Problem Statement

At some point or the other almost each one of us has used an Ola or Uber for taking a ride.

Ride hailing services are services that use online-enabled platforms to connect between passengers and local drivers using their personal vehicles. In most cases they are a comfortable method for door-to-door transport. Usually they are cheaper than using licensed taxicabs. Examples of ride hailing services include Uber and Lyft.

To improve the efficiency of taxi dispatching systems for such services, it is important to be able to predict how long a driver will have his taxi occupied. If a dispatcher knew approximately when a taxi driver would be ending their current ride, they would be better able to identify which driver to assign to each pickup request.

In this competition, we are challenged to build a model that predicts the total ride duration of taxi trips in New York City.

1. Exploratory Data Analysis

Let's check the data files! According to the data description we should find the following columns:

- **id** - a unique identifier for each trip
- **vendor_id** - a code indicating the provider associated with the trip record
- **pickup_datetime** - date and time when the meter was engaged
- **dropoff_datetime** - date and time when the meter was disengaged
- **passenger_count** - the number of passengers in the vehicle (driver entered value)
- **pickup_longitude** - the longitude where the meter was engaged
- **pickup_latitude** - the latitude where the meter was engaged
- **dropoff_longitude** - the longitude where the meter was disengaged
- **dropoff_latitude** - the latitude where the meter was disengaged
- **store_and_fwd_flag** - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server (Y=store and forward; N=not a store and forward trip)
- **trip_duration** - (target) duration of the trip in seconds

Here, we have 2 variables dropoff_datetime and store_and_fwd_flag which are not available before the trip starts and hence will not be used as features to the model.

1.1 Load Libraries

In [286]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt
from datetime import timedelta
import sklearn
import warnings
warnings.filterwarnings('ignore')
```

Load Data

In [328]:

```
dfMain=pd.read_csv("D:/Internshala Course/EDA & ML module/EDA-ML-Final-Project/EDA+ML-Final Project/nyc_taxi

#df.head()
#copy_column=df[['pickup_latitude','pickup_longitude','dropoff_longitude','dropoff_latitude']].copy()
#copy_column.head()
#copy_column.to_csv("D:\Internshala Course\EDA & ML module", index= False)
```

In [329]:

```
df=dfMain.copy()
```

In [ ]:

```

```

In [330]:

```
df.dtypes
```

Out[330]:

```
id                   object
vendor_id             int64
pickup_datetime      object
dropoff_datetime     object
passenger_count       int64
pickup_longitude    float64
pickup_latitude     float64
dropoff_longitude   float64
dropoff_latitude    float64
store_and_fwd_flag   object
trip_duration         int64
dtype: object
```

In [331]:

```
df.isnull().sum()
```

Out[331]:

```
id                  0
vendor_id           0
pickup_datetime     0
dropoff_datetime    0
passenger_count     0
pickup_longitude    0
pickup_latitude     0
dropoff_longitude   0
dropoff_latitude    0
store_and_fwd_flag  0
trip_duration       0
dtype: int64
```

In [332]:

```
df.describe()
```

Out[332]:

	vendor_id	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_d
count	729322.000000	729322.000000	729322.000000	729322.000000	729322.000000	729322.000000	7.2932
mean	1.535403	1.662055	-73.973513	40.750919	-73.973422	40.751775	9.5222
std	0.498745	1.312446	0.069754	0.033594	0.069588	0.036037	3.8646
min	1.000000	0.000000	-121.933342	34.712234	-121.933304	32.181141	1.0000
25%	1.000000	1.000000	-73.991859	40.737335	-73.991318	40.735931	3.9700
50%	2.000000	1.000000	-73.981758	40.754070	-73.979759	40.754509	6.6300
75%	2.000000	2.000000	-73.967361	40.768314	-73.963036	40.769741	1.0750
max	2.000000	9.000000	-65.897385	51.881084	-65.897385	43.921028	1.9397

File structure and content

In [333]:

```
# Convert the pickup and dropoff datetime columns to datetime format
df["pickup_datetime"]=pd.to_datetime(df["pickup_datetime"])
df["dropoff_datetime"]=pd.to_datetime(df["dropoff_datetime"])
```

Reformatting features & Checking consistency

In [334]:

```
df['store_and_fwd_flag'] = df['store_and_fwd_flag'].str.strip().map({'N': 0, 'Y': 1})
#df.head()
```

In [335]:

```
#print(df['pickup_datetime'].dtype)
#print(df['dropoff_datetime'].dtype)
```

In [336]:

```
# Create new columns for pickup date, pickup hour, pickup weekday, and trip duration in seconds
df['pickup_date']=df['pickup_datetime'].dt.date
df['pickup_day']=df['pickup_datetime'].dt.day
df['pickup_hour']=df['pickup_datetime'].dt.hour
df['pickup_weekday']=df['pickup_datetime'].dt.weekday
df['trip_duration']=df['trip_duration']/3600
df['dropoff_weekday']=df['dropoff_datetime'].dt.weekday
df['dropoff_day']=df['dropoff_datetime'].dt.day
#df['pickup_minute']=df['pickup_datetime'].dt.minute
#df['pickup_seconds']=df['pickup_datetime'].dt.second
df['total_time']=(df['dropoff_datetime']-df['pickup_datetime']).dt.total_seconds()
df['dropoff_hour']=df['dropoff_datetime'].dt.hour
df.shape
```

Out[336]:

```
(729322, 19)
```

In [337]:

```
def time_of_day(x):
    # to calculate what time of it is now
    if x in range(6,12):
        return 'Morning'
    elif x in range(12,16):
        return 'Afternoon'
    elif x in range(16,22):
        return 'Evening'
    else:
        return 'Late night'

df['pickup_time_of_day'] = df['pickup_hour'].apply(time_of_day)
df['dropoff_time_of_day'] = df['dropoff_hour'].apply(time_of_day)
```

In [338]:

```
df.head()
```

Out[338]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	

5 rows × 21 columns

In [339]:

```
# Define the dictionary mapping weekday numbers to weekday names
weekday_map1 = {0: 'pickup_Sunday', 1: 'pickup_Monday', 2: 'pickup_Tuesday', 3: 'pickup_Wednesday', 4: 'pic
# Use the dt.weekday method to get the weekday number and map it to the corresponding weekday name
df['pickup_days'] = df['pickup_datetime'].dt.weekday.map(weekday_map1)
```

In [340]:

```
df.head()
```

Out[340]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	

5 rows × 22 columns

In [341]:

```
# Define the dictionary mapping weekday numbers to weekday names
weekday_map2 = {0: 'dropoff_Sunday', 1: 'dropoff_Monday', 2: 'dropoff_Tuesday', 3: 'dropoff_Wednesday', 4:

# Use the dt.weekday method to get the weekday number and map it to the corresponding weekday name
df['dropoff_day'] = df['pickup_datetime'].dt.weekday.map(weekday_map2)
```

In [342]:

```
df.head()
```

Out[342]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	

5 rows × 22 columns

In [343]:

```
check_trip_duration = (df['dropoff_datetime']-df['pickup_datetime']).dt.total_seconds()
df['duration_difference']= np.abs(check_trip_duration-df['trip_duration'])
duration_difference = df.query('duration_difference > 1')
```

In [344]:

```
duration_difference.shape
```

Out[344]:

```
(729309, 23)
```

Target Exploration

In [345]:

```
#df.head()
df['trip_duration'].describe()
```

Out[345]:

```
count    729322.000000
mean          0.264508
std           1.073507
min           0.000278
25%           0.110278
50%           0.184167
75%           0.298611
max         538.815556
Name: trip_duration, dtype: float64
```

In [346]:

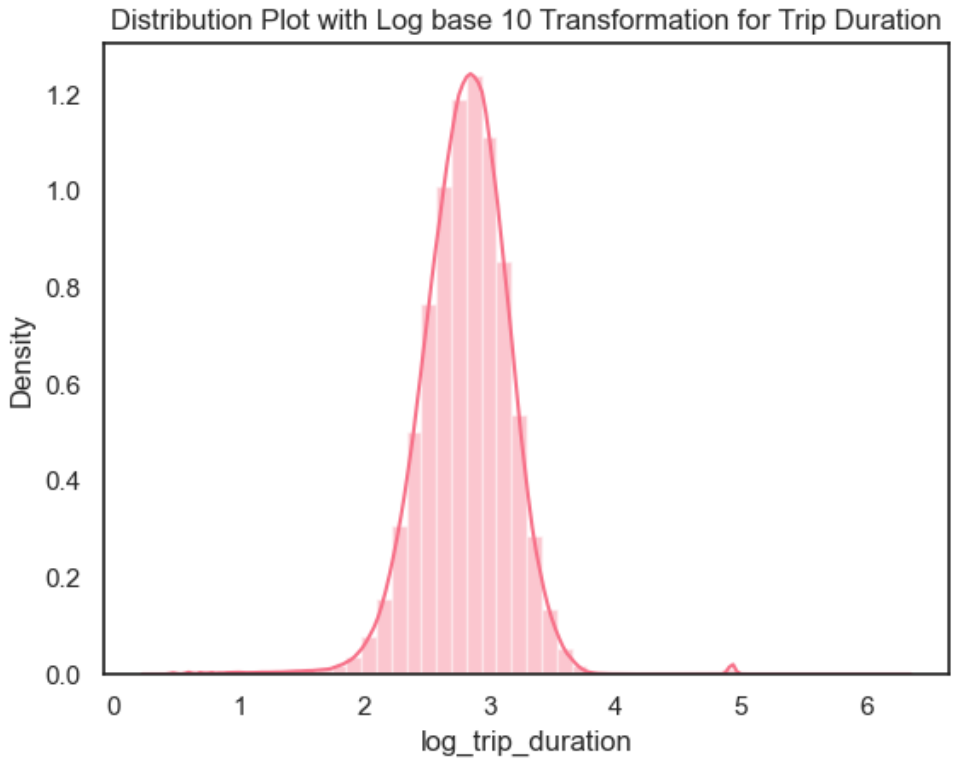
```
df['log_trip_duration']=np.log10((df['trip_duration']*3600).values+1)
```

In [347]:

```
#df.describe()
```

In [348]:

```
sns.distplot(df['log_trip_duration']).set(title='Distribution Plot with Log base 10 Transformation for Trip
#plt.axvline(x=2.7, color="b")
plt.show()
```



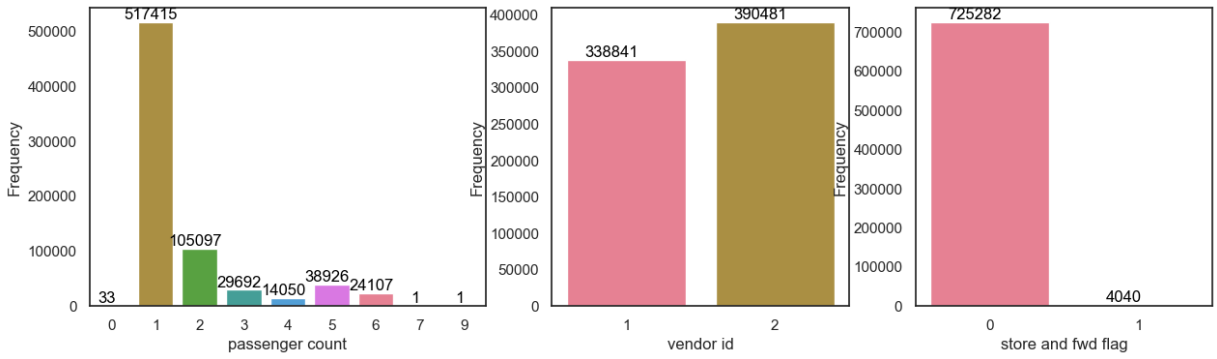
In [ ]:

```
```

Univariate Visualization

In [349]:

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 4),
                         gridspec_kw={'width_ratios': [4, 3, 3]})
#plt.figure(figsize=(33,10))
#passenger count plot
plt.subplot(131)
ax=sns.countplot(x='passenger_count',data=df, palette=sns.color_palette("husl"))
plt.xlabel('passenger count')
plt.ylabel('Frequency')
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black',fontsize=12)

#vendor_id plot
plt.subplot(132)
ax=sns.countplot(x='vendor_id', data=df, palette=sns.color_palette("husl"))
plt.xlabel("vendor id")
plt.ylabel("Frequency")
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black',fontsize=12)


#store_and_fwd_flag plot
plt.subplot(133)
ax=sns.countplot(x='store_and_fwd_flag', data=df, palette=sns.color_palette('husl'))
plt.xlabel("store and fwd flag")
plt.ylabel("Frequency")
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black',fontsize=12)
```



In [ ]:

```
```

Observations:

1. Most of the trips involve only 1 passenger. There are trips with 7-9 passengers but they are very low in number.
2. Vendor 2 has more number of trips as compared to vendor 1
3. The store_and_fwd_flag values, indicating whether the trip data was sent immediately to the vendor (“0”) or held in the memory of the taxi because there was no connection to the server (“1”), show that there was almost no storing taking place

In [350]:

```
df['pickup_datetime'].max(), df['pickup_datetime'].min()
```

Out[350]:

```
(Timestamp('2016-06-30 23:59:37'), Timestamp('2016-01-01 00:01:14'))
```

In [351]:

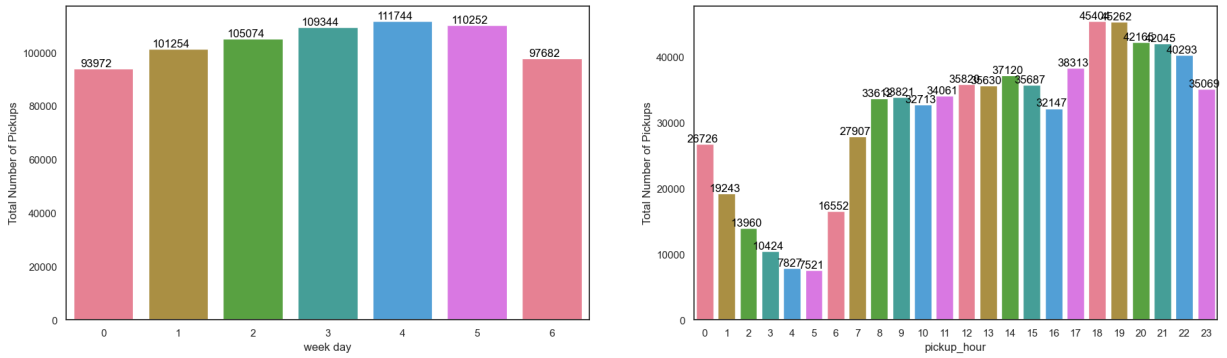
```
df['dropoff_datetime'].max(), df['dropoff_datetime'].min()
```

Out[351]:

```
(Timestamp('2016-07-01 23:02:03'), Timestamp('2016-01-01 00:05:54'))
```

In [352]:

```
#datetime plot
fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(22,6),gridspec_kw={'width_ratios':[1,1]})

#weekday pickup count plot
ax=plt.subplot(121)
sns.countplot(x='pickup_weekday',data=df, palette=sns.color_palette('husl'))
plt.xlabel('week day')
plt.ylabel('Total Number of Pickups')
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black')

#hour wise pickup count plot
ax=plt.subplot(122)
sns.countplot(x='pickup_hour',data=df, palette=sns.color_palette('husl'))
plt.xlabel('pickup_hour')
plt.ylabel('Total Number of Pickups')
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black')
```



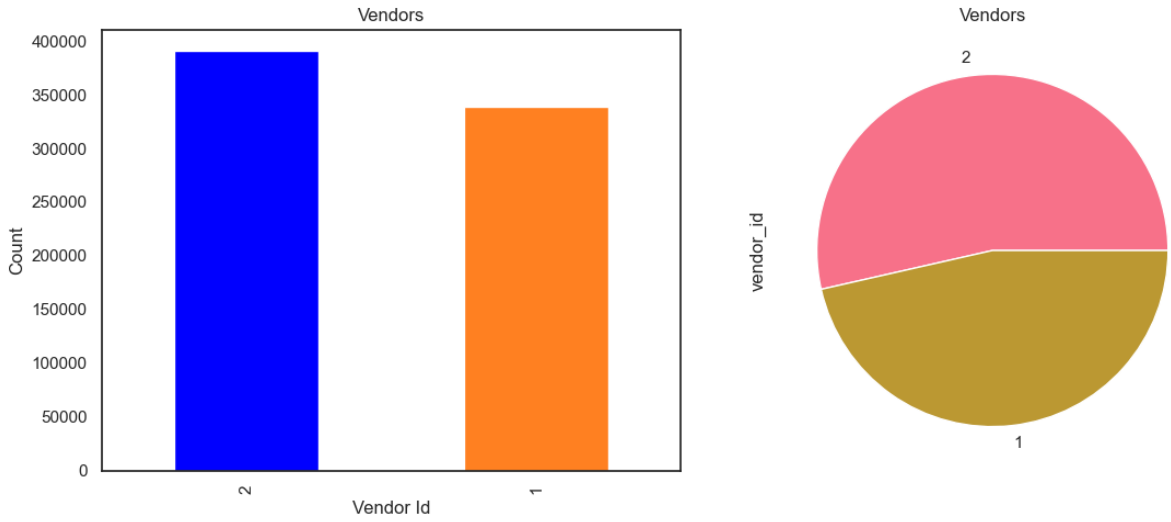
- Number of pickups for weekends is much lower than week days with a peak on Thursday (4). Note that here weekday is a decimal number, where 0 is Sunday and 6 is Saturday.
- Number of pickups as expected is highest in late evenings. However, it is much lower during the morning peak hours.

In [ ]:

```

```

In [353]:

```
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(12,5))
ax = df['vendor_id'].value_counts().plot(kind='bar',title="Vendors",ax=axes[0],color = ('blue',(1, 0.5, 0.1
df['vendor_id'].value_counts().plot(kind='pie',title="Vendors",ax=axes[1])
ax.set_ylabel("Count")
ax.set_xlabel("Vendor Id")
fig.tight_layout()
```

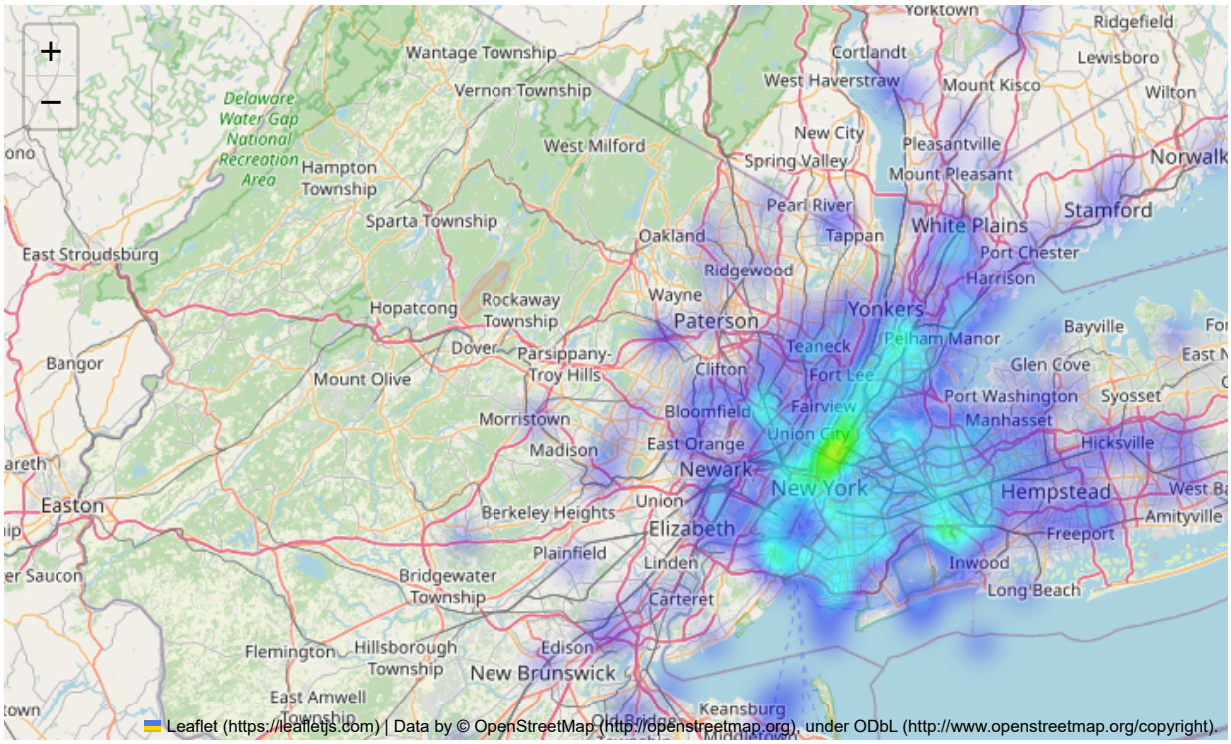


In [ ]:

In [354]:

```
import folium
from folium.plugins import HeatMap

#Create map centered on New York City
nyc_coords = (40.712776, -74.005974)
m = folium.Map(location=nyc_coords, zoom_start=11,width="100%", height="100%")

#Create HeatMap using pickup latitude and longitude
heat_data = df[['pickup_latitude', 'pickup_longitude']].values.tolist()
HeatMap(heat_data, radius=8, max_zoom=13).add_to(m)

#Display map
m
```

Out[354]:

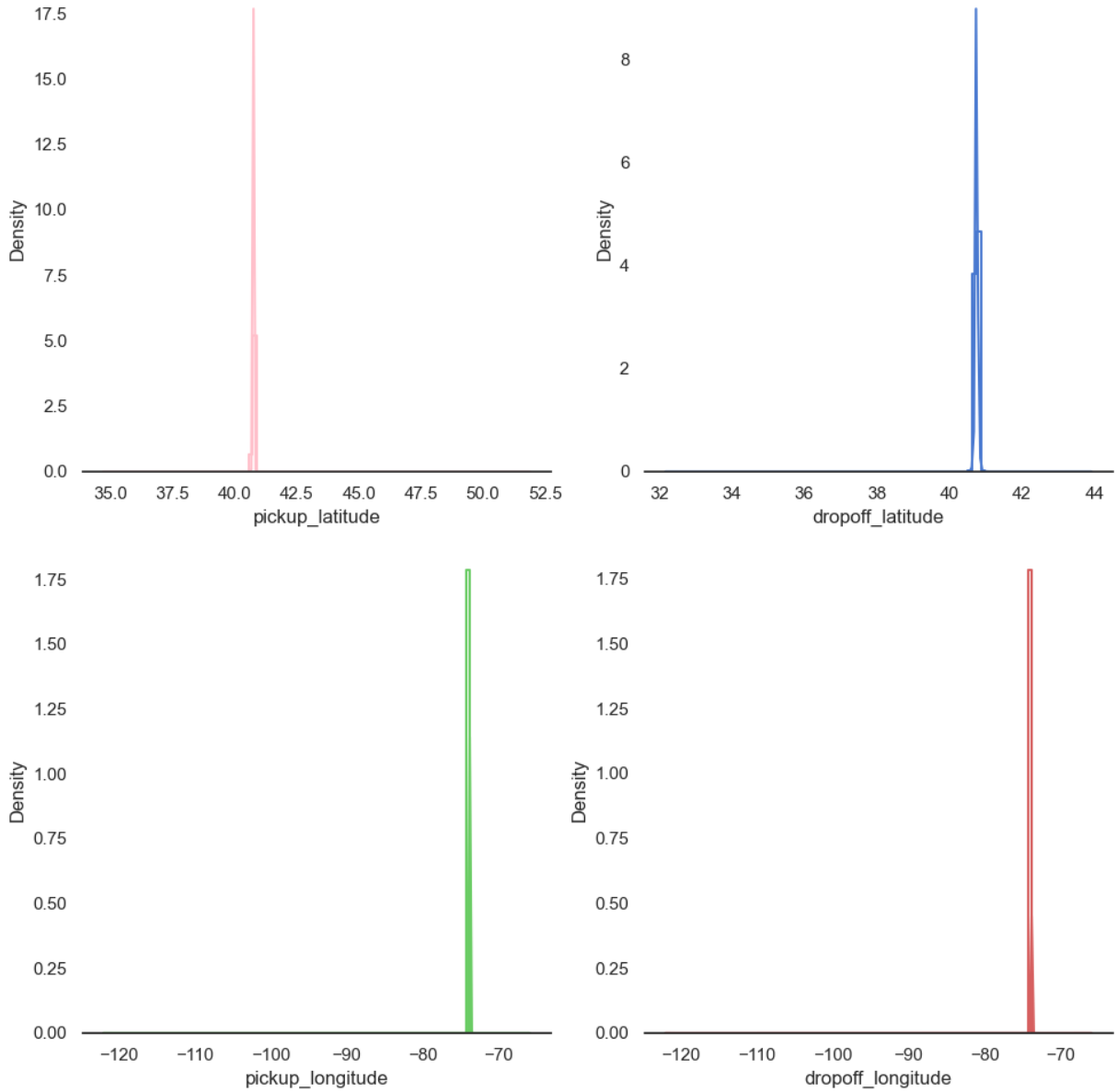


Lattitude & Longitude

In [ ]:

In [355]:

```
with sns.plotting_context("notebook"):
            sns.set(style="white", palette=sns.color_palette("husl"), color_codes=True)
            f, axes = plt.subplots(2,2,figsize=(10, 10), sharex=False, sharey = False)
            sns.despine(left=True)
#creating 4 different plots on four different variable selected
            sns.histplot(df, x="pickup_latitude",  element="step", kde=True, stat="density",color="pink", c
            sns.histplot(df, x="pickup_longitude",  element="step", kde=True, stat="density", color="g",com
            sns.histplot(df, x="dropoff_latitude",  element="step", kde=True, stat="density", color="b",com
            sns.histplot(df, x="dropoff_longitude", element="step", kde=True, stat="density", color="r",com

            #plt.setp(axes, yticks=[])
            plt.tight_layout()
            plt.show()
```



In [ ]:

In [ ]:

In [ ]:

```
```

In [ ]:

```
```

In [356]:

```
df.head()
```

Out[356]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	

5 rows × 24 columns

Bivariate Relations with Target

Now that we have gone through all the basic features one by one. Let us start looking at their relation with the target. This will help us in selecting and extracting features at the modelling stage.

Trip Duration vs Weekday

In [357]:

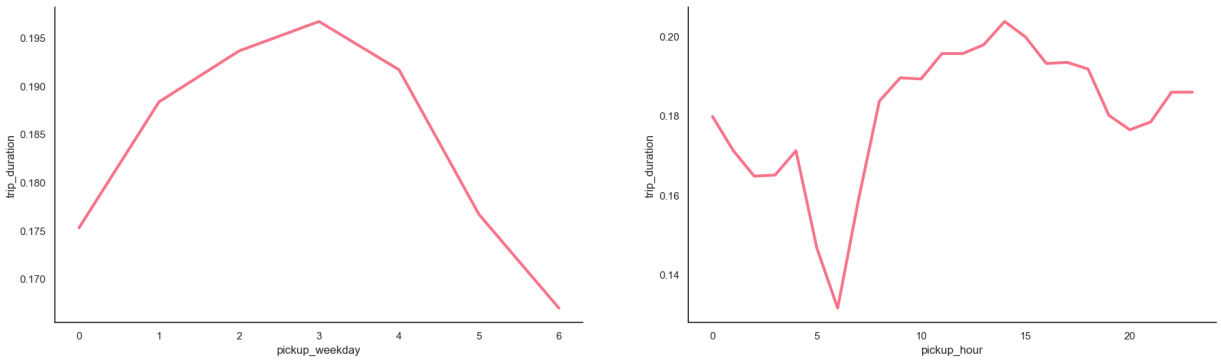
```
df.columns
```

Out[357]:

```
Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration', 'pickup_date', 'pickup_day', 'pickup_hour',
       'pickup_weekday', 'dropoff_weekday', 'dropoff_day', 'total_time',
       'dropoff_hour', 'pickup_time_of_day', 'dropoff_time_of_day',
       'pickup_days', 'duration_difference', 'log_trip_duration'],
      dtype='object')
```

In [358]:

```
# Setting the size of the plot

fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(22,6),gridspec_kw={'width_ratios':[1,1]})
#selecting the spot between the two places to draw the plot
plt.subplot(121)
#Using median, we are creating the plot for pickup days with trip duration
summary_wdays_avg_duration = pd.DataFrame(df.groupby(['pickup_weekday'])['trip_duration'].median())
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.lineplot(data=summary_wdays_avg_duration, x="pickup_weekday",units="unit", y="trip_duration")
sns.despine(bottom = False)

plt.subplot(122)
#Using median, we are creating the plot for pickup hours with trip duration
summary_hourly_avg_duration = pd.DataFrame(df.groupby(['pickup_hour'])['trip_duration'].median())
summary_hourly_avg_duration.reset_index(inplace = True)
summary_hourly_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.lineplot(data=summary_hourly_avg_duration, x="pickup_hour", units = "unit", y="trip_duration")
sns.despine(bottom = False)
```

