CHICAGO TAXI SERVICE – EDA Analysis USING PYSPARK

PROJECT REPORT

Done By

BHASHEYAM KRISHNAN

A20380078

Table of Content:

1. Introduction:

City life style force people to use to Taxi or cab service as maintaining a parking system for all city user is not feasible. For the cost of parking people can commute with Taxi cab. So, the taxi use is one of the essentials for downtown travellers. Though Downtown tend to have more public transport than suburbs. But taxi is one of the comfortable and less time-consuming means. We are not very far from automated Cab services many car firm has already invested in the development of Automated cab.

2. Problem Statement:

Use of the Chicago taxi trip dataset to perform an EDA analysis to see the interesting facts, trend, user preference, stats of the usage, Timeline, peak hour. This will give a rough idea of the people expectation and usage of taxi. This can used to develop new feature to improve the business, Modify features.

3. Dataset Description:

Data became one of the essential like water, gas, electricity, internet. Everyone of us generate enormous data everyday though electronic Gadgets, sensors, APPs. To understand the Trend, flow, drawbacks, improvement areas, Predict, Classify, Measure Performance, develop models, Maintain Business. For the Problem statement we need data of user who used Taxi in the Past 4-5 years. Details like time of use, Cost, Payment mode, Peoples likes, amount spent on fare, toll, tip. These are the some of the factor which will be useful to see how good this service was accepted in market, which are the areas needed to improve when we want to establish the market in new city or different place.

Best means to collect these data are collecting information from the Taxi company. They would have stored all this info collected through Mobile app used by the user as it in-built map service as well. Since people started using Mobile App for Taxi service it is feasible to collect all these information with ease. It is impossible to collect the information manually and it won't accurate as well.

Dataset consist of u'Trip ID,Taxi ID,Trip Start Timestamp,Trip End Timestamp,Trip Seconds,Trip Miles,Pickup Census Tract,Dropoff Census Tract,Pickup Community Area,Dropoff Community Area,Fare,Tips,Tolls,Extras,Trip Total,Payment Type,Company,Pickup Centroid Latitude,Pickup Centroid Longitude,Pickup Centroid Location,Dropoff Centroid Latitude,Dropoff Centroid Longitude,Dropoff Centroid Location,Community Areas.

4. Technologies Used:

- Since dataset is large we need shared file system and parallel computing technologies to work on the stat.
- To perform these tasks, we need environment which can support parallel processing and file system.
- Physical and local system are expensive to maintain and purchase, so using Cloud service setting the environment.
- When it comes to count and stat, map reduce, and Spark technologies features perform the best.
- Spark provide the advantages over the loop processing, using Pyspark performing the analysis.
- Store the data in cloud store service and access the data using EMR cluster.

- For large quantity of Data place, the data in the redshift cluster and preform the analysis.
- Results of the query and analysis are stored in S3 and using excel or R(GGplot) to draw graph.

- **Technologies:**

a. **AWS EMR:**

Amazon EMR provides a managed Hadoop framework that makes it easy, fast, and cost-effective to process vast amounts of data across dynamically scalable Amazon EC2 instances.

b. **AWS s3:**

S3 – Simple Store Service is the data store Service used to store a virtually unlimited data in cloud. It is highly scalable, secure, durable and available. Data in S3 are accessed using Rest calls.

c. **Apache pySpark:**

Python spark is analytics service provided by Apache which run top HDFS, Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.

Load the data in S3 and Using EMR Cluster service to setup a parallel master-slave environment to run pyspark on the cluster. AWS EMR which provide high performance cluster and compatibility with S3 Storage. AWS S3 is virtually unlimited data storable option. It is easy to maintain the data in the cloud with highly scalable and highly available solution. AWS EMR and S3 provide an efficient way to maintain Data and clean up.

5. Analysis plan and setup:

- When it comes to EDA all are stats and number are important factor. To analysis the trend or to understand how it works we need the numbers. So, for this problem statement areas to concentrated are Fare, Distance, count, usage, frequency, trend in the year.
- Find the overall stats for the data we have and calculate year wise to See the variation in the usage over the years. Year is differentiated using timestamp and most of the data are numeric for find the count and stat. If we have the user reviews and comments we can do sentimental analysis on the user ratings.
- These stats will be useful to find the trend, useful information, user preference, Features need to introduce, improved, can be removed, area where we need research & marketing.

