In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
'''
1 ) "uber-raw-data-janjune-15.csv" ->> this data contains all the entries/pickups from 'January' to 'June'
    Quite huge dataset having approx 15M data pts , so lets consider its sample which have approx 1M


2 ) "uber-raw-data-janjune-15_sample.csv" ->> this data is a sample of "uber-raw-data-janjune-15.csv"
    'Since above data is quite huge ~15 Million data pts , hence it is good to work with some sample
     if u do not have good specifications in your systems

'''
```

In [ ]:

1.. Lets Read data for Analysis

In [1]:
```
### lets import all the necessary packages !

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
import os
```

In [3]:
```
os.listdir(r"Z:\Data_Analysis_Projects\Uber\Datasets")
```

Out[3]:
```
['other-American_B01362.csv',
 'other-Carmel_B00256.csv',
 'other-Dial7_B00887.csv',
 'other-Diplo_B01196.csv',
 'other-Federal_02216.csv',
 'other-FHV-services_jan-aug-2015.csv',
 'other-Firstclass_B01536.csv',
 'other-Highclass_B01717.csv',
 'other-Lyft_B02510.csv',
 'other-Prestige_B01338.csv',
 'other-Skyline_B00111.csv',
 'Uber-Jan-Feb-FOIL.csv',
 'uber-raw-data-apr14.csv',
 'uber-raw-data-aug14.csv',
 'uber-raw-data-janjune-15.csv',
 'uber-raw-data-janjune-15_sample.csv',
 'uber-raw-data-jul14.csv',
 'uber-raw-data-jun14.csv',
 'uber-raw-data-may14.csv',
 'uber-raw-data-sep14.csv']
```

In [5]:
```
uber_15 = pd.read_csv(r"Z:\Data_Analysis_Projects\Uber\Datasets/uber-raw-data-janjune-15_sample.csv")
```

In [6]:
```
uber_15.shape
```

Out[6]:
```
(100000, 4)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2.. Lets Perform Data pre-processing/Data cleaning !

check data-type , check missing values , check whether duplicated values or not !
ie Prepare Data for Analysis !

In [7]:
```
type(uber_15)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [9]:
```
uber_15.duplicated().sum()
```

Out[9]:
```
54
```

In [10]:
```
uber_15.drop_duplicates(inplace=True)
```

In [11]:
```
uber_15.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
uber_15.shape
```

Out[12]:
```
(99946, 4)
```

In [13]:
```
uber_15.dtypes
```

Out[13]:
```
Dispatching_base_num    object
Pickup_date             object
Affiliated_base_num     object
locationID               int64
dtype: object
```

In [15]:
```
uber_15.isnull().sum()
```

Out[15]:
```
Dispatching_base_num       0
Pickup_date                0
Affiliated_base_num     1116
locationID                 0
dtype: int64
```

In [17]:
```
uber_15['Pickup_date'][0]
```

Out[17]:
```
'2015-05-02 21:43:00'
```

In [18]:
```
type(uber_15['Pickup_date'][0])
```

Out[18]:
```
str
```

In [20]:
```
uber_15['Pickup_date'] = pd.to_datetime(uber_15['Pickup_date'])
```

In [21]:
```
uber_15['Pickup_date'].dtype
```

Out[21]:
```
dtype('<M8[ns]')
```

In [22]:
```
uber_15['Pickup_date'][0]
```

Out[22]:
```
Timestamp('2015-05-02 21:43:00')
```

In [23]:
```
type(uber_15['Pickup_date'][0])
```

Out[23]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

In [24]:
```
uber_15.dtypes
```

Out[24]:
```
Dispatching_base_num           object
Pickup_date             datetime64[ns]
Affiliated_base_num            object
locationID                      int64
dtype: object
```

In [ ]:
```
'''
datetime64[ns] is a general dtype, while <M8[ns] is a specific dtype , ns is basicaly nano second..
Both are similar , it entirely how your numpy was compiled..

If u want to cross check using Code :
np.dtype('datetime64[ns]') == np.dtype('<M8[ns]')



'''
```

In [ ]:
```
'''
Categorical data has : Object & bool data-types
Numerical data have : Integer & Float data-type


Categorical data refers to a data type that can be stored into groups/categories/labels
Examples of categorical variables are  age group, blood type etc..


Numerical data refers to the data that is in the form of numbers,
Examples of numerical data are height, weight, age etc..

Numerical data has two categories: discrete data and continuous data


Discrete data : It basically takes countable numbers like 1, 2, 3, 4, 5, and so on.
                age of a fly : 8 , 9 day etc..

Continuous data : which is continuous in nature
                  amount of sugar , 11.2 kg  , temp of a city  , your bank balance !



'''
```

In [ ]:
```
'''

Variations of int are : ('int64','int32','int16') in numpy library..


Int16 is a 16 bit signed integer , it means it can store both positive & negative values
int16 has has a range of  (2^15 – 1) to -2^15
int16 has a length of 16 bits (2 bytes).. ie Int16 uses 16 bits


Int32 is a 32 bit signed integer , it means it storesboth positive & negative values
int32 has has a range of (2³¹ – 1) to  -2^31
int32 has a length of 32 bits (4 bytes),, ie Int32 uses 32 bits


Int64 is a 64 bit signed integer , it means it can store both positive & negative values
int64 has has a range of  (2^63 – 1) to -2^63
int64 has a length of 64 bits (8 bytes) , ie Int64 uses 64 bits.


The only difference is that int64 has max range of storing numbers , then comes int32 , then 16 , then int8

That means that Int64’s take up twice as much memory-and doing
operations on them may be a lot slower in some machine architectures.

However, Int64’s can represent numbers much more accurately than
32 bit floats.They also allow much larger numbers to be stored..

