Code Guidelines :

1. **Built using what ?**

a) **Python** – The code is written using Jupyter notebook

b) Matplotlib for plotting and graph generation

c) Pandas for extraction and cleaning

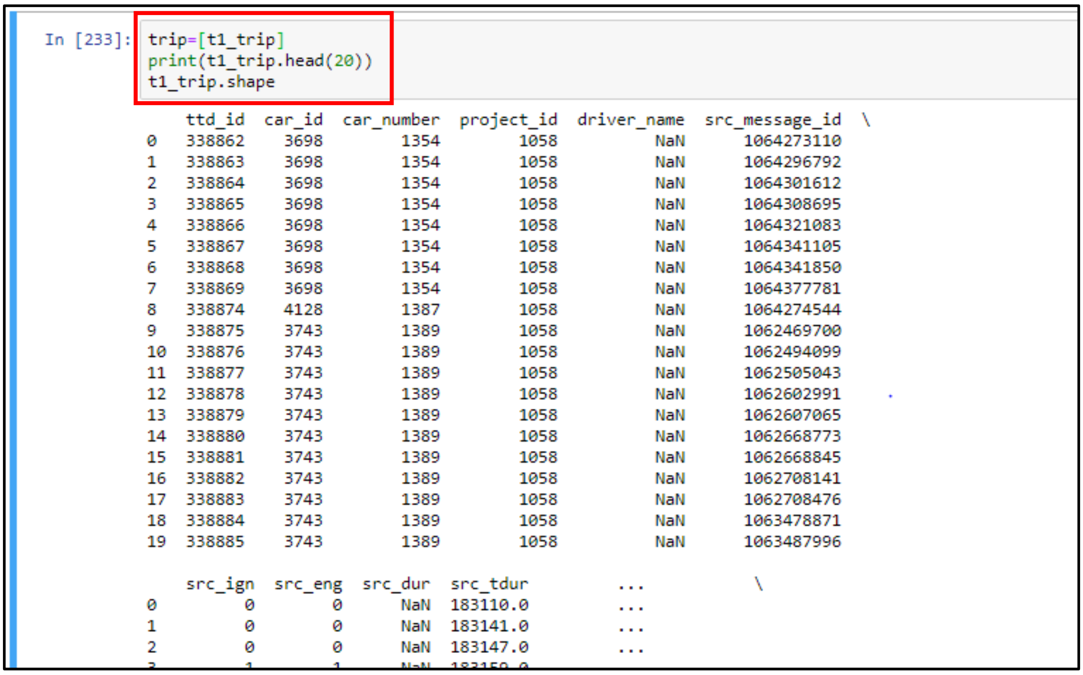
d) Flask for launching an API

2. **Explanations for the tasks - according to KDD (Knowledge & Discovery in Datasets)**

   a) **Selection**

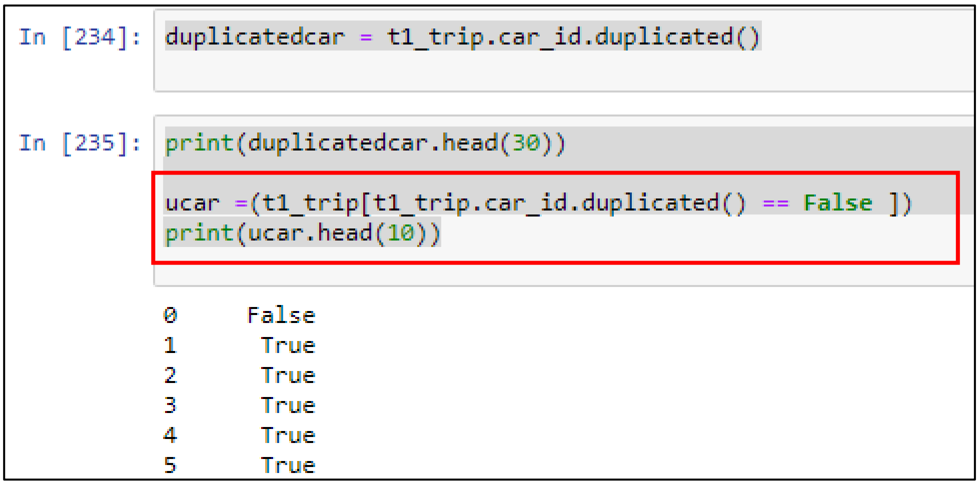
      I. **Removing duplicates – Identifying unique car_ids and car numbers**

      The dataset has duplicate entries of car_ids.



```
In [233]: trip=[t1_trip]
          print(t1_trip.head(20))
          t1_trip.shape

    ttd_id  car_id  car_number  project_id  driver_name  src_message_id  \
0   338862    3698        1354        1058          NaN      1064273110
1   338863    3698        1354        1058          NaN      1064296792
2   338864    3698        1354        1058          NaN      1064301612
3   338865    3698        1354        1058          NaN      1064308695
4   338866    3698        1354        1058          NaN      1064321083
5   338867    3698        1354        1058          NaN      1064341105
6   338868    3698        1354        1058          NaN      1064341850
7   338869    3698        1354        1058          NaN      1064377781
8   338874    4128        1387        1058          NaN      1064274544
9   338875    3743        1389        1058          NaN      1062469700
10  338876    3743        1389        1058          NaN      1062494099
11  338877    3743        1389        1058          NaN      1062505043
12  338878    3743        1389        1058          NaN      1062602991
13  338879    3743        1389        1058          NaN      1062607065
14  338880    3743        1389        1058          NaN      1062668773
15  338881    3743        1389        1058          NaN      1062668845
16  338882    3743        1389        1058          NaN      1062708141
17  338883    3743        1389        1058          NaN      1062708476
18  338884    3743        1389        1058          NaN      1063478871
19  338885    3743        1389        1058          NaN      1063487996

    src_ign  src_eng  src_dur  src_tdur         ...             \
0         0        0      NaN  183110.0         ...
1         0        0      NaN  183141.0         ...
2         0        0      NaN  183147.0         ...
```