6. Analysis on the Dataset:

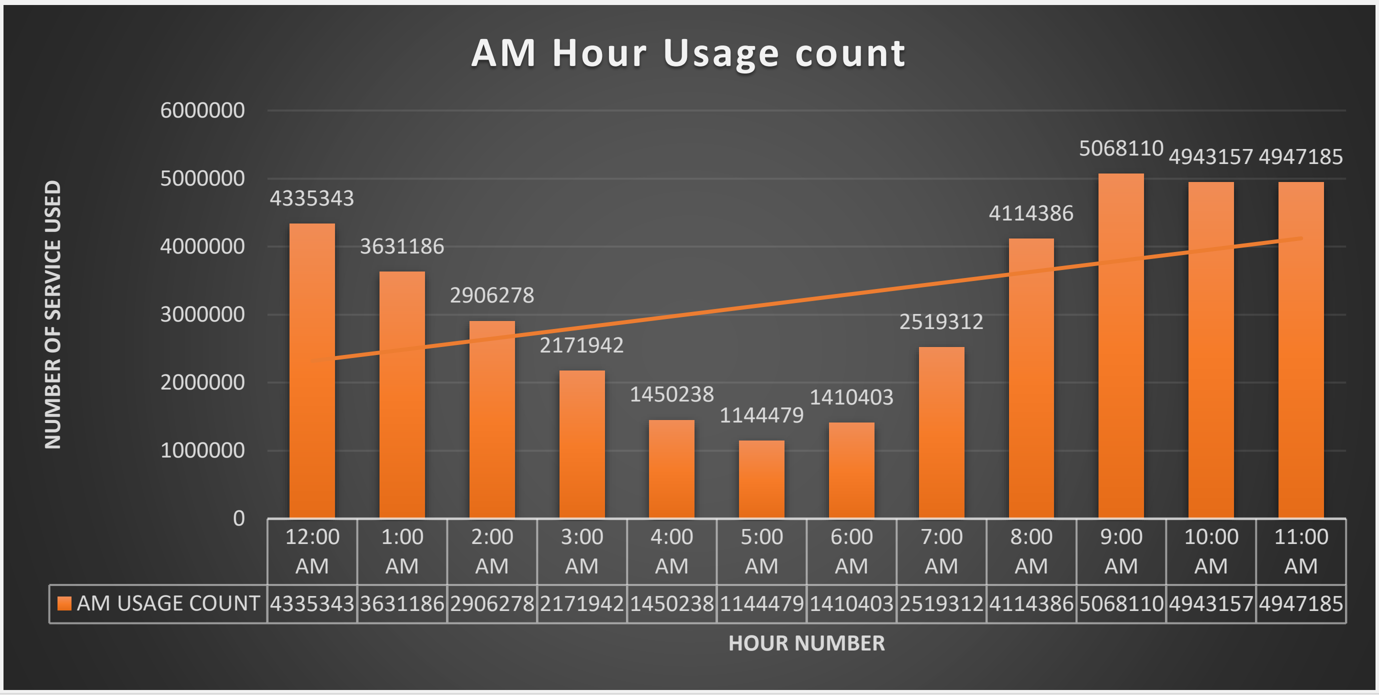
- The dataset consist of data from 2013 – 2017(mid), calculated important numbers which will be helpful to looking to the trend. Also, to see how much it was used and liked by the people.
- Stats on factors like mile, hours, Fare, Total_Fare, other Fares and how much use has paid.
- Figure in the Dataset Stats section, show important numbers involved in the dataset.
- Totally **112860054 instances** over 2013 – 2017 were used in the dataset to calculate the stats
- In total taxi service was used for **23151627 working hours** which is 964651 days.
- In total all taxi has been drove for **313221449 Mile** inside Chicago site alone in 4 years
- People have liked the service and employee are also provides good service as almost **11% of total fare is tips.**
- Firm used just **0.6% to attract with offer**, this show it is one of the essential in the city and they didn't plan or used Strategy to attract customers with Offers.

a. Dataset Stats:

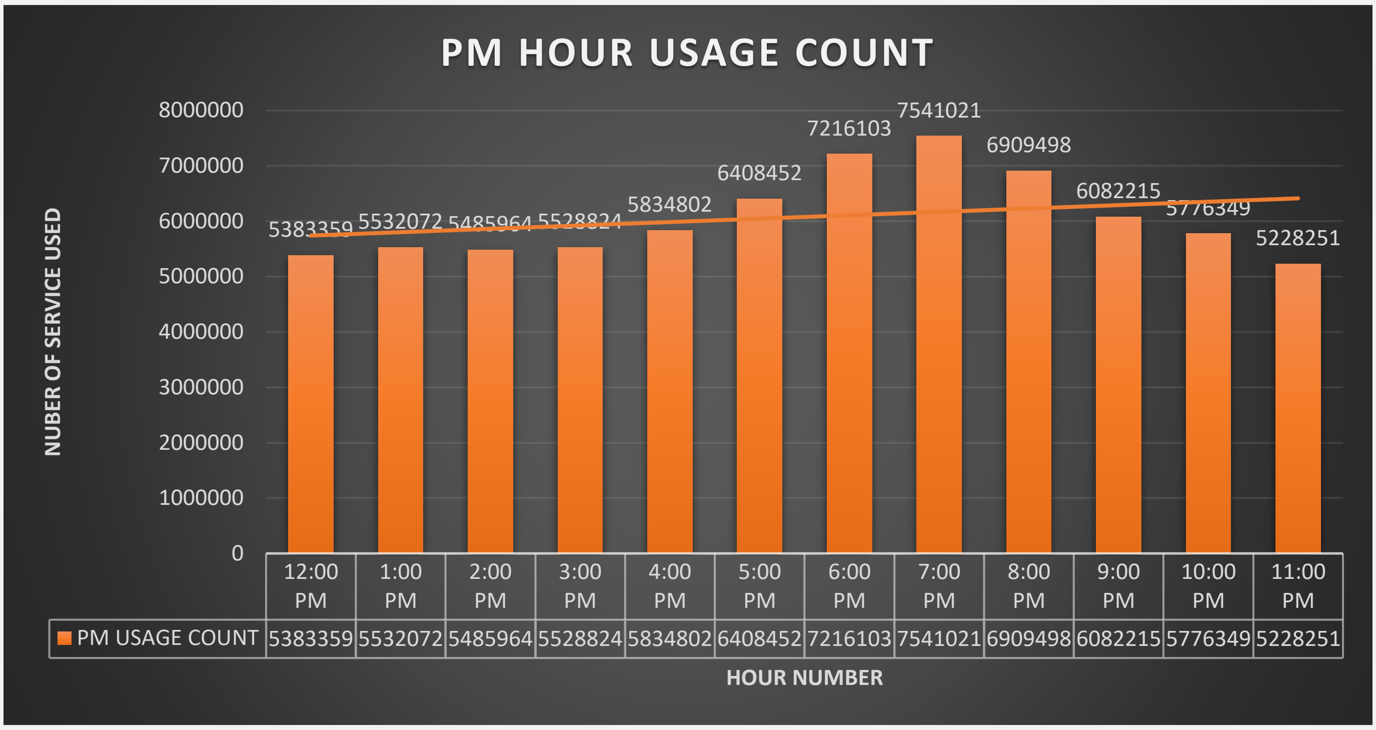
Total Stats of the Dataset		
Seconds	Hours	Days
83345858073	23151627	964651.125
Miles	Trip_Total	Per Mile
313221448.8	1597501966	5.100231712
Fare	Tips	Tip Percentage
1366630692	140897324.4	10.30983171
Tolls	Extra	Extra Percentage
884848.4	91158753.2	6.670328257
Credit card Count	Cash Count	Usage Ratio
42875936	67694669	39%:61%
NO Charge Count	Dispute Count	No Charge Percentage
665345	61084	0.60%

- Above stats shows people prefer more physical than digital transaction, but in future digital transaction are expected take over the cash payment type.
- Considerable amount of Tolls amount shows people who commute to suburbs use taxi service. But, the frequency will be very low.
- People were rEDAy to spend $5.1 per mile, which is expensive when compared to CTA or Bike service(Divy Bikes).