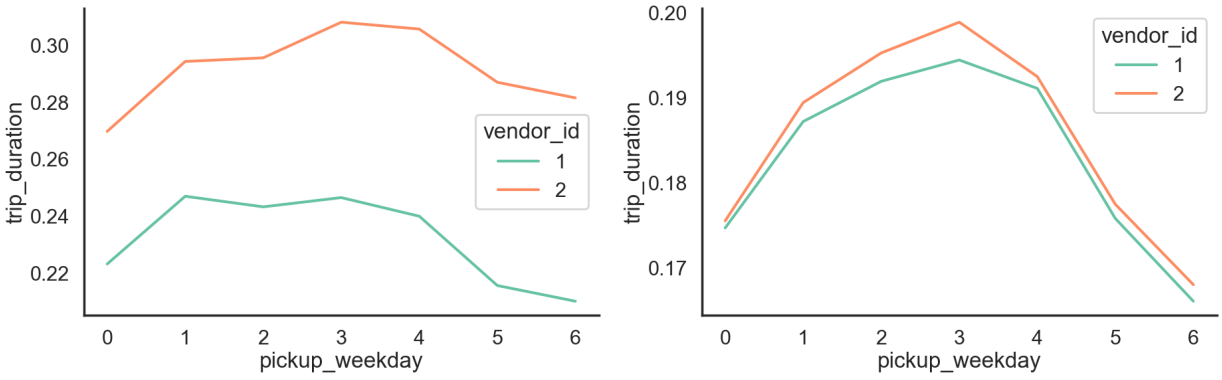


In [ ]:

Mean Trip Duration Vendor Wise

In [359]:

```
#setting the size of the plot and the ratio of two plots

fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(22,6),gridspec_kw={'width_ratios':[1,1]})

plt.subplot(121)

# Generating plot with two variable on x-axis using mean

summary_wdays_avg_duration = pd.DataFrame(df.groupby(['vendor_id','pickup_weekday'])['trip_duration'].mean(
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1

# Setting style for the plot

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.lineplot(data=summary_wdays_avg_duration, x="pickup_weekday",hue="vendor_id",palette="Set2", y="trip_du
sns.despine(bottom = False)

# Generating the same plot as above, the difference only is this plot will be in reference to median

plt.subplot(122)
summary_wdays_avg_duration = pd.DataFrame(df.groupby(['vendor_id','pickup_weekday'])['trip_duration'].media
summary_wdays_avg_duration.reset_index(inplace = True)
summary_wdays_avg_duration['unit']=1

sns.set(style="white", palette="muted", color_codes=True)
sns.set_context("poster")
sns.lineplot(data=summary_wdays_avg_duration, x='pickup_weekday', units = "unit",palette='Set2', hue="vendo
sns.despine(bottom =False)
```



Trip Duration vs Passenger Count

In [360]:

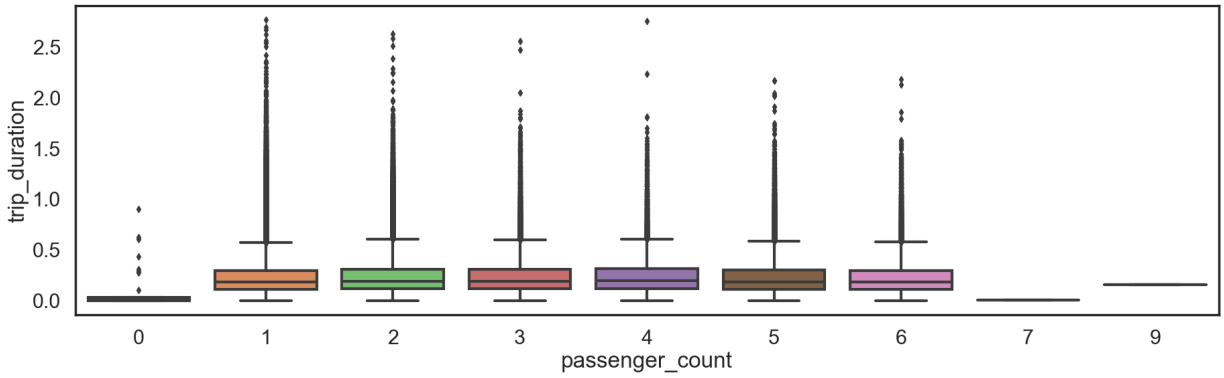
```
# couting the trips with respect to number of passenger in each trip to find outlier
df.passenger_count.value_counts()
```

Out[360]:

```
1    517415
2    105097
5     38926
3     29692
6     24107
4     14050
0        33
7         1
9         1
Name: passenger_count, dtype: int64
```

In [361]:

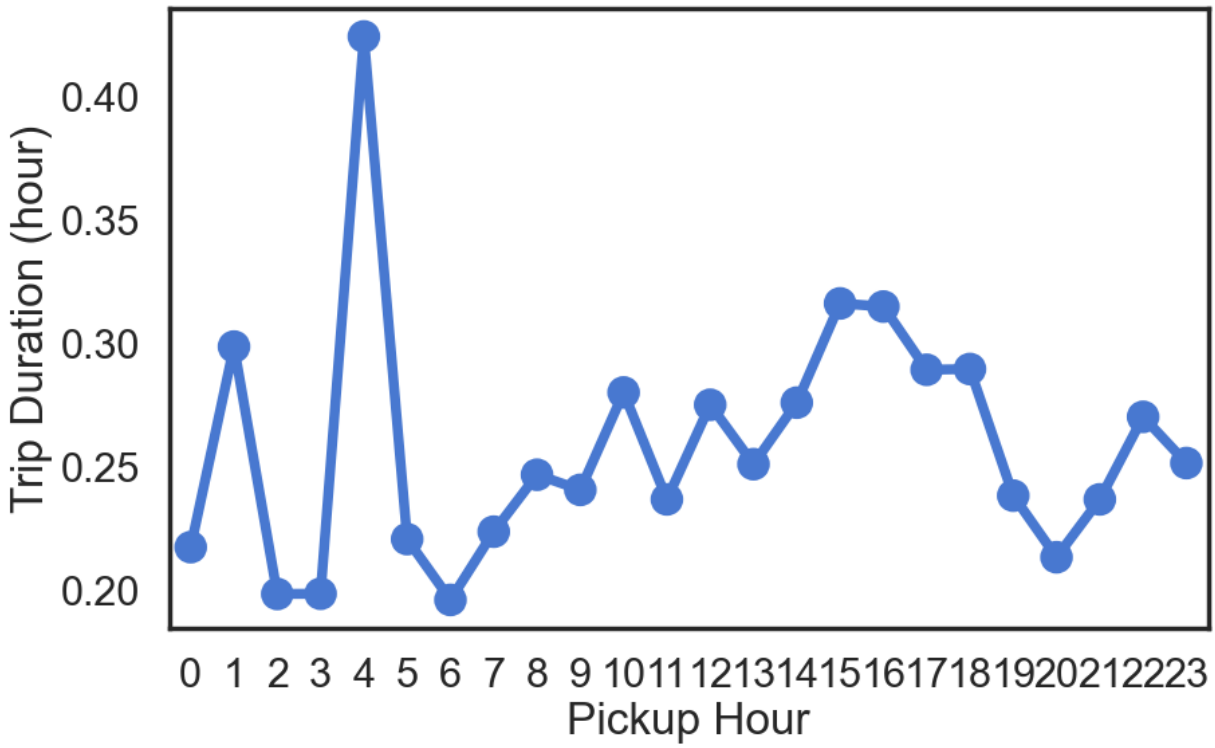
```
# Generating box plot with respect to passenger_count and trip_duration

df.passenger_count.value_counts()

plt.figure(figsize=(22, 6))

df_sub = df[df['trip_duration']*3600 < 10000]

sns.boxplot(x="passenger_count", y="trip_duration", data=df_sub)

plt.show()
```



The relationship between vendor id and duration

In [362]:

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

# Generating line plot for pickout_hour

group1 = data.groupby('pickup_hour').trip_duration.mean().to_frame().reset_index()

plt.figure(figsize=(10,6))

sns.pointplot(x='pickup_hour', y='trip_duration', data=group1)

plt.ylabel('Trip Duration (hour)')

plt.xlabel('Pickup Hour')

plt.show()
```



In [ ]:

```

```

Pick Up Points v/s Dropoff Points

In [363]:

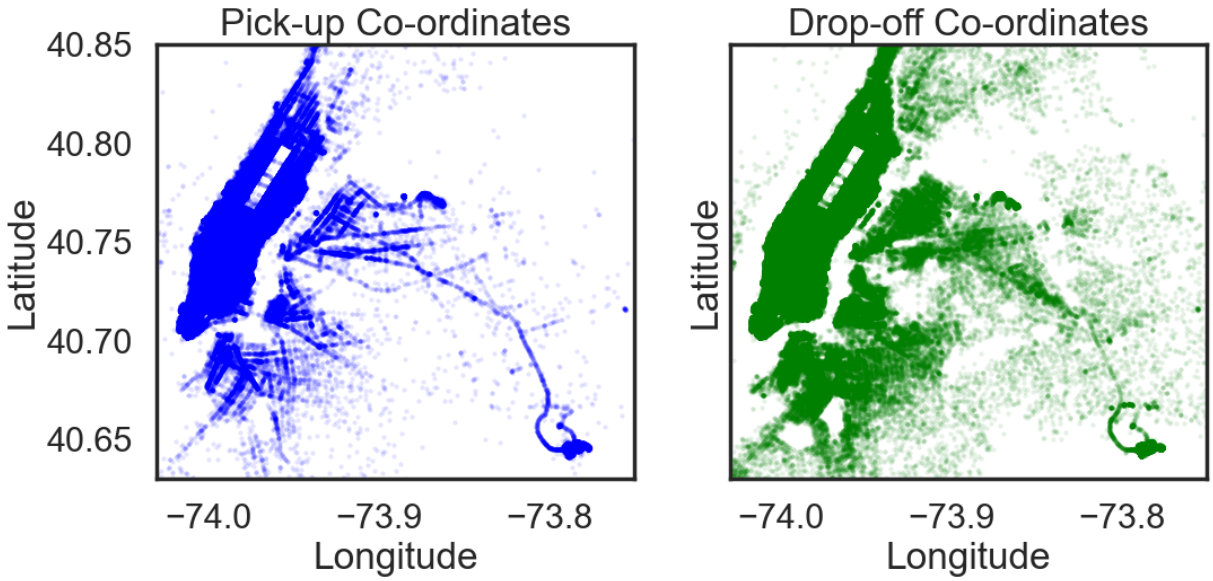
```
df.columns
```

Out[363]:

```
Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration', 'pickup_date', 'pickup_day', 'pickup_hour',
       'pickup_weekday', 'dropoff_weekday', 'dropoff_day', 'total_time',
       'dropoff_hour', 'pickup_time_of_day', 'dropoff_time_of_day',
       'pickup_days', 'duration_difference', 'log_trip_duration'],
      dtype='object')
```

In [364]:

```
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)
fig, ax = plt.subplots(ncols=2, sharex=True, sharey=True,figsize = (12,5))
ax[0].scatter(df['pickup_longitude'].values, df['pickup_latitude'].values,
color='blue', s=1, label='train', alpha=0.1)
ax[1].scatter(df['dropoff_longitude'].values, df['dropoff_latitude'].values,
color='green', s=1, label='train', alpha=0.1)
ax[1].set_title('Drop-off Co-ordinates')
ax[0].set_title('Pick-up Co-ordinates')
ax[0].set_ylabel('Latitude')
ax[0].set_xlabel('Longitude')
ax[1].set_ylabel('Latitude')
ax[1].set_xlabel('Longitude')
plt.ylim(city_lat_border)
plt.xlim(city_long_border)
plt.show()
```



In [ ]:

In [ ]:

Evaluation Metric (MAE)

- A suitable evaluation metric would be Mean Absolute Error (MAE). MAE is the average of the absolute differences between the predicted and actual values. This metric is preferred because it gives equal weightage to all errors and is less sensitive to outliers compared to other metrics such as Root Mean Squared Error (RMSE).

Bencmark Model

In [433]:

```
dfben=df.copy()
```

In [434]:

```
from sklearn.utils import shuffle
dfben= shuffle(dfben, random_state=10)
```

In [435]:

```
# diividing dataset in 3:1 ratio

div=int(df.shape[0]/4)

Train=dfben.loc[:3*div+1,:]
Test= dfben.loc[3*div+1:]
```

In [436]:

```
#Train.head()
```

In [437]:

```
Test.head()
```

Out[437]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude
546991	id2240736	1	2016-05-25 07:59:16	2016-05-25 08:05:02	1	-73.991364	40.732590
337027	id3071362	2	2016-03-09 16:47:45	2016-03-09 16:50:42	1	-73.966812	40.761429
542933	id1516922	1	2016-05-06 22:54:58	2016-05-06 23:14:21	1	-73.991211	40.749744
169349	id0023602	1	2016-03-24 18:59:29	2016-03-24 19:14:00	1	-74.007965	40.704826
535328	id1214106	1	2016-01-31 16:27:46	2016-01-31 16:58:13	1	-73.870811	40.773827

5 rows × 24 columns

In [438]:

```
# Taking mean of trip_duration  column from Train part of dataset

Test['mean_trip_duration']=Train['trip_duration'].mean()
#Test.head()
```

In [439]:

```
from sklearn.metrics import mean_absolute_error as MAE
# Finding Mean Absolute Error
simple_mean_error=MAE(Test['trip_duration'],Test['mean_trip_duration'])
simple_mean_error
```

Out[439]:

```
0.17339154852296407
```

mean error for pickup_day with respect to mean_trip_duration

In [440]:

```
# Making a pivot table for particular columns
week_day=pd.pivot_table(Train, values='trip_duration', index=['pickup_weekday'], aggfunc=np.mean)
week_day
```

Out[440]:

pickup_weekday	trip_duration
0	0.245145
1	0.274441
2	0.267113
3	0.280243
4	0.277122
5	0.253585
6	0.248005

In [441]:

```
# Applying for loop in order to store unique values in a given column
Test['week_day_mean']=0

for i in Train['pickup_weekday'].unique():
# Assigning the unique value to a new created column after posing some operations on train dataset

    Test['week_day_mean'][Test['pickup_weekday']==str(i)]=Train['trip_duration'][Train['pickup_weekday']==s
```

In [442]:

```
# Againg finding Mean Absolute Error
pickup_weekday_error=MAE(Test['trip_duration'],Test['week_day_mean'])
pickup_weekday_error
```

Out[442]:

```
0.26549705639134824
```

Mean trip_duration_hour with respect to passenger_count

In [443]:

```
#trip_duration_hour mean with respect to the mean of passenger_count

pass_count = pd.pivot_table(Train, values='trip_duration', index = ["passenger_count"], aggfunc=np.mean)
pass_count
```

Out[443]:

passenger_count	trip_duration
0	0.124222
1	0.255903
2	0.274139
3	0.285524
4	0.282831
5	0.303148
6	0.298740
7	0.005278

In [444]:

```
# Applying for loop to collect unique values
Test['pass_count_mean']=0
for i in Train['passenger_count'].unique():

    Test['pass_count_mean'][Test['passenger_count']==str(i)]=Train['trip_duration'][Train['passenger_count'
```

In [445]:

```
pass_count_error=MAE(Test['trip_duration'],Test['pass_count_mean'])
pass_count_error
```

Out[445]:

```
0.26549705639134824
```

pickup_day vs trip_duration

In [446]:

```
pick_up=pd.pivot_table(Train, values='trip_duration',index='pickup_day', aggfunc=np.mean)
pick_up
```

Out[446]:

pickup_day	trip_duration
1	0.266310
2	0.252257
3	0.270433
4	0.258755
5	0.290538
6	0.258197
7	0.254653
8	0.268010
9	0.262873
10	0.255959
11	0.257020
12	0.259904
13	0.264483
14	0.267257
15	0.270911
16	0.260339
17	0.261967
18	0.262952
19	0.265211
20	0.258378
21	0.262061
22	0.266806
23	0.272498
24	0.272777
25	0.274208
26	0.269431
27	0.257617
28	0.260653
29	0.266058
30	0.250931
31	0.276745

In [447]:

```
Test['pick_mean']=0
for i in Train['pickup_day'].unique():
    Test['pick_mean'][Test['pickup_day']==str(i)]=Train['trip_duration'][Train['pickup_day']==str(i)].mean(
```

In [ ]:

```
```

In [448]:

```
pickup_day_error=MAE(Test['trip_duration'],Test['pick_mean'])
pickup_day_error
```

Out[448]:

```
0.26549705639134824
```

In [449]:

```
Test.head()
#Test.shape
#Test.describe()
```

Out[449]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude
546991	id2240736	1	2016-05-25 07:59:16	2016-05-25 08:05:02	1	-73.991364	40.732590
337027	id3071362	2	2016-03-09 16:47:45	2016-03-09 16:50:42	1	-73.966812	40.761429
542933	id1516922	1	2016-05-06 22:54:58	2016-05-06 23:14:21	1	-73.991211	40.749744
169349	id0023602	1	2016-03-24 18:59:29	2016-03-24 19:14:00	1	-74.007965	40.704826
535328	id1214106	1	2016-01-31 16:27:46	2016-01-31 16:58:13	1	-73.870811	40.773827

5 rows × 28 columns

Mean trip_duration_hour with respect to passenger_count, pickup_time_of_day and dropoff_time_of_day

In [450]:

```
combo = pd.pivot_table(Train, values = 'trip_duration', index = ['passenger_count','pickup_time_of_day','dr
combo
```

Out[450]:

			trip_duration
passenger_count	**pickup_time_of_day**	**dropoff_time_of_day**	
0	**Afternoon**	**Afternoon**	0.506574
	Evening	**Evening**	0.009352
		Late night	0.106944
	Late night	**Late night**	0.007525
	Morning	**Afternoon**	0.432222
...	**...**	**...**	...
6	**Late night**	**Morning**	0.286772
	Morning	**Afternoon**	0.377804
		Late night	16.731944
		Morning	0.264533
7	**Morning**	**Morning**	0.005278

76 rows × 1 columns

In [ ]:

```
```

In [451]:

```
# Define the three columns to group by
group_cols = ['passenger_count', 'pickup_time_of_day', 'dropoff_time_of_day']

# Calculate the mean of trip_duration for each group in the train dataset
train_group_means = Train.groupby(group_cols)['trip_duration'].transform('mean')

# Assign the mean value to the Super_mean column in the test dataset
Test['Super_mean'] = train_group_means
```

In [452]:

```
Test.dropna(subset=['Super_mean'], inplace=True)
```

In [453]:

```
#calculating mean absolute error
super_mean_error = MAE(Test['trip_duration'] , Test['Super_mean'] )
super_mean_error
```

Out[453]:

```
0.13979233317509132
```

Mean trip_duration_hour with respect store_and_fwd_flag

In [454]:

```
flag=pd.pivot_table(Train, values='trip_duration', index='store_and_fwd_flag', aggfunc=np.mean)
flag
```

Out[454]:

store_and_fwd_flag	trip_duration
0	0.263956
1	0.306166

In [455]:

```
Test['Flag_mean']=0

for i in Train['store_and_fwd_flag'].unique():

    Test['Flag_mean'][Test['store_and_fwd_flag']==str(i)]=Train['trip_duration'][Train['store_and_fwd_flag'
```

In [456]:

```
flag_mean_error=MAE(Test['trip_duration'],Test['Flag_mean'])
flag_mean_error
```

Out[456]:

```
0.0961111111111111
```

Conclusion

1. The error of simple mean of trip duration hour is 0.17277651586027712 which is also almost equal to MAE of dropp time and pickup time of the day is 0.17263892934259373, 0.17262796487353846 respectively
2. The str_fwd_error and passanger count error is same i.e. 0.2652592807074405
3. The mean of trip_duration_hour with respect to pickup_time_error , dropoff_time_error and pass_count error is 0.16813712281281915

KNN Analysis

In [465]:

```
# Create a copy of initial dataset df so that it doesn't get modified too much
dfmath=df.copy()
```

In [466]:

```
#dfmath.shape
#df.describe()
dfmath.columns
```

Out[466]:

```
Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration', 'pickup_date', 'pickup_day', 'pickup_hour',
       'pickup_weekday', 'dropoff_weekday', 'dropoff_day', 'total_time',
       'dropoff_hour', 'pickup_time_of_day', 'dropoff_time_of_day',
       'pickup_days', 'duration_difference', 'log_trip_duration'],
      dtype='object')
```

In [ ]:

```

```

In [467]:

```
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Radius of the earth in km
    dlat = np.radians(lat2 - lat1)
    dlon = np.radians(lon2 - lon1)
    a = np.sin(dlat / 2) ** 2 + np.cos(np.radians(lat1)) * np.cos(np.radians(lat2)) * np.sin(dlon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    distance_km = R * c
    return distance_km

# Calculate the distance for each row in the dataset
dfmath['distance_km'] = haversine(dfmath['pickup_latitude'], dfmath['pickup_longitude'], dfmath['dropoff_la
print(dfmath['distance_km'].describe())
```

```
count    729322.000000
mean          3.441139
std           4.353132
min           0.000000
25%           1.232695
50%           2.095672
75%           3.876481
max        1240.908677
Name: distance_km, dtype: float64
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [469]:

```
dfknn=dfmath.copy()
```

In [ ]:

```
```

In [470]:

```
# pd.options.display.float_format = '{:.6f}'.format
# df['trip_duration']
#df.head()
#
dfknn.columns
```

Out[470]:

```
Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration', 'pickup_date', 'pickup_day', 'pickup_hour',
       'pickup_weekday', 'dropoff_weekday', 'dropoff_day', 'total_time',
       'dropoff_hour', 'pickup_time_of_day', 'dropoff_time_of_day',
       'pickup_days', 'duration_difference', 'log_trip_duration',
       'distance_km'],
      dtype='object')
```

In [471]:

```
print(dfknn['distance_km'].describe())
```

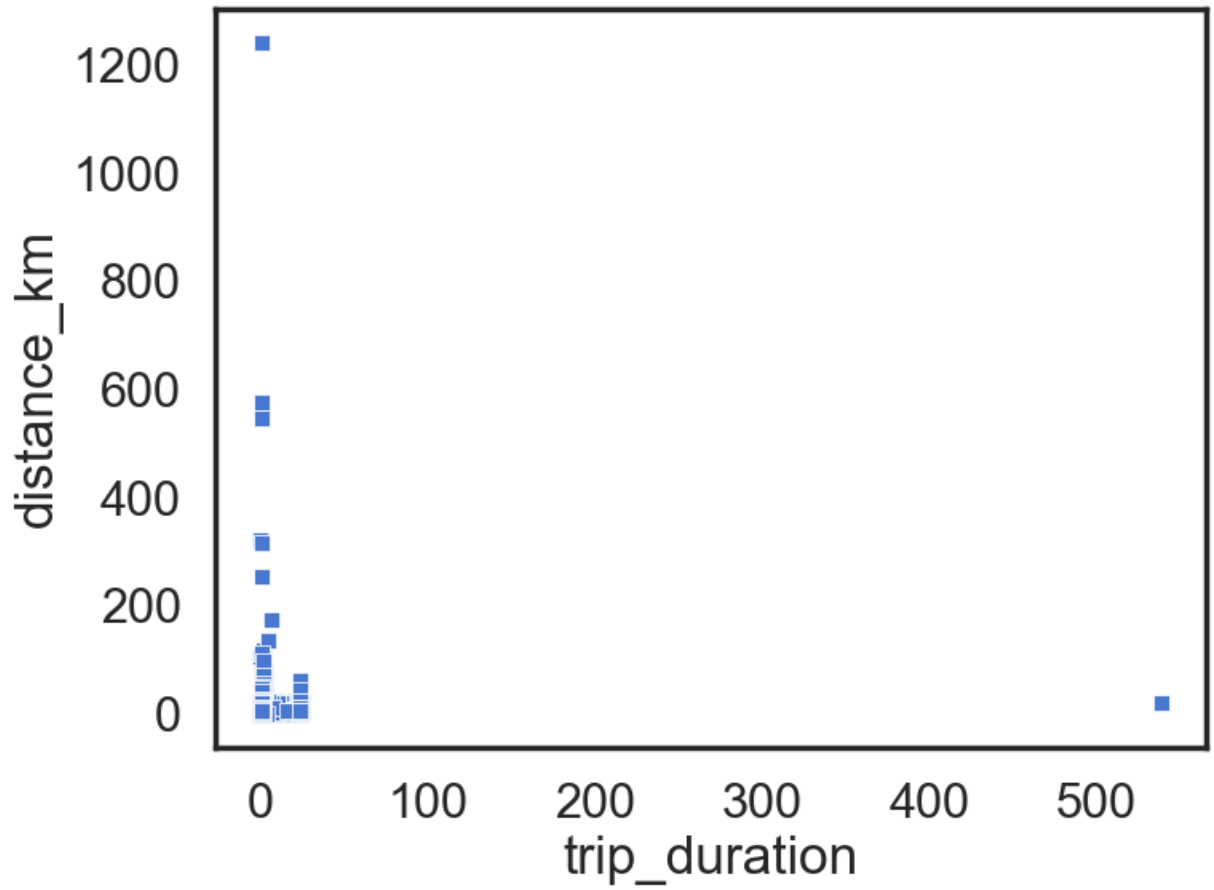
```
count    729322.000000
mean          3.441139
std           4.353132
min           0.000000
25%           1.232695
50%           2.095672
75%           3.876481
max        1240.908677
Name: distance_km, dtype: float64
```

In [472]:

```
# Creating a scatter plot inorder to identify outlier
plt.figure(figsize=(8, 6))
a=dfknn['trip_duration']
ax=sns.scatterplot(data=dfknn, x=a, y="distance_km", s=50, palette='viridis', marker="s")
a.describe()
```

Out[472]:

```
count    729322.000000
mean          0.264508
std           1.073507
min           0.000278
25%           0.110278
50%           0.184167
75%           0.298611
max         538.815556
Name: trip_duration, dtype: float64
```



In [1444]:

```
dfknn[a>500]
```

Out[1444]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude
21813	id1864733	1	2016-01-05 00:19:42	2016-01-27 11:08:38	1	-73.78965	40.643559

1 rows × 25 columns

In [474]:

```
dfknn[dfknn['distance_km']>300]
```

Out[474]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude
244913	id2306955	1	2016-05-07 18:58:53	2016-05-07 19:12:05	1	-72.809669	51.881084
441429	id0982904	1	2016-04-28 13:32:14	2016-04-28 14:14:09	2	-73.870682	40.773598
621372	id2644780	1	2016-05-03 16:24:07	2016-05-03 17:18:34	2	-73.991325	40.750023
654569	id0116374	1	2016-04-02 20:33:19	2016-04-02 20:38:01	1	-74.007095	40.717113
697620	id0978162	1	2016-02-24 16:20:59	2016-02-24 16:35:34	4	-75.354332	34.712234

5 rows × 25 columns

In [475]:

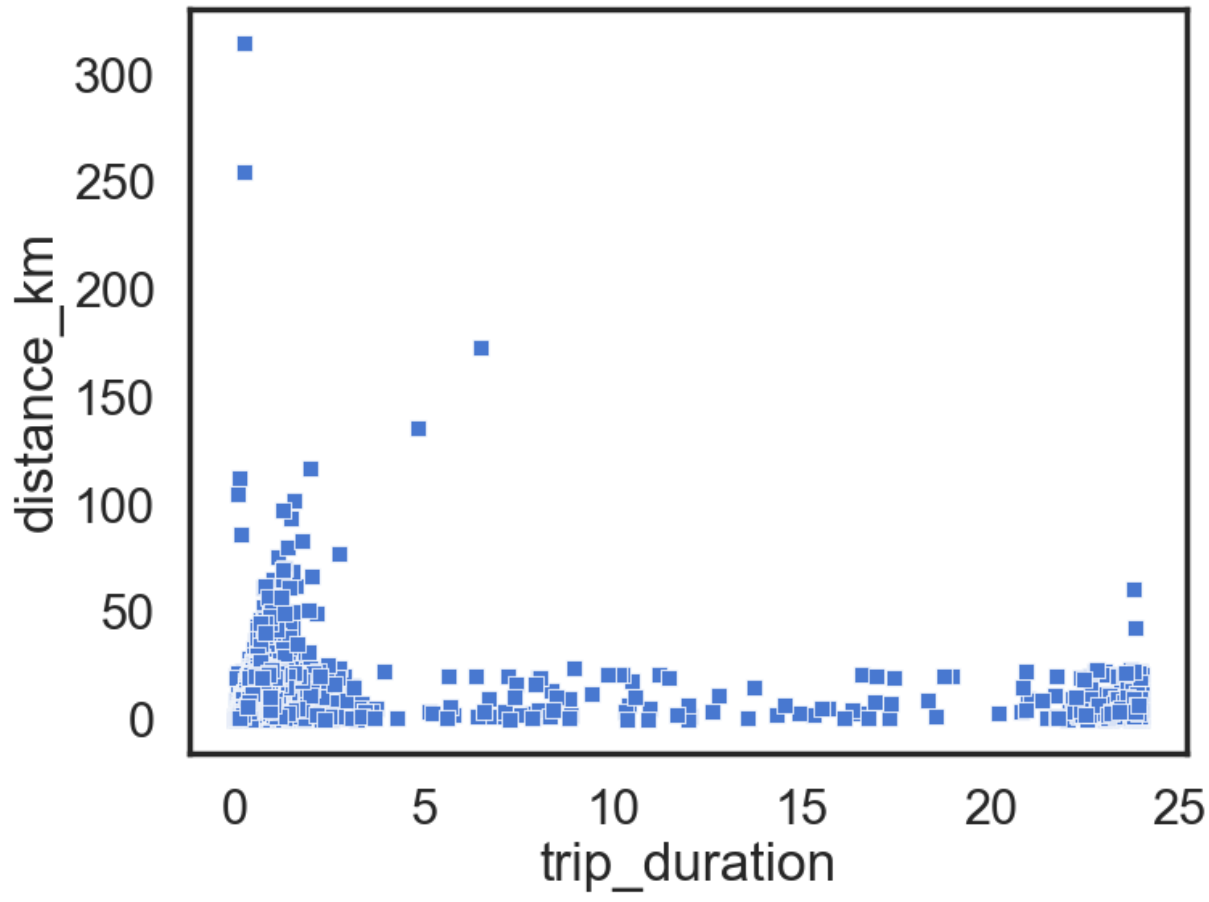
```
#Dropping the value from the dataset
dfknn.drop(dfknn[dfknn['id'].isin(['id1864733', 'id2306955','id0982904','id2644780','id0116374','id0978162
```