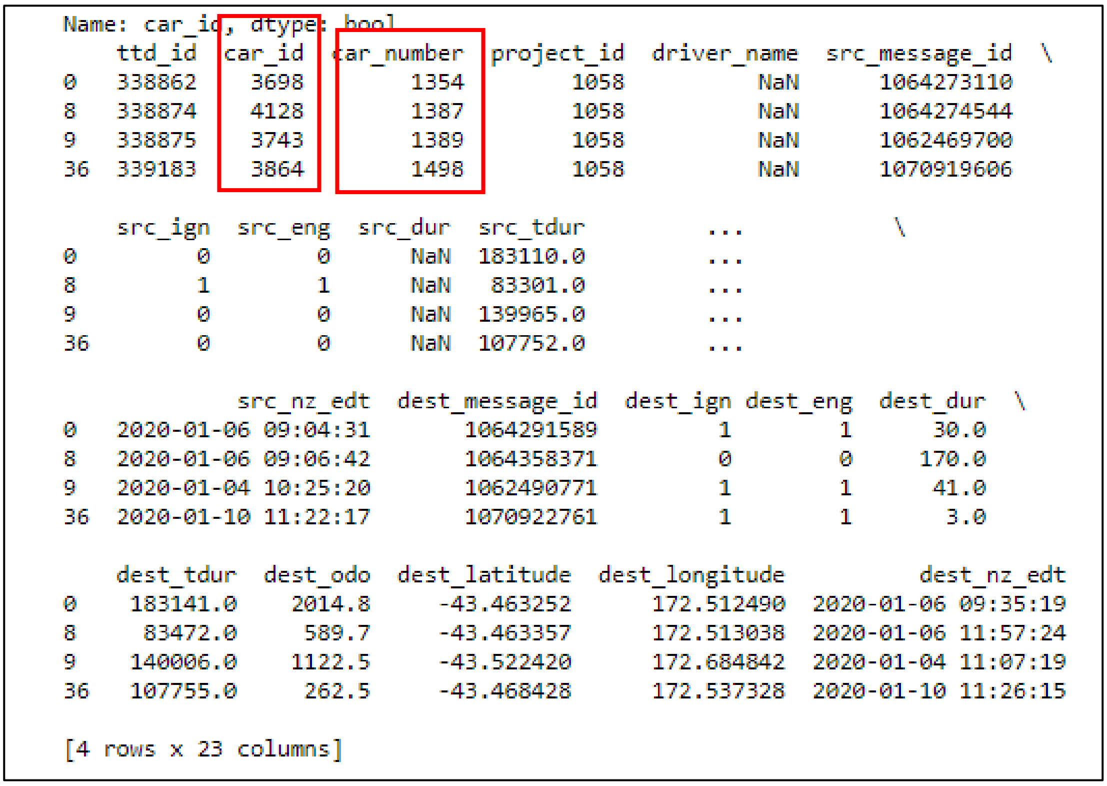
Identifying duplicates is done by duplicated() method. The query below segregates the car_ids which are unique in the list and saves them in ucar,



```
In [234]: duplicatedcar = t1_trip.car_id.duplicated()

In [235]: print(duplicatedcar.head(30))

          ucar =(t1_trip[t1_trip.car_id.duplicated() == False ])
          print(ucar.head(10))

          0     False
          1      True
          2      True
          3      True
          4      True
          5      True
```

II. Task 1 & Task2

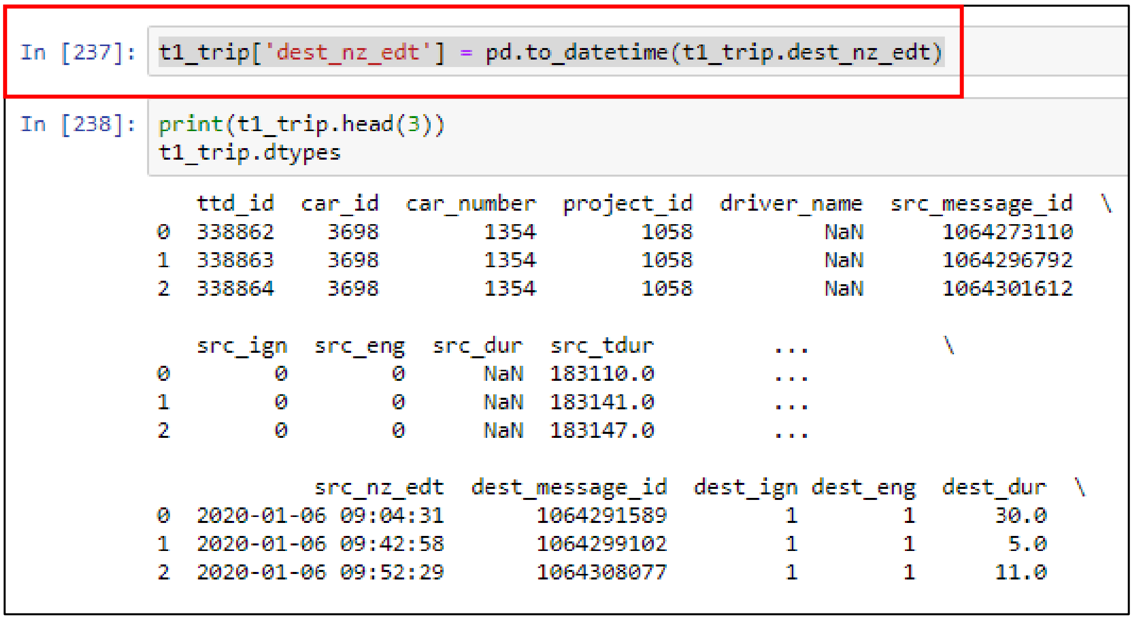
The unique car_ids and the car numbers are extracted out by the query and the output is as follows.



```
Name: car_id, dtype: bool
    ttd_id  car_id  car_number  project_id  driver_name  src_message_id  \
0   338862    3698        1354        1058          NaN      1064273110
8   338874    4128        1387        1058          NaN      1064274544
9   338875    3743        1389        1058          NaN      1062469700
36  339183    3864        1498        1058          NaN      1070919606

    src_ign  src_eng  src_dur  src_tdur        ...           \
0         0        0      NaN  183110.0        ...
8         1        1      NaN   83301.0        ...
9         0        0      NaN  139965.0        ...
36        0        0      NaN  107752.0        ...

             src_nz_edt  dest_message_id  dest_ign dest_eng  dest_dur  \
0   2020-01-06 09:04:31       1064291589         1        1      30.0
8   2020-01-06 09:06:42       1064358371         0        0     170.0
9   2020-01-04 10:25:20       1062490771         1        1      41.0
36  2020-01-10 11:22:17       1070922761         1        1       3.0

    dest_tdur  dest_odo  dest_latitude  dest_longitude          dest_nz_edt
0    183141.0    2014.8     -43.463252      172.512490  2020-01-06 09:35:19
8     83472.0     589.7     -43.463357      172.513038  2020-01-06 11:57:24
9    140006.0    1122.5     -43.522420      172.684842  2020-01-04 11:07:19
36   107755.0     262.5     -43.468428      172.537328  2020-01-10 11:26:15

[4 rows x 23 columns]
```

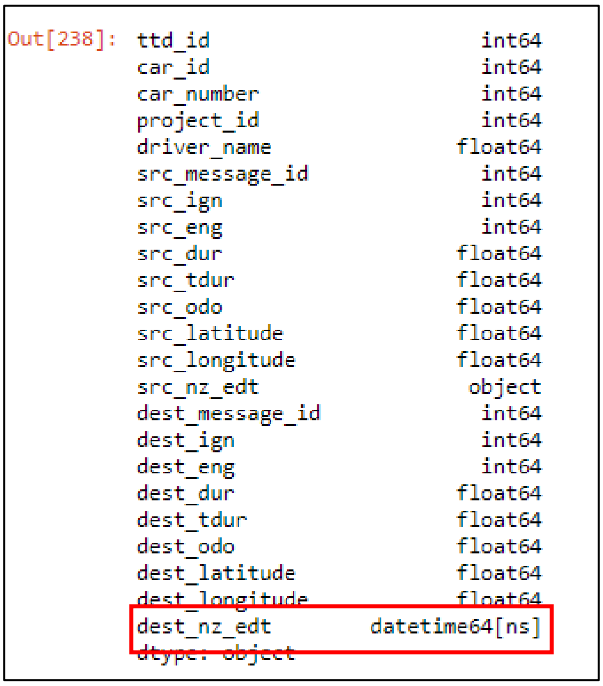
b) Pre-processing

I. To get the latest information out (ign, lat, long, tdur,driver_name, address) of all car numbers , we need to process the format of the field, dest_nz_edt. Assuming that the destination time marks the journey of a car to be complete for a day, I have brought it into the right format for further processing.



```
In [237]: t1_trip['dest_nz_edt'] = pd.to_datetime(t1_trip.dest_nz_edt)

In [238]: print(t1_trip.head(3))
          t1_trip.dtypes

             ttd_id  car_id  car_number  project_id  driver_name  src_message_id  \
          0  338862    3698        1354        1058          NaN      1064273110
          1  338863    3698        1354        1058          NaN      1064296792
          2  338864    3698        1354        1058          NaN      1064301612

             src_ign  src_eng  src_dur  src_tdur         ...             \
          0        0        0      NaN  183110.0         ...
          1        0        0      NaN  183141.0         ...
          2        0        0      NaN  183147.0         ...

                      src_nz_edt  dest_message_id  dest_ign dest_eng  dest_dur  \
          0  2020-01-06 09:04:31       1064291589         1        1      30.0
          1  2020-01-06 09:42:58       1064299102         1        1       5.0
          2  2020-01-06 09:52:29       1064308077         1        1      11.0
```