'''
```

In [ ]:
```
'''

Variations of unsigned integer are : ('uint64','uint32','uint16','uint8') in numpy library..
By the way , all the variations of signed integers comes sub-class numpy.unsignedinteger

uint8 is a 8 bit un-signed integer , it means it can store only positive values
Range->> Integer values from (0 to 255) ie [0 to 2^8 -1]
uint8 has a length of 8 bits (1 bytes).

uint16 is a 16 bit un-signed integer , it means it can store only positive values
Range->> Integer values from (0 to 65535) ie [0 to 2^16 -1]
uint16 has a length of 16 bits (2 bytes).


uint32 is a 32 bit un-signed integer , it means it can store only positive values
Range->> Integer values from (0 to 4294967295) ie [0 to 2^32 -1]
uint32 has a length of 32 bits (4 bytes).


uint64 is a 64 bit un-signed integer , it means it can store only positive values
Range->> Integer values from (0 to 18446744073709551615) ie [0 to 2^64 -1]
uint64 has a length of 64 bits (8 bytes).

'''
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

3.. Which month have max. Uber pickups in New York City ?

In [25]:
```
uber_15
```

Out[25]:

	Dispatching_base_num	Pickup_date	Affiliated_base_num	locationID
0	B02617	2015-05-02 21:43:00	B02764	237
1	B02682	2015-01-20 19:52:59	B02682	231
2	B02617	2015-03-19 20:26:00	B02617	161
3	B02764	2015-04-10 17:38:00	B02764	107
4	B02764	2015-03-23 07:03:00	B00111	140
...	...	...	...	...
99995	B02764	2015-04-13 16:12:00	B02764	234
99996	B02764	2015-03-06 21:32:00	B02764	24
99997	B02598	2015-03-19 19:56:00	B02598	17
99998	B02682	2015-05-02 16:02:00	B02682	68
99999	B02764	2015-06-24 16:04:00	B02764	125

99946 rows × 4 columns

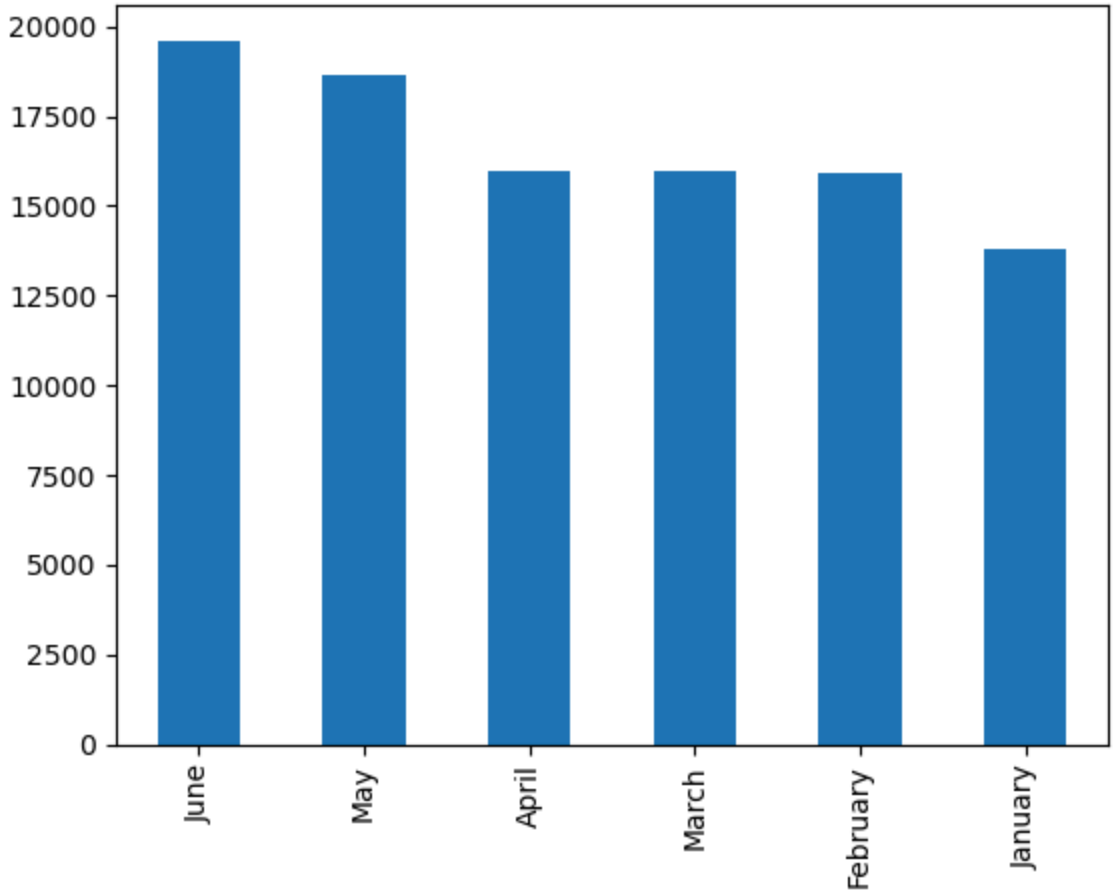
In [29]:
```
uber_15['month'] = uber_15['Pickup_date'].dt.month_name()
```

In [30]:
```
uber_15['month']
```

Out[30]:
```
0           May
1       January
2         March
3         April
4         March
          ...
99995     April
99996     March
99997     March
99998       May
99999      June
Name: month, Length: 99946, dtype: object
```

In [33]:
```
uber_15['month'].value_counts().plot(kind='bar')
```

Out[33]: <AxesSubplot:>



In [ ]:
```
'''
Inference : June seems to have max Uber Pickups

'''
```

In [ ]:
```

```

In [ ]:

In [36]:
```
## extracting dervied features (weekday ,day ,hour ,month ,minute) from 'Pickup_date'..

uber_15['weekday'] = uber_15['Pickup_date'].dt.day_name()
uber_15['day'] = uber_15['Pickup_date'].dt.day
uber_15['hour'] = uber_15['Pickup_date'].dt.hour
uber_15['minute'] = uber_15['Pickup_date'].dt.minute
```

In [37]:
```
uber_15.head(4)
```

Out[37]:

	Dispatching_base_num	Pickup_date	Affiliated_base_num	locationID	month	weekday	day	hour	minute
0	B02617	2015-05-02 21:43:00	B02764	237	May	Saturday	2	21	43
1	B02682	2015-01-20 19:52:59	B02682	231	January	Tuesday	20	19	52
2	B02617	2015-03-19 20:26:00	B02617	161	March	Thursday	19	20	26
3	B02764	2015-04-10 17:38:00	B02764	107	April	Friday	10	17	38

In [ ]:

In [39]:
```
## pd.crosstab() is used to create pivot table ..

pivot = pd.crosstab(index=uber_15['month'] , columns=uber_15['weekday'])
```

In [40]:
```
pivot
```

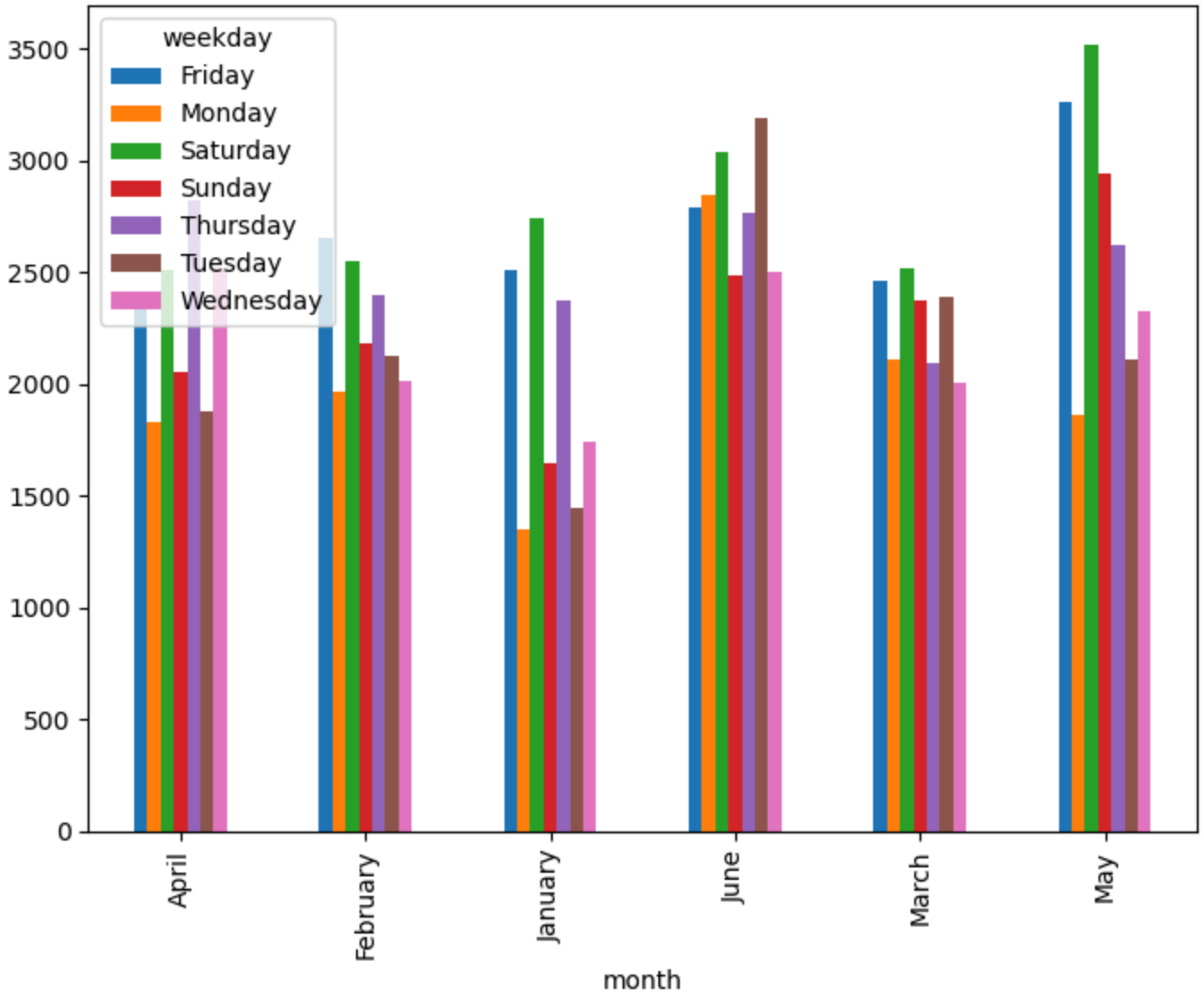
Out[40]:

weekday	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
month							
April	2365	1833	2508	2052	2823	1880	2521
February	2655	1970	2550	2183	2396	2129	2013
January	2508	1353	2745	1651	2378	1444	1740
June	2793	2848	3037	2485	2767	3187	2503
March	2465	2115	2522	2379	2093	2388	2007
May	3262	1865	3519	2944	2627	2115	2328

In [ ]:

In [42]:
```
## grouped-bar plot using Pandas ..
pivot.plot(kind='bar' , figsize=(8,6))
```

Out[42]: <AxesSubplot:xlabel='month'>



In [ ]:
```
'''

On Saturday & Friday, u are getting more Uber pickups in each month , it seems that New Yorkers used to go for
shopping , Malls , fun activities alot on these days

'''
```

In [ ]:

In [ ]:

In [ ]:

4.. Lets Find out Hourly Rush in New york city on all days

In [45]:
```
summary = uber_15.groupby(['weekday' , 'hour'] , as_index=False).size()
```

In [46]:
```
summary
```

Out[46]:

	weekday	hour	size
0	Friday	0	581
1	Friday	1	333
2	Friday	2	197
3	Friday	3	138
4	Friday	4	161
...	...	...	...
163	Wednesday	19	1044
164	Wednesday	20	897
165	Wednesday	21	949
166	Wednesday	22	900
167	Wednesday	23	669

168 rows × 3 columns

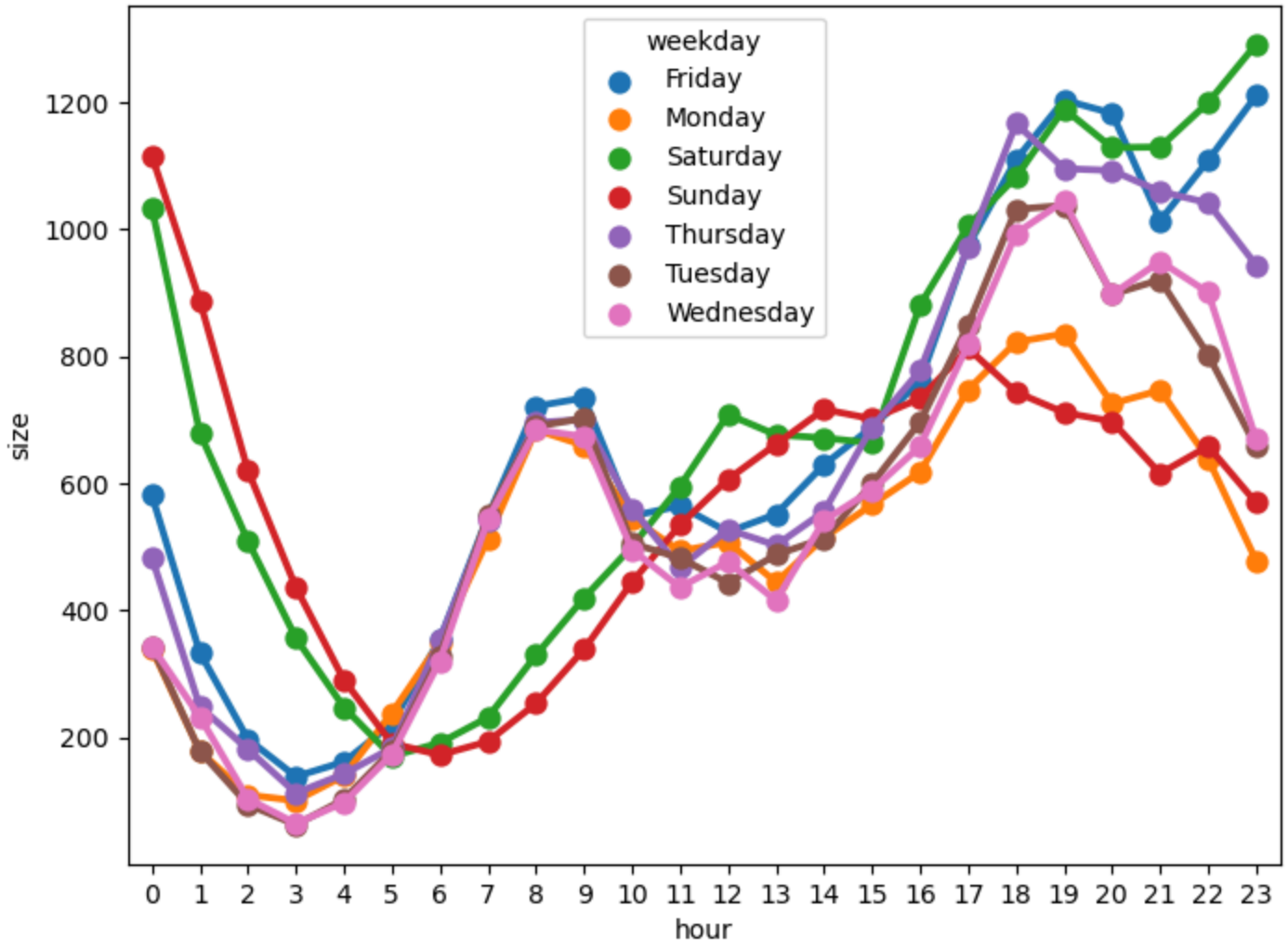
In [ ]:
```

```

In [48]:
```
## pointplot between 'hour' & 'size' for all the weekdays..

plt.figure(figsize=(8,6))
sns.pointplot(x="hour" , y="size" , hue="weekday" , data=summary)
```

Out[48]: <AxesSubplot:xlabel='hour', ylabel='size'>



In [ ]:
```
'''
It's interesting to see that Saturday and Sunday exhibit similar demand throughout the late night/morning/afte
but it exhibits opposite trends during the evening. In the evening, Saturday pickups continue to increase thro
but Sunday pickups takes a downward turn after evening..

We can see that there the weekdays that has the most demand during the late evening is Friday and Saturday,
which is expected, but what strikes me is that Thursday nights also exhibits very similar trends as Friday and

It seems like New Yorkers are starting their 'weekends' on Thursday nights. :)


'''
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

5.. Which Base_number has most number of Active Vehicles ??

In [49]:
```
uber_15.columns
```

Out[49]:
```
Index(['Dispatching_base_num', 'Pickup_date', 'Affiliated_base_num',
       'locationID', 'month', 'weekday', 'day', 'hour', 'minute'],
      dtype='object')
```

In [ ]:

In [50]:
```
os.listdir(r"Z:\Data_Analysis_Projects\Uber\Datasets")
```

Out[50]:
```
['other-American_B01362.csv',
 'other-Carmel_B00256.csv',
 'other-Dial7_B00887.csv',
 'other-Diplo_B01196.csv',
 'other-Federal_02216.csv',
 'other-FHV-services_jan-aug-2015.csv',
 'other-Firstclass_B01536.csv',
 'other-Highclass_B01717.csv',
 'other-Lyft_B02510.csv',
 'other-Prestige_B01338.csv',
 'other-Skyline_B00111.csv',
 'Uber-Jan-Feb-FOIL.csv',
 'uber-raw-data-apr14.csv',
 'uber-raw-data-aug14.csv',
 'uber-raw-data-janjune-15.csv',
 'uber-raw-data-janjune-15_sample.csv',
 'uber-raw-data-jul14.csv',
 'uber-raw-data-jun14.csv',
 'uber-raw-data-may14.csv',
 'uber-raw-data-sep14.csv']
```

In [52]:
```
uber_foil = pd.read_csv(r"Z:\Data_Analysis_Projects\Uber\Datasets/Uber-Jan-Feb-FOIL.csv")
```

In [53]:
```
uber_foil.shape
```

Out[53]:
```
(354, 4)
```

In [54]:
```
uber_foil.head(3)
```

Out[54]:

	dispatching_base_number	date	active_vehicles	trips
0	B02512	1/1/2015	190	1132
1	B02765	1/1/2015	225	1765
2	B02764	1/1/2015	3427	29421

In [ ]:
```
```

In [ ]:
```
### establishing the entire set-up of Plotly..
```

In [ ]:
```
!pip install chart_studio ## chart_studio provides a web-service for hosting graphs!
!pip install plotly
```

In [55]:
```
import chart_studio.plotly as py
import plotly.graph_objs as go
import plotly.express as px

from plotly.offline import download_plotlyjs , init_notebook_mode , plot , iplot
## iplot() when working in a Jupyter Notebook to display the plot in the notebook.
## U have to do a proper setup of plotly , otherwise plotly plots gets open in a web-browser instead of Jupyte
```

In [56]:
```
init_notebook_mode(connected=True)
```

In [57]:
```
uber_foil.columns
```

Out[57]:
```
Index(['dispatching_base_number', 'date', 'active_vehicles', 'trips'], dtype='object')
```

In [59]:
```
px.box(x='dispatching_base_number' , y='active_vehicles' , data_frame=uber_foil)
```

In [ ]:
```

```

In [60]:
```
### if u need distribution +  5-summary stats of data , its good to go with violinplot
px.violin(x='dispatching_base_number' , y='active_vehicles' , data_frame=uber_foil)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

6.. Collect entire data & Make it ready for the Data Analysis..

In [63]:
```
files = os.listdir(r"Z:\Data_Analysis_Projects\Uber\Datasets")[-8:]
```

In [65]:
```
files.remove('uber-raw-data-janjune-15.csv')
```

In [66]:
```
files
```

Out[66]:
```
['uber-raw-data-apr14.csv',
 'uber-raw-data-aug14.csv',
 'uber-raw-data-janjune-15_sample.csv',
 'uber-raw-data-jul14.csv',
 'uber-raw-data-jun14.csv',
 'uber-raw-data-may14.csv',
 'uber-raw-data-sep14.csv']
```

In [67]:
```
files.remove('uber-raw-data-janjune-15_sample.csv')
```

In [68]:
```
files
```

Out[68]:
```
['uber-raw-data-apr14.csv',
 'uber-raw-data-aug14.csv',
 'uber-raw-data-jul14.csv',
 'uber-raw-data-jun14.csv',
 'uber-raw-data-may14.csv',
 'uber-raw-data-sep14.csv']
```

In [ ]:

In [ ]:

In [69]:
```
#blank dataframe
final = pd.DataFrame()

path = r"Z:\Data_Analysis_Projects\Uber\Datasets"

for file in files :
    current_df = pd.read_csv(path+'/'+file)
    final = pd.concat([current_df , final])
```

In [70]:
```
final.shape
```

Out[70]: (4534327, 4)

In [ ]:
```
### After Collecting entire data ,u might ask is : Do we have duplicate entires in data ?
### We are going to remove duplicates data when the entire row is duplicated
```

In [71]:
```
### first lets figure out total observations where we have duplicate values..
final.duplicated().sum()
```

Out[71]: 82581

In [72]:
```
## drop duplicate rows ..
final.drop_duplicates(inplace=True)
```

In [73]:
```
final.shape
```

Out[73]: (4451746, 4)

In [74]:
```
final.head(3)
```

Out[74]:

	Date/Time	Lat	Lon	Base
0	9/1/2014 0:01:00	40.2201	-74.0021	B02512
1	9/1/2014 0:01:00	40.7500	-74.0027	B02512
2	9/1/2014 0:03:00	40.7559	-73.9864	B02512

Dataset Information :

The dataset contains information about the Datetime, Latitude, Longitude and Base of each uber ride that happened in the month of July 2014 at New York City, USA

Date/Time : The date and time of the Uber pickup

Lat : The latitude of the Uber pickup

Lon : The longitude of the Uber pickup

Base : The TLC base company code affiliated with the Uber pickup

```
The Base codes are for the following Uber bases:
B02512 : Unter
B02598 : Hinter
B02617 : Weiter
B02682 : Schmecken
B02764 : Danach-NY
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

7.. at what locations of New York City we are getting rush ??

In [ ]:
```
### ie where-ever we have more data-points or more density, it means more rush is at there !
```

In [76]:
```
rush_uber = final.groupby(['Lat' , 'Lon'] , as_index=False).size()
```

In [77]:
```
rush_uber.head(6)
```

Out[77]:

	Lat	Lon	size
0	39.6569	-74.2258	1
1	39.6686	-74.1607	1
2	39.7214	-74.2446	1
3	39.8416	-74.1512	1
4	39.9055	-74.0791	1
5	39.9196	-74.1112	1

In [ ]:

In [ ]:

In [ ]:
```
!pip install folium
```

In [78]:
```
import folium
```

In [79]:
```
basemap = folium.Map()
```

In [80]:
```
basemap
```

Out[80]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

In [ ]:

In [81]:
```
from folium.plugins import HeatMap
```

In [82]:
```
HeatMap(rush_uber).add_to(basemap)
```

Out[82]:
```
<folium.plugins.heat_map.HeatMap at 0x23becbedd60>
```

In [83]:
```
basemap
```

Out[83]: Make this Notebook Trusted to load map: File -> Trust Notebook

We can see a number of hot spots here. Midtown Manhattan is clearly a huge bright spot
& these are made from Midtown to Lower Manhattan followed by Upper Manhattan and the Heights of Brook
lyn.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

8.. Examine rush on Hour and Weekday (Perform Pair wise Analysis)

In [84]:
```
final.columns
```

Out[84]:
```
Index(['Date/Time', 'Lat', 'Lon', 'Base'], dtype='object')
```

In [85]:
```
final.head(3)
```

Out[85]:

	Date/Time	Lat	Lon	Base
0	9/1/2014 0:01:00	40.2201	-74.0021	B02512
1	9/1/2014 0:01:00	40.7500	-74.0027	B02512
2	9/1/2014 0:03:00	40.7559	-73.9864	B02512

In [86]:
```
final.dtypes
```

Out[86]:
```
Date/Time     object
Lat          float64
Lon          float64
Base          object
dtype: object
```

In [88]:
```
final['Date/Time'][0]
```

Out[88]:
```
0    9/1/2014 0:01:00
0    5/1/2014 0:02:00
0    6/1/2014 0:00:00
0    7/1/2014 0:03:00
0    8/1/2014 0:03:00
0    4/1/2014 0:11:00
Name: Date/Time, dtype: object
```

In [89]:
```
### converting 'Date/Time' feature into date-time..

final['Date/Time'] = pd.to_datetime(final['Date/Time'] , format="%m/%d/%Y %H:%M:%S")
```

In [91]:
```
final['Date/Time'].dtype
```

Out[91]:
```
dtype('<M8[ns]')
```

In [ ]:
```

```

In [93]:
```
### extracting 'weekday' & 'hour' from 'Date/Time' feature..

final['day'] = final['Date/Time'].dt.day
final['hour'] = final['Date/Time'].dt.hour
```

In [94]:
```
final.head(4)
```

Out[94]:

	Date/Time	Lat	Lon	Base	day	hour
0	2014-09-01 00:01:00	40.2201	-74.0021	B02512	1	0
1	2014-09-01 00:01:00	40.7500	-74.0027	B02512	1	0
2	2014-09-01 00:03:00	40.7559	-73.9864	B02512	1	0
3	2014-09-01 00:06:00	40.7450	-73.9889	B02512	1	0

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
'''
Earlier we have learnt how to create pivot table using pd.crosstab() , now let me show u one more way to build
pivot_table without pd.crosstab()

'''
```

In [100]:
```
pivot = final.groupby(['day' , 'hour']).size().unstack()
```

In [101]:
```
pivot

### pivot table is all about  , we have Rows*columns & having value in each cell !
```

Out[101]:

hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22
day																				
1	3178	1944	1256	1308	1429	2126	3664	5380	5292	4617	...	6933	7910	8633	9511	8604	8001	7315	7803	6268
2	2435	1569	1087	1414	1876	2812	4920	6544	6310	4712	...	6904	8449	10109	11100	11123	9474	8759	8357	6998
3	3354	2142	1407	1467	1550	2387	4241	5663	5386	4657	...	7226	8850	10314	10491	11239	9599	9026	8531	7142
4	2897	1688	1199	1424	1696	2581	4592	6029	5704	4744	...	7158	8515	9492	10357	10259	9097	8358	8649	7706
5	2733	1541	1030	1253	1617	2900	4814	6261	6469	5530	...	6955	8312	9609	10699	10170	9430	9354	9610	8853
6	4537	2864	1864	1555	1551	2162	3642	4766	4942	4401	...	7235	8612	9444	9929	9263	8405	8117	8567	7852
7	3645	2296	1507	1597	1763	2422	4102	5575	5376	4639	...	7276	8474	10393	11013	10573	9472	8691	8525	7194
8	2830	1646	1123	1483	1889	3224	5431	7361	7357	5703	...	7240	8775	9851	10673	9687	8796	8604	8367	6795
9	2657	1724	1222	1480	1871	3168	5802	7592	7519	5895	...	7877	9220	10270	11910	11449	9804	8909	8665	7499
10	3296	2126	1464	1434	1591	2594	4664	6046	6158	5072	...	7612	9578	11045	11875	10934	9613	9687	9240	7766
11	3036	1665	1095	1424	1842	2520	4954	6876	6871	5396	...	7503	8920	10125	10898	10361	9327	8824	8730	7771
12	3227	2147	1393	1362	1757	2710	4576	6250	6231	5177	...	7743	9390	10734	11713	12216	10393	9965	10310	9992
13	5408	3509	2262	1832	1705	2327	4196	5685	6060	5631	...	8200	9264	10534	11826	11450	9921	8705	8423	7363
14	3748	2349	1605	1656	1756	2629	4257	5781	5520	4824	...	6963	8192	9511	10115	9553	9146	9182	8589	6891
15	2497	1515	1087	1381	1862	2980	5050	6837	6729	5201	...	7633	8505	10285	11959	11728	11032	10509	9105	7153
16	2547	1585	1119	1395	1818	2966	5558	7517	7495	5958	...	7597	9290	10804	11773	10855	10924	10142	10374	8094
17	3155	2048	1500	1488	1897	2741	4562	6315	5882	4934	...	7472	8997	10323	11236	11089	9919	9935	9823	8362
18	3390	2135	1332	1626	1892	2959	4688	6618	6451	5377	...	7534	9040	10274	10692	10338	9551	9310	9285	8015
19	3217	2188	1604	1675	1810	2639	4733	6159	6014	5006	...	7374	8898	9893	10741	10429	9701	10051	10049	9090
20	4475	3190	2100	1858	1618	2143	3584	4900	5083	4765	...	7462	8630	9448	10046	9272	8592	8614	8703	7787
21	4294	3194	1972	1727	1926	2615	4185	5727	5529	4707	...	7064	8127	9483	9817	9291	8317	8107	8245	7362
22	2787	1637	1175	1468	1934	3151	5204	6872	6850	5198	...	7337	9148	10574	10962	9884	8980	8772	8430	6784
23	2546	1580	1136	1429	1957	3132	5204	6890	6436	5177	...	7575	9309	9980	10341	10823	11347	11447	10347	8637
24	3200	2055	1438	1493	1798	2754	4484	6013	5913	5146	...	7083	8706	10366	10786	9772	9080	9213	8831	7480
25	2405	1499	1072	1439	1943	2973	5356	7627	7078	5994	...	7298	8732	9922	10504	10673	9048	8751	9508	8522
26	3810	3065	2046	1806	1730	2337	3776	5172	5071	4808	...	7269	8815	9885	10697	10867	10122	9820	10441	9486

hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22
day																				
27	5196	3635	2352	2055	1723	2336	3539	4937	5053	4771	...	7519	8803	9793	9838	9228	8267	7908	8507	7720
28	4123	2646	1843	1802	1883	2793	4290	5715	5671	5206	...	7341	8584	9671	9975	9132	8255	8309	7949	6411
29	2678	1827	1409	1678	1948	3056	5213	6852	6695	5481	...	7630	9249	10105	11113	10411	9301	9270	9114	6992
30	2401	1510	1112	1403	1841	3216	5757	7596	7611	6064	...	8396	10243	11554	12126	12561	11024	10836	10042	8275
31	2174	1394	1087	919	773	997	1561	2169	2410	2525	...	4104	5099	5386	5308	5350	4898	4819	5064	5164

31 rows × 24 columns

In [ ]:

In [103]:
```
### styling dataframe

pivot.style.background_gradient()
```

Out[103]:

hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
day																				
1	3178	1944	1256	1308	1429	2126	3664	5380	5292	4617	4607	4729	4930	5794	6933	7910	8633	9511	8604	8001
2	2435	1569	1087	1414	1876	2812	4920	6544	6310	4712	4797	4975	5188	5695	6904	8449	10109	11100	11123	9474
3	3354	2142	1407	1467	1550	2387	4241	5663	5386	4657	4788	5065	5384	6093	7226	8850	10314	10491	11239	9599
4	2897	1688	1199	1424	1696	2581	4592	6029	5704	4744	4743	4975	5193	6175	7158	8515	9492	10357	10259	9097
5	2733	1541	1030	1253	1617	2900	4814	6261	6469	5530	5141	5011	5047	5690	6955	8312	9609	10699	10170	9430
6	4537	2864	1864	1555	1551	2162	3642	4766	4942	4401	4801	5174	5426	6258	7235	8612	9444	9929	9263	8405
7	3645	2296	1507	1597	1763	2422	4102	5575	5376	4639	4905	5166	5364	6214	7276	8474	10393	11013	10573	9472
8	2830	1646	1123	1483	1889	3224	5431	7361	7357	5703	5288	5350	5483	6318	7240	8775	9851	10673	9687	8796
9	2657	1724	1222	1480	1871	3168	5802	7592	7519	5895	5406	5443	5496	6419	7877	9220	10270	11910	11449	9804
10	3296	2126	1464	1434	1591	2594	4664	6046	6158	5072	4976	5415	5506	6527	7612	9578	11045	11875	10934	9613
11	3036	1665	1095	1424	1842	2520	4954	6876	6871	5396	5215	5423	5513	6486	7503	8920	10125	10898	10361	9327
12	3227	2147	1393	1362	1757	2710	4576	6250	6231	5177	5157	5319	5570	6448	7743	9390	10734	11713	12216	10393
13	5408	3509	2262	1832	1705	2327	4196	5685	6060	5631	5442	5720	5914	6678	8200	9264	10534	11826	11450	9921
14	3748	2349	1605	1656	1756	2629	4257	5781	5520	4824	4911	5118	5153	5747	6963	8192	9511	10115	9553	9146
15	2497	1515	1087	1381	1862	2980	5050	6837	6729	5201	5347	5517	5503	6997	7633	8505	10285	11959	11728	11032
16	2547	1585	1119	1395	1818	2966	5558	7517	7495	5958	5626	5480	5525	6198	7597	9290	10804	11773	10855	10924
17	3155	2048	1500	1488	1897	2741	4562	6315	5882	4934	5004	5306	5634	6507	7472	8997	10323	11236	11089	9919
18	3390	2135	1332	1626	1892	2959	4688	6618	6451	5377	5150	5487	5490	6383	7534	9040	10274	10692	10338	9551
19	3217	2188	1604	1675	1810	2639	4733	6159	6014	5006	5092	5240	5590	6367	7374	8898	9893	10741	10429	9701
20	4475	3190	2100	1858	1618	2143	3584	4900	5083	4765	5135	5650	5745	6656	7462	8630	9448	10046	9272	8592
21	4294	3194	1972	1727	1926	2615	4185	5727	5529	4707	4911	5212	5465	6085	7064	8127	9483	9817	9291	8317
22	2787	1637	1175	1468	1934	3151	5204	6872	6850	5198	5277	5352	5512	6342	7337	9148	10574	10962	9884	8980
23	2546	1580	1136	1429	1957	3132	5204	6890	6436	5177	5066	5304	5504	6232	7575	9309	9980	10341	10823	11347
24	3200	2055	1438	1493	1798	2754	4484	6013	5913	5146	4947	5311	5229	5974	7083	8706	10366	10786	9772	9080
25	2405	1499	1072	1439	1943	2973	5356	7627	7078	5994	5432	5504	5694	6204	7298	8732	9922	10504	10673	9048
26	3810	3065	2046	1806	1730	2337	3776	5172	5071	4808	5061	5179	5381	6166	7269	8815	9885	10697	10867	10122

hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
day																				
27	5196	3635	2352	2055	1723	2336	3539	4937	5053	4771	5198	5732	5839	6820	7519	8803	9793	9838	9228	8267
28	4123	2646	1843	1802	1883	2793	4290	5715	5671	5206	5247	5500	5486	6120	7341	8584	9671	9975	9132	8255
29	2678	1827	1409	1678	1948	3056	5213	6852	6695	5481	5234	5163	5220	6305	7630	9249	10105	11113	10411	9301
30	2401	1510	1112	1403	1841	3216	5757	7596	7611	6064	5987	6090	6423	7249	8396	10243	11554	12126	12561	11024
31	2174	1394	1087	919	773	997	1561	2169	2410	2525	2564	2777	2954	3280	4104	5099	5386	5308	5350	4898

In [ ]:

In [ ]:

In [ ]:

In [ ]:

9.. How to Automate Your Analysis..?

In [104]:
```
## creating a user-defined function..

def gen_pivot_table(df , col1 , col2):
    pivot = final.groupby([col1 , col2]).size().unstack()
    return pivot.style.background_gradient()
```

In [106]:
```
final.columns
```

Out[106]:
```
Index(['Date/Time', 'Lat', 'Lon', 'Base', 'day', 'hour'], dtype='object')
```

In [107]:
```
gen_pivot_table(final , "day" , "hour")
```

Out[107]:

hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
day																				
1	3178	1944	1256	1308	1429	2126	3664	5380	5292	4617	4607	4729	4930	5794	6933	7910	8633	9511	8604	8001
2	2435	1569	1087	1414	1876	2812	4920	6544	6310	4712	4797	4975	5188	5695	6904	8449	10109	11100	11123	9474
3	3354	2142	1407	1467	1550	2387	4241	5663	5386	4657	4788	5065	5384	6093	7226	8850	10314	10491	11239	9599
4	2897	1688	1199	1424	1696	2581	4592	6029	5704	4744	4743	4975	5193	6175	7158	8515	9492	10357	10259	9097
5	2733	1541	1030	1253	1617	2900	4814	6261	6469	5530	5141	5011	5047	5690	6955	8312	9609	10699	10170	9430
6	4537	2864	1864	1555	1551	2162	3642	4766	4942	4401	4801	5174	5426	6258	7235	8612	9444	9929	9263	8405
7	3645	2296	1507	1597	1763	2422	4102	5575	5376	4639	4905	5166	5364	6214	7276	8474	10393	11013	10573	9472
8	2830	1646	1123	1483	1889	3224	5431	7361	7357	5703	5288	5350	5483	6318	7240	8775	9851	10673	9687	8796
9	2657	1724	1222	1480	1871	3168	5802	7592	7519	5895	5406	5443	5496	6419	7877	9220	10270	11910	11449	9804
10	3296	2126	1464	1434	1591	2594	4664	6046	6158	5072	4976	5415	5506	6527	7612	9578	11045	11875	10934	9613
11	3036	1665	1095	1424	1842	2520	4954	6876	6871	5396	5215	5423	5513	6486	7503	8920	10125	10898	10361	9327
12	3227	2147	1393	1362	1757	2710	4576	6250	6231	5177	5157	5319	5570	6448	7743	9390	10734	11713	12216	10393
13	5408	3509	2262	1832	1705	2327	4196	5685	6060	5631	5442	5720	5914	6678	8200	9264	10534	11826	11450	9921
14	3748	2349	1605	1656	1756	2629	4257	5781	5520	4824	4911	5118	5153	5747	6963	8192	9511	10115	9553	9146
15	2497	1515	1087	1381	1862	2980	5050	6837	6729	5201	5347	5517	5503	6997	7633	8505	10285	11959	11728	11032
16	2547	1585	1119	1395	1818	2966	5558	7517	7495	5958	5626	5480	5525	6198	7597	9290	10804	11773	10855	10924
17	3155	2048	1500	1488	1897	2741	4562	6315	5882	4934	5004	5306	5634	6507	7472	8997	10323	11236	11089	9919
18	3390	2135	1332	1626	1892	2959	4688	6618	6451	5377	5150	5487	5490	6383	7534	9040	10274	10692	10338	9551
19	3217	2188	1604	1675	1810	2639	4733	6159	6014	5006	5092	5240	5590	6367	7374	8898	9893	10741	10429	9701
20	4475	3190	2100	1858	1618	2143	3584	4900	5083	4765	5135	5650	5745	6656	7462	8630	9448	10046	9272	8592
21	4294	3194	1972	1727	1926	2615	4185	5727	5529	4707	4911	5212	5465	6085	7064	8127	9483	9817	9291	8317
22	2787	1637	1175	1468	1934	3151	5204	6872	6850	5198	5277	5352	5512	6342	7337	9148	10574	10962	9884	8980
23	2546	1580	1136	1429	1957	3132	5204	6890	6436	5177	5066	5304	5504	6232	7575	9309	9980	10341	10823	11347
24	3200	2055	1438	1493	1798	2754	4484	6013	5913	5146	4947	5311	5229	5974	7083	8706	10366	10786	9772	9080
25	2405	1499	1072	1439	1943	2973	5356	7627	7078	5994	5432	5504	5694	6204	7298	8732	9922	10504	10673	9048
26	3810	3065	2046	1806	1730	2337	3776	5172	5071	4808	5061	5179	5381	6166	7269	8815	9885	10697	10867	10122

hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
day																				
27	5196	3635	2352	2055	1723	2336	3539	4937	5053	4771	5198	5732	5839	6820	7519	8803	9793	9838	9228	8267
28	4123	2646	1843	1802	1883	2793	4290	5715	5671	5206	5247	5500	5486	6120	7341	8584	9671	9975	9132	8255
29	2678	1827	1409	1678	1948	3056	5213	6852	6695	5481	5234	5163	5220	6305	7630	9249	10105	11113	10411	9301
30	2401	1510	1112	1403	1841	3216	5757	7596	7611	6064	5987	6090	6423	7249	8396	10243	11554	12126	12561	11024
31	2174	1394	1087	919	773	997	1561	2169	2410	2525	2564	2777	2954	3280	4104	5099	5386	5308	5350	4898

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: