Project Name : **CAB FARE PREDICTION**

Duration : August 11 - 16, 2019

Author : Aashit Singh

Index

Part 1: **Introduction**

1.1 Problem Statement

Aim: To predict cab fare

Given: Pick-up date and time, pick-up and drop coordinates, number of passengers

Dataset: A dataset of 16000 observations of a pilot program conducted in in New York area is given.

Problem Statement: Predict the fare_amount of renting a cab, given pickup & dropoff coordinates & number of passengers

Model to be developed: Because we are required to predict a **continuous value**, we will be building a **regression model**.

1.2 Data Exploration

```
# Any results you write to the current directory are saved as output.
# train_file_path = '/kaggle/input/train_cab.csv'
# test_file_path = '/kaggle/input/test.csv'
train_file_path = 'train_cab.csv'
test_file_path = 'test.csv'
training_data_full = pd.read_csv(filepath_or_buffer = train_file_path, header = 0, encoding='utf-8')
testing_data_full = pd.read_csv(filepath_or_buffer = test_file_path, header = 0, encoding='utf-8')
```

In [2]:
```
training_data_full
```

Out[2]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.0
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.0
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.0
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.0
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.0
5	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	40.758233	1.0
6	7.5	2012-11-20 20:35:00 UTC	-73.980002	40.751662	-73.973802	40.764842	1.0
7	16.5	2012-01-04 17:22:00 UTC	-73.951300	40.774138	-73.990095	40.751048	1.0
8	NaN	2012-12-03 13:10:00 UTC	-74.006462	40.726713	-73.993078	40.731628	1.0
9	8.9	2009-09-02 01:11:00 UTC	-73.980658	40.733873	-73.991540	40.758138	2.0
10	5.3	2012-04-08 07:30:50 UTC	-73.996335	40.737142	-73.980721	40.733559	1.0

Variable Name	Nature of Variable	Type of Data	Variable Datatype
fare_amount	Dependent - Target	Numerical - Continuous	Float
pickup_datetime	Independent - Feature	Numerical - Continuous	Datetime - UTC*
pickup_longitude	Independent - Feature	Numerical - Continuous	Float
pickup_latitude	Independent - Feature	Numerical - Continuous	Float
dropoff_longitude	Independent - Feature	Numerical - Continuous	Float
dropoff_latitude	Independent - Feature	Numerical - Continuous	Float
passenger_count	Independent - Feature	Numerical - Discrete	Integer

*: UTC - Universal Coordinated Time

1.3 Overview of Procedures & Steps followed

1.3.1 Initial Data Exploration

- On going through the **train_cab.csv** file it was found that there are certain errors that are man-made or data entry errors.
- For example some values in fare_amount were not numeric.
- Some entries in a datetime column were not in the specified format.
- Some latitudes and longitudes were missing.
- Passenger count was not strictly discrete.

1.3.2 Basic Data Preprocessing

- The above errors were corrected in basic preprocessing.
- Rows not having numeric fare amount were dropped.
- Rows not in specified datetime format were dropped
- Rows with missing values were dropped
- Rows with non integer passenger count was dropped

```
def preprocess_data(pandas_data_frame):
    """
    Function to perform all the above preprocessing functions on a pandas_data_frame
    """
    pandas_data_frame = column_to_numeric_type(pandas_data_frame, 'fare_amount')
    pandas_data_frame = strip_pickup_datetime(pandas_data_frame)
    pandas_data_frame = drop_rows_with_0s_NAs(pandas_data_frame, pandas_data_frame.columns)
    # Changing the datatype of passenger_count to integer from float
    pandas_data_frame['passenger_count'] = pandas_data_frame['passenger_count'].astype('int')
    # Resetting the index numbers
    pandas_data_frame = pandas_data_frame.reset_index(drop=True)
    return pandas_data_frame
```

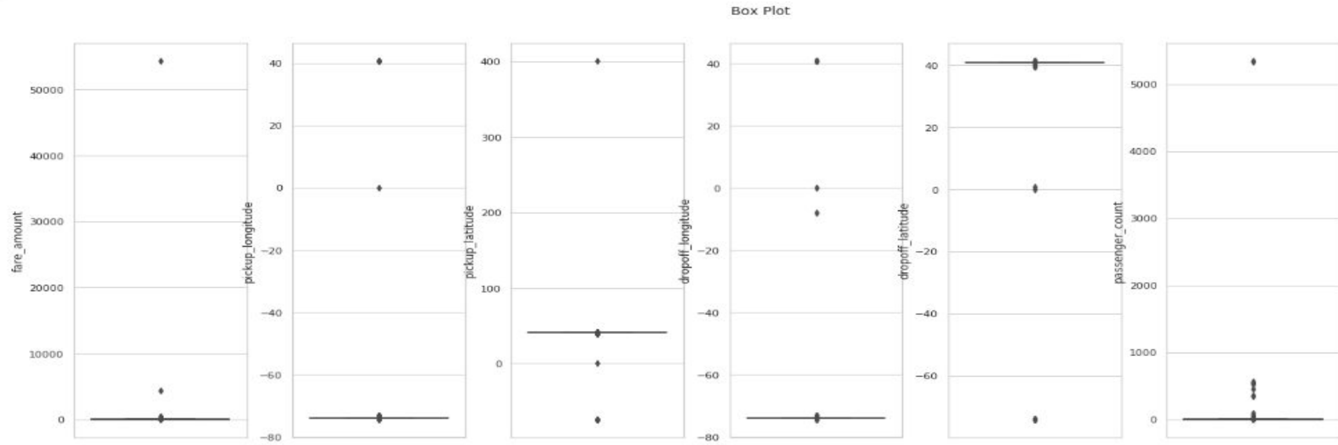
```
In [5]: clean_training_data = preprocess_data(training_data_full)
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:64: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versu
s-copy

Number of Rows removed = 1

Number of Rows removed = 1

Number of Rows removed = 459

====================== Shape ======================
(15606, 7)

====================== Missing Values Info ======================
           variables  n_missing_values  missing_percentage
0        fare_amount                 0                 0.0
1    pickup_datetime                 0                 0.0
2   pickup_longitude                 0                 0.0
3    pickup_latitude                 0                 0.0
4  dropoff_longitude                 0                 0.0
5   dropoff_latitude                 0                 0.0
6    passenger_count                 0                 0.0
```

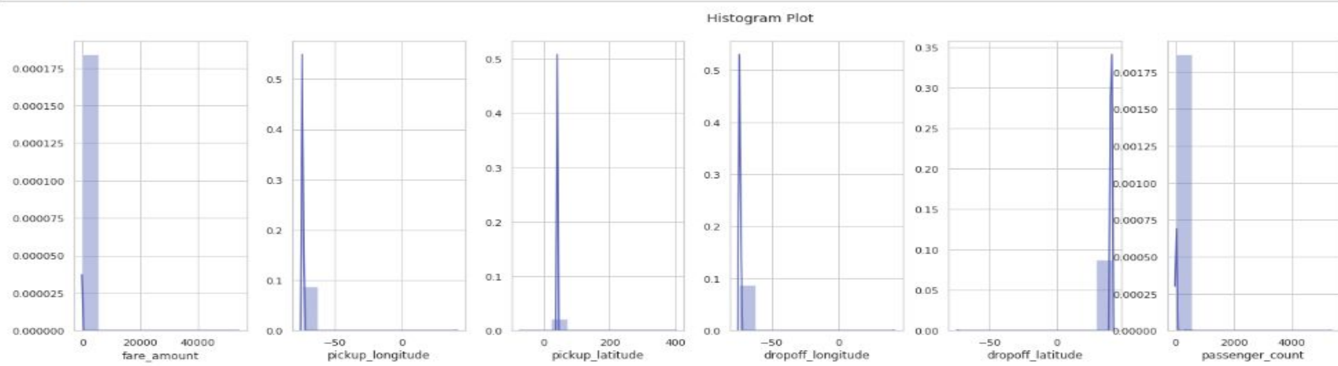
1.3.3 Dataset Visualisation

- The remaining dataset was visualised by making a boxplot & histogram plot

- Shown below is a screenshot of the boxplot & histogram. It doesn't make any sense because the data has a lot of noise



In [8]:
```
bins_list = [10,10,10,10,10,10]
hist_plot(y_axis_cols_list=y_axis_cols_list, x_axis_data_list=x_axis_data_list, bins_list=bins_list, subplot_nrows=
1, subplot_ncols=6)
```



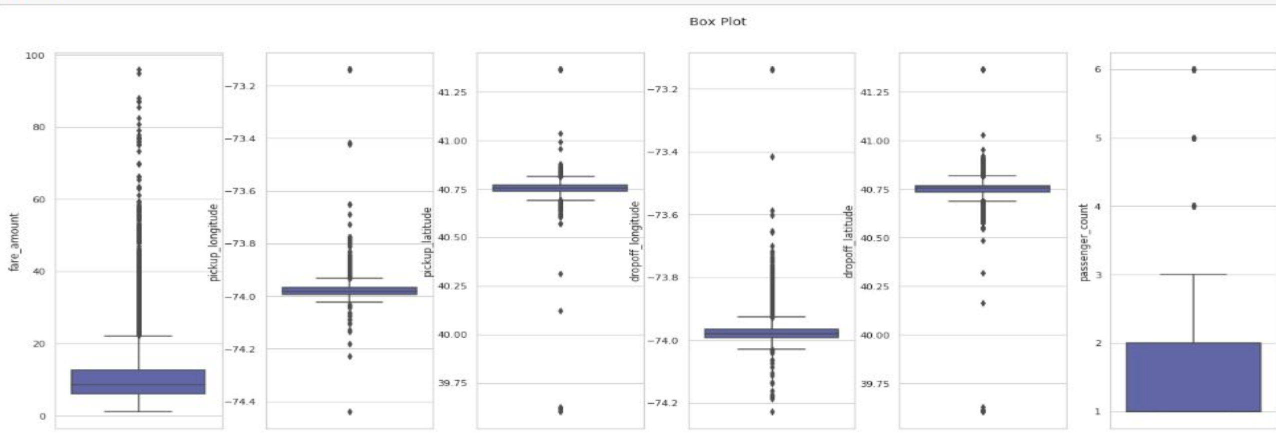
1.3.4 Removing Noisy Data & visualising the data

Noisy data is data that doesn't make any sense being in the place where it is. For example:

- fare_amount: some observations run into 1000s, which is not practical
- passenger_count: passengers more than 6 in a cab is not practical
- Latitude & longitude = 0 is not practical
- latitude & longitude interchanged

In [10]:
```
clean_training_data = preprocess_data_2(clean_training_data)
```
```
Number of Rows removed = 42
```

In [11]:
```
numerical_data_cols_list = ['fare_amount', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_lat
itude', 'passenger_count']
x_axis_data_list = [clean_training_data,clean_training_data,clean_training_data,clean_training_data,clean_training_
data,clean_training_data]
box_plot(y_axis_cols_list=numerical_data_cols_list, x_axis_data_list=x_axis_data_list, subplot_nrows=1, subplot_nco
ls=6)
```

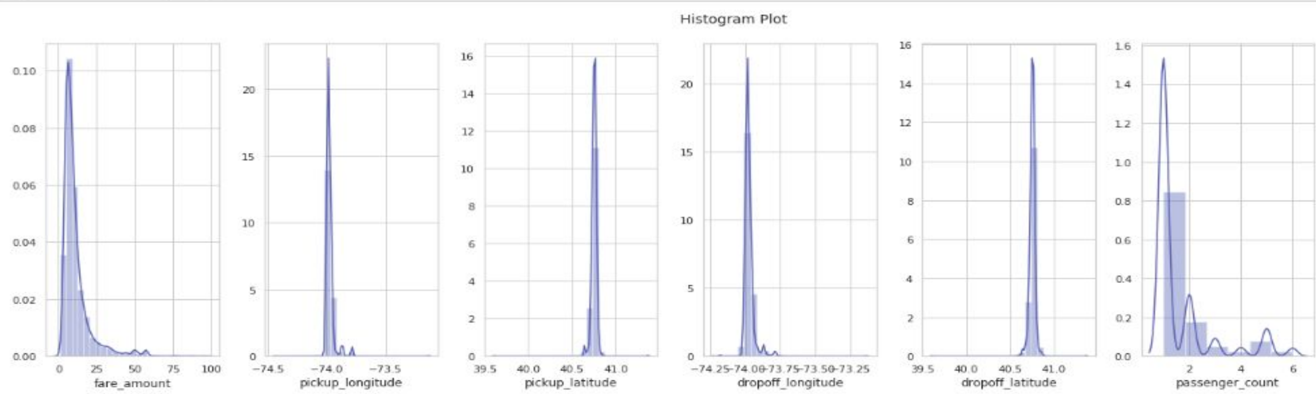


All such data was removed. After which the following graphs were plotted. And now we can make some sense out of the data. To remove the noisy data, manually lower & upper ranges for each feature was added into a dataframe.

- 'fare_amount' : [1, 100],
- 'pickup_longitude' : [-74.8, -72.8],
- 'pickup_latitude' : [39.45, 41.45],
- 'dropoff_longitude' : [-74.8, -72.8],
- 'dropoff_latitude' : [39.45, 41.45],
- 'passenger_count' : [1, 6],
- 'amnt_per_km' : [1, 20],
- 'amnt_per_hr' : [20, 250],
- 'onroad_distance' : [50, 100000000000000],
- 'onroad_time' : [60, 1000000],
- 'aerial_distance' : [10, 100000000000000]

This includes some new features added over the course of time while programming.

```
In [12]: bins_list = [25,25,25,25,25,6]
hist_plot(y_axis_cols_list=numerical_data_cols_list, x_axis_data_list=x_axis_data_list, bins_list=bins_list, subplo
t_nrows=1, subplot_ncols=6)
```



1.4 UNDERSTANDING THE BUSINESS:

After exploring the data, it is important to UNDERSTAND the BUSINESS before we start building models to predict. After doing extensive research on the cab rental industry and the factors that affect the prices, based on the data given and considering the limitations I had, the following new features were engineered to develop a good model.

Part 2: **Feature Engineering**

Feature Engineering is creating new features out of the features already present in the dataset.
I will be creating the following new features in the methods given.

2.1 Making New Features

1. Using the *pickup_datetime* to make:
   year
   month
   day of the week
   hour of the day
2. Making a tuple of origin & destination coordinates using the 4 coordinates column
3. Using the *pickup_latitude, pickup_longitude, dropoff_latitude, dropoff_longitude* to make:
   aerial_distance - using *Vincenty Formula* from **geopy** library
   onroad_distance - using GoogleMaps *Distance Matrix API* from the **googlemaps** library
   onroad_time - using GoogleMaps *Distance Matrix API* from the **googlemaps** library

How did we use GoogleMaps Distance Matrix API?

1. Make an account on GoogleCloud Platform
2. Obtain an API Key
3. Setup client connection to the server using API key & googlemaps library

```
!pip install -U googlemaps
!pip install geopy
import googlemaps, time
import geopy.distance
gkey = 'AIzaSyAC5eKZ1qd_mpC9zKnnm9JNYlMjZkNxtJs'
gmaps = googlemaps.Client(key=gkey)
coords_columns = ['pickup_latitude', 'pickup_longitude', 'dropoff_longitude', 'dropoff_latitude']
```

```
Requirement already up-to-date: googlemaps in /home/shitbot009/anaconda3/lib/python3.7/site-packages (3.0.2)
Requirement already satisfied, skipping upgrade: requests<3.0,>=2.11.1 in /home/shitbot009/anaconda3/lib/python3.7/
site-packages (from googlemaps) (2.21.0)
Requirement already satisfied, skipping upgrade: urllib3<1.25,>=1.21.1 in /home/shitbot009/anaconda3/lib/python3.7/
site-packages (from requests<3.0,>=2.11.1->googlemaps) (1.24.1)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /home/shitbot009/anaconda3/lib/python3.7/sit
e-packages (from requests<3.0,>=2.11.1->googlemaps) (2019.6.16)
Requirement already satisfied, skipping upgrade: chardet<3.1.0,>=3.0.2 in /home/shitbot009/anaconda3/lib/python3.7/
site-packages (from requests<3.0,>=2.11.1->googlemaps) (3.0.4)
Requirement already satisfied, skipping upgrade: idna<2.9,>=2.5 in /home/shitbot009/anaconda3/lib/python3.7/site-pa
ckages (from requests<3.0,>=2.11.1->googlemaps) (2.8)
Requirement already satisfied: geopy in /home/shitbot009/anaconda3/lib/python3.7/site-packages (1.20.0)
Requirement already satisfied: geographiclib<2,>=1.49 in /home/shitbot009/anaconda3/lib/python3.7/site-packages (fr
om geopy) (1.49)
```

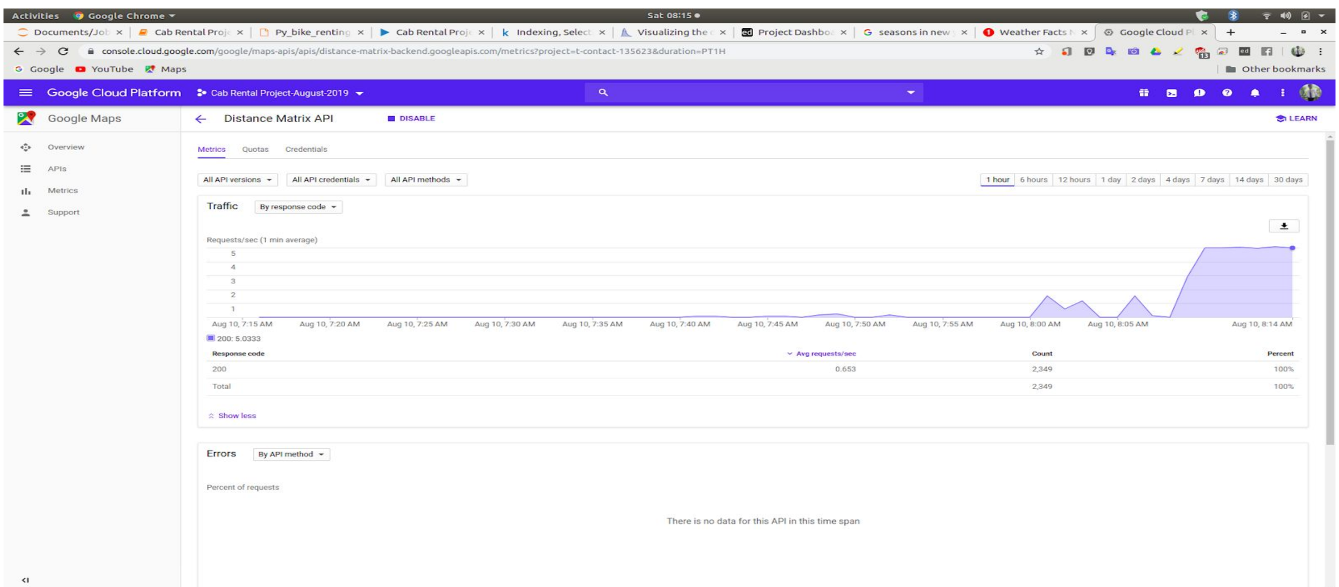
4. Write a function to fetch the JSON Response by sending the origin & destination coordinates & **store the JSON response in a data frame to be used later**.

```
def find_onroad_distance_2(coord):
    """
    Finds the ON-ROAD DISTANCE between the origin coordinates & destination coordinates
    1. Uses Google Maps API
    2. A Google Cloud Platform API Key is needed
    3. Install googlemaps library - !pip install -U googlemaps
    4. Result of the API is in JSON Format. Extracting data from it using conventional Python tools
    5. Taking distance value in meter
    6. Taking duration value in seconds
    7. Doesn't use loops, instead works with apply() function and returns a tuple, so FASTER
    """
    origin = coord[0]
    dest = coord[1]
    #time = time.mktime(year=2019,month=8,day=)
    distance_results = gmaps.distance_matrix(origin, dest, mode='driving')
    return distance_results
```

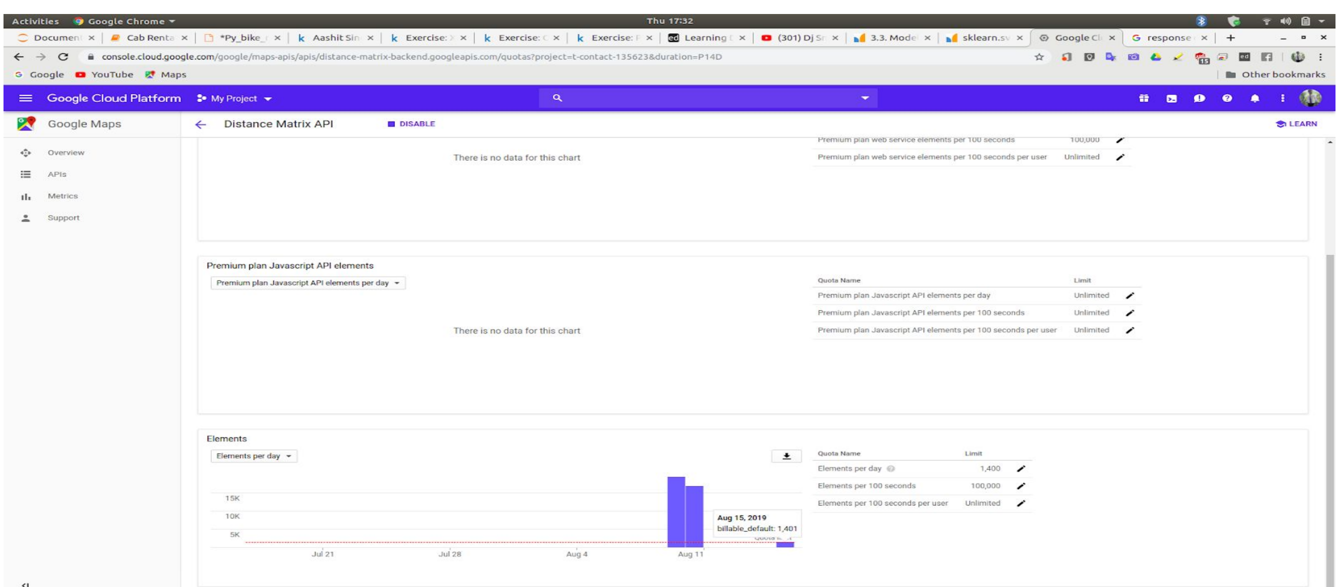
5. Stripping the JSON response to get the onroad_distance in **meter** & onroad_time in **seconds**

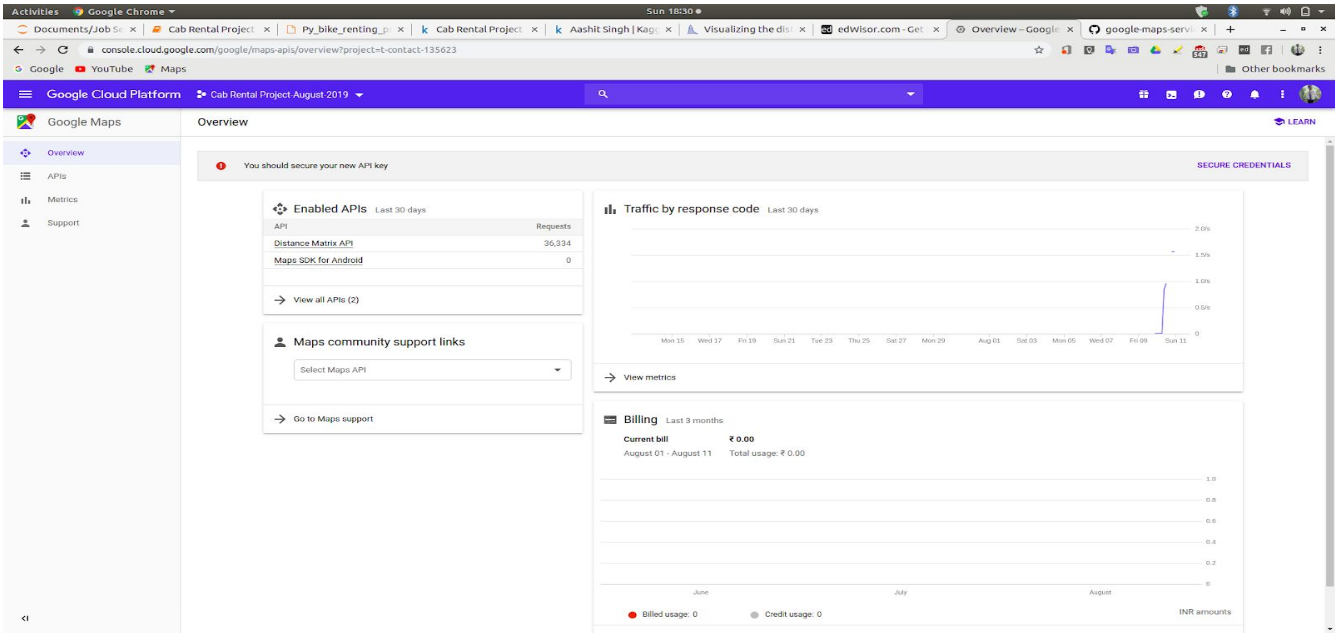
```
In [18]: def extract_feature(json_response, feature_name):
             """
             Function to parse JSON & return value distance & duration
             """
             if json_response['rows'][0]['elements'][0]['status'] == 'OK':
                 return json_response['rows'][0]['elements'][0][feature_name]['value']
             else:
                 return 0
```

NOTE: The connection to the GoogleMapsAPI was not stable for more than 20 minutes at a time & only 5 queries were being answered in 1 second, so getting 15600 (approx.) queries in 1 session was tough. So the data was divided into 4 dataframes of 4000 rows each and then they were individually queried. The resulting response was stored in a dataframe & printed to a file so as to not repeat the whole procedure again. The APIs had a quota limit per day & to avoid exceeding that, the data in printed file is used & attached to this project as well.
Below are some screenshots of my GoogleCloud Console



Shows the speed of query response(above) & daily quota limit (below)





Shows total number of queries answered

2.2 Making Categorical Features

Using the new features made out of datetime, I have made new categorical features that have an impact on the fare_amount, like

- weekday or not - Monday (0) to Friday(4) have this value as 1, and Saturday (5) & Sunday (6) have this value as 0.
- winter month or not - December to February has this value as 1, and other months as 0
- night hour or not - 0400 to 0659 has this value as 1 & other hours as 0

```python
def set_winter_month(mnth):
    if (mnth in range(3, 12)):
        return 0 # NOT WINTER MONTH
    else:
        return 1 # WINTER MONTH

def set_weekday(wday):
    if (wday in range(0,5)):
        return 1 # IS WEEKDAY
    else:
        return 0 # NOT WEEKDAY

def set_late_night(hr): # NYC time is UTC -4hr, so latenight hours are 12AM - 3AM. UTC - 0400 to 0659
    if (hr in range(4, 7)):
        return 1
    else:
        return 0
```

2.3 Making Numerical Features

Using the new features onroad_distance & onroad_time & fare_amount, we are constructing 2 new numerical features that play an important role in the accuracy of the regression model:

- **amnt_per_km**: finding the amount paid per kilometer of the journey. This can be the amount per mile or meter, it all comes down to the same thing. Relation of this feature to the target feature as will be seen later through pair plots or correlation & heatmap plot is **inverse** to the fare_amount. That means, as the fare_amount increases, amnt_per_km keeps reducing.

- **amnt_per_hr**: finding the amount paid per hour of the journey. This feature was also made after extensively studying up on the industry and business. Cabs have a per hour charge apart from the per distance-unit charge.

CONCERN: One concern about having these 2 features into the model is that while predicting the fare amount in real time, we will not have the amount beforehand to calculate this. It is a genuine concern. My response to this concern is - that is the reason we are doing it on a training set. So that an estimate of an average amnt_per_km and amnt_per_hr can be found out, which can then be engineered into the final model without the fare amount being present beforehand.

2.4 Removing NOISY data

Once all the new features have been engineered in, we are removing some noisy data that makes the visualisation much intuitive and better. Noisy data can be, observations where onroad_distance is less than 100m. That kind of observation is obviously doing no help in training the model.

In [35]:
```
def drop_noisy_data(data_value):
    """
    Function to drop noisy data value
    1. If on-road distance < 50, drop that column. Not good data for our training
    """
    if data_value < 100:
        return False
    else:
        return True
```

In [36]:
```
initial_n_rows = clean_training_data.shape[0]
clean_training_data = clean_training_data[clean_training_data['onroad_distance'].apply(lambda x: drop_noisy_data(x
))]
after_n_rows = clean_training_data.shape[0]
n_rows_removed = initial_n_rows - after_n_rows
print("Number of Rows removed = " + str(n_rows_removed) + "\n")
```

```
Number of Rows removed = 221
```

2.5 Checking summary of the data to see if everything is fine

Checking on the summary and missing_values information to see if all the above processes are fine or not.

In [34]:
```
print_missing_values_info(clean_training_data)
```

```
======================= Shape =======================
(15343, 12)

======================= Missing Values Info =======================
          variables  n_missing_values  missing_percentage
0       fare_amount                 0                 0.0
1   passenger_count                 0                 0.0
2   aerial_distance                 0                 0.0
3              wday                 0                 0.0
4              mnth                 0                 0.0
5                hr                 0                 0.0
6              year                 0                 0.0
7      winter_month                 0                 0.0
8           weekday                 0                 0.0
9           nght_hr                 0                 0.0
10  onroad_distance                 0                 0.0
11      onroad_time                 0                 0.0
```

```
======================= Info =======================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15343 entries, 0 to 15342
Data columns (total 12 columns):
fare_amount        15343 non-null float64
passenger_count    15343 non-null int64
aerial_distance    15343 non-null float64
wday               15343 non-null int64
mnth               15343 non-null int64
hr                 15343 non-null int64
year               15343 non-null int64
winter_month       15343 non-null int64
weekday            15343 non-null int64
nght_hr            15343 non-null int64
onroad_distance    15343 non-null int64
onroad_time        15343 non-null int64
dtypes: float64(2), int64(10)
memory usage: 1.4 MB
None

======================= Describe =======================
        fare_amount  passenger_count  aerial_distance          wday  \
count  15343.000000     15343.000000     15343.000000  15343.000000
mean      11.285006         1.653001      3422.861019      3.036499
std        9.276900         1.268600      4134.697248      1.970377
min        1.140000         1.000000         0.279186      0.000000
25%        6.000000         1.000000      1289.366948      1.000000
50%        8.500000         1.000000      2201.708227      3.000000
75%       12.500000         2.000000      3944.057927      5.000000
max       95.000000         6.000000    101166.019546      6.000000

               mnth            hr          year  winter_month       weekday  \
count  15343.000000  15343.000000  15343.000000  15343.000000  15343.000000
mean       6.276478     13.501466   2011.738448      0.252493      0.711204
std        3.449821      6.507207      1.871637      0.434457      0.453218
min        1.000000      0.000000   2009.000000      0.000000      0.000000
25%        3.000000      9.000000   2010.000000      0.000000      0.000000
50%        6.000000     14.000000   2012.000000      0.000000      1.000000
75%        9.000000     19.000000   2013.000000      1.000000      1.000000
max       12.000000     23.000000   2015.000000      1.000000      1.000000

            nght_hr  onroad_distance   onroad_time
count  15343.000000     15343.000000  15343.000000
mean       0.044124      4831.676334    819.114319
std        0.205378      5933.900667    475.810536
min        0.000000       110.000000     34.000000
25%        0.000000      1756.000000    470.000000
50%        0.000000      2907.000000    718.000000
75%        0.000000      5272.500000   1072.500000
max        1.000000    124966.000000   5975.000000
```

Part 3: **Data Visualization & Feature Selection**

Data Visualization is an important part in understanding what kind of data we are working with. I have used the following plots to visualize data to the best of my abilities.

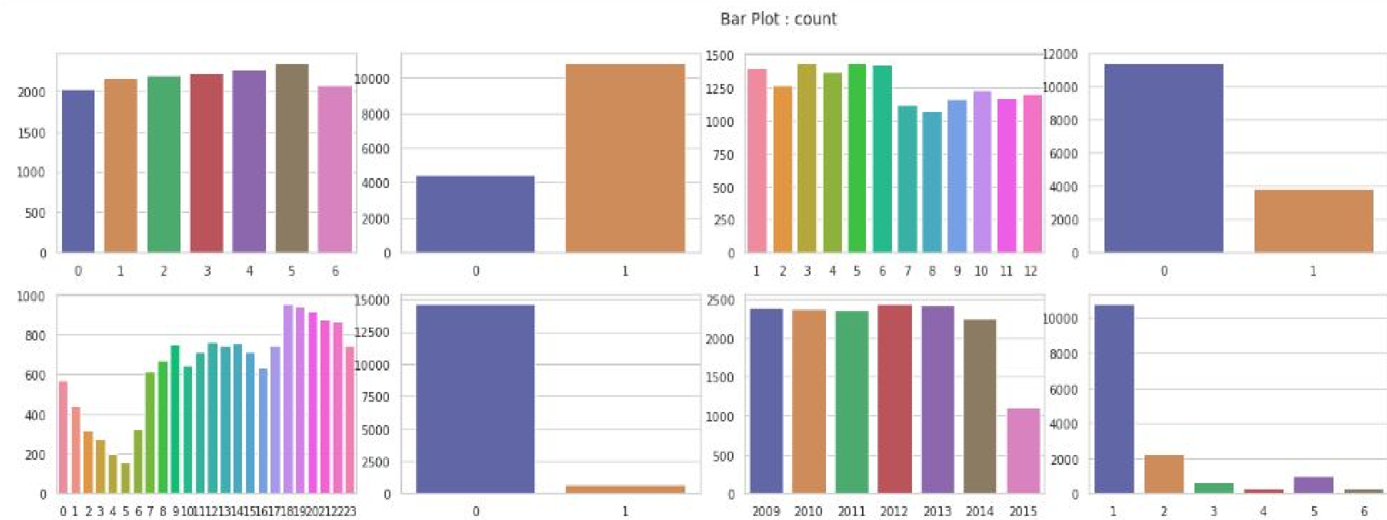
Library used - **seaborn**

- **Bar Graphs** - these show the relation between a categorical variable (plotted on X-axis) and a continuous variable (plotted on Y-axis)
- **Histogram** - these show the distribution trend of a continuous variable over some intervals
- **Pairplot** - A grid of scatter plots between all the features in data
- **Heatmap** - Correlation between 2 numerical variables
- **Box Plot** - Shows data distribution with outlier range