In [476]:

```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=dfknn, x=a, y="distance_km", s=50, palette='viridis', marker="s")
```

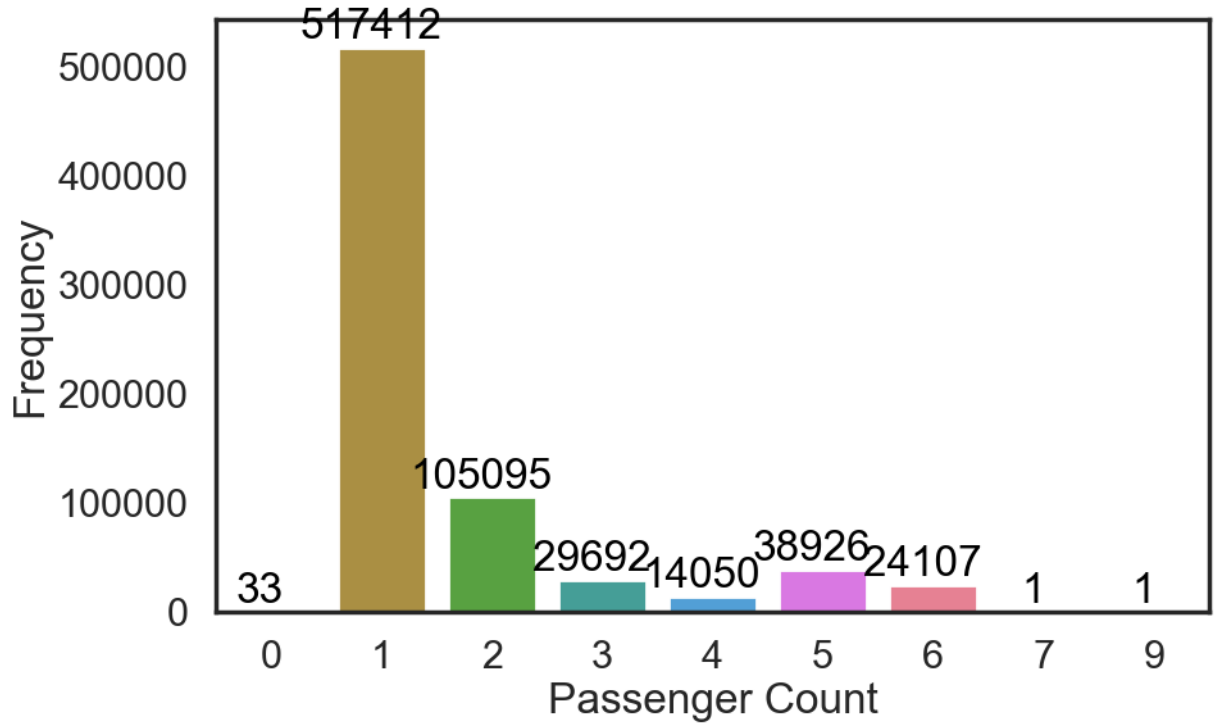
Out[476]:

```
<Axes: xlabel='trip_duration', ylabel='distance_km'>
```



In [477]:

```
# Binary Features
plt.figure(figsize=(22, 6))
#fig, axs = plt.subplot(ncols=2)

# Passenger Count
plt.subplot(121)
ax=sns.countplot(data=dfknn, x='passenger_count',palette=sns.color_palette('husl'))
plt.xlabel('Passenger Count')
plt.ylabel('Frequency')
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black')
```

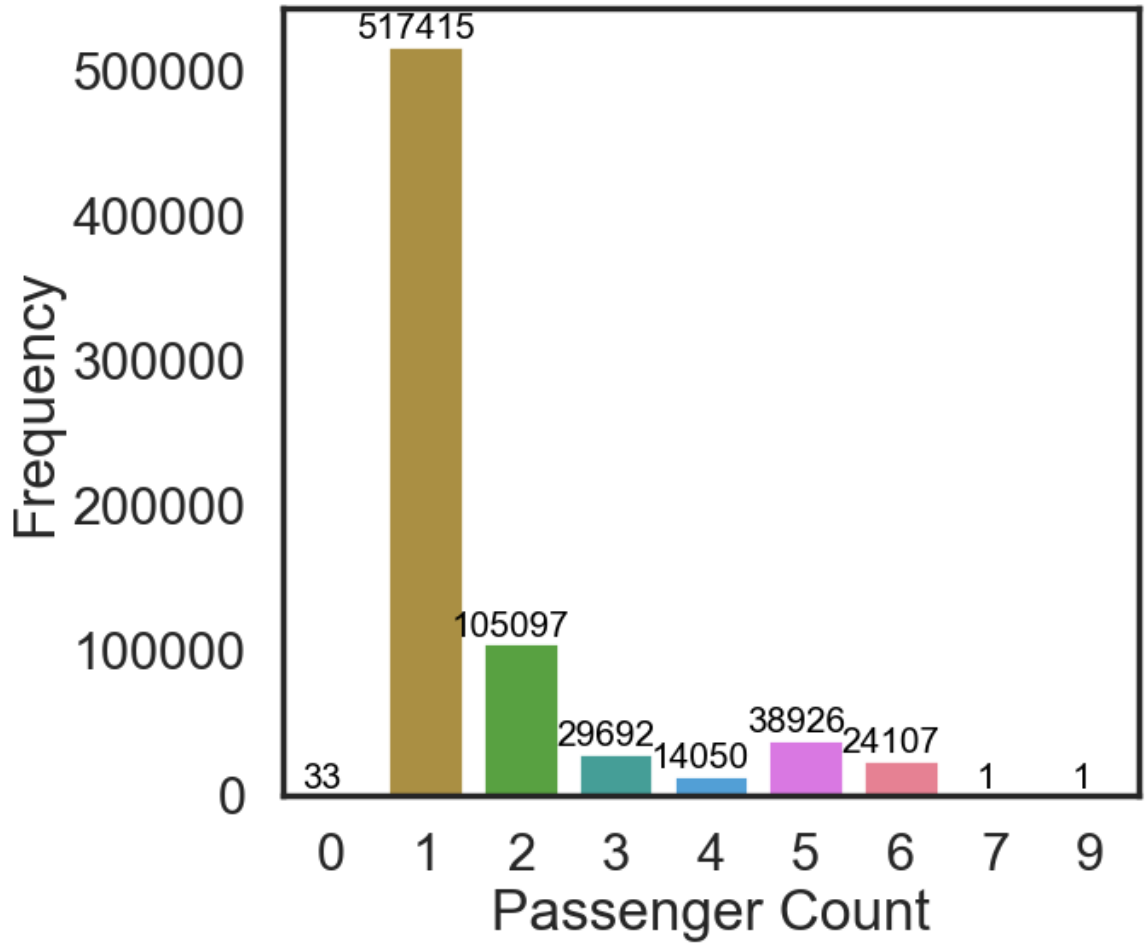


In [478]:

```
dfknn = dfknn.loc[~dfknn['passenger_count'].isin([0, 7, 9])]
```

In [479]:

```
# Binary Features
plt.figure(figsize=(22, 6))
#fig, axs = plt.subplot(ncols=2)

# Passenger Count
plt.subplot(131)
ax=sns.countplot(data=df, x='passenger_count',palette=sns.color_palette('husl'))
plt.xlabel('Passenger Count')
plt.ylabel('Frequency')
for p in ax.patches:
        ax.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()),
                    ha='center', va='bottom',
                    color= 'black',fontsize=15)
```

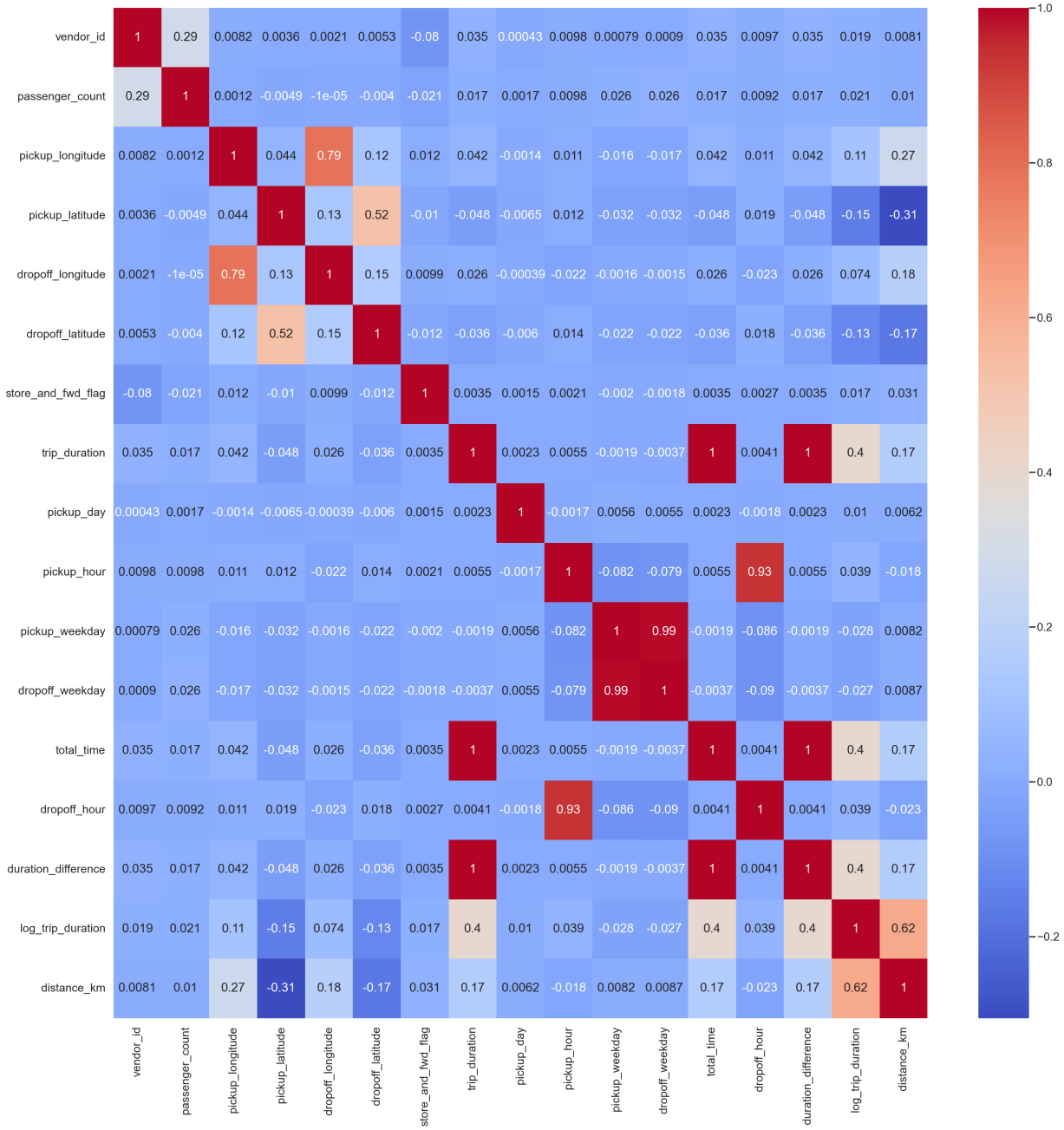


In [480]:

```
import seaborn as sns
import pandas as pd

# Load the dataset

# Compute the correlation matrix
corr = dfknn.corr()
fig, ax = plt.subplots(figsize=(35, 35))
# Create a heatmap to visualize the correlation matrix
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[480]:

<Axes: >

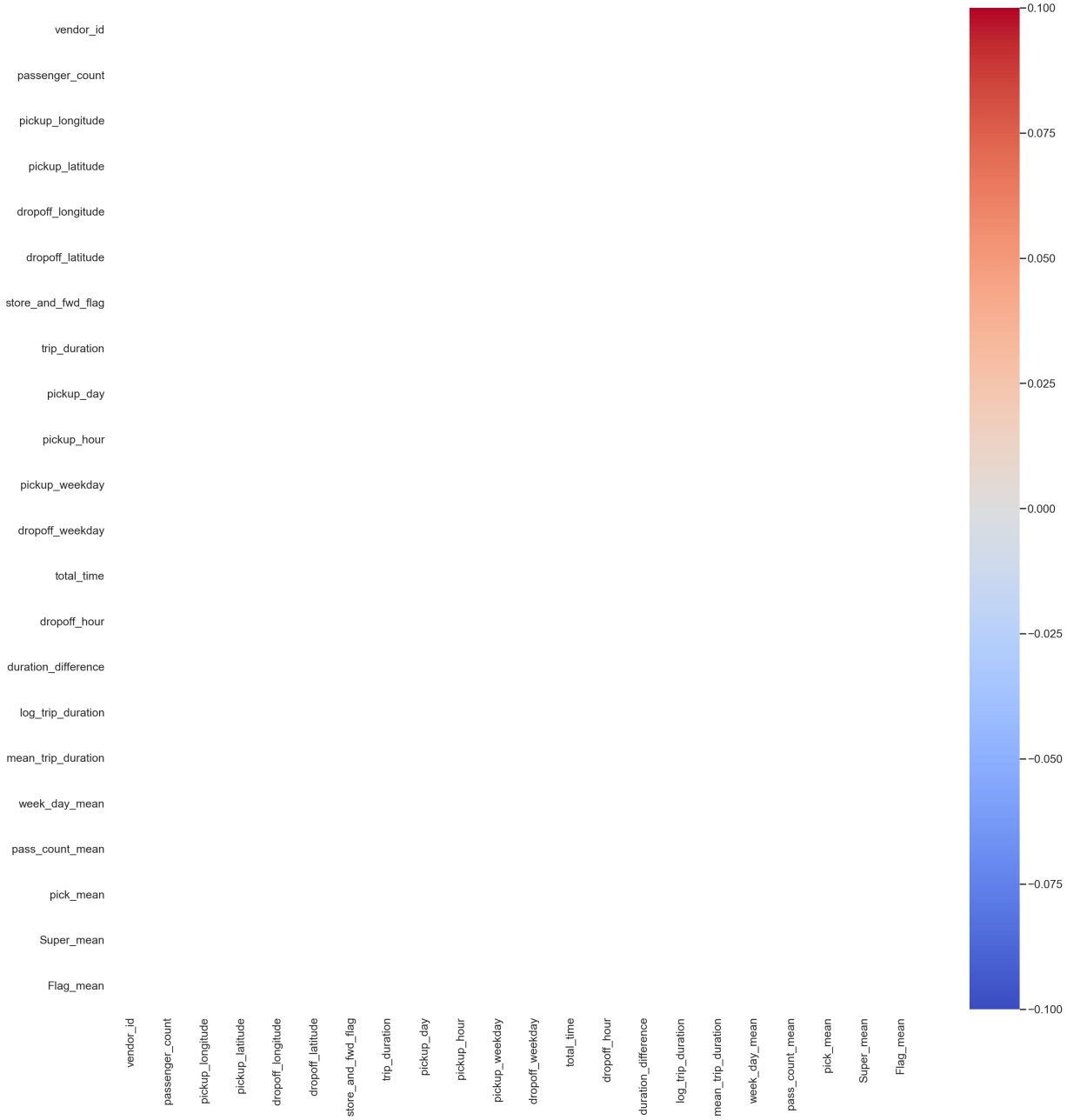


In [481]:

```
import seaborn as sns
import pandas as pd

# Load the dataset

# Compute the correlation matrix
corr = Test.corr()
fig, ax = plt.subplots(figsize=(35, 35))
# Create a heatmap to visualize the correlation matrix
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[481]:

<Axes: >



In [482]:

```
dfknn.dtypes
```

Out[482]:

```
id                             object
vendor_id                       int64
pickup_datetime        datetime64[ns]
dropoff_datetime       datetime64[ns]
passenger_count                 int64
pickup_longitude              float64
pickup_latitude               float64
dropoff_longitude             float64
dropoff_latitude              float64
store_and_fwd_flag              int64
trip_duration                 float64
pickup_date                    object
pickup_day                      int64
pickup_hour                     int64
pickup_weekday                  int64
dropoff_weekday                 int64
dropoff_day                    object
total_time                    float64
dropoff_hour                    int64
pickup_time_of_day             object
dropoff_time_of_day            object
pickup_days                    object
duration_difference           float64
log_trip_duration             float64
distance_km                   float64
dtype: object
```

In [483]:

```
#d=dfknn
#d.drop(columns=['pickup_day','dropoff_day','weekday_name''pickup_day','dropoff_day'],inplace=True)
#d.drop(columns=['id','vendor_id','pickup_datetime','dropoff_datetime','pickup_date','dropoff_weekday','weel
#data.dtypes
#print(type(df.loc[1, "id"]))
#d.dtypes
```

In [484]:

```
data = dfknn.iloc[1:18001,]
cat_cols = ['pickup_time_of_day', 'dropoff_time_of_day']
data = pd.concat([data, pd.get_dummies(data[cat_cols].astype('str'))], axis=1)
data.drop(columns = ['pickup_time_of_day','pickup_datetime','dropoff_datetime', 'dropoff_time_of_day','id',
data
```

Out[484]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	tı
1	2	-73.988312	40.731743	-73.994751	40.694931	0	
2	2	-73.997314	40.721458	-73.948029	40.774918	0	
3	6	-73.961670	40.759720	-73.956779	40.780628	0	
4	1	-74.017120	40.708469	-73.988182	40.740631	0	
5	2	-73.993614	40.751884	-73.995422	40.723862	0	
...	...	...	...	...	...	...	
17997	1	-73.985321	40.741386	-74.014412	40.712418	0	
17998	3	-73.978935	40.764229	-74.000969	40.757381	0	
17999	1	-73.961990	40.778889	-73.971840	40.781971	0	
18000	1	-73.984154	40.725342	-73.988571	40.758640	0	
18001	1	-73.978752	40.736553	-73.974030	40.746498	0	

18000 rows × 24 columns

In [501]:

```
#seperate features and target
x = data.drop(['trip_duration'], axis=1)
y = data["trip_duration"]
x.shape,y.shape
```

Out[501]:

```
((18000, 23), (18000,))
```

In [502]:

```
#dfknn.drop(columns=['pickup_days'],inplace=True)
```

In [503]:

```
# Importing MinMax Scaler

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [504]:

```
x = pd.DataFrame(x_scaled)
```

In [505]:

```
# Importing Train test split
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56)
```

In [506]:

```
#importing KNN regressor and metric mse
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.metrics import mean_absolute_error as MAE
```

In [507]:

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Creating instance of KNN
reg = KNN(n_neighbors = 5)

# Fitting the model
reg.fit(train_x, train_y)

# Predicting over the Train Set and calculating MSE
test_predict = reg.predict(test_x)
k = MAE(test_predict, test_y)



mse = mean_squared_error(test_y,test_predict,squared=False)
rmse = np.sqrt(mse)
r2 = r2_score(test_y,test_predict)

print("RMSE:       ", rmse)
print("R2:         ", r2)

print('Test MAE    ', k )
```

```
RMSE:        0.6381830734402639
R2:          0.813186163341647
Test MAE     0.07527864197530865
```

In [508]:

```
def Elbow(K):
  #initiating empty list
    test_MAE = []

  #training model for evey value of K
    for i in K:
        #Instance of KNN
        reg = KNN(n_neighbors = i)
        reg.fit(train_x, train_y)
        #Appending MAE value to empty list claculated using the predictions
        tmp = reg.predict(test_x)
        tmp = MAE(tmp,test_y)
        test_MAE.append(tmp)

    return test_MAE
```

In [509]:

```
#Defining K range
k = range(1,10)
```

In [510]:

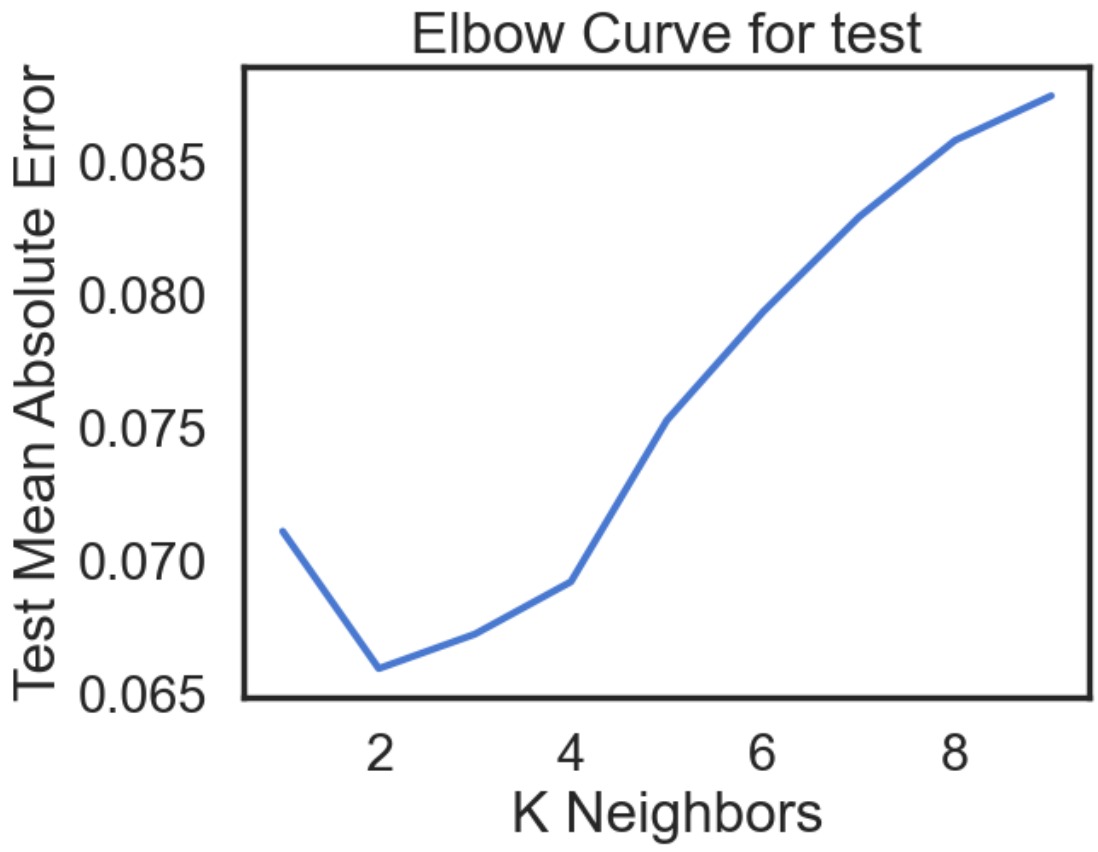
```
test=Elbow(k)
```

In [511]:

```
# plotting the Curves
plt.plot(k, test)
plt.xlabel('K Neighbors')
plt.ylabel('Test Mean Absolute Error')
plt.title('Elbow Curve for test')
```

Out[511]:

```
Text(0.5, 1.0, 'Elbow Curve for test')
```



In [512]:

```
knn_train_score = reg.score(train_x,train_y)
print(knn_train_score*100,"%")
```

```
81.68884968470309 %
```

In [ ]:

```

```

In [513]:

```
knn_test_score = reg.score(test_x,test_y)
knn_test_score*100
```

Out[513]:

```
81.3186163341647
```

In [514]:

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Creating instance of KNN
reg = KNN(n_neighbors = 3)

# Fitting the model
reg.fit(train_x, train_y)

# Predicting over the Train Set and calculating MSE
test_predict = reg.predict(test_x)
k = MAE(test_predict, test_y)

mse = mean_squared_error(test_y,test_predict,squared=False)
rmse = np.sqrt(mse)
r2 = r2_score(test_y,test_predict)


knn_train_score = reg.score(train_x,train_y)

knn_test_score = reg.score(test_x,test_y)

print("Knn_test_score:  ", knn_test_score)

print("knn_train_score: ",knn_train_score)
print("RMSE:            ", rmse)
print("R2:              ", r2)


print('Test MAE:        ', k )
```

```
Knn_test_score:   0.9135591687208122
knn_train_score:  0.9222628456590701
RMSE:             0.5263477307961296
R2:               0.9135591687208122
Test MAE:         0.06723621399176954
```

In [ ]:

In [ ]:

Observation

1. Preprocessing the columns 'pickup_time_of_day' and 'dropoff_time_of_day' and divide them into 4 parts. using general method of finding the error in KNN algorithm, the MAE comes out to be as follows
2. For the features used in above calculations, we got a bit under fit results. Although the results are not that bad.
3. for K=5, we ha R2 value of 0.813186163341647 and for K=3, we have R2 value of 0.9135591687208122.
4. These values are pretty impressive but there is a big room for improvement in the hypothesis.

KNN analysis for multiple features

In [515]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Select the top 5 features
X = dfknn[['pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude', 'distance_km']]
y = dfknn['trip_duration']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the KNN regressor
knn = KNeighborsRegressor()

# Fit the model on the training data
knn.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn.predict(X_test)

# Evaluate the model performance using mean squared error
mse = mean_squared_error(y_test, y_pred)
print('Mean squared error:', mse)

r2 = r2_score(y_test, y_pred)
print('R-squared:', r2)

k = MAE(y_pred, y_test)
print('Test MAE:          ', k )
```

```
Mean squared error: 0.8275736057535062
R-squared: -0.18640353880937188
Test MAE:           0.13300704268269462
```

In [ ]:

```

```

Linear Regression

In [516]:

```
dflr=df.copy()
```

In [517]:

```
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Radius of the earth in km
    dlat = np.radians(lat2 - lat1)
    dlon = np.radians(lon2 - lon1)
    a = np.sin(dlat / 2) ** 2 + np.cos(np.radians(lat1)) * np.cos(np.radians(lat2)) * np.sin(dlon / 2) ** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    distance_km = R * c
    return distance_km

# Calculate the distance for each row in the dataset
dflr['distance_km'] = haversine(dflr['pickup_latitude'], dflr['pickup_longitude'], dflr['dropoff_latitude']
```

In [518]:

```
dflr.shape
```

Out[518]:

```
(729322, 25)
```

In [ ]:

In [519]:

```
dflr.head()
```

Out[519]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dro
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	

5 rows × 25 columns

Linear Regression on ('pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude', 'distance_km') features

In [520]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Select the top 5 features
X = dflr[['pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude', 'distance_km']]
y = dflr['trip_duration']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the linear regression model
lr = LinearRegression()

# Fit the model on the training data
lr.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = lr.predict(X_test)

# Evaluate the model performance using metrics such as mean squared error and R-squared
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
k = MAE(y_pred, y_test)
print('Test MAE:          ', k )
print('Mean squared error:', mse)
print('R-squared:', r2)
```

```
Test MAE:           0.12785732055218849
Mean squared error: 2.0311781617547813
R-squared: 0.010295806448668898
```

Linear Regression on ('distance_km','pickup_hour','passenger_count') features

In [521]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


X = dflr[['distance_km','pickup_hour','passenger_count']]
y = dflr['trip_duration']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the linear regression model
lr = LinearRegression()

# Fit the model on the training data
lr.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = lr.predict(X_test)

# Calculate the mean squared error and R-squared value
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

k = MAE(y_pred, y_test)
print('Test MAE:          ', k )
print('Mean squared error:', mse)
print('R-squared:', r2)
```

```
Test MAE:           0.12728821350623243
Mean squared error: 2.030587754599998
R-squared: 0.010583486007259002
```

Linear Regression on preprocessed columns(days of the week for pickup and dropoff)

In [522]:

```
dflr.head()
```

Out[522]:

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	drop
0	id1080784	2	2016-02-29 16:40:21	2016-02-29 16:47:01	1	-73.953918	40.778873	
1	id0889885	1	2016-03-11 23:35:37	2016-03-11 23:53:57	2	-73.988312	40.731743	
2	id0857912	2	2016-02-21 17:59:33	2016-02-21 18:26:48	2	-73.997314	40.721458	
3	id3744273	2	2016-01-05 09:44:31	2016-01-05 10:03:32	6	-73.961670	40.759720	
4	id0232939	1	2016-02-17 06:42:23	2016-02-17 06:56:31	1	-74.017120	40.708469	

5 rows × 25 columns

In [523]:

```
# Selecting a part of dataset for train dataset
data = dflr.iloc[1:18001,]
cat_cols = ['pickup_days', 'dropoff_day']
# Creating dummy columns
data = pd.concat([data, pd.get_dummies(data[cat_cols].astype('str'))], axis=1)
data.drop(columns = ['pickup_days', 'dropoff_day','id','vendor_id','pickup_time_of_day','dropoff_time_of_da
data
```

Out[523]:

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	tı
1	2	-73.988312	40.731743	-73.994751	40.694931	0	
2	2	-73.997314	40.721458	-73.948029	40.774918	0	
3	6	-73.961670	40.759720	-73.956779	40.780628	0	
4	1	-74.017120	40.708469	-73.988182	40.740631	0	
5	2	-73.993614	40.751884	-73.995422	40.723862	0	
...	...	...	...	...	...	...	
17996	1	-73.995468	40.759930	-73.988617	40.769260	0	
17997	1	-73.985321	40.741386	-74.014412	40.712418	0	
17998	3	-73.978935	40.764229	-74.000969	40.757381	0	
17999	1	-73.961990	40.778889	-73.971840	40.781971	0	
18000	1	-73.984154	40.725342	-73.988571	40.758640	0	

18000 rows × 30 columns

In [524]:

```
data.dtypes
```

Out[524]:

```
passenger_count                  int64
pickup_longitude               float64
pickup_latitude                float64
dropoff_longitude              float64
dropoff_latitude               float64
store_and_fwd_flag               int64
trip_duration                  float64
pickup_day                       int64
pickup_hour                      int64
pickup_weekday                   int64
dropoff_weekday                  int64
total_time                     float64
dropoff_hour                     int64
duration_difference            float64
log_trip_duration              float64
distance_km                    float64
pickup_days_pickup_Friday        uint8
pickup_days_pickup_Monday        uint8
pickup_days_pickup_Saturday      uint8
pickup_days_pickup_Sunday        uint8
pickup_days_pickup_Thursday      uint8
pickup_days_pickup_Tuesday       uint8
pickup_days_pickup_Wednesday     uint8
dropoff_day_dropoff_Friday       uint8
dropoff_day_dropoff_Monday       uint8
dropoff_day_dropoff_Saturday     uint8
dropoff_day_dropoff_Sunday       uint8
dropoff_day_dropoff_Thursday     uint8
dropoff_day_dropoff_Tuesday      uint8
dropoff_day_dropoff_Wednesday    uint8
dtype: object
```

In [525]:

```
#seperating independent and dependent variables
x = data.drop(['trip_duration'], axis=1)
y = data['trip_duration']
x.shape, y.shape
```

Out[525]:

```
((18000, 29), (18000,))
```

In [526]:

```
# Performing the train test split function
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56)
```

In [527]:

```
#importing Linear Regression
from sklearn.linear_model import LinearRegression as LR
```

In [528]:

```
# Dropping all the columns that are not numerical

#d.drop(columns=['pickup_days','dropoff_day'],inplace=True)
#dflr.drop(columns=['id','vendor_id','pickup_time_of_day','dropoff_time_of_day','pickup_days','dropoff_day'
#data.drop(columns=[],inplace=True)
```

In [529]:

```
dflr.dtypes
```

Out[529]:

```
id                             object
vendor_id                       int64
pickup_datetime        datetime64[ns]
dropoff_datetime       datetime64[ns]
passenger_count                 int64
pickup_longitude              float64
pickup_latitude               float64
dropoff_longitude             float64
dropoff_latitude              float64
store_and_fwd_flag              int64
trip_duration                 float64
pickup_date                    object
pickup_day                      int64
pickup_hour                     int64
pickup_weekday                  int64
dropoff_weekday                 int64
dropoff_day                    object
total_time                    float64
dropoff_hour                    int64
pickup_time_of_day             object
dropoff_time_of_day            object
pickup_days                    object
duration_difference           float64
log_trip_duration             float64
distance_km                   float64
dtype: object
```

In [530]:

```
# Creating instance of Linear Regresssion
lr = LR()

# Fitting the model
lr.fit(train_x, train_y)
```

Out[530]:

LinearRegression

LinearRegression()

In [531]:

```
# Predicting over the Train Set and calculating error
train_predict = lr.predict(train_x)
k = MAE(train_predict, train_y)


from sklearn.metrics import mean_squared_error, r2_score

# Calculate RMSE
rmse = mean_squared_error(train_y, train_predict, squared=False)
print('RMSE:', rmse)


print('Training Mean Absolute Error', k )
```

```
RMSE: 9.894114161864067e-16
Training Mean Absolute Error 2.91627792837777e-16
```

In [532]:

```
# Predicting over the Test Set and calculating error
test_predict = lr.predict(test_x)
k = MAE(test_predict, test_y)


from sklearn.metrics import mean_squared_error, r2_score

# Calculate RMSE
rmse = mean_squared_error(test_y, test_predict, squared=False)
print('RMSE:', rmse)
k = MAE(test_predict, test_y)
print('Test MAE:          ', k )

print('Test Mean Absolute Error    ', k )
```

```
RMSE: 9.072692330583305e-16
Test MAE:           2.8050664125583373e-16
Test Mean Absolute Error     2.8050664125583373e-16
```

Parameters of Linear Regression

In [533]:

```
lr.coef_
```

Out[533]:

```
array([-7.96934555e-18,  3.35725913e-15, -1.42795689e-15,  8.46950651e-17,
       -2.01819454e-16,  4.11252892e-17, -4.99325922e-19,  1.54604689e-18,
        1.44547612e-07,  2.14509082e-16,  3.39261289e-04, -3.10532866e-18,
       -6.15005946e-05, -4.87756511e-17, -6.18312876e-18, -1.00623057e-07,
        1.58669042e-07, -1.89248007e-07,  2.15211904e-07,  4.20525049e-08,
        3.48746781e-08,  5.42240780e-08, -1.60172367e-07,  1.58725981e-07,
       -2.16095028e-07,  2.46730731e-07, -1.58300317e-07,  1.37972733e-07,
       -2.59242785e-08])
```

Plotting the coefficients

In [534]:

```
range(len(train_x.columns))
```

Out[534]:

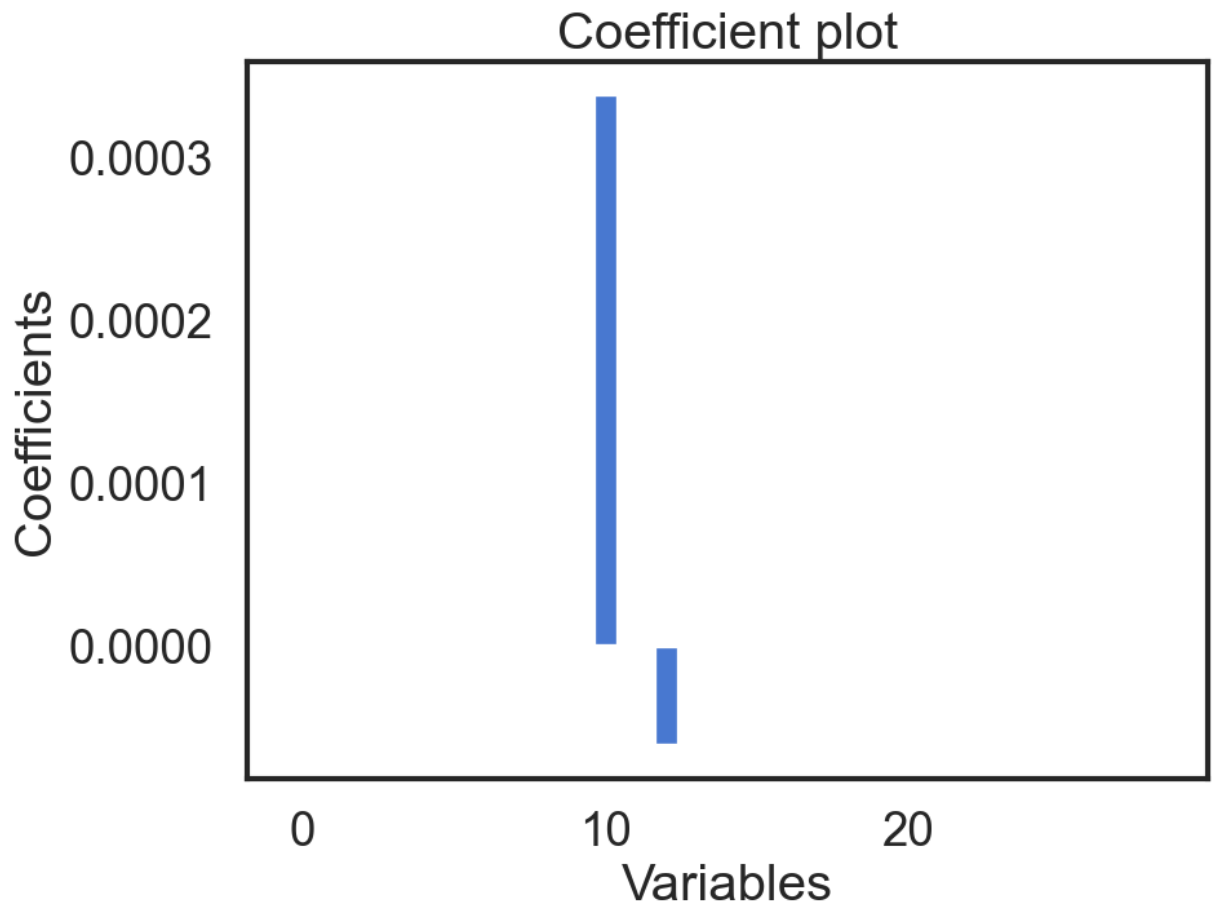
```
range(0, 29)
```

In [535]:

```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Coefficient plot')
```

Out[535]:

```
Text(0.5, 1.0, 'Coefficient plot')
```



Checking assumptions of Linear Model

In [536]:

```
# Arranging and calculating the Residuals
residuals = pd.DataFrame({
    'fitted values' : test_y,
    'predicted values' : test_predict,
})

residuals['residuals'] = residuals['fitted values'] - residuals['predicted values']
residuals.head()
```

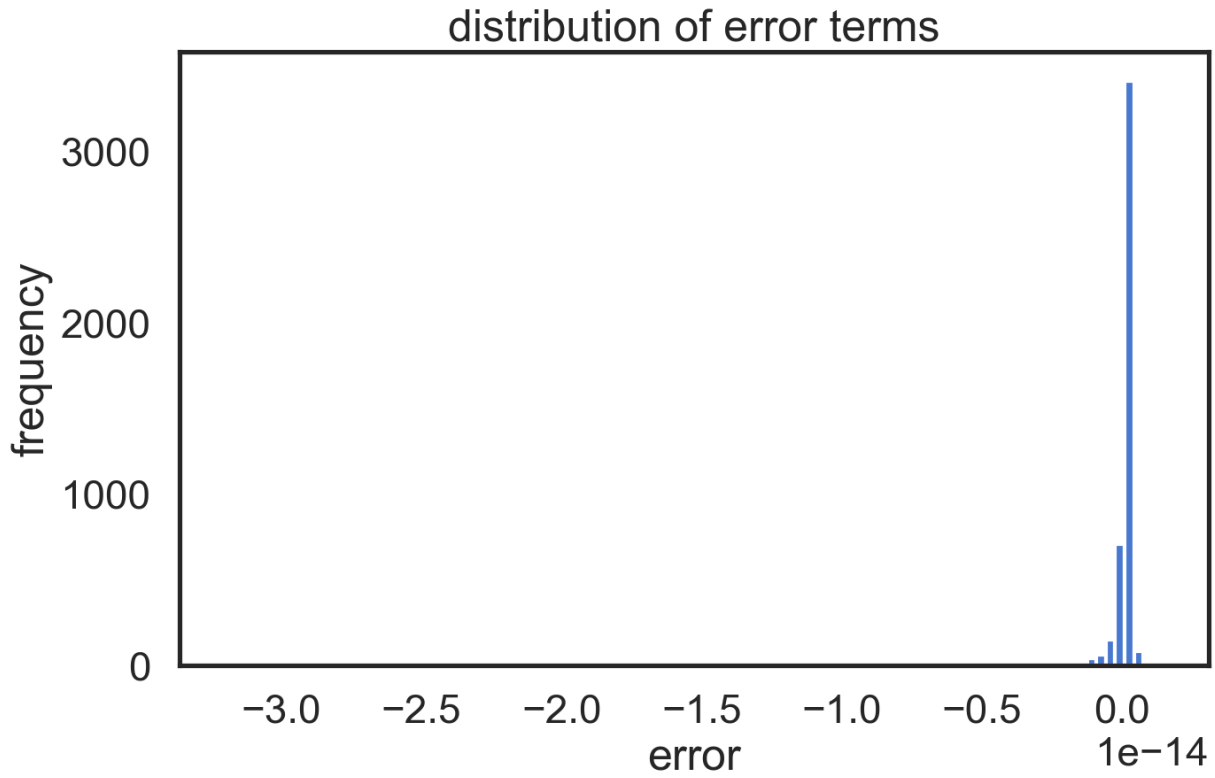
Out[536]:

	fitted values	predicted values	residuals
10811	0.171389	0.171389	1.665335e-16
16319	0.559444	0.559444	-9.992007e-16
13951	0.227222	0.227222	1.665335e-16
1693	0.170833	0.170833	1.942890e-16
1532	0.396111	0.396111	5.551115e-17

Checking Distribution of Residuals

In [537]:

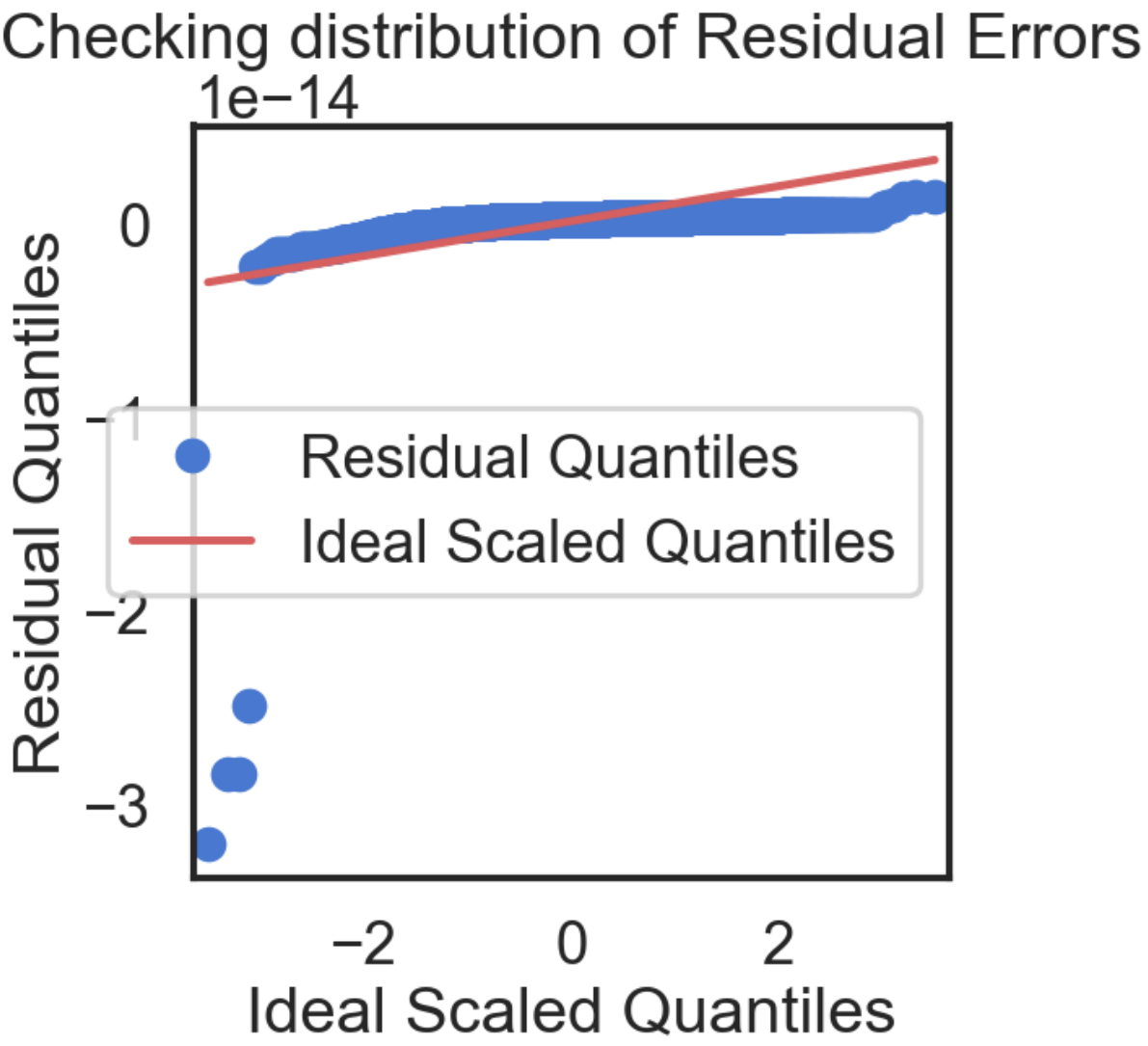
```
plt.figure(figsize=(10,6),dpi=150,facecolor="w",edgecolor="b")
plt.hist(residuals.residuals,bins=100)
plt.xlabel("error")
plt.ylabel("frequency")
plt.title("distribution of error terms")
plt.show()
```



QQ-Plot

In [538]:

```
# importing the QQ-plot from the from the statsmodels
from statsmodels.graphics.gofplots import qqplot

## Plotting the QQ plot
fig, ax = plt.subplots(figsize=(5,5) , dpi = 120)
qqplot(residuals.residuals, line = 's' , ax = ax)
plt.ylabel('Residual Quantiles')
plt.xlabel('Ideal Scaled Quantiles')
plt.legend(["Residual Quantiles","Ideal Scaled Quantiles"])
plt.title('Checking distribution of Residual Errors')
plt.show()
```



In [539]:

```
# Importing Variance_inflation_Factor funtion from the Statsmodels
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Calculating VIF for every column (only works for the not Catagorical)
VIF = pd.Series([variance_inflation_factor(data.values, i) for i in range(data.shape[1])], index =data.colu
VIF
```

Out[539]:

```
passenger_count                  1.002172
pickup_longitude                 1.453814
pickup_latitude                  1.374584
dropoff_longitude                1.229682
dropoff_latitude                 1.310605
store_and_fwd_flag               1.001919
trip_duration                         inf
pickup_day                       1.004197
pickup_hour                      8.198811
pickup_weekday                        inf
dropoff_weekday                 76.498533
total_time                            inf
dropoff_hour                     8.221951
duration_difference                   inf
log_trip_duration                2.033893
distance_km                      2.666365
pickup_days_pickup_Friday             inf
pickup_days_pickup_Monday             inf
pickup_days_pickup_Saturday           inf
pickup_days_pickup_Sunday             inf
pickup_days_pickup_Thursday           inf
pickup_days_pickup_Tuesday            inf
pickup_days_pickup_Wednesday          inf
dropoff_day_dropoff_Friday            inf
dropoff_day_dropoff_Monday            inf
dropoff_day_dropoff_Saturday          inf
dropoff_day_dropoff_Sunday            inf
dropoff_day_dropoff_Thursday          inf
dropoff_day_dropoff_Tuesday           inf
dropoff_day_dropoff_Wednesday         inf
dtype: float64
```

Model Interpretability

In [540]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# Create an instance of StandardScaler
scaler = StandardScaler()

# Normalize the input features
train_x_scaled = scaler.fit_transform(train_x)
test_x_scaled = scaler.transform(test_x)

# Create an instance of Linear Regression
lr = LinearRegression()

# Fit the model
lr.fit(train_x_scaled, train_y)
```

Out[540]:

▾ LinearRegression

LinearRegression()

In [541]:

```
# Predicting over the Train Set and calculating error
train_predict = lr.predict(train_x)
k = MAE(train_predict, train_y)
print('Training Mean Absolute Error', k )
```

```
Training Mean Absolute Error 777.3798336696274
```

In [542]:

```
# Predicting over the Test Set and calculating error
test_predict = lr.predict(test_x)
k = MAE(test_predict, test_y)


from sklearn.metrics import mean_squared_error, r2_score

# Calculate RMSE
rmse = mean_squared_error(test_y, test_predict, squared=False)
print('RMSE:', rmse)



print('Test Mean Absolute Error    ', k )
```

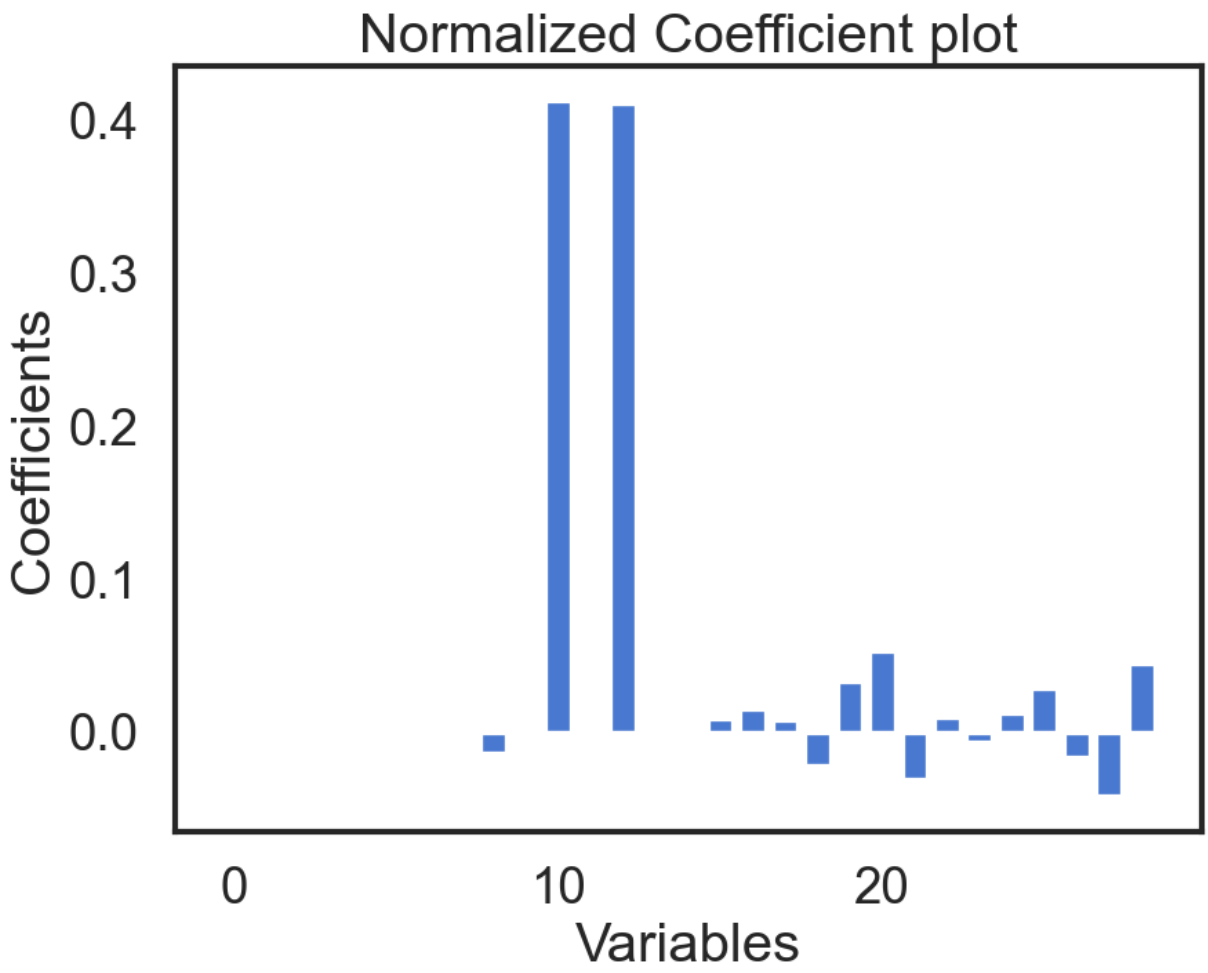
```
RMSE: 2275.087625738153
Test Mean Absolute Error     741.6494813209888
```

In [543]:

```
plt.figure(figsize=(8, 6), dpi=120, facecolor='w', edgecolor='b')
x = range(len(train_x.columns))
y = lr.coef_
plt.bar( x, y )
plt.xlabel( "Variables")
plt.ylabel('Coefficients')
plt.title('Normalized Coefficient plot')
```

Out[543]:

```
Text(0.5, 1.0, 'Normalized Coefficient plot')
```



Conclusions

1. We have done regression on 4 different set of features. Unfortunately, no any hypothesis gave us very good results.
2. Our Linear regression model will fall under the catagory of either under fit or over fit.
3. On computing the coefficients we observed that there are some negative values as well
4. On plotting the qqplot we see that the residual quantile line doesn't fit over all ideal scaled quantiles

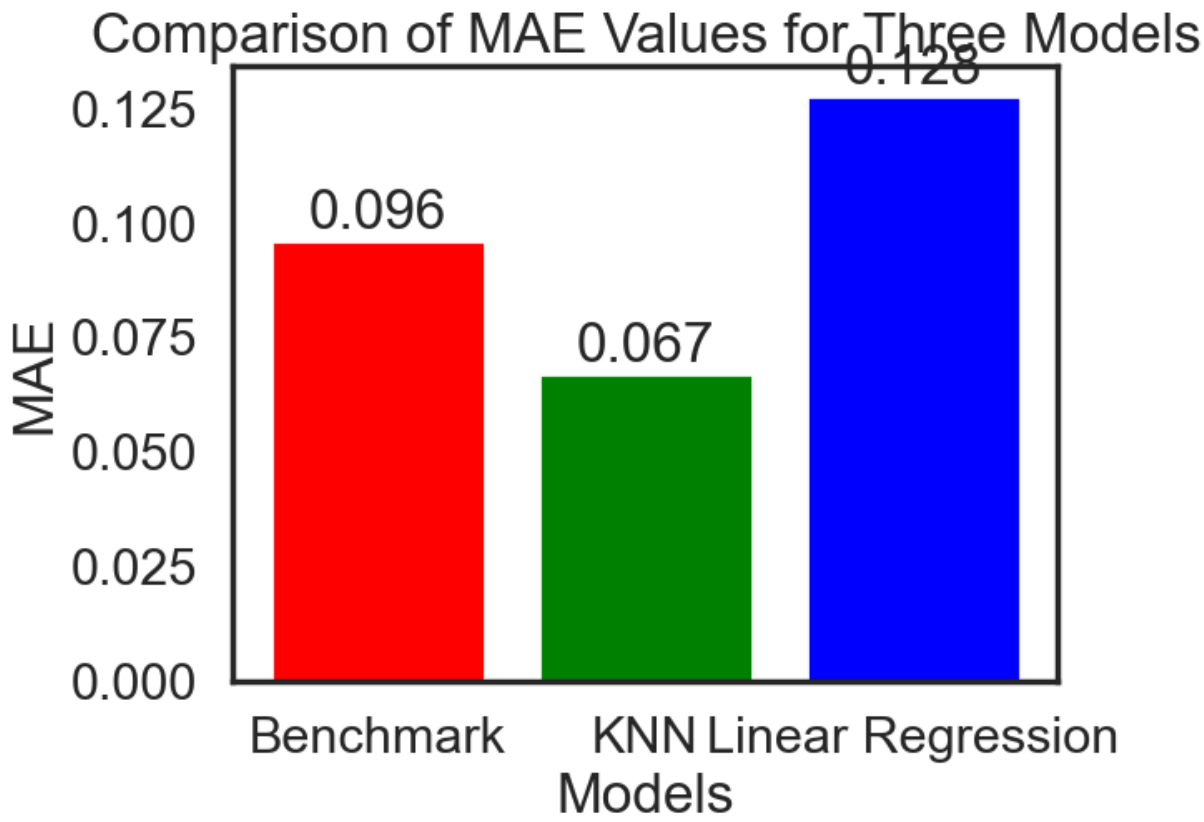
Comparison of best MAE for three models

Note: This is not a best way to comparing the models as we are comparing only MAE.

- Other parameters are more important determining if the model is good or not like, R2, RMSE, MSE.
- This is just a general comparison of three models because our model didn't perform well in linear regression.

In [544]:

```
import matplotlib.pyplot as plt

mae_benchmark = 0.0961111111111111
mae_knn =  0.06723621399176954
mae_linear = 0.12785732055218849

models = ['Benchmark', 'KNN', 'Linear Regression']
mae_values = [mae_benchmark, mae_knn, mae_linear]

plt.bar(models, mae_values, color=['red', 'green', 'blue'])
plt.title('Comparison of MAE Values for Three Models')
plt.xlabel('Models')
plt.ylabel('MAE')

for i in range(len(models)):
    plt.annotate(str(round(mae_values[i], 3)), xy=(models[i], mae_values[i]), ha='center', va='bottom')

plt.show()
```



In [ ]:

In [ ]: