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.sampl
```

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_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.

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 iwhether 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 statastical unimportants 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()    #statastical 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

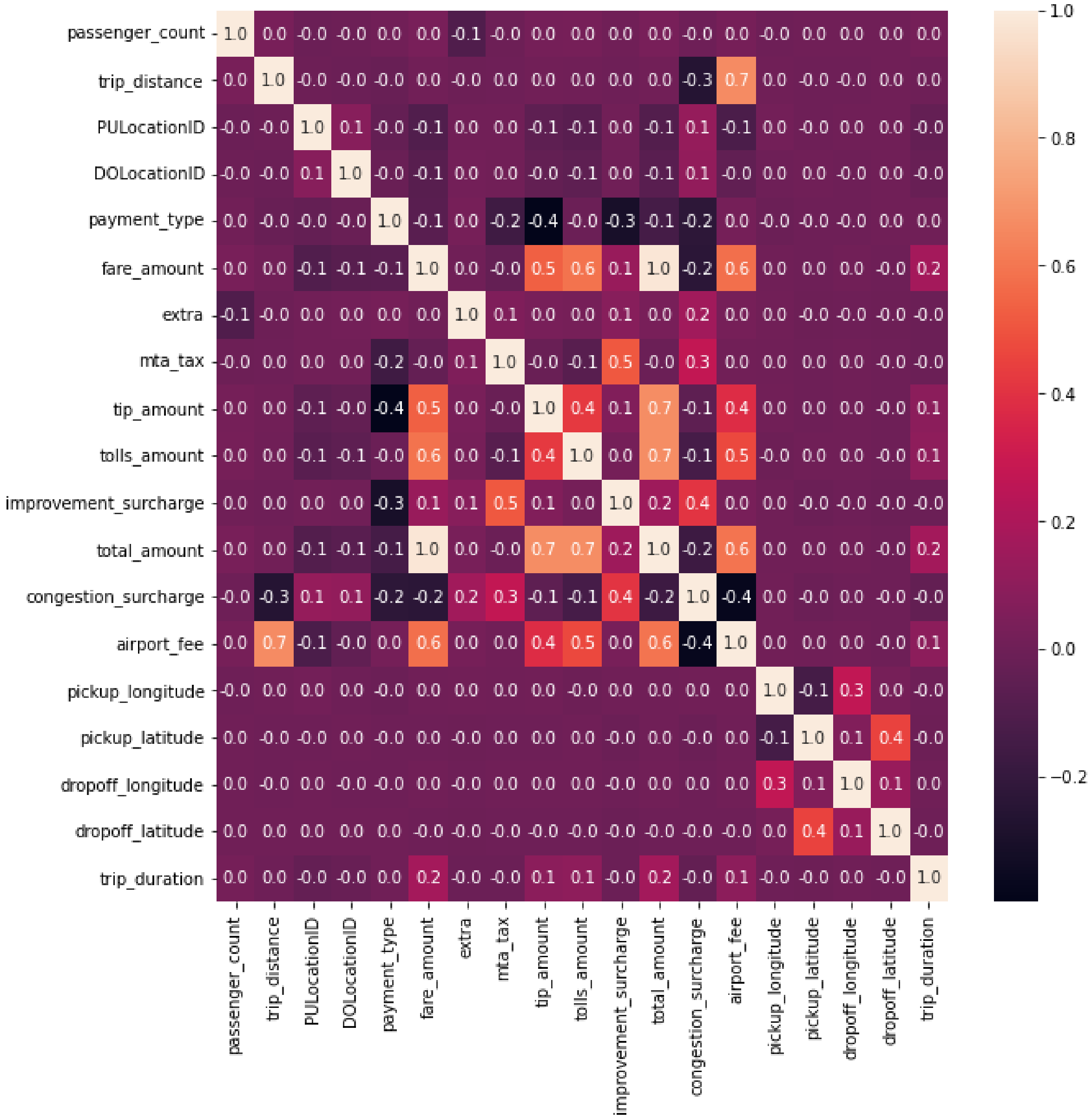
Importing Visualization Libraries

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()
```



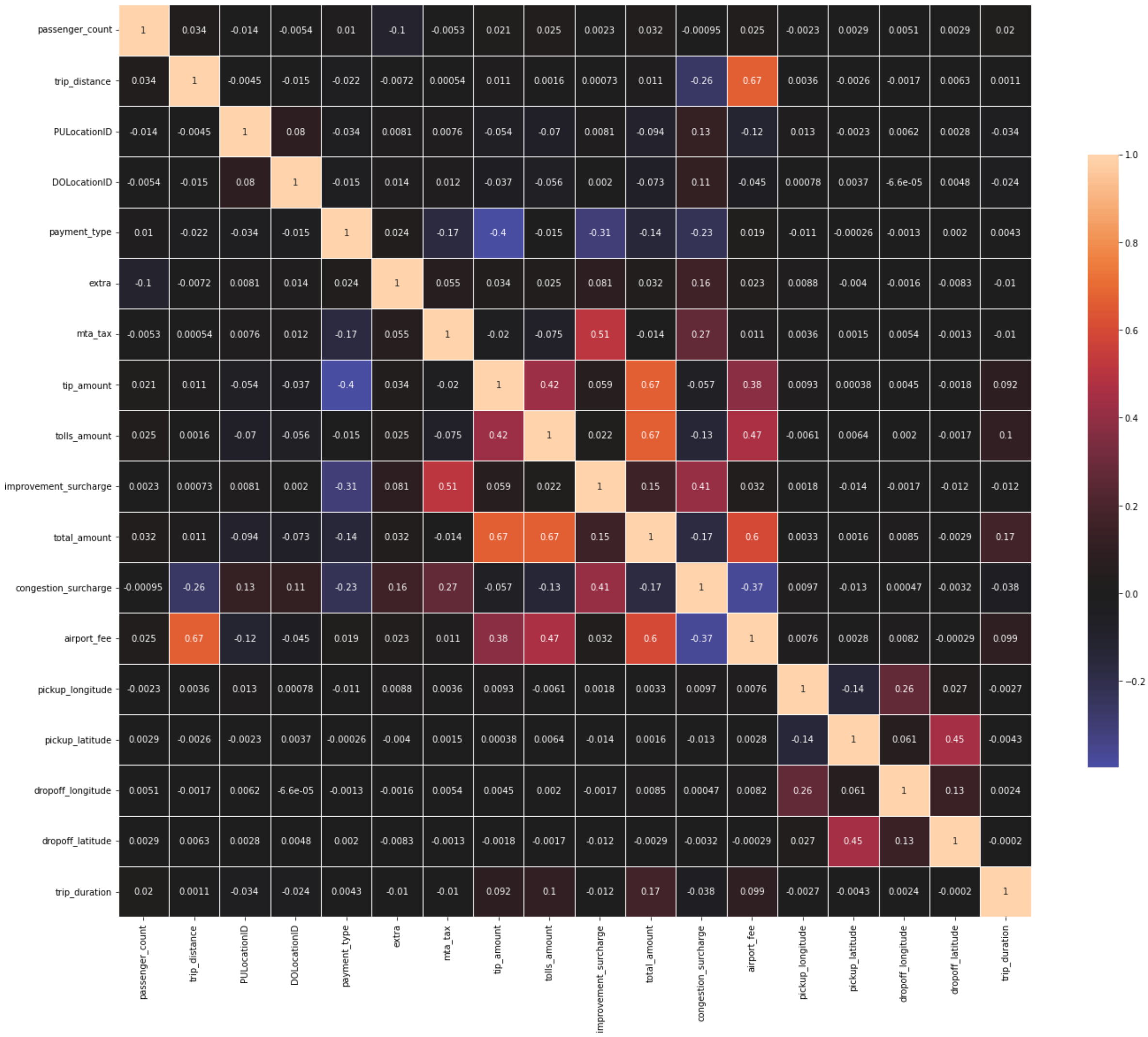
Plot heat map to observe the relationship between the variables.

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 varey significantly.

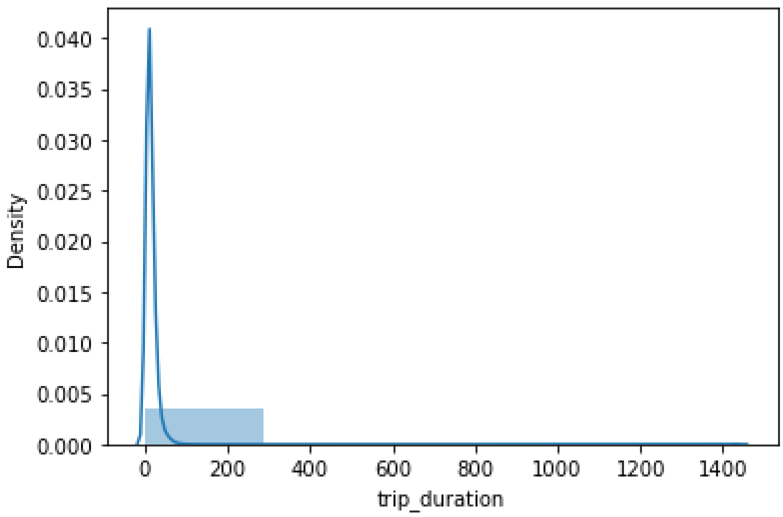
In [22]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},annot=True)
plt.tight_layout()
plt.show()
```



Heat map after removing multicollinearity

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

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

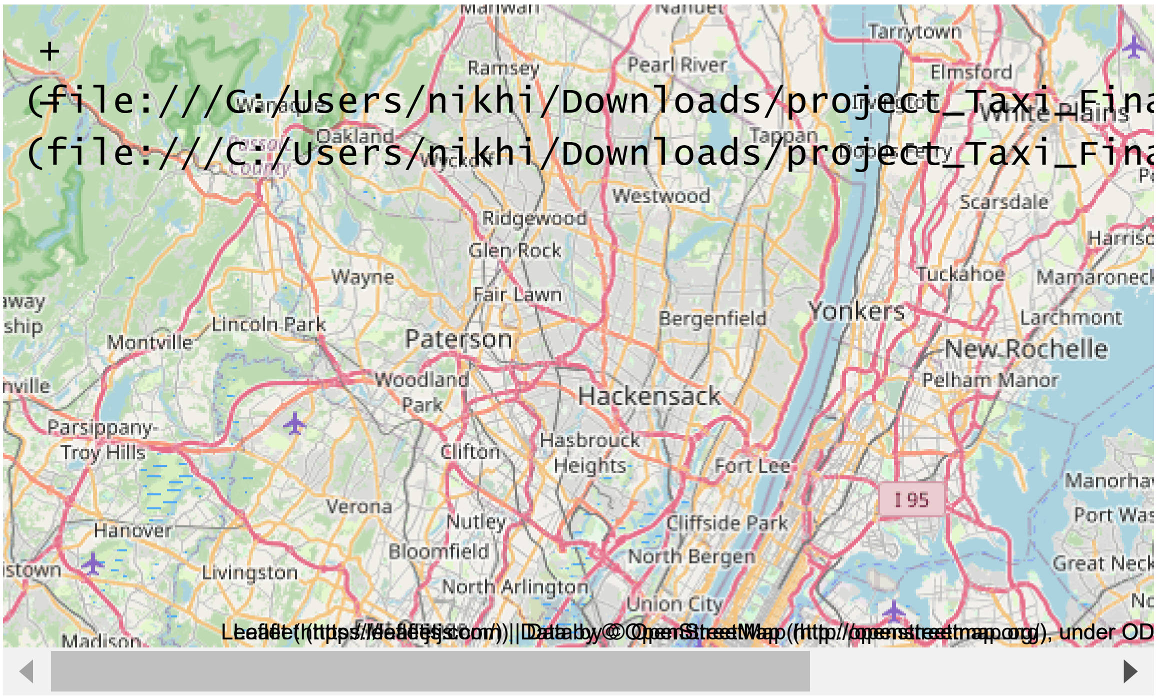


Analysis of target colunmn

Map view for Pickup Points

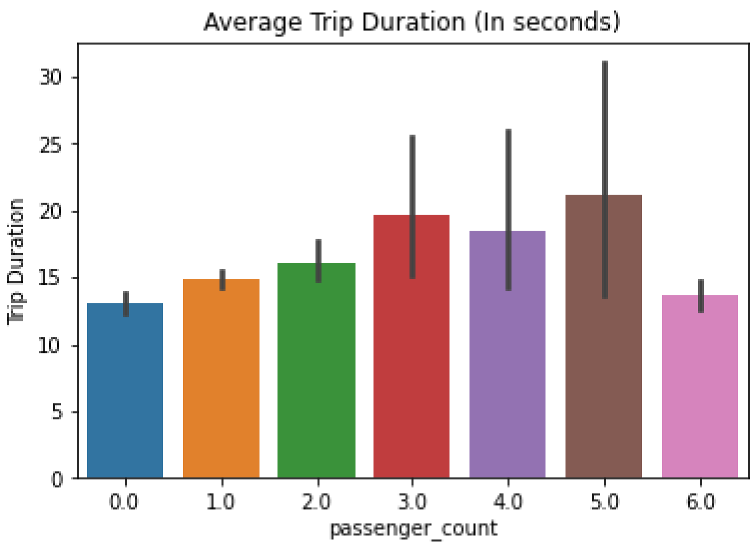
In [24]:
```python
import folium # folium is python library for geospatial analysis

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

map=folium.Map(location=location ,width=800,hight=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]:
```python
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]:
```python
pickup =df[["pickup_longitude",'pickup_latitude']]
pickup
```

```python
data = list(zip(pickup.pickup_latitude.values,
                            pickup.pickup_longitude.values,
                            ))
type(data[0][0])
```

Out[26]: numpy.float64

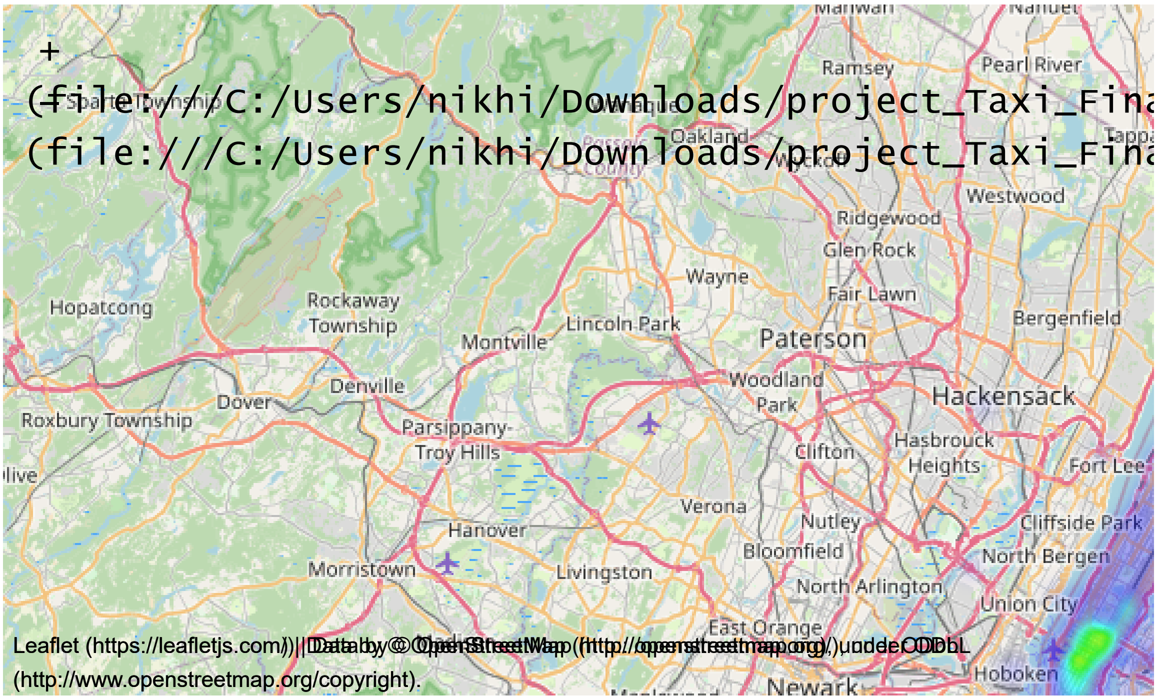
In [27]:
```python
pickup = pd.DataFrame(data)
pickup.head()
```

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]:
```python
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
```

Out[28]:



Map view for Dropoff Point

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

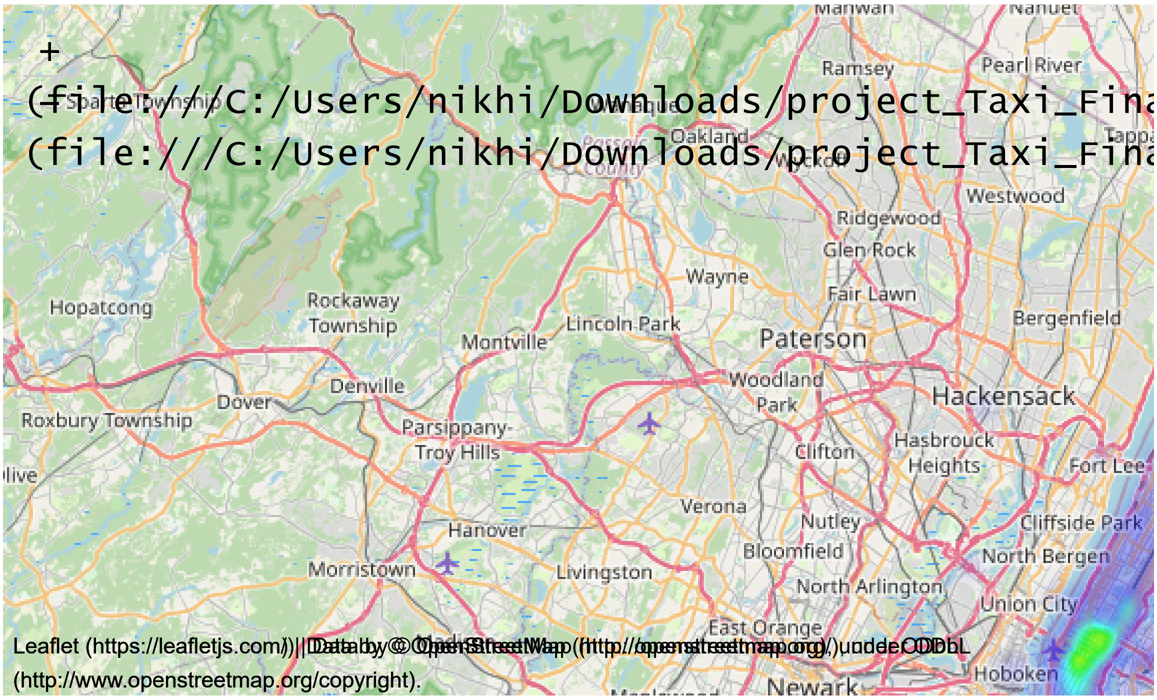
Out[29]:

	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
```

Out[30]:



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	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	

5 rows × 28 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	ti
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
```

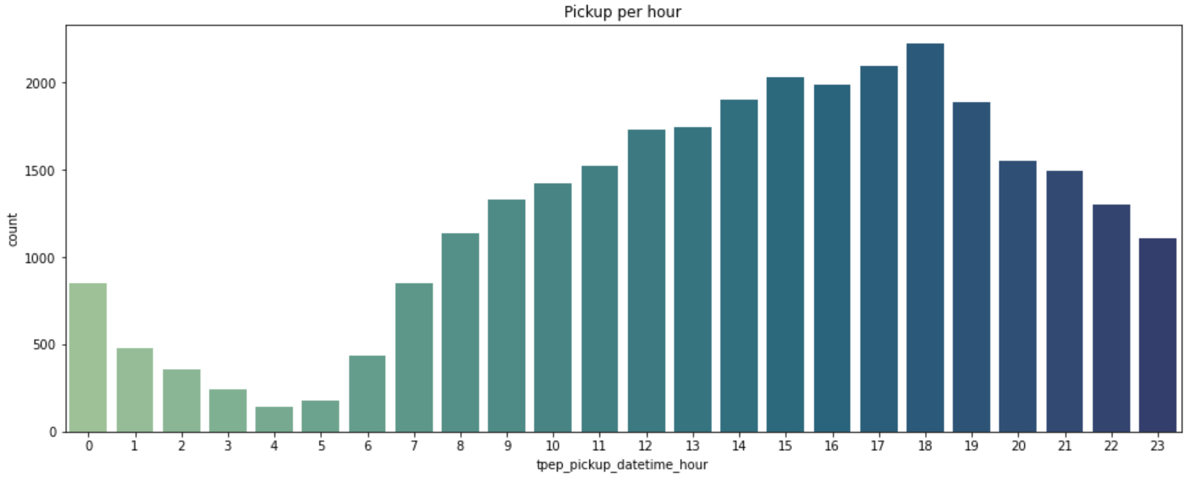
```
In [44]: df["passenger_count"].unique()   #cheking data element in the columns}
```

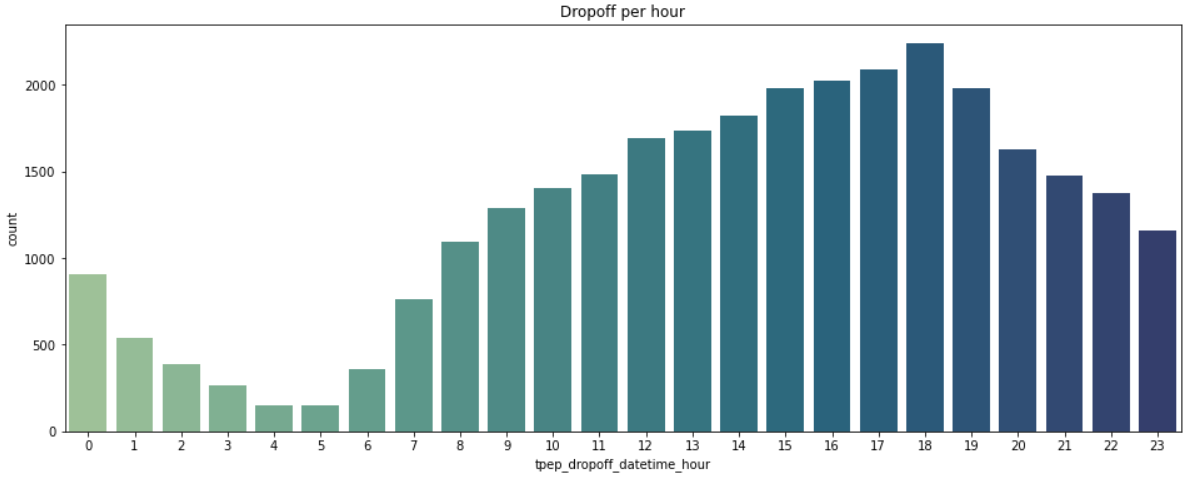
```
Out[44]: array([1., 6., 2., 0., 3., 4., 5.])
```

```
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'>
```





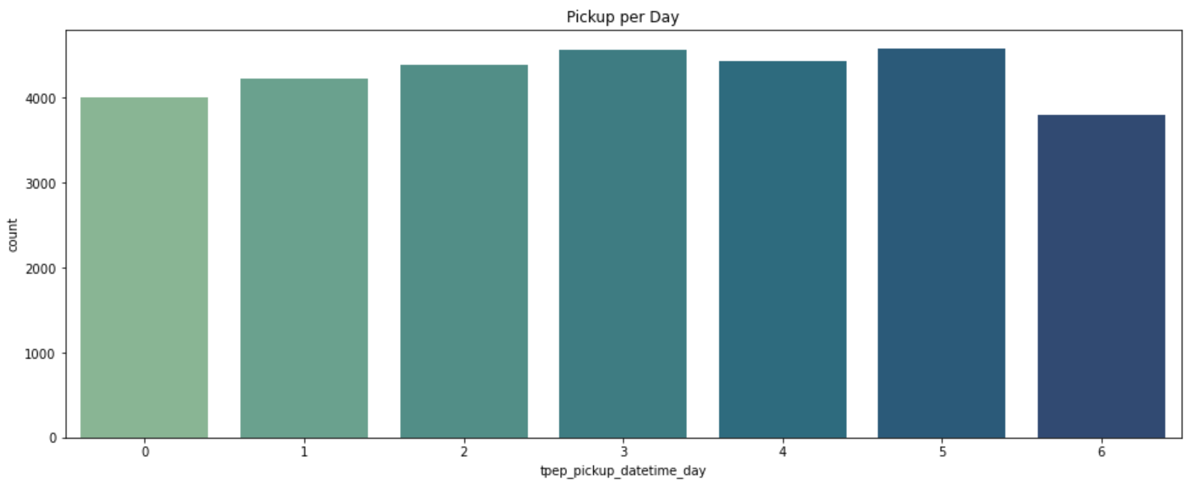
The graph shows count of passengers using the taxi at each hour of the day. It is clear that the city has high traffic from 8 am to 11 pm as most people tend to go for work and traffic peaks at evening 5pm to 6 pm.

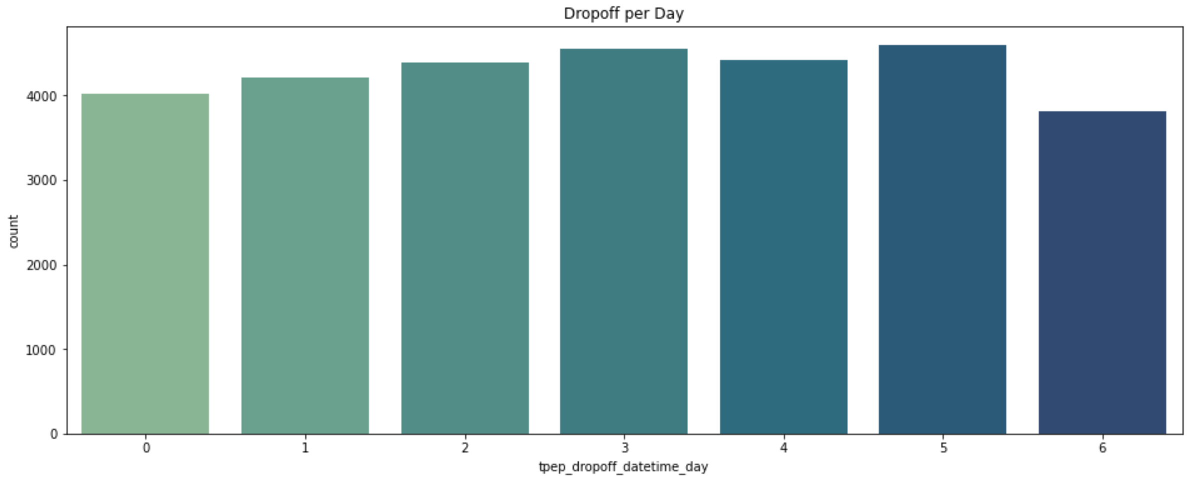
In [47]:

```
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_amou
        'improvement_surcharge','congestion_surcharge',
        'airport_fee'],axis=1)

df
```

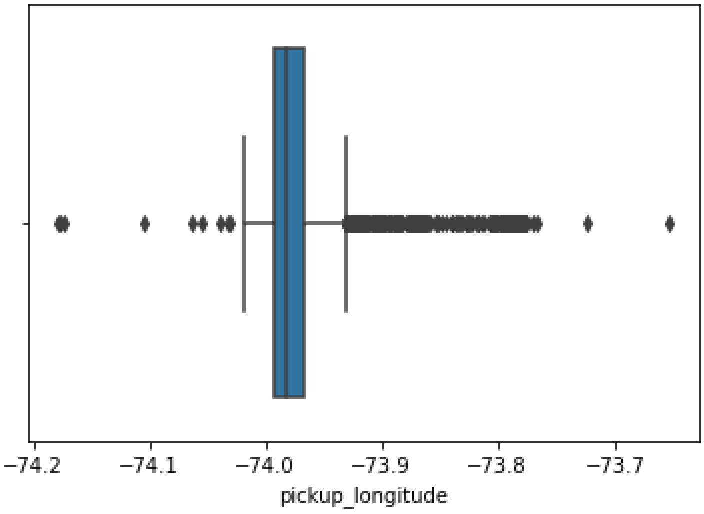
Out[48]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	t
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	
...	...	...	...	...	...	
29995	1.0	-73.984726	40.759773	-73.865250	40.770683	
29996	2.0	-73.980057	40.754837	-73.974434	40.759300	
29997	1.0	-73.976624	40.788319	-73.970436	40.785755	
29998	6.0	-74.005669	40.714779	-73.992653	40.687550	
29999	1.0	-73.963661	40.777065	-73.965836	40.754364	

30000 rows × 14 columns

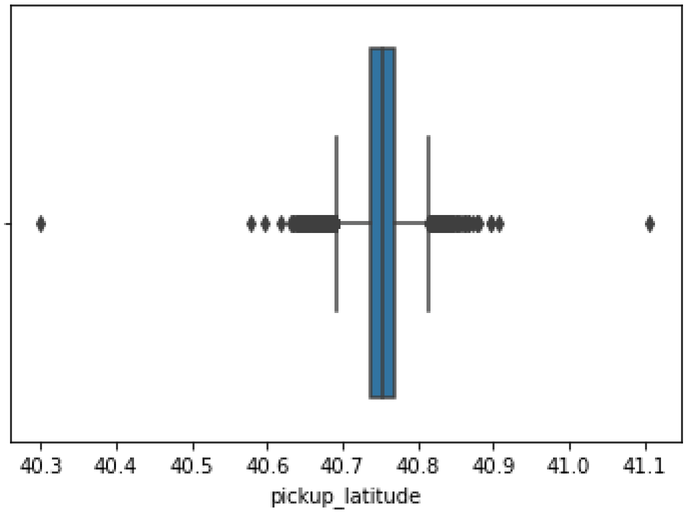
In [49]:
```
sns.boxplot(x=df["pickup_longitude"])
```

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



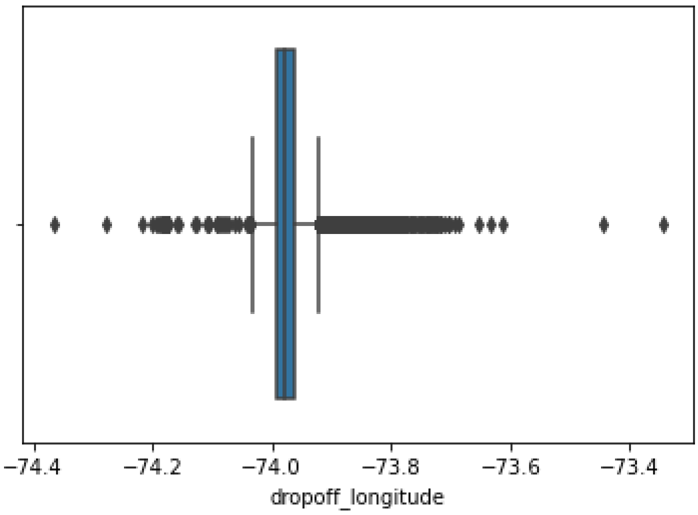
In [50]:
```
sns.boxplot(x=df["pickup_latitude"])
```

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



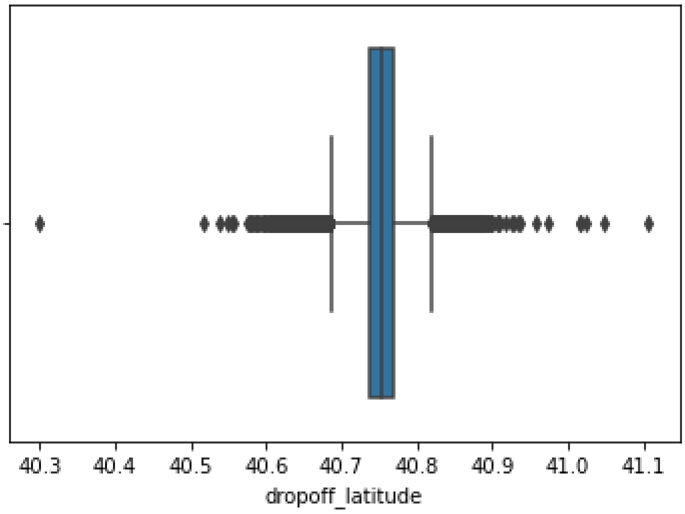
In [51]:
```
sns.boxplot(x=df["dropoff_longitude"])
```

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



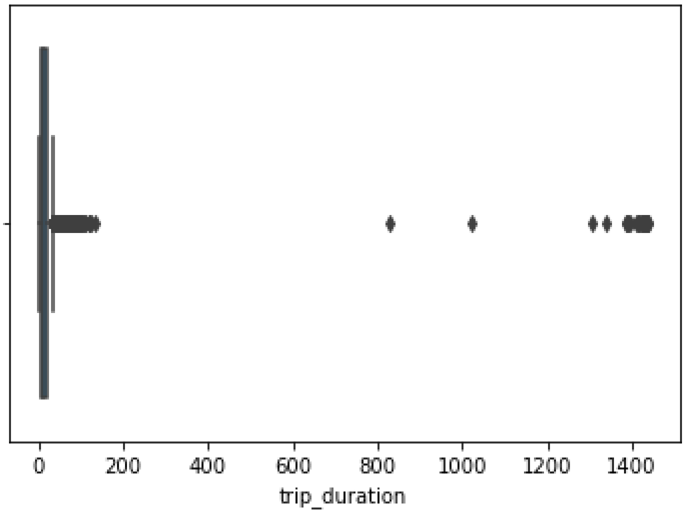
In [52]:
```
sns.boxplot(x=df["dropoff_latitude"])
```

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

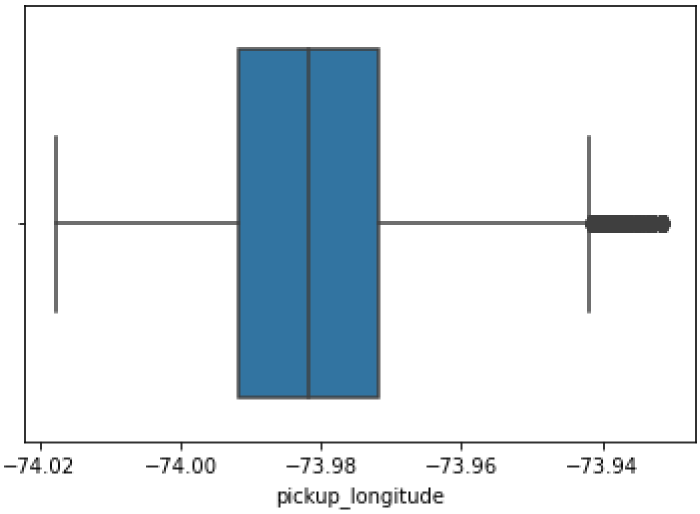
30000 rows × 1 columns

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

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

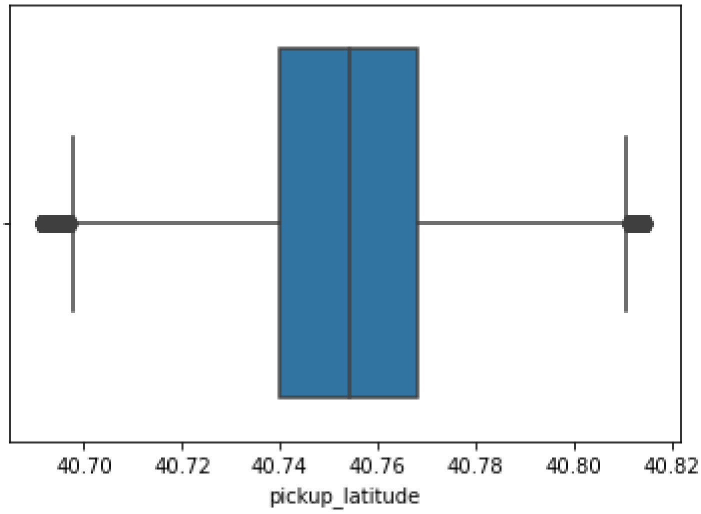
In [60]:
```
sns.boxplot(x=df["pickup_longitude"])
```

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



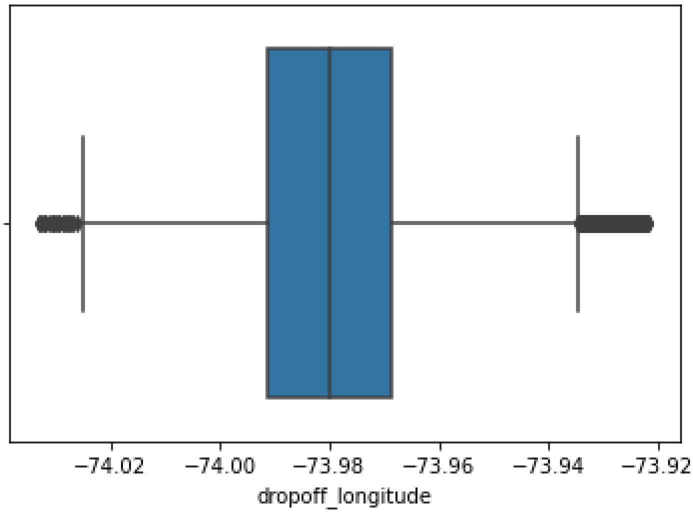
In [61]:
```
sns.boxplot(x=df["pickup_latitude"])
```

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



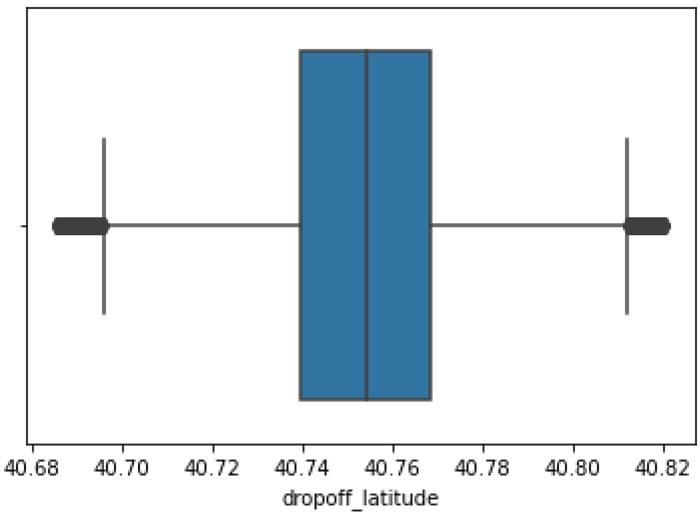
In [62]:
```
sns.boxplot(x=df["dropoff_longitude"])
```

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



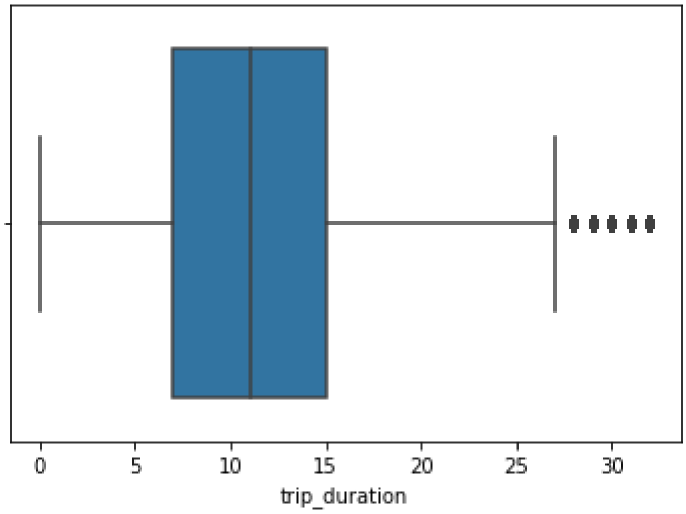
In [63]:
```
sns.boxplot(x=df["dropoff_latitude"])
```

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_long
              'dropoff_latitude','tpep_pickup_datetime_day','tpep_pickup_datetime_r
              'tpep_dropoff_datetime_day',
              'tpep_dropoff_datetime_month','tpep_dropoff_datetime_hour','tpep_drop
```

In [76]:

```
Xnew
```

Out[76]:

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

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')
```

Spliting 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,accura
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
```

In [84]:
```
from sklearn.linear_model import Ridge,Lasso
rr = Ridge()
```

```
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,accura
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,accura
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 4.38439393939394
5 4.343777777777775
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_square
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
```

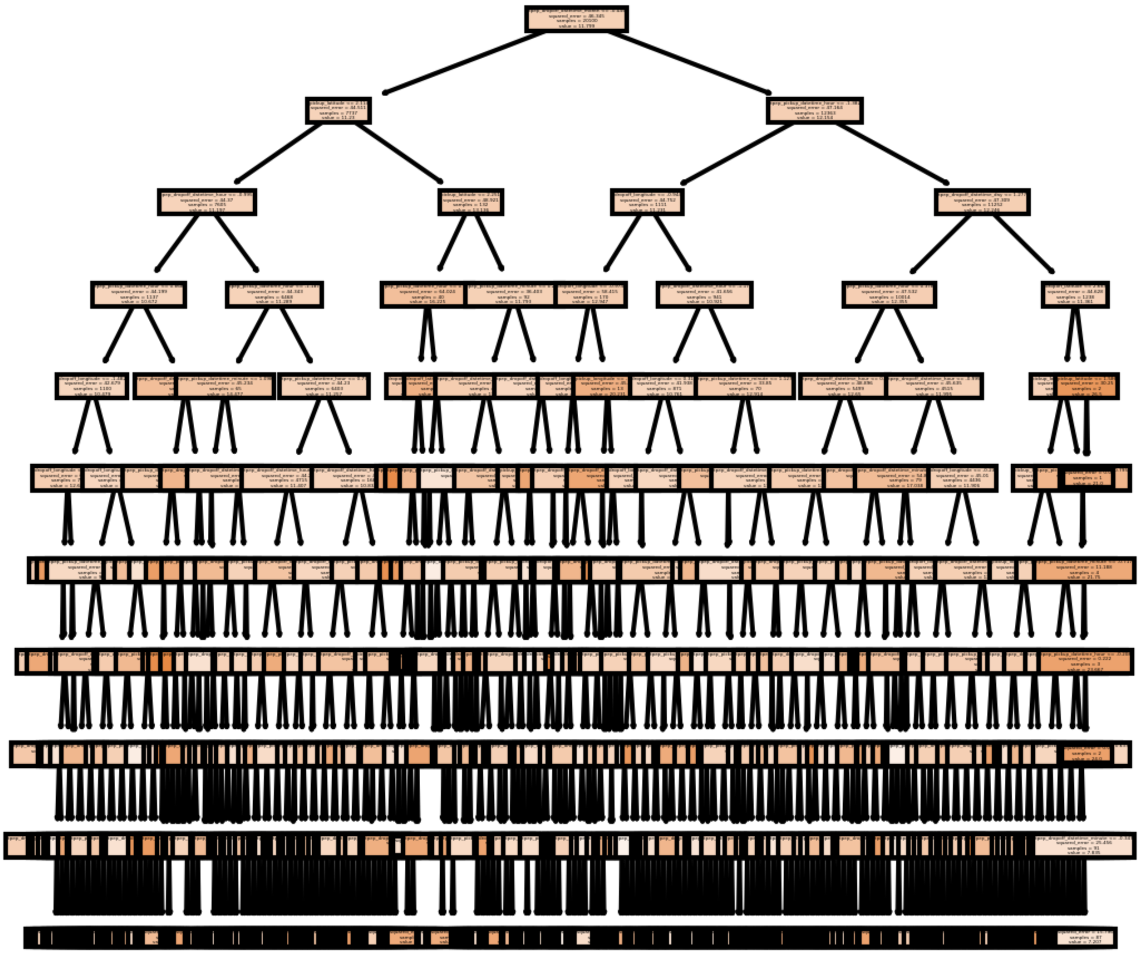
plt.plot(xtrain,pred_tr,color='r') plt.plot(X,Y,'b.') plt.xlabel("xtrain") plt.ylabel("pred_tr") plt.show()

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

```
Requirement already satisfied: graphviz in c:\users\nikhi\anaconda3\lib\site-packa
ges (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_square
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
```

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,learnin
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_squar
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 ro 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-packag
es (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_squar
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_squar
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 [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```