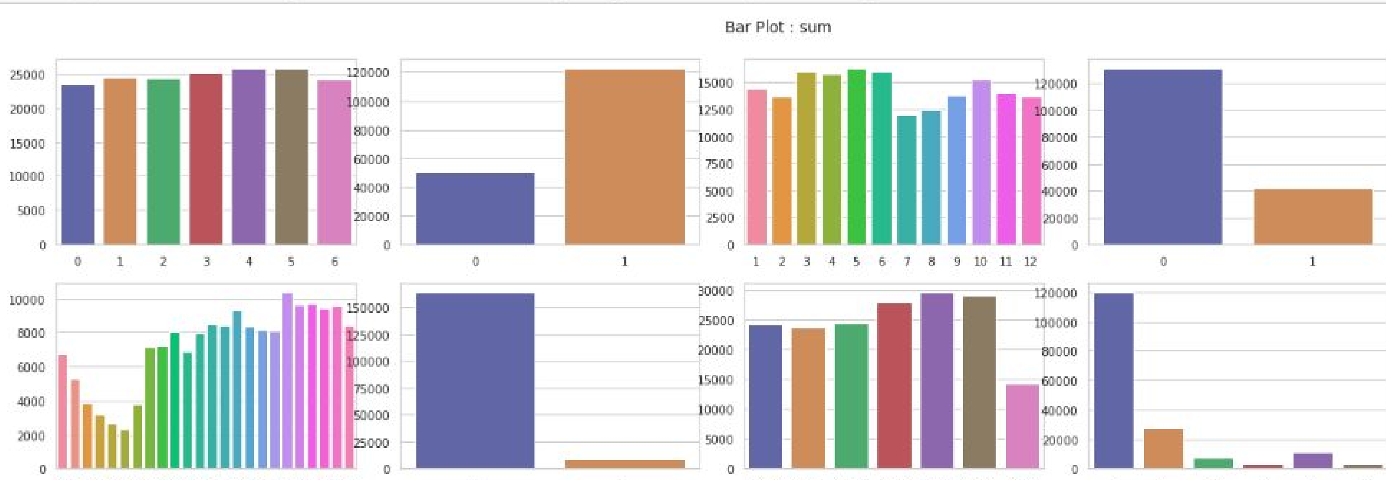
1. Bar Graph **number of cab rentals** & **sum of fare_amount of ALL the cab rentals**

i) Day of the week	ii) Weekday or not
iii) Month of the year	iv) Winter month or not
v) Hour of the day	vi) Night hour or not
vii) Year	viii) Passenger count

In [41]:
```
bar_plot(['wday','weekday', 'mnth', 'winter_month', 'hr', 'nght_hr', 'year', 'passenger_count'], subplot_nrows=2, s
ubplot_ncols=4, plotting_method='count', pandas_data_frame=clean_training_data)
```



In [42]:
```
bar_plot(['wday','weekday', 'mnth', 'winter_month', 'hr', 'nght_hr', 'year', 'passenger_count'], subplot_nrows=2, s
ubplot_ncols=4, plotting_method='sum', pandas_data_frame=clean_training_data)
```



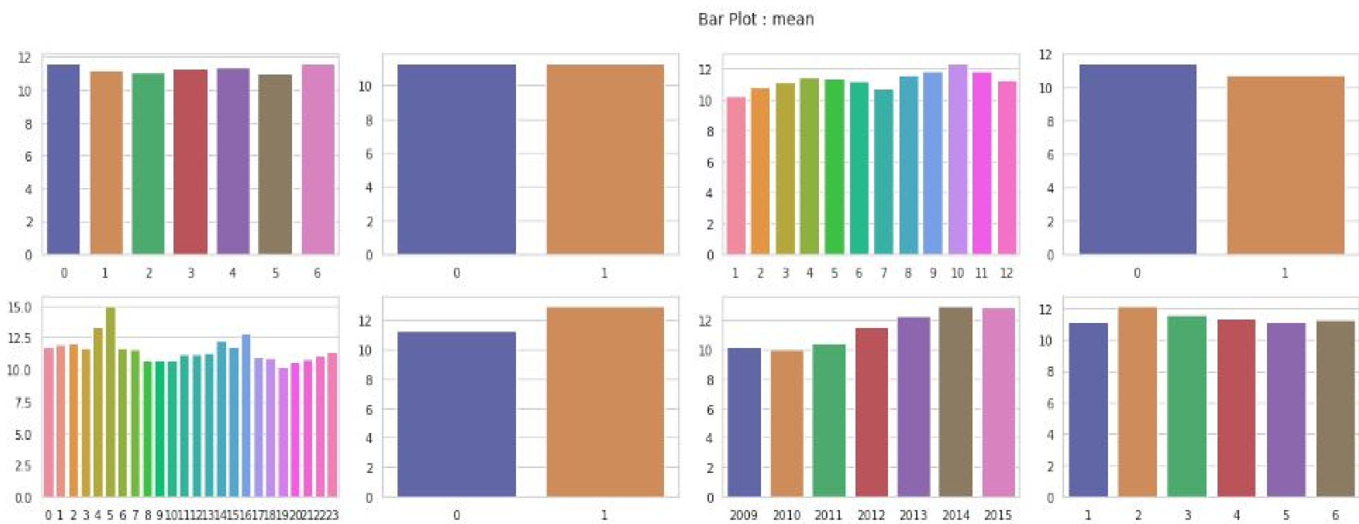
Insights:

- ➔ Between 1AM and 6AM, number of rentals drastically drop. Night hour bookings are low.
- ➔ Over 90% of cab bookings are for **1 passenger**.

2. Bar Graph **mean of fare_amount of ALL the cab rentals**

i) Day of the week	ii) Weekday or not
iii) Month of the year	iv) Winter month or not
v) Hour of the day	vi) Night hour or not
vii) Year	viii) Passenger count

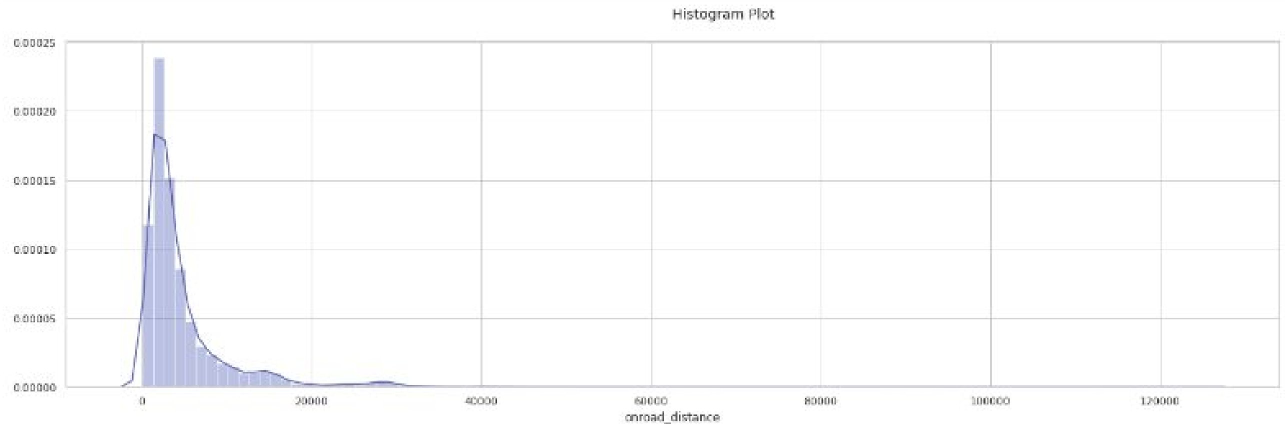
In [43]:
```
bar_plot(['wday','weekday', 'mnth', 'winter_month', 'hr', 'nght_hr', 'year', 'passenger_count'], subplot_nrows=2, s
ubplot_ncols=4, plotting_method='mean', pandas_data_frame=clean_training_data)
```



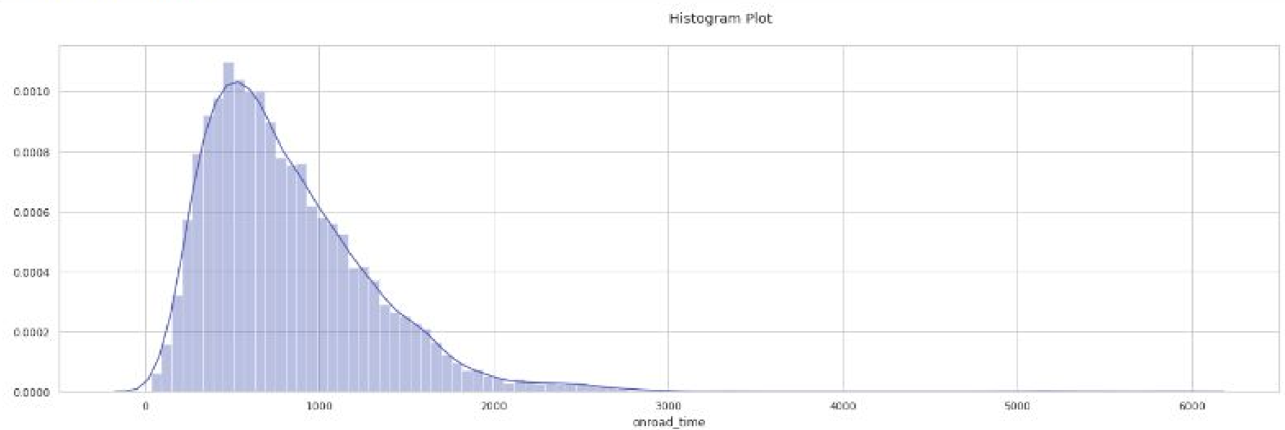
Insights:

- ➔ Between 5AM and 7AM, mean of fare amount of rentals drastically rise. Night hour bookings are low but expensive
- ➔ Over the years the average booking amount is rising consistently from 2009 to 2015.
- ➔ Average rental amount stands at around $13 in 2015, up from $10 in 2009. A 30% growth in per-capita revenue

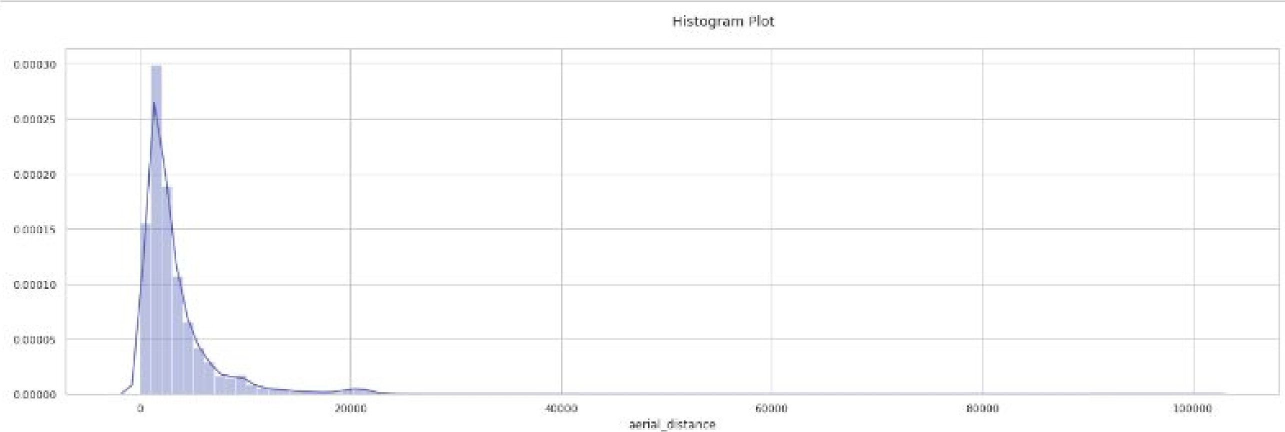
3. Histogram plot to see distribution of **onroad_distance, onroad_time & aerial_distance**



In [47]:
```
hist_plot(y_axis_cols_list=['onroad_time'], x_axis_data_list=[clean_training_data], subplot_nrows=1, subplot_ncols=1
, bins_list=[100])
```



In [48]:
```
hist_plot(y_axis_cols_list=['aerial_distance'], x_axis_data_list=[clean_training_data], subplot_nrows=1, subplot_nco
ls=1, bins_list=[100])
```

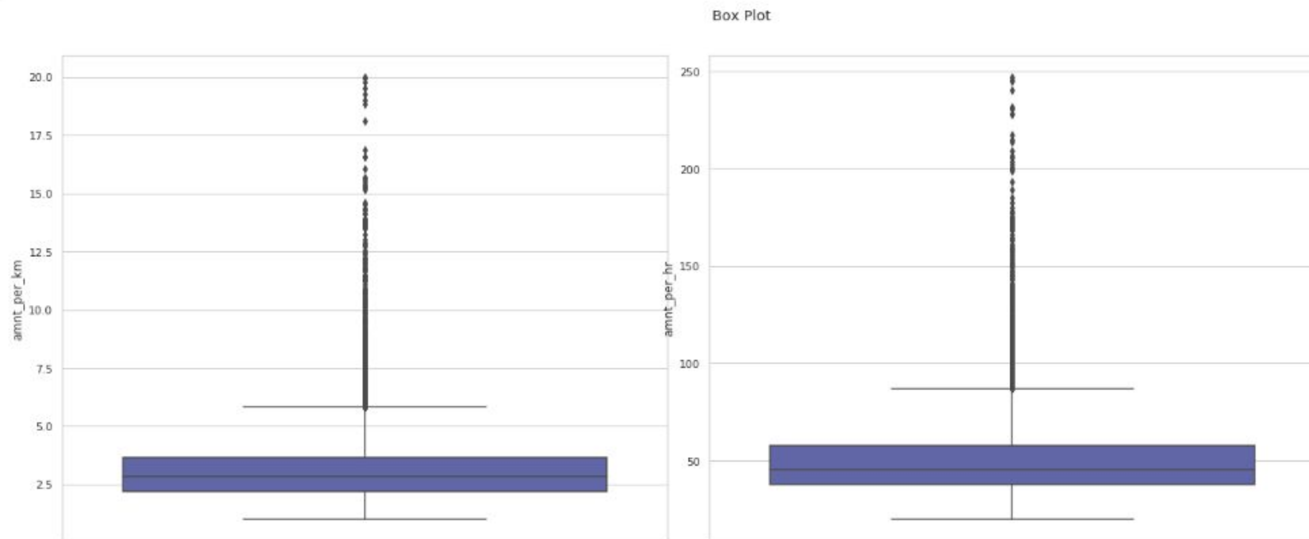


Insights:

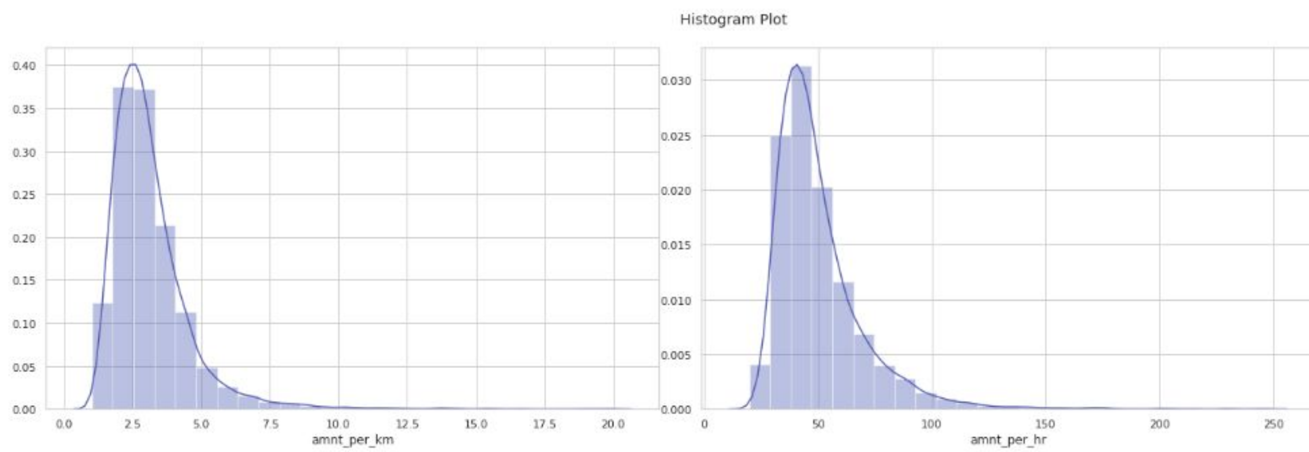
- ➔ Most of the onroad_distance is between 1km to 4km or 1000m to 4000m
- ➔ Most of the onroad_time is between 300 seconds to 1200 seconds, i.e. 5 minutes to 20 minutes
- ➔ Most of the aerial_distance is between 1km to 3km, peaking at 2km.

4. Box plot between amnt_per_km & amnt_per_hr & its histogram plots

In [62]:
```
box_plot(y_axis_cols_list=['amnt_per_km', 'amnt_per_hr'], x_axis_data_list=[clean_training_data, clean_training_data], subplot_nrows=1, subplot_ncols=2)
```



In [63]:
```
hist_plot(y_axis_cols_list=['amnt_per_km', 'amnt_per_hr'], x_axis_data_list=[clean_training_data, clean_training_data], subplot_nrows=1, subplot_ncols=2, bins_list=[25, 25])
```



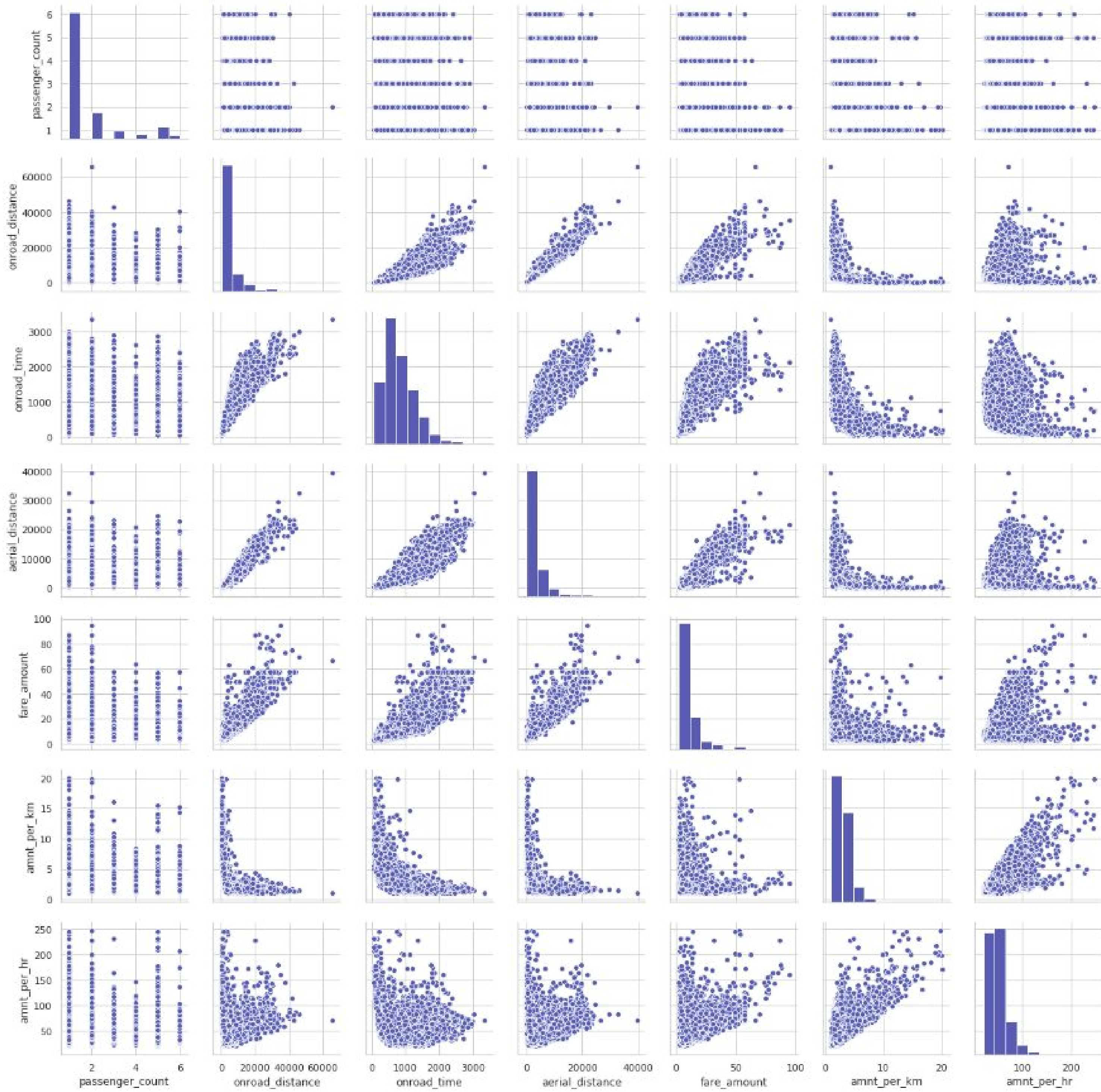
Insights:

- ➔ Most of the observations have amnt_per_km around $1.5 - $3
- ➔ Most of the observations have amnt_per_hr around $40 - $50

5. Pair plot between all the features, predictors & target as well

n [64]:
```
sns.pairplot(clean_training_data[numerical_columns])
```

ut[64]: <seaborn.axisgrid.PairGrid at 0x7ff2169ce358>



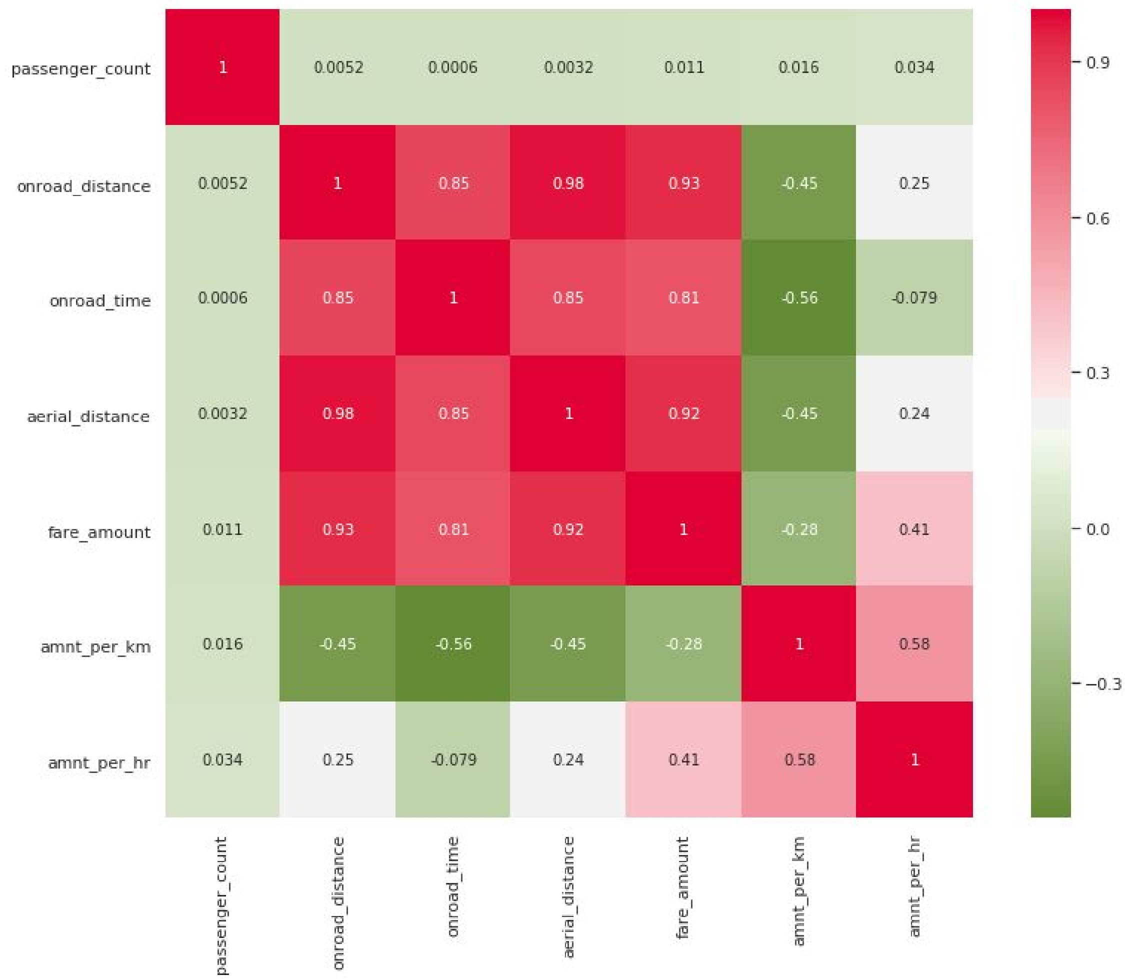
Insights

- ➔ onroad_distance & onroad_time & aerial_distance are linearly related to fare_amount (target)
- ➔ amnt_per_km is rectangular hyperbolically related to fare_amount
- ➔ amnt_per_hr doesn't have a very distinct relationship

6. Heatmap plot to see correlation between the numerical variables

In [65]:
```
correlation_coeff = clean_training_data[numerical_columns].corr()
```

In [66]:
```
fig = plt.figure(figsize = (14,10))
heatmap_plot = sn_plt = sns.heatmap(data=correlation_coeff, square = True,\
                                    cmap = sns.diverging_palette(as_cmap=True, h_neg=100, h_pos=10),\
                                    annot= True)
```



0.00-0.19: very weak 0.20-0.39: weak 0.40-0.59: moderate 0.60-0.79: strong 0.80-1.00: very strong. Keeping correlation coefficient threshold at 0.8, we can drop aerial_distance or onroad_distance, but as onroad_distance has higher correlation with fare_amount, we will drop aerial_distance

Insights:

- ➔ onroad_distance is highly positively correlated with aerial_distance and fare_amount. So we have to drop one of the two, aerial or onroad distance
- ➔ onroad_time & onroad_distance are also correlated positively
- ➔ amnt_per_km is negatively correlated with fare_amount & onroad_time & distance

I now drop the columns that are of no use in prediction. Coordinate columns & datetime

Features in our dataset:

1. passenger_count (**num**)
2. aerial_distance (**num**)
3. wday (day of the week)
4. mnth (month of the year)
5. hr (hour of the day)
6. Year
7. winter_month (**cat**)
8. weekday (**cat**)
9. nght_hr (**cat**)
10. onroad_distance (**num**)
11. onroad_time (**num**)
12. amnt_per_km (**num**)
13. amnt_per_hr (**num**)

Selection & Importance of Features

“Garbage In, Garbage Out”

This is an important rule in data science. Feeding well-structured, properly labelled & correct data will give relevant and meaningful output. Feeding non-sense & useless data will give irrelevant & garbage output. So selection of features to include in out training model is of utmost importance.
For this, we perform certain tasks. Like correlation analysis that had already been done before through the heatmap.
For categorical data we perform Chi-Square Test along with Hypothesis testing to determine if 2 variables are independent or not.

What is Hypothesis Testing?

A: Hypothesis Testing is a statistical test to check dependency of 2 categorical variables. We assume a null hypothesis - H_0 & an alternate hypothesis - H_1.
H_0 - The 2 categorical variables are **independent** - there is no relationship between them
H_1 - The 2 categorical variables are **not independent** - they are not totally random, there is some relationship.
Accepting or rejecting the null hypothesis is based on the p-value of the test & the p-value threshold that we assume before conducting the experiment. We assume p-value threshold to be **0.05**.
This means, under totally random circumstances, given the observations of the 2 variables, there is **atleast 5%** chance of such values to occur. We say our confidence interval is **95%**.
If the experiment’s p-value comes out to be less than our threshold, then we say, under totally random circumstances, for such observations of these 2 variables to occur has less than 5% chance, and thus with 95% confidence we can say that the 2 variables are not totally random or independent, i.e. we reject the null hypothesis and accept the alternative hypothesis.

After conducting the hypothesis test, we drop the interdependent categorical variables.

From out test, we have decided to drop the following variables:

1. year 2. mnth 3. wday 4. hr

The results summary is as shown below.
Pairs with p-values less than 0.05 are not independent & one of them must be removed.

A snapshot of the result of the chi-square test.

```
In [80]: chi_test_res
Out[80]:
```

	Feature_1	Feature_2	P-Values
0	wday	wday	-1.00
1	wday	weekday	0.00
2	wday	mnth	0.23
3	wday	winter_month	0.59
4	wday	hr	0.00
5	wday	nght_hr	0.03
6	wday	year	0.09
7	weekday	wday	0.00
8	weekday	weekday	-1.00
9	weekday	mnth	0.07
10	weekday	winter_month	0.42
11	weekday	hr	0.00
12	weekday	nght_hr	0.61
13	weekday	year	0.01
14	mnth	wday	0.23
15	mnth	weekday	0.07
16	mnth	mnth	-1.00
17	mnth	winter_month	0.00
18	mnth	hr	0.73
19	mnth	nght_hr	0.27
20	mnth	year	0.00
21	winter_month	wday	0.59
22	winter_month	weekday	0.42
23	winter_month	mnth	0.00
24	winter_month	winter_month	-1.00
25	winter_month	hr	0.04
26	winter_month	nght_hr	0.34
27	winter_month	year	0.00
28	hr	wday	0.00
29	hr	weekday	0.00
30	hr	mnth	0.73
31	hr	winter_month	0.04
32	hr	hr	-1.00

Importance of the features in our model

```
In [70]: feature_importance.sort_values(asc
Out[70]:
```

	Feature	Importance_Fraction
9	onroad_distance	0.820092
11	amnt_per_km	0.079963
1	aerial_distance	0.064912
12	amnt_per_hr	0.024251
10	onroad_time	0.008314
5	year	0.001580
4	hr	0.000273
2	wday	0.000232
3	mnth	0.000194
0	passenger_count	0.000079
7	weekday	0.000062
6	winter_month	0.000034
8	nght_hr	0.000014

Importance of features

This was found by exploiting a feature of the randomforest regressor, that has an attribute ***feature_importances_***

It basically explains how much role does each feature play in the prediction of the target variable. As we can see, a huge importance is played by onroad_distance - over **80%**, and intuitively, we can understand it to be correct also.

Then the 2nd most important feature is amnt_per_km, which explains about **8%** of the variability in the data. Subsequently, onroad_time & amnt_per_hr also play important contributions.

I want to drop one of the 2 variables, aerial_distance or onroad_distance, but after seeing this I am pretty much sure I will be dropping aerial_distance.

Just to make sure that my decision is correct, I will verify it with another correlation analysis metric for numerical variables - **Variance Inflation Factor** (VIF).

A VIF value of over 10, indicates a really high collinearity and one of the 2 variables should and must be dropped from the model.

BEFORE removing one of the 2 aerial_distance & onroad_distance

In [67]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from statsmodels.tools.tools import add_constant
numerical_data = add_constant(clean_training_data[numerical_columns])
variance_if = pd.Series([vif(numerical_data.values, i) for i in range(numerical_data.shape[1])],\
                         index = numerical_data.columns)
variance_if.round(2)
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2389: FutureWarning: Method .ptp i
s deprecated and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

Out[67]:
```
const              33.40
passenger_count     1.00
onroad_distance    26.13
onroad_time         8.27
aerial_distance    22.92
fare_amount        14.64
amnt_per_km         3.61
amnt_per_hr         5.62
dtype: float64
```

AFTER removing aerial_distance

In [68]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
from statsmodels.tools.tools import add_constant
numerical_data = add_constant(clean_training_data[numerical_columns].drop(columns=['aerial_distance']))
variance_if = pd.Series([vif(numerical_data.values, i) for i in range(numerical_data.shape[1])],\
                         index = numerical_data.columns)
variance_if.round(2)
```

Out[68]:
```
const              33.40
passenger_count     1.00
onroad_distance    11.49
onroad_time         8.19
fare_amount        14.30
amnt_per_km         3.60
amnt_per_hr         5.61
dtype: float64
```

Dropping un-required columns

I am dropping the columns that are not required or are not independent which are, aerial_distance, mnth, year, wday, hr.

So remaining features are

1. passenger_count	2. onroad_distance	3. onroad_time
4. amnt_per_km	5. amnt_per_hr	6. winter_month
7. weekday	8. nght_hr	

Part 4: **Preparing Test Data & Building Models**

All the preprocessing & feature engineering that we have so far done on the training dataset, we now have to perform on the test data set to prepare it to predict the output from the trained model.

4.1 Preparing the test data

The following steps are performed on the test data:

1. Loading into a pandas dataframe
2. Stripping datetime
3. Removing NAs & 0s
4. Converting passenger_count to integer type
5. Fetching onroad_distance & onroad_time from GoogleMapsAPI & storing it in a file to be read from later, and extracting data from the JSON response.
   NOTE: File containing the GoogleMaps data for the training dataset & test dataset are in the folder that will be delivered, so please keep them while running or the program will not run. And a demo snippet of code is also given in the code to re run the code to fetch the data from APIs. Keep in mind that running that code again, the results obtained might not EXACTLY match with the one stored by me in the file, as the traffic conditions might be different from when I fetched the data & when the code will be run again, as the GoogleMapsAPI returns real-time data.

After this, the new features are prepared, and added into the dataset & unwanted features are dropped. NO OUTLIERS ARE REMOVED.

4.2 Making a training set without outliers

The training data is split into 2 parts, one with the original complete dataset consisting of the outliers & the other where the outliers are totally removed using InterQuartile Range method.

Removing IQR Outliers

In [84]:
```
def is_in_iqr_range(num, col_name, pandas_data_frame):
    """
    Function to check if a given value is within the IQR range of that column or not
    num - float or integer type
    col_name - string type
    """
    q1 = np.quantile(np.array(pandas_data_frame[col_name]), 0.25)
    q3 = np.quantile(np.array(pandas_data_frame[col_name]), 0.75)
    iqr = q3 - q1
    lower_range = q1 - 1.5*iqr
    upper_range = q3 + 1.5*iqr
    if num >= lower_range and num <= upper_range:
        return True
    return False

def remove_outliers_iqr(cols_list, pandas_data_frame):
    """
    Function to remove the rows containing outlier values using IQR Range
    col_list - list of strings of columns names
    """
    initial_n_rows = pandas_data_frame.shape[0]
    for col in cols_list:
        pandas_data_frame = pandas_data_frame[pandas_data_frame[col].apply(lambda x: is_in_iqr_range(x, col, pandas
_data_frame))]
    after_n_rows = pandas_data_frame.shape[0]
    n_rows_removed = initial_n_rows - after_n_rows
    print("Number of Rows removed = " + str(n_rows_removed) + "\n")
    return pandas_data_frame
```

In [85]:
```
training_data_iqr = remove_outliers_iqr(pandas_data_frame=training_data, cols_list=['passenger_count','onroad_dista
nce','onroad_time','amnt_per_km','amnt_per_hr'])
```

```
Number of Rows removed = 4028
```

4.3 Randomizing the test data & training dataset;

The prepared & preprocessed test & training data is randomized to avoid any patterns formed in the dataset chronologically. The indexes are reset.

4.4 Determining Metrics to test the accuracy and correctness of the model

Because it is a regression model that I am building, the following metrics can be used:

1. **RMSE** (Root Mean Squared Error)
2. **MAE** (Mean Absolute Error)
3. **R-Squared** - variability of dataset explained, percentage or fraction
4. **Adjusted R-Squared**

4.5 Model to be built

I will be building the following models:

4.5.1 Linear Regression:

- With outliers training dataset, this is the performance of the model

With Outliers Data

In [170]:
```
linreg_model.fit(X_train, y_train)
cv_folds = 5
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train, y=y_train, estimator=linreg_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train.shape[0]
p = X_train.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(linreg_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 93.01228%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 93.00859%
```

- Without outliers training dataset, this is the performance of the model

Without IQR Outliers Data

In [172]:
```
linreg_model.fit(X_train_iqr, y_train_iqr)
cv_folds = 5
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train_iqr, y=y_train_iqr, estimator=linreg_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train_iqr.shape[0]
p = X_train_iqr.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(linreg_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 93.61435%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 93.60975%
```

4.5.2 Decision Trees

- With outliers training dataset, this is the performance of the model

With Outliers Data

In [175]:
```
dt_model.fit(X_train, y_train)
cv_folds = 5
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train, y=y_train, estimator=dt_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train.shape[0]
p = X_train.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(dt_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 99.05100%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 99.05050%
```

- Without outliers training dataset, this is the performance of the model

Without IQR Outliers Data

In [177]:
```
dt_model.fit(X_train_iqr, y_train_iqr)
cv_folds = 5
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train_iqr, y=y_train_iqr, estimator=dt_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train_iqr.shape[0]
p = X_train_iqr.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(dt_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 99.49078%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 99.49041%
```

4.5.3 Random Forest

- With outliers training dataset, this is the performance of the model

With Outliers Data

In [179]:
```
regression_model.fit(X_train, y_train)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train, y=y_train, estimator=regression_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train.shape[0]
p = X_train.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using rave
l().
  """Entry point for launching an IPython kernel.
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 99.49038%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 99.49011%
```

- Without outliers training dataset, this is the performance of the model

Without IQR Outliers Data

```
In [180]: regression_model.fit(X_train_iqr, y_train_iqr)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train_iqr, y=y_train_iqr, estimator=regression_model, cv=cv_folds,\
                             scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train_iqr.shape[0]
p = X_train_iqr.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using rave
l().
  """Entry point for launching an IPython kernel.
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 99.85770%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 99.85760%
```

4.5.4 EXTremely RAndomised Tree (EXTRA Tree)

- With outliers training dataset, this is the performance of the model

With Outliers Data

```
In [182]: regression_model.fit(X_train, y_train)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train, y=y_train, estimator=regression_model, cv=cv_folds,\
                             scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train.shape[0]
p = X_train.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using rave
l().
  """Entry point for launching an IPython kernel.
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 99.67495%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 99.67478%
```

- Without outliers training dataset, this is the performance of the model

Without IQR Outliers Data

```
In [183]: regression_model.fit(X_train_iqr, y_train_iqr)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train_iqr, y=y_train_iqr, estimator=regression_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train_iqr.shape[0]
p = X_train_iqr.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: DataConversionWarning: A column-vec
tor y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using rave
l().
  """Entry point for launching an IPython kernel.
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 99.94334%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 99.94330%
```

4.5.5 K-Nearest Neighbors (KNN)

- With outliers training dataset, this is the performance of the model

With Outliers Data

```
In [185]: regression_model.fit(X_train, y_train)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train, y=y_train, estimator=regression_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train.shape[0]
p = X_train.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 88.17215%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 88.16590%
```

- Without outliers training dataset, this is the performance of the model

Without IQR Outliers Data

In [186]:
```
regression_model.fit(X_train_iqr, y_train_iqr)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train_iqr, y=y_train_iqr, estimator=regression_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train_iqr.shape[0]
p = X_train_iqr.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 78.32104%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 78.30544%
```

4.5.6 SVM (Support Vector Machines) Regressor

- With outliers training dataset, this is the performance of the model

With Outliers Data

In [188]:
```
regression_model.fit(X_train, y_train)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train, y=y_train, estimator=regression_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train.shape[0]
p = X_train.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:724: DataConversionWarning: A co
lumn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)
```

```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 86.47137%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 86.46423%
```

- Without outliers training dataset, this is the performance of the model

Without IQR Outliers Data

In [190]:
```
regression_model.fit(X_train_iqr, y_train_iqr)
cv_folds = 3
scoring_metric = 'r2'
cv_scores = cross_val_score(X=X_train_iqr, y=y_train_iqr, estimator=regression_model, cv=cv_folds,\
                            scoring=scoring_metric, n_jobs=-1)
cv_scores_mean = cv_scores.mean()
r2_score = cv_scores_mean
n = X_train_iqr.shape[0]
p = X_train_iqr.shape[1]
adj_r2 = 1 - ((1-r2_score)*(n-1)/(n-p-1))
y_pred = pd.DataFrame(regression_model.predict(X_test))
print("================ Train Data - R-squared Score ================")
print("Shows the fraction of variance explained by our model.")
print("R-Squared Score = {:.5f}%".format(r2_score*100))
print()
print("================ Adjusted R-Squared Details ================")
print("Penalises using excessive features")
print("Adjusted R-Squared Score = {:.5f}%".format(adj_r2*100))
```

```
/home/shitbot009/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:724: DataConversionWarning: A co
lumn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)
```

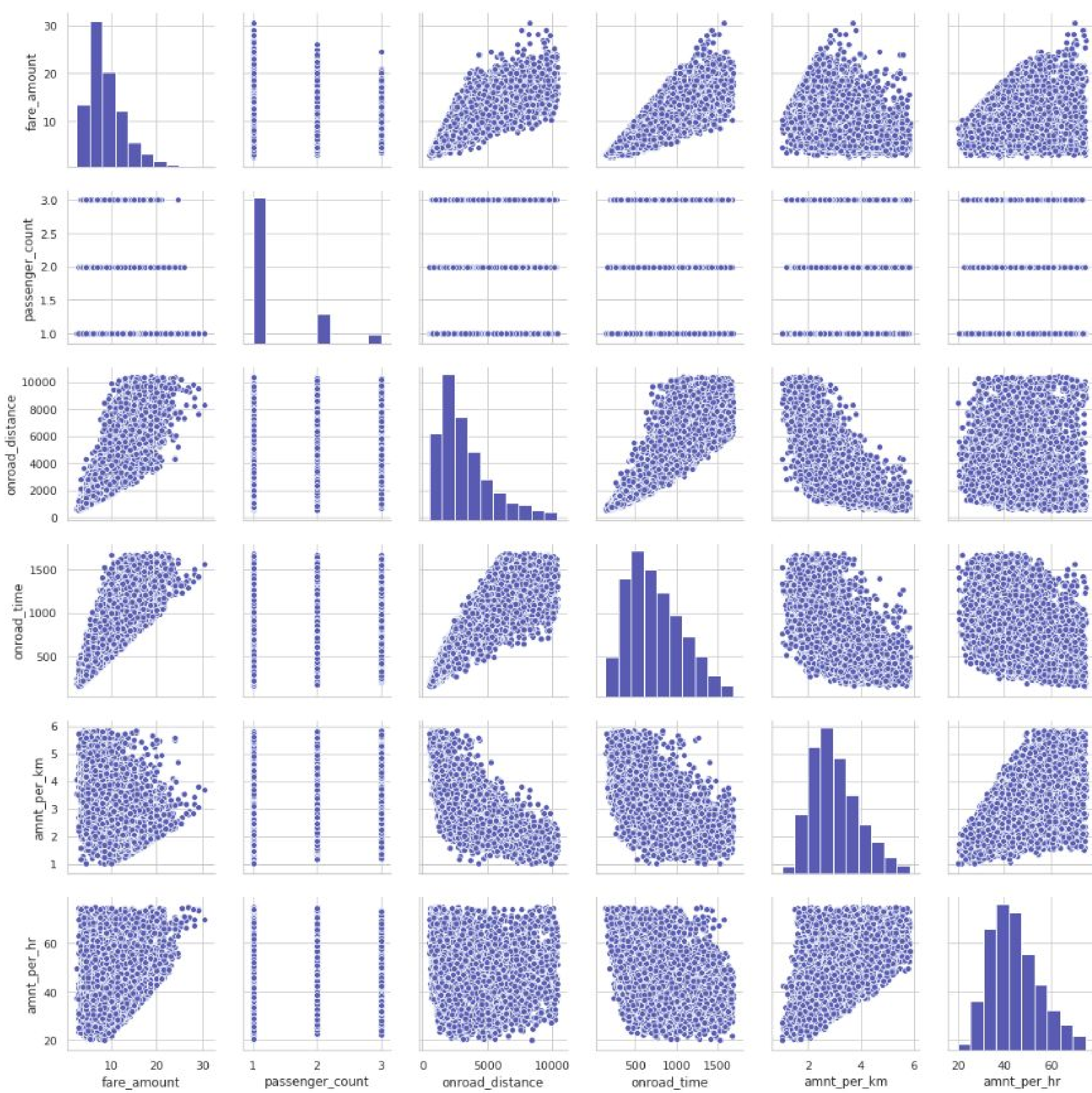
```
================ Train Data - R-squared Score ================
Shows the fraction of variance explained by our model.
R-Squared Score = 76.64304%

================ Adjusted R-Squared Details ================
Penalises using excessive features
Adjusted R-Squared Score = 76.62623%
```

4.5.7 Neural Networks

Library used - **Keras**

Plotting training set without outliers to visualize

```
In [195]: sns.pairplot(training_data_iqr.drop(columns=['nght_hr','winter_month','weekday']))
Out[195]: <seaborn.axisgrid.PairGrid at 0x7ff1f03cd198>
```



Scaling: As data is NOT normally distributed, we go for *MinMaxScaler*

Architecture of NN - Sequential

Number of Input Nodes: 8 (same as number of features)

Depth: 1

Number of Output Nodes: 1

Optimizer: Adam (Adaptive Moment)

Loss Function: MSE (Mean Squared Error)

Metric: MAE (Mean Absolute Error)

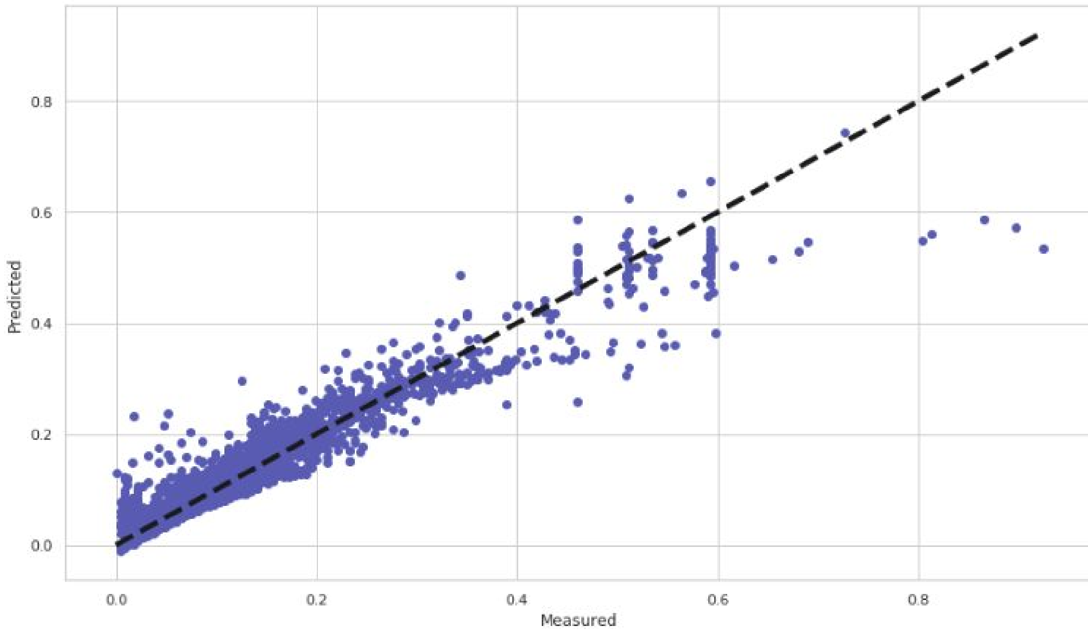
Batch Size: 512

Epochs: 100

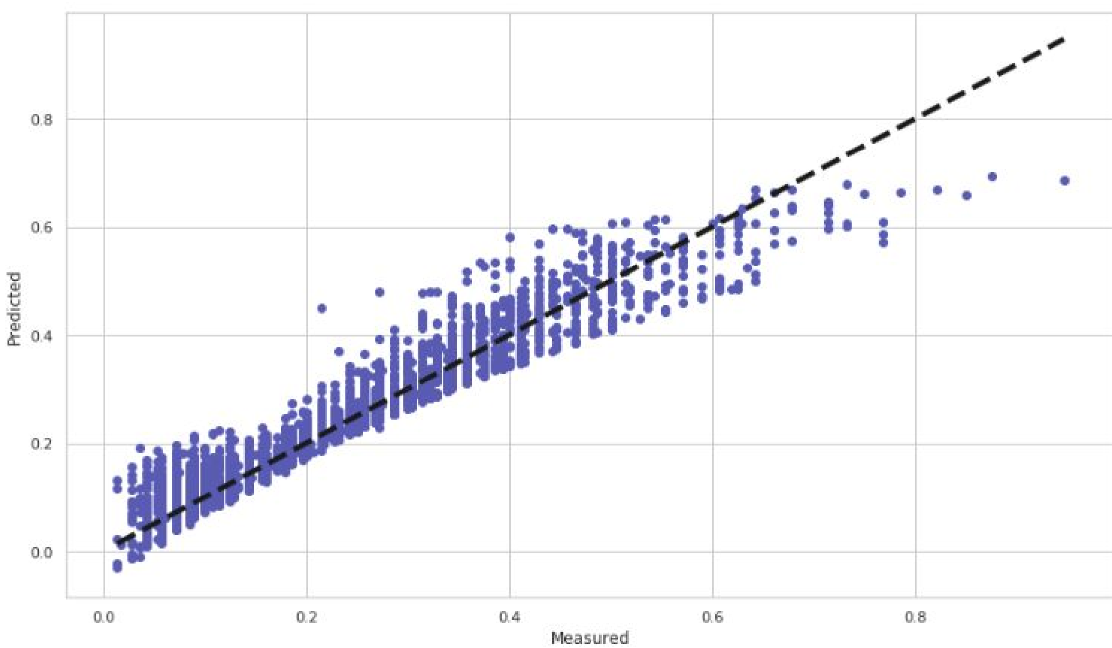
With outliers training dataset, this is the performance of the model

As can be seen, with every epoch, the MAE is falling.

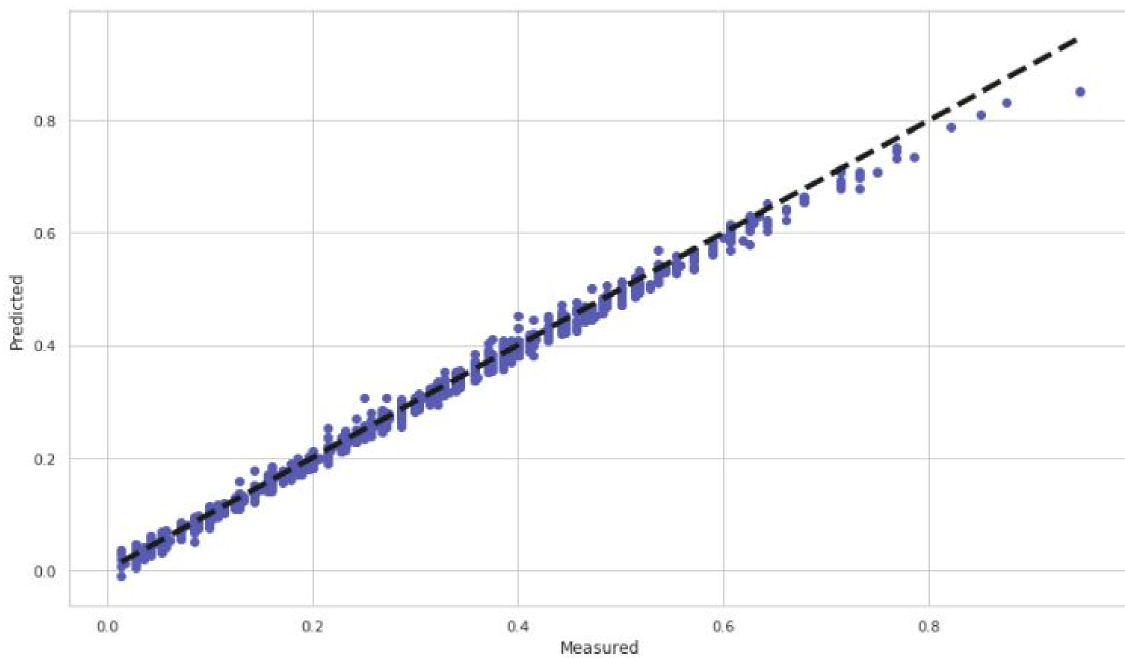
```
Epoch 1/100
10609/10609 [==============================] - 2s 204us/step - loss: 0.0570 - mean_absolute_error: 0.1785 - acc: 2.8
278e-04
Epoch 2/100
10609/10609 [==============================] - 0s 4us/step - loss: 0.0327 - mean_absolute_error: 0.1354 - acc: 2.827
8e-04
Epoch 3/100
10609/10609 [==============================] - 0s 2us/step - loss: 0.0198 - mean_absolute_error: 0.1009 - acc: 2.827
8e-04
Epoch 4/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0129 - mean_absolute_error: 0.0775 - acc: 2.827
8e-04
Epoch 5/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0090 - mean_absolute_error: 0.0617 - acc: 2.827
8e-04
Epoch 6/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0068 - mean_absolute_error: 0.0512 - acc: 2.827
8e-04
Epoch 7/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0052 - mean_absolute_error: 0.0439 - acc: 2.827
8e-04
Epoch 8/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0042 - mean_absolute_error: 0.0389 - acc: 2.827
8e-04
Epoch 9/100
10609/10609 [==============================] - 0s 2us/step - loss: 0.0035 - mean_absolute_error: 0.0354 - acc: 2.827
8e-04
Epoch 10/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0031 - mean_absolute_error: 0.0330 - acc: 2.827
8e-04
Epoch 11/100
10609/10609 [==============================] - 0s 2us/step - loss: 0.0028 - mean_absolute_error: 0.0315 - acc: 3.770
4e-04
Epoch 12/100
10609/10609 [==============================] - 0s 2us/step - loss: 0.0025 - mean_absolute_error: 0.0305 - acc: 3.770
4e-04
Epoch 13/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0024 - mean_absolute_error: 0.0297 - acc: 3.770
4e-04
Epoch 14/100
10609/10609 [==============================] - 0s 3us/step - loss: 0.0023 - mean_absolute_error: 0.0290 - acc: 3.770
4e-04
Epoch 15/100
10609/10609 [==============================] - 0s 2us/step - loss: 0.0022 - mean_absolute_error: 0.0284 - acc: 3.770
```



Without Outliers:



4.5.8 Deep Neural Networks
<u>Architecture of DNN</u> **- Sequential**
Number of Input Nodes: 8 (same as number of features)
Depth: 3
Hidden Layers Size: 2 hidden layers, 32 nodes + 16 nodes
Number of Output Nodes: 1
Optimizer: Adam (Adaptive Moment)
Loss Function: MSE (Mean Squared Error)
Metric: MAE (Mean Absolute Error)
Batch Size: 128
Epochs: 50



To make this model better, I have to deal with underfitting, overfitting, regularization. Which can be done as an extended version of this project.

Conclusion:
After seeing and analysing all the above models, which are completely raw. They can be further tuned to perform even better, which I will do as an extended project. But based on this raw analysis, **EXTRA Trees** model seems to work fine. I have used **joblib** library to store these models, so as to not repeat the training again and again. The model can be directly loaded to predict values on new unseen data.