7. Analysis on time:



	12:00 AM	1:00 AM	2:00 AM	3:00 AM	4:00 AM	5:00 AM	6:00 AM	7:00 AM	8:00 AM	9:00 AM	10:00 AM	11:00 AM
AM USAGE COUNT	4335343	3631186	2906278	2171942	1450238	1144479	1410403	2519312	4114386	5068110	4943157	4947185



	12:00 PM	1:00 PM	2:00 PM	3:00 PM	4:00 PM	5:00 PM	6:00 PM	7:00 PM	8:00 PM	9:00 PM	10:00 PM	11:00 PM
🟧 PM USAGE COUNT	5383359	5532072	5485964	5528824	5834802	6408452	7216103	7541021	6909498	6082215	5776349	5228251

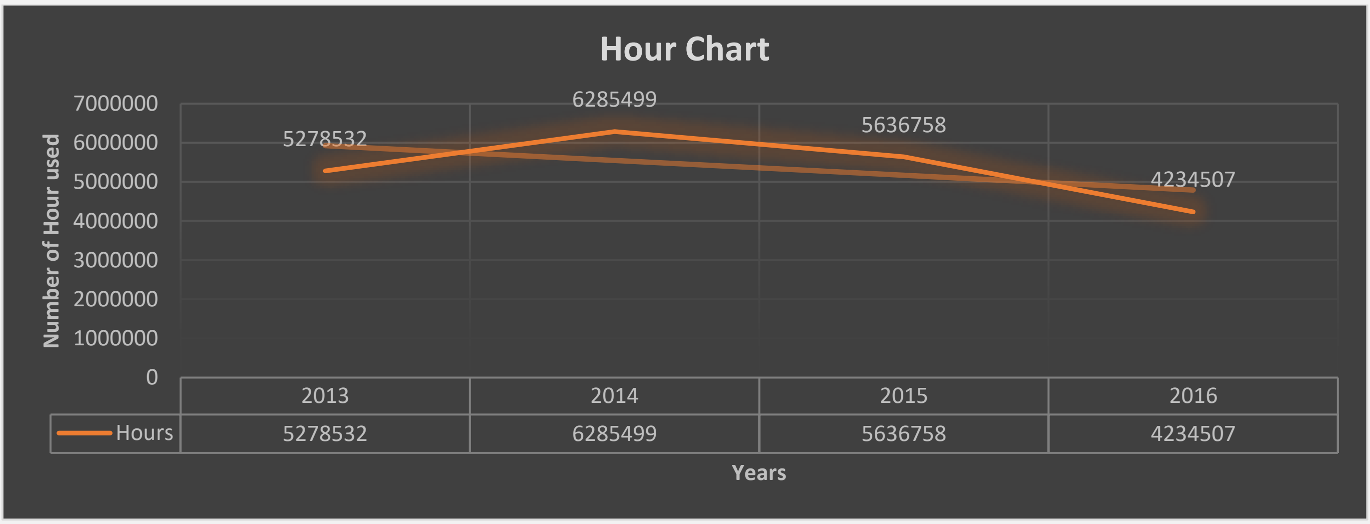
- From the above chart day time is taxi service is most used and peak hours evening 5PM – 9PM where people depend on cab more.
- Early morning is very least time it used, only Airport or Train commuter alone use at that time.
- This chart giving facts which we alrEDAy know when taxi is used more, this just add proof for the fact.

8. Analysis on the Trend:

- Trend analysis is done from 2013-2016, as we don't have complete data on the year 2017.
- Trend is interesting to see as it a curve structure, there many inference, correlation and information from the trend.
- Various external factor is causing the curve structure in many of the stats. To conclude to a point is difficult using this stat. But there are many observations and inference from the Trend.

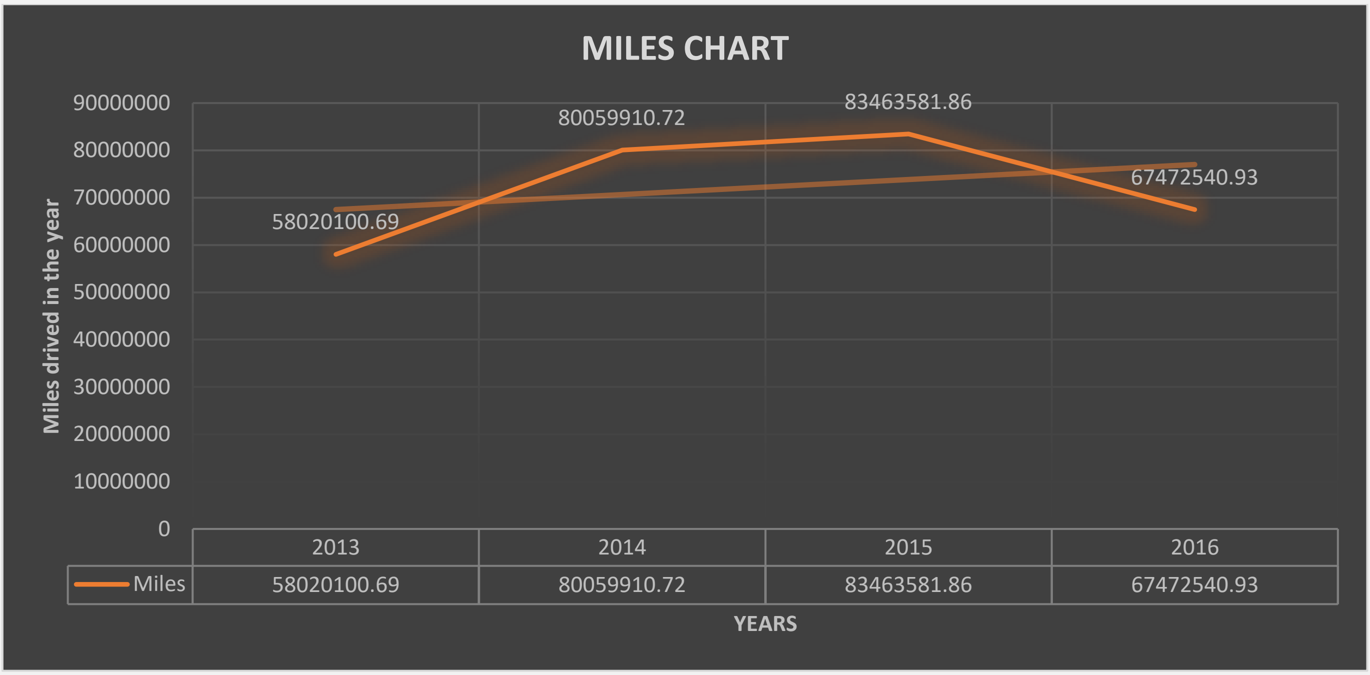
a. Stats and Graph:

i. Stats on Hours:



	2013	2014	2015	2016
Hours	5278532	6285499	5636758	4234507

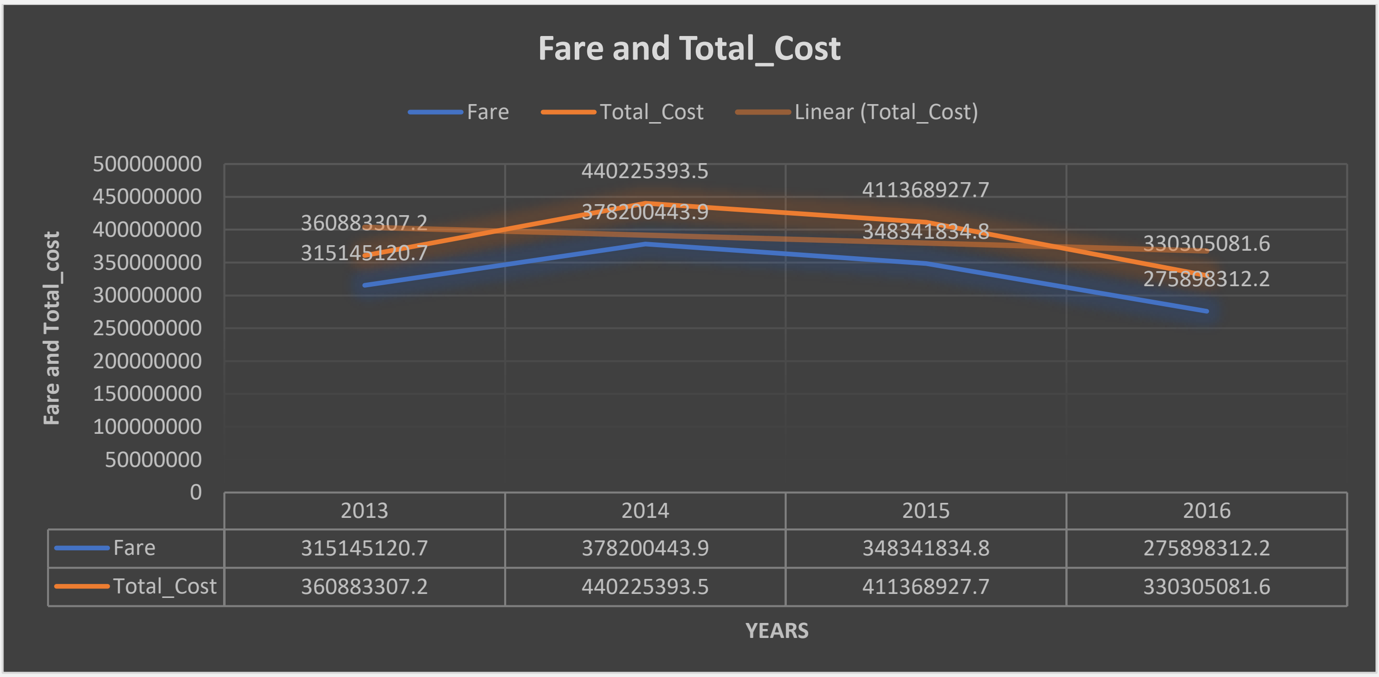
ii.Stats on Miles:



	2013	2014	2015	2016
Miles	58020100.69	80059910.72	83463581.86	67472540.93

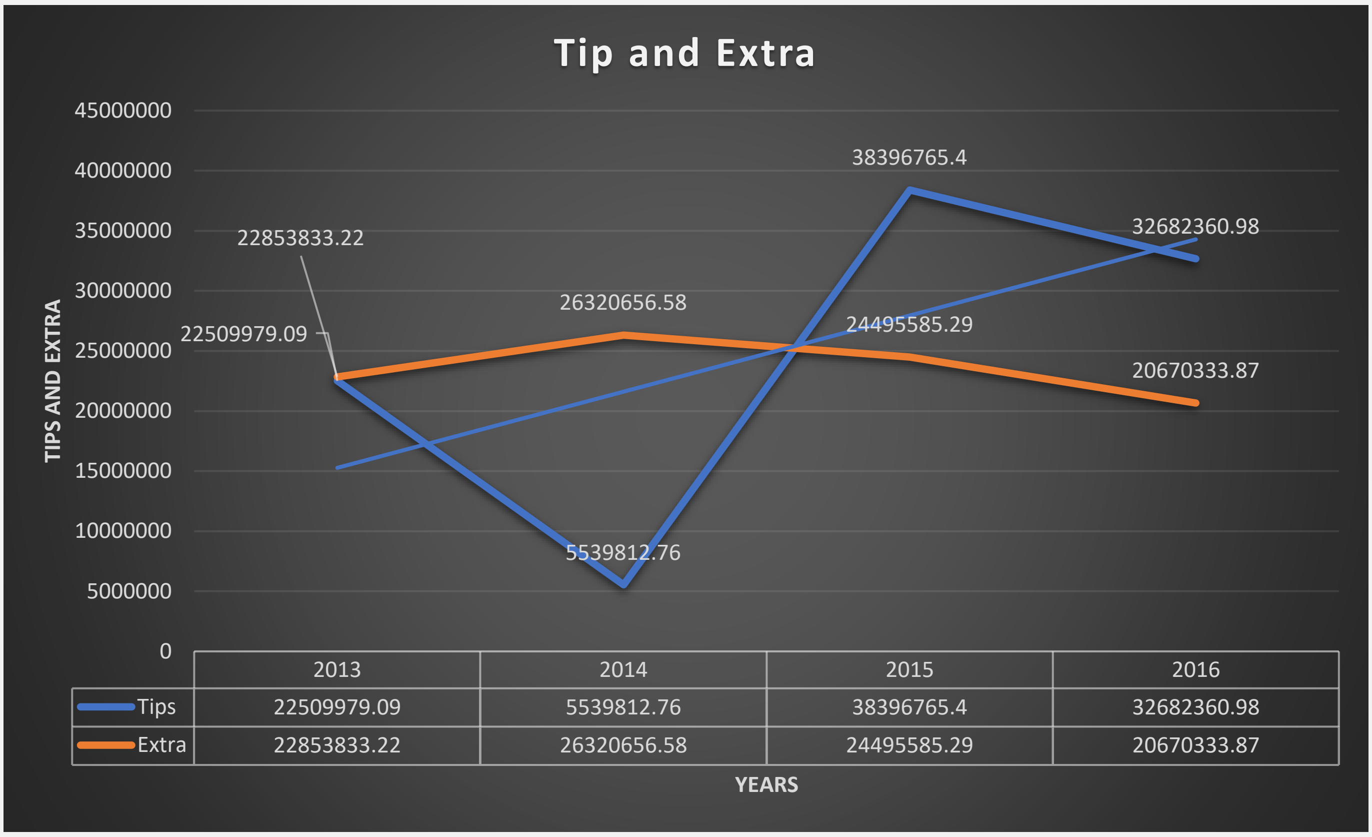
- