0.609459
tpep_pickup_datetime_minute      -0.006582
tpep_dropoff_datetime_day      -0.019484
tpep_dropoff_datetime_month      0.272423
tpep_dropoff_datetime_hour      -0.651121
tpep_dropoff_datetime_minute      -0.000121
dtype: float64
```

skewness before outlier treatment

```
In [56]: def outlier_detect(df):
        for i in df.describe().columns:
            Q1=df.describe().at['25%',i]
            Q3=df.describe().at['75%',i]
            IQR=Q3 - Q1
            LTV=Q1 - 1.5 * IQR
            UTV=Q3 + 1.5 * IQR
            x=np.array(df[i])
            p=[]
            for j in x:
                if j < LTV or j>UTV:
                    p.append(df[i].median())
                else:
                    p.append(j)
            df[i]=p
        return df
```

Function for outlier removal : first we teke 25% data as Q1 and 75% data as Q3. Then we define Inter Quartile Range(IQR) as Q3-Q1. Then we define Lower Tube Values(LTV) and Upper Tube Values(UTV).

```
In [57]: new_df=outlier_detect(df)
```

new_df After removing outliers.

```
In [58]: new_df[["trip_duration"]]
```

```
Out[58]:
```

	trip_duration
0	8.0
1	1.0
2	9.0
3	32.0
4	7.0
...	...
29995	22.0
29996	10.0
29997	26.0
29998	11.0
29999	7.0

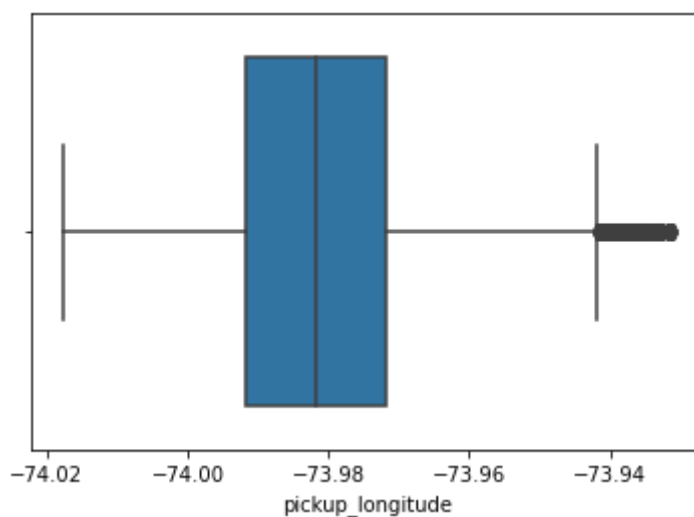
30000 rows × 1 columns

```
In [59]: df["passenger_count"].value_counts()
```

```
Out[59]: 1.0    30000
Name: passenger_count, dtype: int64
```

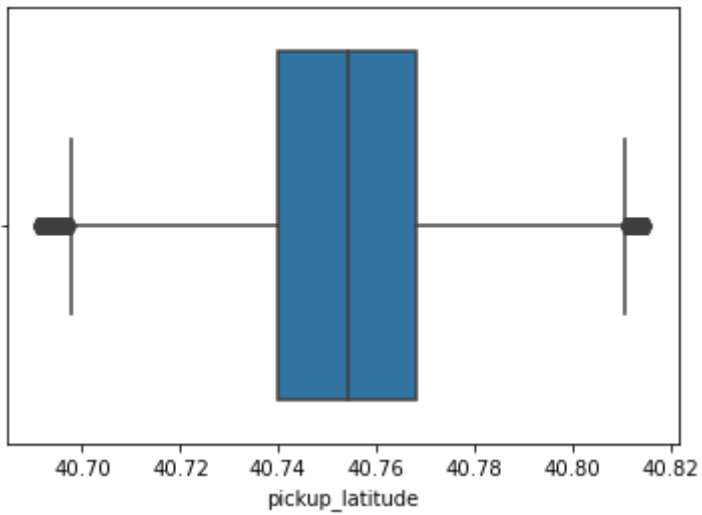
```
In [60]: sns.boxplot(x=df["pickup_longitude"])
```

```
Out[60]: <AxesSubplot:xlabel='pickup_longitude'>
```



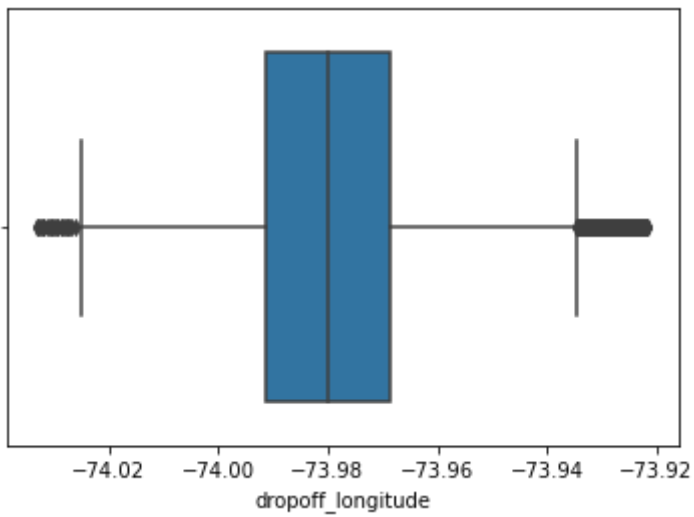
```
In [61]: sns.boxplot(x=df["pickup_latitude"])
```

```
Out[61]: <AxesSubplot:xlabel='pickup_latitude'>
```



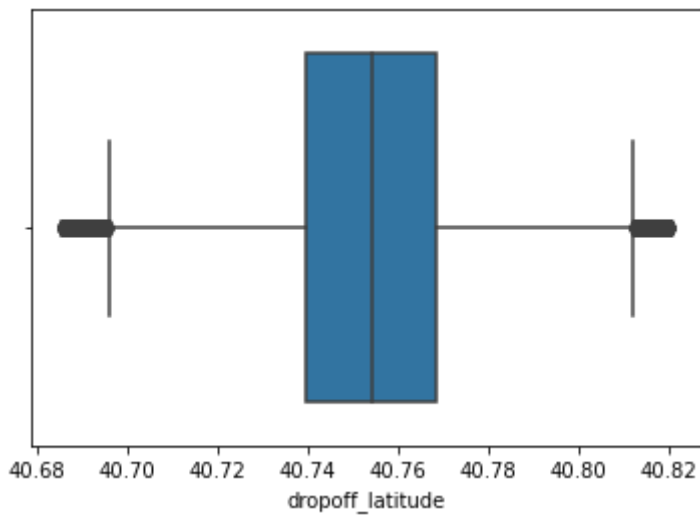
```
In [62]: sns.boxplot(x=df["dropoff_longitude"])
```

```
Out[62]: <AxesSubplot:xlabel='dropoff_longitude'>
```



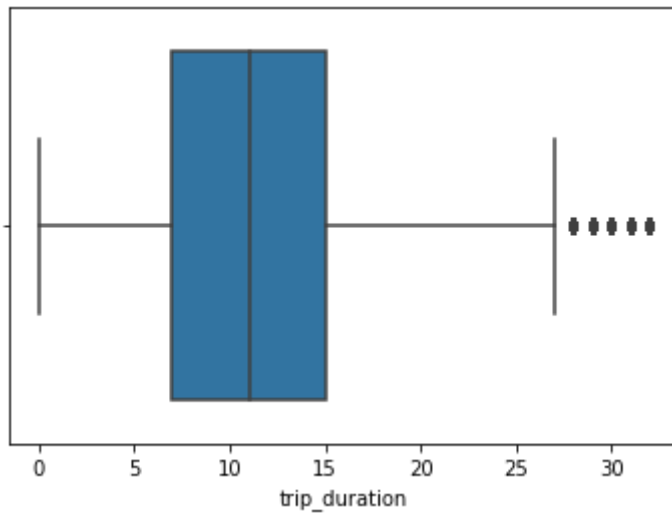
```
In [63]: sns.boxplot(x=df["dropoff_latitude"])
```

```
Out[63]: <AxesSubplot:xlabel='dropoff_latitude'>
```



```
In [64]: sns.boxplot(x=new_df["trip_duration"])
```

```
Out[64]: <AxesSubplot:xlabel='trip_duration'>
```



```
In [65]: # cheking skewness
```

```
In [66]: # new_df.skew()
new_df.skew()
```

```
Out[66]: passenger_count      0.000000
pickup_longitude      0.313151
pickup_latitude     -0.131339
dropoff_longitude      0.377558
dropoff_latitude     -0.175692
trip_duration         0.841548
tpep_pickup_datetime_day -0.019162
tpep_pickup_datetime_month 0.272407
tpep_pickup_datetime_hour -0.474223
tpep_pickup_datetime_minute -0.006582
tpep_dropoff_datetime_day -0.019484
tpep_dropoff_datetime_month 0.272056
tpep_dropoff_datetime_hour -0.651121
tpep_dropoff_datetime_minute -0.000121
dtype: float64
```

```
In [67]: # new_df.shape
new_df.shape
```

```
Out[67]: (30000, 14)
```

Dividing columns into categorical and continious

```
In [68]: cat=[]
con=[]
for i in new_df:
    if(df[i].dtypes=="object"):
        cat.append(i)
    else:
        con.append(i)
```

```
In [69]: cat
```

```
Out[69]: []
```

```
In [70]: new_df.head()
```

```
Out[70]:
```

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_d
0	1.0	-73.953918	40.778873	-73.963875	40.771164	
1	1.0	-73.988312	40.731743	-73.994751	40.694931	
2	1.0	-73.997314	40.721458	-73.948029	40.774918	
3	1.0	-73.961670	40.759720	-73.956779	40.780628	
4	1.0	-74.017120	40.708469	-73.988182	40.740631	

```
In [71]: con
```

```
Out[71]: ['passenger_count',
'pickup_longitude',
'pickup_latitude',
'dropoff_longitude',
'dropoff_latitude',
'trip_duration',
'tpep_pickup_datetime_day',
'tpep_pickup_datetime_month',
'tpep_pickup_datetime_hour',
'tpep_pickup_datetime_minute',
'tpep_dropoff_datetime_day',
'tpep_dropoff_datetime_month',
'tpep_dropoff_datetime_hour',
'tpep_dropoff_datetime_minute']
```

Model Building

```
In [72]: Y=new_df[["trip_duration"]] #Dependent Variable
X=new_df.drop(labels=["trip_duration"],axis=1) #Independent Variables
```

Standerdizing of the Data

Standardization is about making sure that data is internally consistant and each data type has same content and format so that all variables are contributing in prediction.

```
In [73]: from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
Xnew=pd.DataFrame(sc.fit_transform(X))
```

```
In [74]: Xnew
```

Out[74]:

	0	1	2	3	4	5	6	7	8
0	0.0	1.723993	1.190026	0.861446	0.752170	-1.538782	-1.098978	0.280684	0.896266
1	0.0	-0.428633	-1.028677	-0.857699	-2.469473	-0.001981	-1.098978	-1.479989	-1.293750
2	0.0	-0.992098	-1.512823	1.743745	0.910802	1.534821	-1.098978	-0.110577	-1.697174
3	0.0	1.238840	0.288358	1.256505	1.152135	-0.001981	-1.098978	-1.284359	0.147050
4	0.0	-2.231720	-2.124291	-0.491950	-0.538164	1.022554	-1.098978	-1.088729	-1.409014
...
29995	0.0	-0.204202	0.290872	-0.038694	0.731857	-0.514248	1.503077	-0.501838	-1.524278
29996	0.0	0.088035	0.058496	0.273530	0.250803	-1.026515	1.503077	-2.458141	-0.717430
29997	0.0	0.302916	1.634664	0.496122	1.368803	-0.001981	1.503077	0.867575	-0.429270
29998	0.0	-1.514974	-1.827267	-0.740880	-2.781417	-1.026515	1.503077	-0.893098	0.031786
29999	0.0	1.114210	1.104905	0.752274	0.042196	1.022554	1.503077	1.258836	-0.198742

30000 rows × 13 columns

In [75]: `Xnew.columns=['passenger_count','pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','tpep_pickup_datetime','tpep_dropoff_datetime','tpep_pickup_datetime_day','tpep_dropoff_datetime_day','tpep_pickup_datetime_month','tpep_dropoff_datetime_month','tpep_pickup_datetime_hour','tpep_dropoff_datetime_hour']`

In [76]: `Xnew`

Out[76]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	tpep_pickup_datetime	tpep_dropoff_datetime
0	0.0	1.723993	1.190026	0.861446	0.752170	2010-01-01 00:00:00	2010-01-01 00:00:00
1	0.0	-0.428633	-1.028677	-0.857699	-2.469473	2010-01-01 00:00:00	2010-01-01 00:00:00
2	0.0	-0.992098	-1.512823	1.743745	0.910802	2010-01-01 00:00:00	2010-01-01 00:00:00
3	0.0	1.238840	0.288358	1.256505	1.152135	2010-01-01 00:00:00	2010-01-01 00:00:00
4	0.0	-2.231720	-2.124291	-0.491950	-0.538164	2010-01-01 00:00:00	2010-01-01 00:00:00
...
29995	0.0	-0.204202	0.290872	-0.038694	0.731857	2010-01-01 00:00:00	2010-01-01 00:00:00
29996	0.0	0.088035	0.058496	0.273530	0.250803	2010-01-01 00:00:00	2010-01-01 00:00:00
29997	0.0	0.302916	1.634664	0.496122	1.368803	2010-01-01 00:00:00	2010-01-01 00:00:00
29998	0.0	-1.514974	-1.827267	-0.740880	-2.781417	2010-01-01 00:00:00	2010-01-01 00:00:00
29999	0.0	1.114210	1.104905	0.752274	0.042196	2010-01-01 00:00:00	2010-01-01 00:00:00

30000 rows × 13 columns

In [77]: `Xnew.columns` [#columns overview](#)


```
Out[77]: Index(['passenger_count', 'pickup_longitude', 'pickup_latitude',
            'dropoff_longitude', 'dropoff_latitude', 'tpep_pickup_datetime_day',
            'tpep_pickup_datetime_month', 'tpep_pickup_datetime_hour',
            'tpep_pickup_datetime_minute', 'tpep_dropoff_datetime_day',
            'tpep_dropoff_datetime_month', 'tpep_dropoff_datetime_hour',
            'tpep_dropoff_datetime_minute'],
            dtype='object')
```

Splitting Data into training and testing set

Split the data into training and testing as 80% and 20% respectively.

```
In [78]: from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(Xnew,Y,test_size=0.33,random_state=21)
```

```
In [79]: xtest
```

```
Out[79]:
```

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	t
6677	0.0	0.543582	0.126018	-0.148716	-0.530103	
20672	0.0	0.796664	0.521093	-0.477932	0.032040	
19682	0.0	-1.588511	-2.245686	-0.308864	0.042599	
23589	0.0	-1.523569	-0.626240	-0.852601	-0.122562	
16352	0.0	0.891211	0.474582	1.808313	0.538565	
...
15293	0.0	-0.549921	-1.089554	-1.209004	-0.490123	
11864	0.0	-1.196473	-1.013233	-0.483879	-0.176406	
20750	0.0	-0.410965	0.055264	-0.103263	-0.252175	
20273	0.0	0.484848	-0.136168	-0.038694	0.875819	
16655	0.0	-1.278128	-0.366029	-1.283767	-1.447717	

9900 rows × 13 columns

```
In [80]: xtrain
```

Out[80]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	t
3837	0.0	-0.095329	0.153494	-0.038694	0.042599	
14777	0.0	-0.616295	-0.149636	0.155437	0.188253	
8380	0.0	0.459063	1.383253	-0.038694	0.042599	
22950	0.0	-0.063336	-0.237271	0.021202	0.223881	
4542	0.0	1.738795	1.210677	-0.549298	0.394765	
...	
16432	0.0	0.402239	0.022491	-0.065881	-0.212356	
8964	0.0	0.263283	1.596593	-0.109210	0.370422	
5944	0.0	0.332044	-0.237450	0.361038	0.314159	
5327	0.0	0.684448	0.208265	0.311762	-0.157383	
15305	0.0	0.398896	0.371683	0.913696	-1.749988	

20100 rows × 13 columns

LinearRegression

```
In [81]: from sklearn.linear_model import LinearRegression
lr=LinearRegression()
Model=lr.fit(xtrain,ytrain)
pred_tr=Model.predict(xtrain)
pred_ts=Model.predict(xtest)

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score,accuracy_score
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest,pred_ts))

MAE: 5.279196232111102
MSE: 45.07322823362592
RMSE: 6.713659824091918
R2 0.005271572759711574
```

```
In [82]: # Visualize the performance of the model
```

Regularization of the model

Ridge

```
In [83]: tg = {"alpha": [0.991,0.992,0.993,0.994,0.995,0.996,0.997,0.998,0.999,1.001,1.002,1.003,1.004,1.005,1.006,1.007,1.008,1.009,1.01,1.011,1.012,1.013,1.014,1.015,1.016,1.017,1.018,1.019,1.02,1.021,1.022,1.023,1.024,1.025,1.026,1.027,1.028,1.029,1.03,1.031,1.032,1.033,1.034,1.035,1.036,1.037,1.038,1.039,1.04,1.041,1.042,1.043,1.044,1.045,1.046,1.047,1.048,1.049,1.05,1.051,1.052,1.053,1.054,1.055,1.056,1.057,1.058,1.059,1.06,1.061,1.062,1.063,1.064,1.065,1.066,1.067,1.068,1.069,1.07,1.071,1.072,1.073,1.074,1.075,1.076,1.077,1.078,1.079,1.08,1.081,1.082,1.083,1.084,1.085,1.086,1.087,1.088,1.089,1.09,1.091,1.092,1.093,1.094,1.095,1.096,1.097,1.098,1.099,1.1,1.101,1.102,1.103,1.104,1.105,1.106,1.107,1.108,1.109,1.11,1.111,1.112,1.113,1.114,1.115,1.116,1.117,1.118,1.119,1.12,1.121,1.122,1.123,1.124,1.125,1.126,1.127,1.128,1.129,1.13,1.131,1.132,1.133,1.134,1.135,1.136,1.137,1.138,1.139,1.14,1.141,1.142,1.143,1.144,1.145,1.146,1.147,1.148,1.149,1.15,1.151,1.152,1.153,1.154,1.155,1.156,1.157,1.158,1.159,1.16,1.161,1.162,1.163,1.164,1.165,1.166,1.167,1.168,1.169,1.17,1.171,1.172,1.173,1.174,1.175,1.176,1.177,1.178,1.179,1.18,1.181,1.182,1.183,1.184,1.185,1.186,1.187,1.188,1.189,1.19,1.191,1.192,1.193,1.194,1.195,1.196,1.197,1.198,1.199,1.2,1.201,1.202,1.203,1.204,1.205,1.206,1.207,1.208,1.209,1.21,1.211,1.212,1.213,1.214,1.215,1.216,1.217,1.218,1.219,1.22,1.221,1.222,1.223,1.224,1.225,1.226,1.227,1.228,1.229,1.23,1.231,1.232,1.233,1.234,1.235,1.236,1.237,1.238,1.239,1.24,1.241,1.242,1.243,1.244,1.245,1.246,1.247,1.248,1.249,1.25,1.251,1.252,1.253,1.254,1.255,1.256,1.257,1.258,1.259,1.26,1.261,1.262,1.263,1.264,1.265,1.266,1.267,1.268,1.269,1.27,1.271,1.272,1.273,1.274,1.275,1.276,1.277,1.278,1.279,1.28,1.281,1.282,1.283,1.284,1.285,1.286,1.287,1.288,1.289,1.29,1.291,1.292,1.293,1.294,1.295,1.296,1.297,1.298,1.299,1.3,1.301,1.302,1.303,1.304,1.305,1.306,1.307,1.308,1.309,1.31,1.311,1.312,1.313,1.314,1.315,1.316,1.317,1.318,1.319,1.32,1.321,1.322,1.323,1.324,1.325,1.326,1.327,1.328,1.329,1.33,1.331,1.332,1.333,1.334,1.335,1.336,1.337,1.338,1.339,1.34,1.341,1.342,1.343,1.344,1.345,1.346,1.347,1.348,1.349,1.35,1.351,1.352,1.353,1.354,1.355,1.356,1.357,1.358,1.359,1.36,1.361,1.362,1.363,1.364,1.365,1.366,1.367,1.368,1.369,1.37,1.371,1.372,1.373,1.374,1.375,1.376,1.377,1.378,1.379,1.38,1.381,1.382,1.383,1.384,1.385,1.386,1.387,1.388,1.389,1.39,1.391,1.392,1.393,1.394,1.395,1.396,1.397,1.398,1.399,1.4,1.401,1.402,1.403,1.404,1.405,1.406,1.407,1.408,1.409,1.41,1.411,1.412,1.413,1.414,1.415,1.416,1.417,1.418,1.419,1.42,1.421,1.422,1.423,1.424,1.425,1.426,1.427,1.428,1.429,1.43,1.431,1.432,1.433,1.434,1.435,1.436,1.437,1.438,1.439,1.44,1.441,1.442,1.443,1.444,1.445,1.446,1.447,1.448,1.449,1.45,1.451,1.452,1.453,1.454,1.455,1.456,1.457,1.458,1.459,1.46,1.461,1.462,1.463,1.464,1.465,1.466,1.467,1.468,1.469,1.47,1.471,1.472,1.473,1.474,1.475,1.476,1.477,1.478,1.479,1.48,1.481,1.482,1.483,1.484,1.485,1.486,1.487,1.488,1.489,1.49,1.491,1.492,1.493,1.494,1.495,1.496,1.497,1.498,1.499,1.5,1.501,1.502,1.503,1.504,1.505,1.506,1.507,1.508,1.509,1.51,1.511,1.512,1.513,1.514,1.515,1.516,1.517,1.518,1.519,1.52,1.521,1.522,1.523,1.524,1.525,1.526,1.527,1.528,1.529,1.53,1.531,1.532,1.533,1.534,1.535,1.536,1.537,1.538,1.539,1.54,1.541,1.542,1.543,1.544,1.545,1.546,1.547,1.548,1.549,1.55,1.551,1.552,1.553,1.554,1.555,1.556,1.557,1.558,1.559,1.56,1.561,1.562,1.563,1.564,1.565,1.566,1.567,1.568,1.569,1.57,1.571,1.572,1.573,1.574,1.575,1.576,1.577,1.578,1.579,1.58,1.581,1.582,1.583,1.584,1.585,1.586,1.587,1.588,1.589,1.59,1.591,1.592,1.593,1.594,1.595,1.596,1.597,1.598,1.599,1.6,1.601,1.602,1.603,1.604,1.605,1.606,1.607,1.608,1.609,1.61,1.611,1.612,1.613,1.614,1.615,1.616,1.617,1.618,1.619,1.62,1.621,1.622,1.623,1.624,1.625,1.626,1.627,1.628,1.629,1.63,1.631,1.632,1.633,1.634,1.635,1.636,1.637,1.638,1.639,1.64,1.641,1.642,1.643,1.644,1.645,1.646,1.647,1.648,1.649,1.65,1.651,1.652,1.653,1.654,1.655,1.656,1.657,1.658,1.659,1.66,1.661,1.662,1.663,1.664,1.665,1.666,1.667,1.668,1.669,1.67,1.671,1.672,1.673,1.674,1.675,1.676,1.677,1.678,1.679,1.68,1.681,1.682,1.683,1.684,1.685,1.686,1.687,1.688,1.689,1.69,1.691,1.692,1.693,1.694,1.695,1.696,1.697,1.698,1.699,1.7,1.701,1.702,1.703,1.704,1.705,1.706,1.707,1.708,1.709,1.71,1.711,1.712,1.713,1.714,1.715,1.716,1.717,1.718,1.719,1.72,1.721,1.722,1.723,1.724,1.725,1.726,1.727,1.728,1.729,1.73,1.731,1.732,1.733,1.734,1.735,1.736,1.737,1.738,1.739,1.74,1.741,1.742,1.743,1.744,1.745,1.746,1.747,1.748,1.749,1.75,1.751,1.752,1.753,1.754,1.755,1.756,1.757,1.758,1.759,1.76,1.761,1.762,1.763,1.764,1.765,1.766,1.767,1.768,1.769,1.77,1.771,1.772,1.773,1.774,1.775,1.776,1.777,1.778,1.779,1.78,1.781,1.782,1.783,1.784,1.785,1.786,1.787,1.788,1.789,1.79,1.791,1.792,1.793,1.794,1.795,1.796,1.797,1.798,1.799,1.8,1.801,1.802,1.803,1.804,1.805,1.806,1.807,1.808,1.809,1.81,1.811,1.812,1.813,1.814,1.815,1.816,1.817,1.818,1.819,1.82,1.821,1.822,1.823,1.824,1.825,1.826,1.827,1.828,1.829,1.83,1.831,1.832,1.833,1.834,1.835,1.836,1.837,1.838,1.839,1.84,1.841,1.842,1.843,1.844,1.845,1.846,1.847,1.848,1.849,1.85,1.851,1.852,1.853,1.854,1.855,1.856,1.857,1.858,1.859,1.86,1.861,1.862,1.863,1.864,1.865,1.866,1.867,1.868,1.869,1.87,1.871,1.872,1.873,1.874,1.875,1.876,1.877,1.878,1.879,1.88,1.881,1.882,1.883,1.884,1.885,1.886,1.887,1.888,1.889,1.89,1.891,1.892,1.893,1.894,1.895,1.896,1.897,1.898,1.899,1.9,1.901,1.902,1.903,1.904,1.905,1.906,1.907,1.908,1.909,1.91,1.911,1.912,1.913,1.914,1.915,1.916,1.917,1.918,1.919,1.92,1.921,1.922,1.923,1.924,1.925,1.926,1.927,1.928,1.929,1.93,1.931,1.932,1.933,1.934,1.935,1.936,1.937,1.938,1.939,1.94,1.941,1.942,1.943,1.944,1.945,1.946,1.947,1.948,1.949,1.95,1.951,1.952,1.953,1.954,1.955,1.956,1.957,1.958,1.959,1.96,1.961,1.962,1.963,1.964,1.965,1.966,1.967,1.968,1.969,1.97,1.971,1.972,1.973,1.974,1.975,1.976,1.977,1.978,1.979,1.98,1.981,1.982,1.983,1.984,1.985,1.986,1.987,1.988,1.989,1.99,1.991,1.992,1.993,1.994,1.995,1.996,1.997,1.998,1.999,2.0,2.001,2.002,2.003,2.004,2.005,2.006,2.007,2.008,2.009,2.01,2.011,2.012,2.013,2.014,2.015,2.016,2.017,2.018,2.019,2.02,2.021,2.022,2.023,2.024,2.025,2.026,2.027,2.028,2.029,2.03,2.031,2.032,2.033,2.034,2.035,2.036,2.037,2.038,2.039,2.04,2.041,2.042,2.043,2.044,2.045,2.046,2.047,2.048,2.049,2.05,2.051,2.052,2.053,2.054,2.055,2.056,2.057,2.058,2.059,2.06,2.061,2.062,2.063,2.064,2.065,2.066,2.067,2.068,2.069,2.07,2.071,2.072,2.073,2.074,2.075,2.076,2.077,2.078,2.079,2.08,2.081,2.082,2.083,2.084,2.085,2.086,2.087,2.088,2.089,2.09,2.091,2.092,2.093,2.094,2.095,2.096,2.097,2.098,2.099,2.1,2.101,2.102,2.103,2.104,2.105,2.106,2.107,2.108,2.109,2.11,2.111,2.112,2.113,2.114,2.115,2.116,2.117,2.118,2.119,2.12,2.121,2.122,2.123,2.124,2.125,2.126,2.127,2.128,2.129,2.13,2.131,2.132,2.133,2.134,2.135,2.136,2.137,2.138,2.139,2.14,2.141,2.142,2.143,2.144,2.145,2.146,2.147,2.148,2.149,2.15,2.151,2.152,2.153,2.154,2.155,2.156,2.157,2.158,2.159,2.16,2.161,2.162,2.163,2.164,2.165,2.166,2.167,2.168,2.169,2.17,2.171,2.172,2.173,2.174,2.175,2.176,2.177,2.178,2.179,2.18,2.181,2.182,2.183,2.184,2.185,2.186,2.187,2.188,2.189,2.19,2.191,2.192,2.193,2.194,2.195,2.196,2.197,2.198,2.199,2.2,2.201,2.202,2.203,2.204,2.205,2.206,2.207,2.208,2.209,2.21,2.211,2.212,2.213,2.214,2.215,2.216,2.217,2.218,2.219,2.22,2.221,2.222,2.223,2.224,2.225,2.226,2.227,2.228,2.229,2.23,2.231,2.232,2.233,2.234,2.235,2.236,2.237,2.238,2.239,2.24,2.241,2.242,2.243,2.244,2.245,2.246,2.247,2.248,2.249,2.25,2.251,2.252,2.253,2.254,2.255,2.256,2.257,2.258,2.259,2.26,2.261,2.262,2.263,2.264,2.265,2.266,2.267,2.268,2.269,2.27,2.271,2.272,2.273,2.274,2.275,2.276,2.277,2.278,2.279,2.28,2.281,2.282,2.283,2.284,2.285,2.286,2.287,2.288,2.289,2.29,2.291,2.292,2.293,2.294,2.295,2.296,2.297,2.298,2.299,2.3,2.301,2.302,2.303,2.304,2.305,2.306,2.307,2.308,2.309,2.31,2.311,2.312,2.313,2.314,2.315,2.316,2.317,2.318,2.319,2.32,2.321,2.322,2.323,2.324,2.325,2.326,2.327,2.328,2.329,2.33,2.331,2.332,2.333,2.334,2.335,2.336,2.337,2.338,2.339,2.34,2.341,2.342,2.343,2.344,2.345,2.346,2.347,2.348,2.349,2.35,2.351,2.352,2.353,2.354,2.355,2.356,2.357,2.358,2.359,2.36,2.361,2.362,2.363,2.364,2.365,2.366,2.367,2.368,2.369,2.37,2.371,2.372,2.373,2.374,2.375,2.376,2.377,2.378,2.379,2.38,2.381,2.382,2.383,2.384,2.385,2.386,2.387,2.388,2.389,2.39,2.391,2.392,2.393,2.394,2.395,2.396,2.397,2.398,2.399,2.4,2.401,2.402,2.403,2.404,2.405,2.406,2.407,2.408,2.409,2.41,2.411,2.412,2.413,2.414,2.415,2.416,2.417,2.418,2.419,2.42,2.421,2.422,2.423,2.424,2.425,2.426,2.427,2.428,2.429,2.43,2.431,2.432,2.433,2.434,2.435,2.436,2.437,2.438,2.439,2.44,2.441,2.442,2.443,2.444,2.445,2.446,2.447,2.448,2.449,2.45,2.451,2.452,2.453,2.454,2.455,2.456,2.457,2.458,2.459,2.46,2.461,2.462,2.463,2.464,2.465,2.466,2.467,2.468,2.469,2.47,2.471,2.472,2.473,2.474,2.475,2.476,2.477,2.478,2.479,2.48,2.481,2.482,2.483,2.484,2.485,2.486,2.487,2.488,2.489,2.49,2.491,2.492,2.493,2.494,2.495,2.496,2.497,2.498,2.499,2.5,2.501,2.502,2.503,2.504,2.505,2.506,2.507,2.508,2.509,2.51,2.511,2.512,2.513,2.514,2.515,2.516,2.517,2.518,2.519,2.52,2.521,2.522,2.523,2.524,2.525,2.526,2.527,2.528,2.529,2.53,2.531,2.532,2.533,2.534,2.535,2.536,2.537,2.538,2.539,2.54,2.541,2.542,2.543,2.544,2.545,2.546,2.547,2.548,2.549,2.55,2.551,2.552,2.553,2.554,2.555,2.556,2.557,2.558,2.559,2.56,2.561,2.562,2.563,2.564,2.565,2.566,2.567,2.568,2.569,2.57,2.571,2.572,2.573,2.574,2.575,2.576,2.577,2.578,2.579,2.58,2.581,2.582,2.583,2.584,2.585,2.586,2.587,2.588,2.589,2.59,2.591,2.592,2.593,2.594,2.595,2.596,2.597,2.598,2.599,2.6,2.601,2.602,2.603,2.604,2.605,2.606,2.607,2.608,2.609,2.61,2.611,2.612,2.613,2.614,2.615,2.616,2.617,2.618,2.619,2.62,2.621,2.622,2.623,2.624,2.625,2.626,2.627,2.628,2.629,2.63,2.631,2.632,2.633,2.634,2.635,2.636,2.637,2.638,2.639,2.64,2.641,2.642,2.643,2.644,2.645,2.646,2.647,2.648,2.649,2.65,2.651,2.652,2.653,2.654,2.655,2.656,2.657,2.658,2.659,2.66,2.661,2.662,2.663,2.664,2.665,2.666,2.667,2.668,2.669,2.67,2.671,2.672,2.673,2.674,2.675,2.676,2.677,2.678,2.679,2.68,2.681,2.682,2.683,2.684,2.685,2.686,2.687,2.688,2.689,2.69,2.691,2.692,2.693,2.694,2.695,2.696,2.697,2.698,2.699,2.7,2.701,2.702,2.703,2.704,2.705,2.706,2.707,2.708,2.709,2.71,2.711,2.712,2.713,2.714,2.715,2.716,2.717,2.718,2.719,2.72,2.721,2.722,2.723,2.724,2.725,2.726,2.727,2.728,2.729,2.73,2.731,2.732,2.733,2.734,2.735,2.736,2.737,2.738,2.739,2.74,2.741,2.742,2.743,2.744,2.745,2.746,2.747,2.748,2.749,2.75,2.751,2.752,2.753,2.754,2.755,2.756,2.757,2.758,2.759,2.76,2.761,2.762,2.763,2.764,2.765,2.766,2.767,2.768,2.769,2.77,2.771,2.772,2.773,2.774,2.775,2.776,2.777,2.778,2.779,2.78,2.781,2.782,2.783,2.784,2.785,2.786,2.787,2.788,2.789,2.79,2.791,2.792,2.793,2.794,2.795,2.796,2.797,2.798,2.799,2.8,2.801,2.802,2.803,2.804,2.805,2.806,2.807,2.808,2.809,2.81,2.811,2.812,2.813,2.814,2.815,2.816,2.817,2.818,2.819,2.82,2.821,2.822,2.823,2.824,2.825,2.826,2.827,2.828,2.829,2.83,2.831,2.832,2.833,2.83
```

```

from sklearn.model_selection import GridSearchCV
cv1 = GridSearchCV(rr,tg,scoring="neg_mean_absolute_error",cv=4)
cvmodel1 = cv1.fit(xtrain,ytrain)
cvmodel1.best_params_

```

Out[84]: {'alpha': 0.991}

```

In [85]: rr = Ridge(alpha=1.009)

model = rr.fit(xtrain,ytrain)

pred_tr = model.predict(xtrain)
pred_ts = model.predict(xtest)

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score,accuracy_score
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest,pred_ts))

MAE: 5.279028385181044
MSE: 45.07282206645367
RMSE: 6.713629574712449
R2 0.005280536529299162

```

Lasso

```

In [86]: ls = Lasso()
from sklearn.model_selection import GridSearchCV
cv2 = GridSearchCV(ls,tg,scoring="neg_mean_absolute_error",cv=4)
cvmodel2 = cv2.fit(xtrain,ytrain)
cvmodel2.best_params_

```

Out[86]: {'alpha': 0.991}

```

In [87]: ls = Lasso(alpha=1.009)

model = ls.fit(xtrain,ytrain)

pred_tr = model.predict(xtrain)
pred_ts = model.predict(xtest)

from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score,accuracy_score
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest,pred_ts))

MAE: 5.288062716719432
MSE: 45.314321540195905
RMSE: 6.731591308167475
R2 -4.915475551015014e-05

```

PolynomialRegression

```

In [88]: from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 3)
xtrain_poly= poly_regs.fit_transform(xtrain)

```

```
xtest_poly= poly_regs.fit_transform(xtest)
lin_reg_2 =LinearRegression()
model=lin_reg_2.fit(xtrain_poly,ytrain)
pred_tr=model.predict(xtrain_poly)
pred_ts=model.predict(xtest_poly)
```

```
In [89]: from sklearn.metrics import mean_absolute_error,mean_squared_error
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest,pred_ts))
```

```
MAE: 64971134.88780944
MSE: 1.3777716933578338e+19
RMSE: 3711834712.5886865
R2 -3.040626827362652e+17
```

```
In [90]: lin_reg_2.score(xtest_poly,ytest)
```

```
Out[90]: -3.040626827362652e+17
```

Support Vector Regression

```
In [91]: # from sklearn.svm import SVR
# svr = SVR(kernel = 'linear')
# model=svr.fit(xtrain,ytrain)
# pred_tr=model.predict(xtrain)
# pred_ts=model.predict(xtest)
# #
# from sklearn.metrics import mean_absolute_error,mean_squared_error
# print('MAE:', mean_absolute_error(ytest, pred_ts))
# print('MSE:', mean_squared_error(ytest, pred_ts))
# print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
# print('R2', r2_score(ytest,pred_ts))
# print("-----")
# tr_err=mean_absolute_error(ytrain,pred_tr)
# ts_err=mean_absolute_error(ytest,pred_ts)
# print("Training error=",tr_err)
# print("Testing error=",ts_err)
# print("-----")

# if(tr_err<ts_err):
#     print("model is overfited")
# else:
#     print("model is underfited")
```

```
plt.plot(xtrain,pred_tr,color='r') plt.plot(X,Y,'b.') plt.xlabel("xtrain") plt.ylabel("pred_tr")
plt.show()
```

KnnRegression

```
In [92]: from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors=5)
model = knr.fit(xtrain,ytrain)
pred = model.predict(xtest)
from sklearn.metrics import mean_absolute_error,mean_squared_error
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
```

```
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest, pred_ts))
```

```
MAE: 64971134.88780944
MSE: 1.3777716933578338e+19
RMSE: 3711834712.5886865
R2 -3.040626827362652e+17
```

Hyperparameter tuning for KNeighborsRegressor

```
In [93]: for i in range(2,10,1):
          from sklearn.neighbors import KNeighborsRegressor
          knr = KNeighborsRegressor(n_neighbors=i)
          model = knr.fit(xtrain,ytrain)
          pred = model.predict(xtest)

          from sklearn.metrics import mean_absolute_error
          print(i,mean_absolute_error(ytest,pred))
```

```
2 4.721262626262626
3 4.488956228956229
4 4.3843939393939394
5 4.3437777777777775
6 4.3146127946127955
7 4.303290043290043
8 4.297739898989899
9 4.300819304152637
```

Decision Tree

```
In [94]: from sklearn.tree import DecisionTreeRegressor
          dtr=DecisionTreeRegressor(max_depth=10,random_state=10)
          Model=dtr.fit(xtrain,ytrain)
          pred_tr=Model.predict(xtrain)
          pred_ts=Model.predict(xtest)

          from sklearn.metrics import mean_absolute_error,accuracy_score,r2_score,mean_squared_error
          print('MAE:', mean_absolute_error(ytest, pred_ts))
          print('MSE:', mean_squared_error(ytest, pred_ts))
          print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
          print('R2', r2_score(ytest, pred_ts))
```

```
MAE: 4.922622563144734
MSE: 40.825302121439286
RMSE: 6.389468062479011
R2 0.09901974714620898
```

```
plt.plot(xtrain,pred_tr,color='r') plt.plot(X,Y,'b.') plt.xlabel("xtrain") plt.ylabel("pred_tr")
plt.show()
```

```
In [95]: !pip install graphviz
```

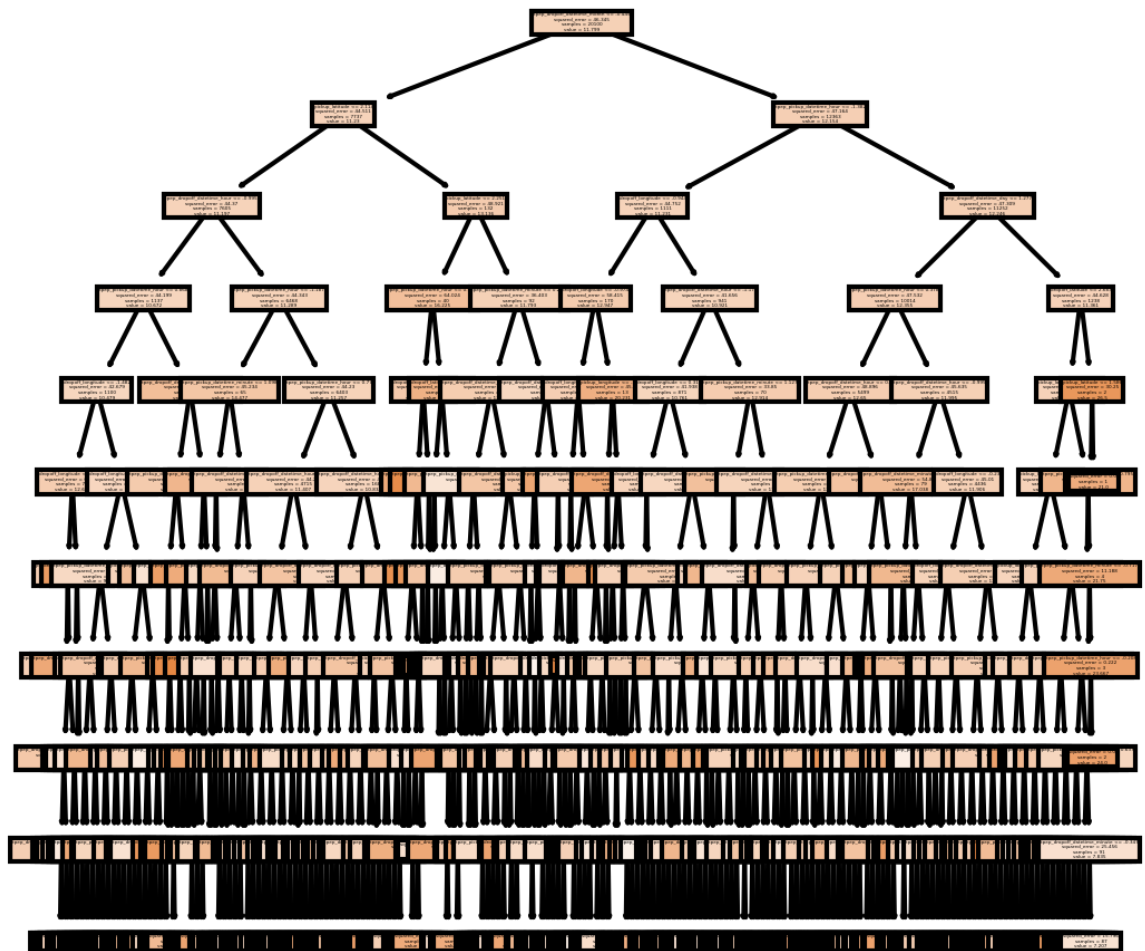
```
Requirement already satisfied: graphviz in c:\users\nikhi\anaconda3\lib\site-packages (0.20.1)
```

```
In [96]: from graphviz import *
          from sklearn.tree import DecisionTreeClassifier, plot_tree
          fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
```

```

plot_tree(dtr,
          feature_names = xtrain.columns,
          class_names=new_df.trip_duration,
          filled = True);
fig.savefig('imagename.png')

```



RandomForest

```

In [97]: from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=20,random_state=21)
Model=rf.fit(xtrain,ytrain)
pred=Model.predict(xtrain)
pred_tr=Model.predict(xtrain)
pred_ts=Model.predict(xtest)
from sklearn.metrics import mean_absolute_error,accuracy_score,r2_score,mean_squared_error
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest,pred_ts))

```

```

MAE: 1.6452828282828282
MSE: 6.361890909090909
RMSE: 2.522789118356657
R2 0.8595983916334363

```

```

plt.plot(xtrain,pred_tr,color='r') plt.plot(X,Y,'b.') plt.xlabel("xtrain") plt.ylabel("pred_tr")
plt.show()

```

AdaboostRegressor

```
In [98]: from sklearn.ensemble import AdaBoostRegressor
adb=AdaBoostRegressor(DecisionTreeRegressor(max_depth=20),n_estimators=500,learning_rate=0.1)
model=adb.fit(xtrain,ytrain)
pred_tr=Model.predict(xtrain)
pred_ts=Model.predict(xtest)
from sklearn.metrics import mean_absolute_error,accuracy_score,r2_score,mean_squared_error
print('MAE:', mean_absolute_error(ytest, pred_ts))
print('MSE:', mean_squared_error(ytest, pred_ts))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts)))
print('R2', r2_score(ytest,pred_ts))

MAE: 1.6452828282828282
MSE: 6.361890909090909
RMSE: 2.5222789118356657
R2 0.8595983916334363
```

We have MAE as 1.7783 and RMSE as 2.7799 which are very close to 0 and R2 score is 0.8298 which is close to 1. Thus we can say that the model is predicting with minimal errors and good accuracy.

```
In [99]: pred_tr
```

```
Out[99]: array([ 5.2 ,  9.6 , 15.05, ..., 12.85, 10.05,  7.2 ])
```

```
plt.plot(xtrain,pred_tr,color='r') plt.plot(X,Y,'b.') plt.xlabel("xtrain") plt.ylabel("pred_tr")
plt.show()
```

XGboost

```
In [100... !pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\nikhi\anaconda3\lib\site-packages (1.7.1)
Requirement already satisfied: scipy in c:\users\nikhi\anaconda3\lib\site-packages (from xgboost) (1.7.3)
Requirement already satisfied: numpy in c:\users\nikhi\anaconda3\lib\site-packages (from xgboost) (1.21.5)
```

```
In [101... from xgboost import XGBRegressor
xgbr=XGBRegressor()
model=xgbr.fit(xtrain,ytrain)
```

```
In [102... pred_tr1=Model.predict(xtrain)
pred_ts1=Model.predict(xtest)
from sklearn.metrics import mean_absolute_error,accuracy_score,r2_score,mean_squared_error
print('MAE:', mean_absolute_error(ytest, pred_ts1))
print('MSE:', mean_squared_error(ytest, pred_ts1))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts1)))
print('R2', r2_score(ytest,pred_ts1))

MAE: 1.6452828282828282
MSE: 6.361890909090909
RMSE: 2.5222789118356657
R2 0.8595983916334363
```

```
plt.plot(xtrain,pred_tr,color='r') plt.plot(X,Y,'b.') plt.xlabel("xtrain") plt.ylabel("pred_tr")
```

```
plt.show()
```

BayesianRidge

```
In [103... from sklearn.linear_model import BayesianRidge
bysn=BayesianRidge()
model=bysn.fit(xtrain,ytrain)
pred_tr1=Model.predict(xtrain)
pred_ts1=Model.predict(xtest)
from sklearn.metrics import mean_absolute_error,accuracy_score,r2_score,mean_squared_error
print('MAE:', mean_absolute_error(ytest, pred_ts1))
print('MSE:', mean_squared_error(ytest, pred_ts1))
print('RMSE:', np.sqrt(mean_squared_error(ytest, pred_ts1)))
print('R2', r2_score(ytest,pred_ts1))
print("-----")
tr_err=mean_absolute_error(ytrain,pred_tr)
ts_err=mean_absolute_error(ytest,pred_ts)
print("Training error=",tr_err)
print("Testing error=",ts_err)
print("-----")

if(tr_err<ts_err):
    print("model is overfited")
else:
    print("model is underfited")
```

```
MAE: 1.6452828282828282
MSE: 6.361890909090909
RMSE: 2.5222789118356657
R2 0.8595983916334363
```

```
-----
Training error= 0.6866343283582089
Testing error= 1.6452828282828282
-----
```

```
model is overfited
```

```
In [104... ytest
```

```
Out[104]:
```

	trip_duration
6677	6.0
20672	7.0
19682	11.0
23589	32.0
16352	11.0
...	...
15293	30.0
11864	6.0
20750	15.0
20273	14.0
16655	22.0

9900 rows × 1 columns


```
In [105...] ytest.value_counts()
```

```
Out[105]: trip_duration
11.0      1120
7.0       646
6.0       634
8.0       612
10.0      574
5.0       555
9.0       535
12.0      502
4.0       475
13.0      435
14.0      380
3.0       344
15.0      343
16.0      319
17.0      252
18.0      225
19.0      222
20.0      201
21.0      178
2.0       157
23.0      153
22.0      149
24.0      122
25.0      108
27.0      106
26.0       93
0.0       93
28.0       72
1.0       69
29.0       65
31.0       59
30.0       53
32.0       49
dtype: int64
```

```
In [106...] pred_ts1
```

```
Out[106]: array([ 6.5 ,  7.45, 10.8 , ..., 15.75, 12.55, 21.35])
```

create a model pickle file

```
In [107...] # Adaboost_model
# pickle.dump(adb, open('AD_model.pkl', 'wb'))
import pickle
pickle.dump(rf, open('R_model.pkl', 'wb'))
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