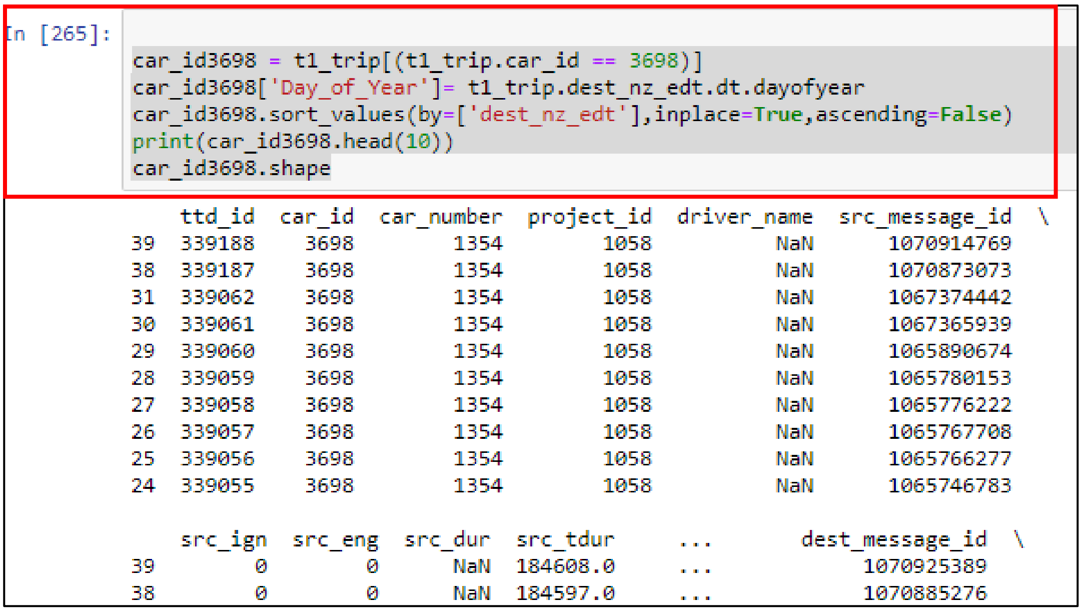
The format has changed and can be confirmed



```
Out[238]: ttd_id                        int64
          car_id                        int64
          car_number                    int64
          project_id                    int64
          driver_name                 float64
          src_message_id                int64
          src_ign                       int64
          src_eng                       int64
          src_dur                     float64
          src_tdur                    float64
          src_odo                     float64
          src_latitude                float64
          src_longitude               float64
          src_nz_edt                   object
          dest_message_id               int64
          dest_ign                      int64
          dest_eng                      int64
          dest_dur                    float64
          dest_tdur                   float64
          dest_odo                    float64
          dest_latitude               float64
          dest_longitude              float64
          dest_nz_edt          datetime64[ns]
          dtype: object
```

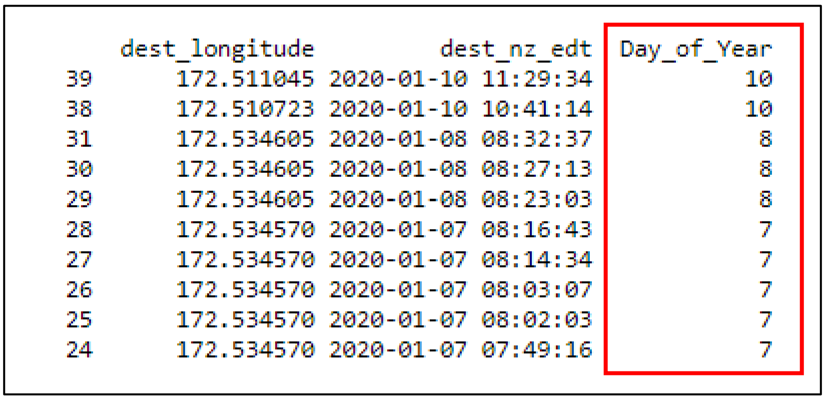
II. Task 3

The query below takes in every unique car_id which and transforms it out according to the destination time.



```
car_id3698 = t1_trip[(t1_trip.car_id == 3698)]
car_id3698['Day_of_Year']= t1_trip.dest_nz_edt.dt.dayofyear
car_id3698.sort_values(by=['dest_nz_edt'],inplace=True,ascending=False)
print(car_id3698.head(10))
car_id3698.shape
```

```
    ttd_id  car_id  car_number  project_id  driver_name  src_message_id  \
39  339188    3698        1354        1058          NaN      1070914769
38  339187    3698        1354        1058          NaN      1070873073
31  339062    3698        1354        1058          NaN      1067374442
30  339061    3698        1354        1058          NaN      1067365939
29  339060    3698        1354        1058          NaN      1065890674
28  339059    3698        1354        1058          NaN      1065780153
27  339058    3698        1354        1058          NaN      1065776222
26  339057    3698        1354        1058          NaN      1065767708
25  339056    3698        1354        1058          NaN      1065766277
24  339055    3698        1354        1058          NaN      1065746783

    src_ign  src_eng  src_dur  src_tdur     ...       dest_message_id  \
39        0        0      NaN  184608.0     ...            1070925389
38        0        0      NaN  184597.0     ...            1070885276
```



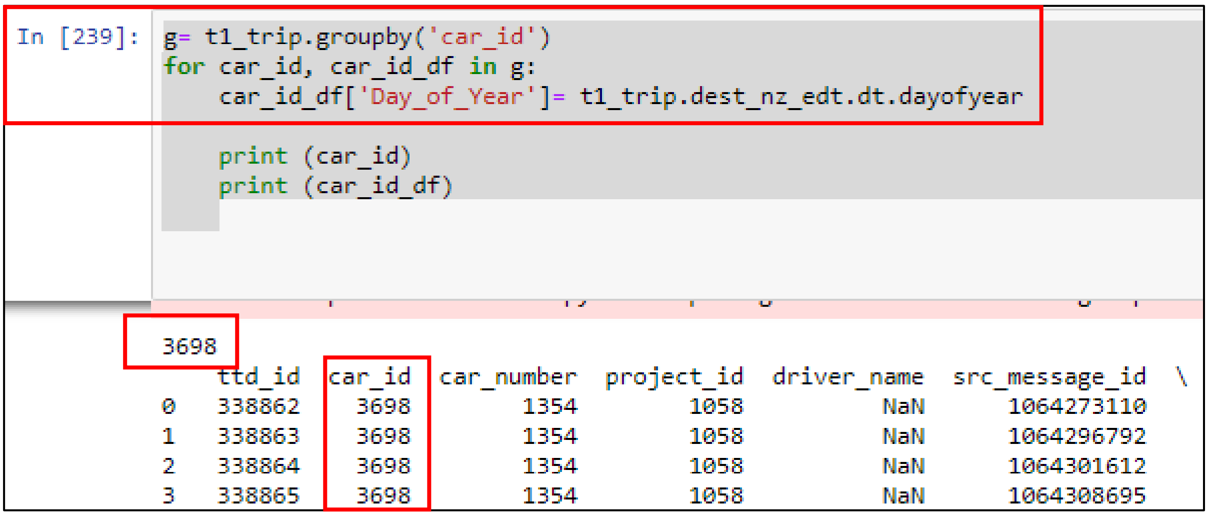
```
     dest_longitude          dest_nz_edt  Day_of_Year
39       172.511045  2020-01-10 11:29:34           10
38       172.510723  2020-01-10 10:41:14           10
31       172.534605  2020-01-08 08:32:37            8
30       172.534605  2020-01-08 08:27:13            8
29       172.534605  2020-01-08 08:23:03            8
28       172.534570  2020-01-07 08:16:43            7
27       172.534570  2020-01-07 08:14:34            7
26       172.534570  2020-01-07 08:03:07            7
25       172.534570  2020-01-07 08:02:03            7
24       172.534570  2020-01-07 07:49:16            7
```

The same process is adopted for all car ids.

Another column denoting day of year is also added to clearly interprate the data of car_ids day wise.

c) Transformation

Alternatively, group_by clause can also be used to save the data as an object file holding categorized car_id data.



```
In [239]: g= t1_trip.groupby('car_id')
          for car_id, car_id_df in g:
              car_id_df['Day_of_Year']= t1_trip.dest_nz_edt.dt.dayofyear

              print (car_id)
              print (car_id_df)
```

```
   3698
      ttd_id  car_id  car_number  project_id  driver_name  src_message_id  \
   0  338862    3698        1354        1058          NaN      1064273110
   1  338863    3698        1354        1058          NaN      1064296792
   2  338864    3698        1354        1058          NaN      1064301612
   3  338865    3698        1354        1058          NaN      1064308695
```

d) Data mining

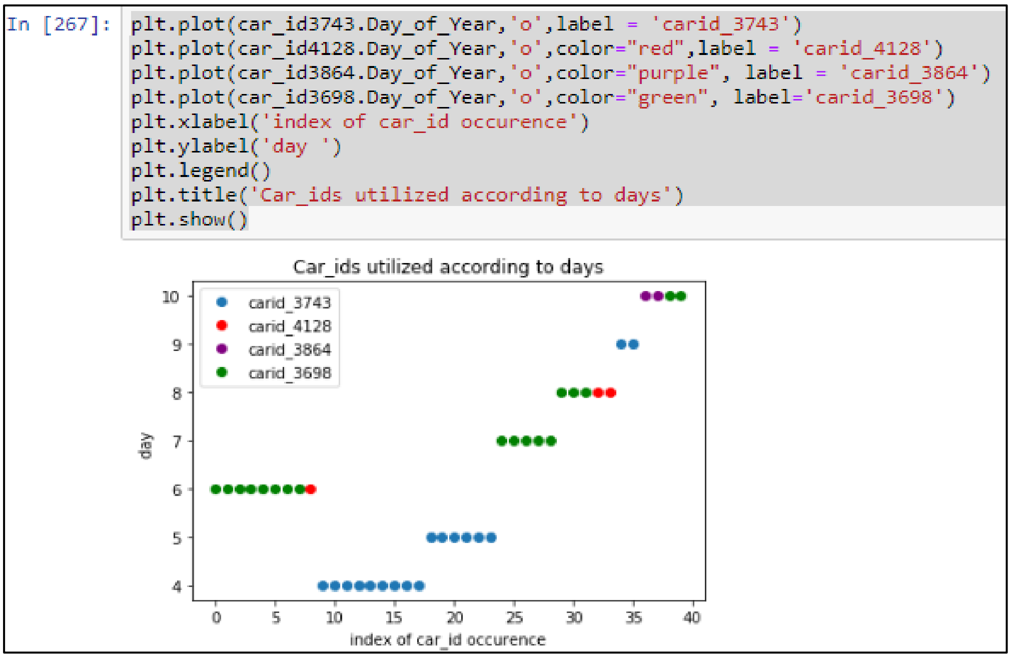
To mine data – driven insights, we need to have all car_ids in a format of days . The additional column defined as ‘Day_of_Year’ is useful out here for the comparison of car_ids according to the day they were utilised.

Alternatively, we can also have a Weekday names and months defined .

e) Interpretation

I. Task 5

To define the car_ids utilised according to the days , matplotlib library is used to put the data through in a simple yet clean format.

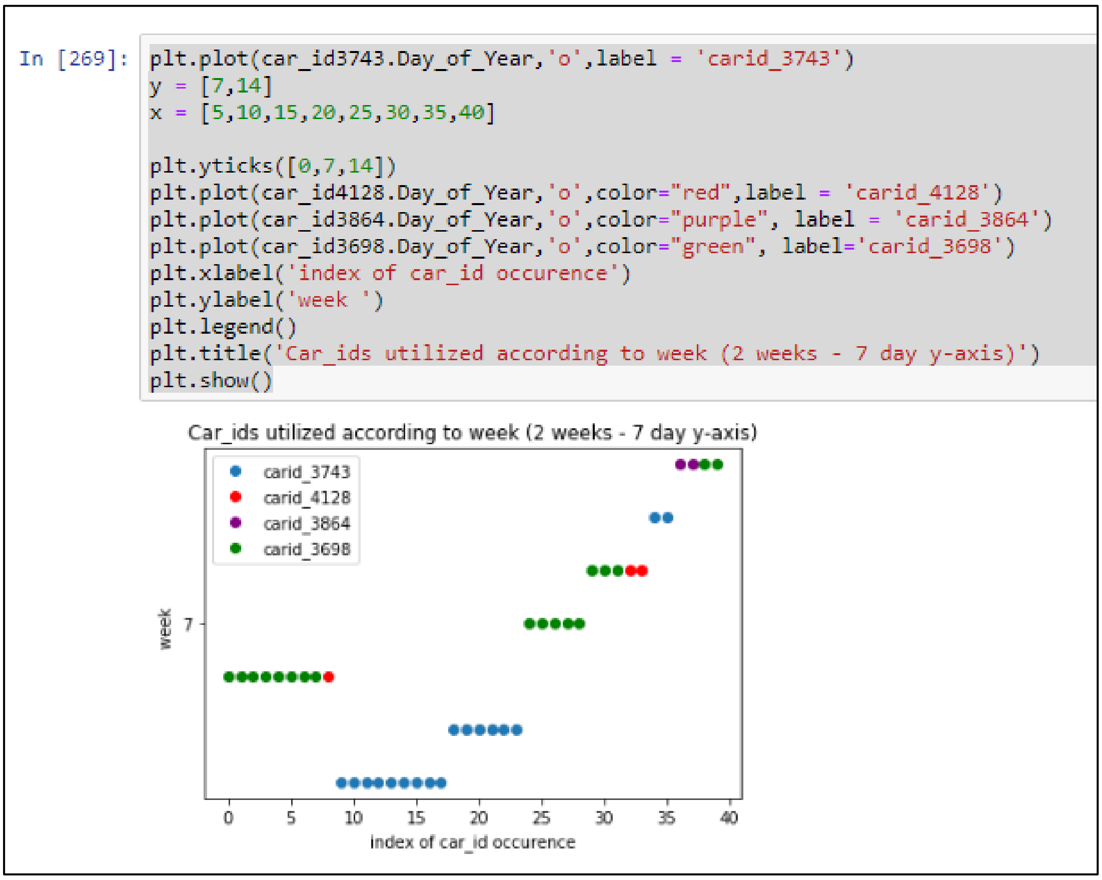


```
In [267]: plt.plot(car_id3743.Day_of_Year,'o',label = 'carid_3743')
          plt.plot(car_id4128.Day_of_Year,'o',color="red",label = 'carid_4128')
          plt.plot(car_id3864.Day_of_Year,'o',color="purple", label = 'carid_3864')
          plt.plot(car_id3698.Day_of_Year,'o',color="green", label='carid_3698')
          plt.xlabel('index of car_id occurence')
          plt.ylabel('day ')
          plt.legend()
          plt.title('Car_ids utilized according to days')
          plt.show()
```

The graph has car_ids according to the index of their occurrence.

II. Task 6

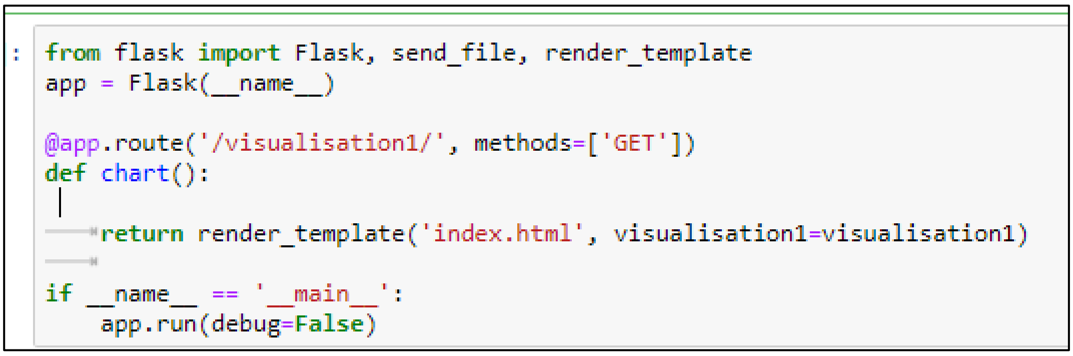
Similarly, by changing the scale we can have a weekly utilisation of the car_id



```
In [269]: plt.plot(car_id3743.Day_of_Year,'o',label = 'carid_3743')
          y = [7,14]
          x = [5,10,15,20,25,30,35,40]

          plt.yticks([0,7,14])
          plt.plot(car_id4128.Day_of_Year,'o',color="red",label = 'carid_4128')
          plt.plot(car_id3864.Day_of_Year,'o',color="purple", label = 'carid_3864')
          plt.plot(car_id3698.Day_of_Year,'o',color="green", label='carid_3698')
          plt.xlabel('index of car_id occurence')
          plt.ylabel('week ')
          plt.legend()
          plt.title('Car_ids utilized according to week (2 weeks - 7 day y-axis)')
          plt.show()
```

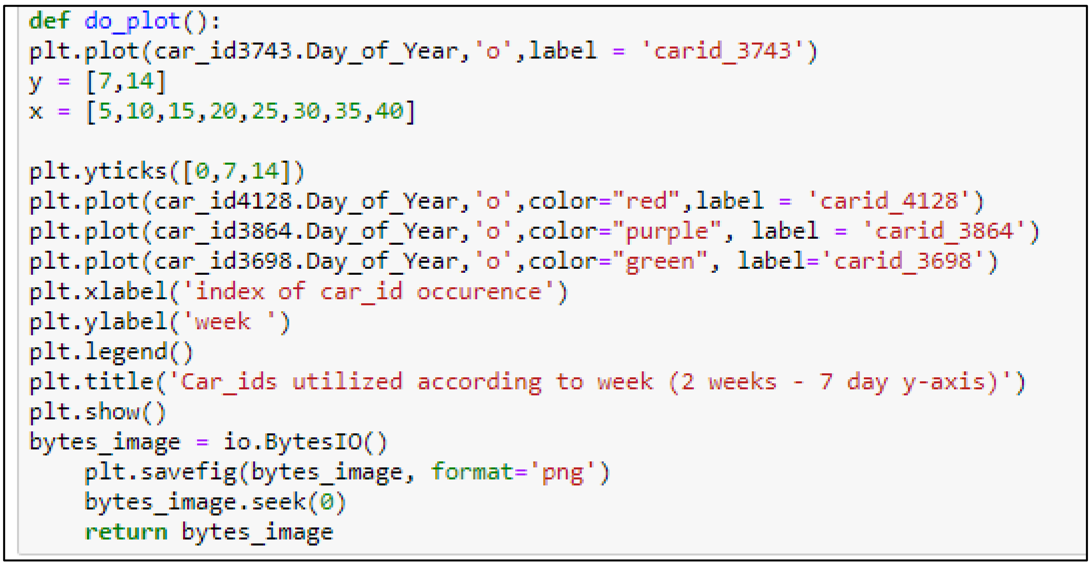
Car_ids utilized according to week (2 weeks - 7 day y-axis)
● carid_3743
● carid_4128
● carid_3864
● carid_3698
week
7
0
5
10
15
20
25
30
35
40
index of car_id occurence

3. Making use of Flask as a tool for launching Web API
   a) Flask provides the ability to launch a web page through the local host.
   b) It requires additional code as follows



```
from flask import Flask, send_file, render_template
app = Flask(__name__)

@app.route('/visualisation1/', methods=['GET'])
def chart():

    return render_template('index.html', visualisation1=visualisation1)

if __name__ == '__main__':
    app.run(debug=False)
```

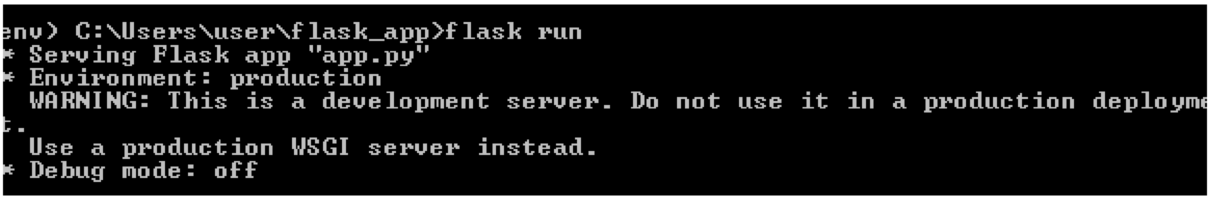
c) It also requires to define the plot commands in a function which can be called through it’s main function.



```
def do_plot():
plt.plot(car_id3743.Day_of_Year,'o',label = 'carid_3743')
y = [7,14]
x = [5,10,15,20,25,30,35,40]

plt.yticks([0,7,14])
plt.plot(car_id4128.Day_of_Year,'o',color="red",label = 'carid_4128')
plt.plot(car_id3864.Day_of_Year,'o',color="purple", label = 'carid_3864')
plt.plot(car_id3698.Day_of_Year,'o',color="green", label='carid_3698')
plt.xlabel('index of car_id occurence')
plt.ylabel('week ')
plt.legend()
plt.title('Car_ids utilized according to week (2 weeks - 7 day y-axis)')
plt.show()
bytes_image = io.BytesIO()
    plt.savefig(bytes_image, format='png')
    bytes_image.seek(0)
    return bytes_image
```

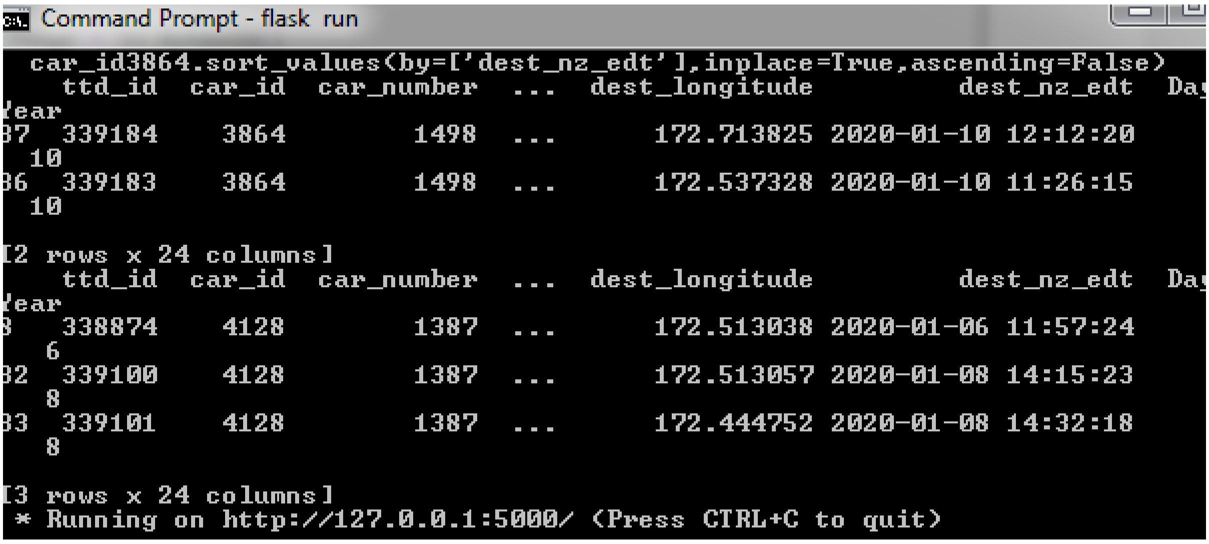
d) The libraries used such as **matplotlib, pandas , io** have to be loaded separately through the command prompt along with flask through the compiler (since I have used **Jupyter notebook** through **Anaconda**)

e) Setting up a virtual environment for the python file containing flask to connect to the localhost.



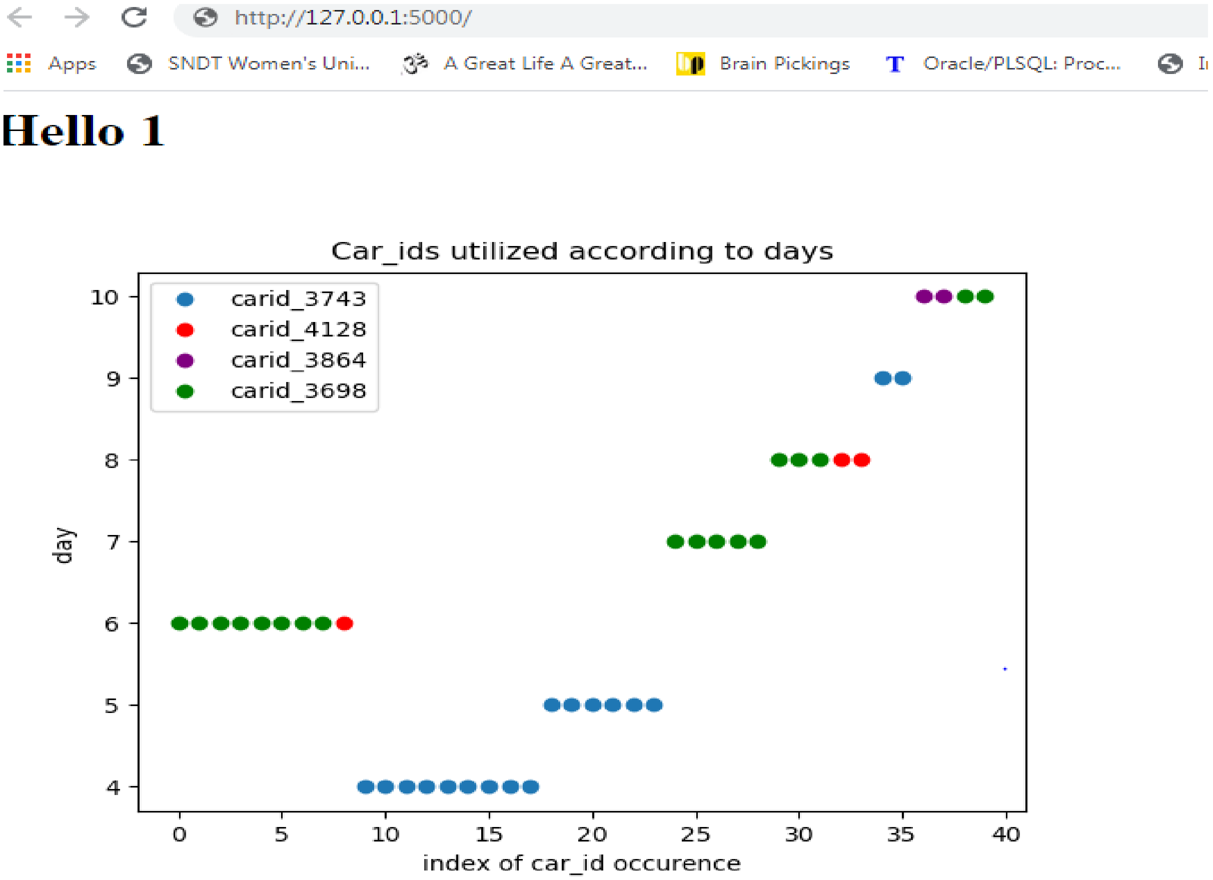
```
env) C:\Users\user\flask_app>flask run
* Serving Flask app "app.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployme
t.
  Use a production WSGI server instead.
* Debug mode: off
```

The localhost would run at the given port.



```
 car_id3864.sort_values(by=['dest_nz_edt'],inplace=True,ascending=False)
    ttd_id  car_id  car_number  ...  dest_longitude          dest_nz_edt  Da
Year
37  339184    3864        1498  ...      172.713825 2020-01-10 12:12:20
 10
36  339183    3864        1498  ...      172.537328 2020-01-10 11:26:15
 10

[2 rows x 24 columns]
    ttd_id  car_id  car_number  ...  dest_longitude          dest_nz_edt  Da
Year
8   338874    4128        1387  ...      172.513038 2020-01-06 11:57:24
  6
32  339100    4128        1387  ...      172.513057 2020-01-08 14:15:23
  8
33  339101    4128        1387  ...      172.444752 2020-01-08 14:32:18
  8

[3 rows x 24 columns]
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Localhost at 127.0.0.1:5000



Hello 1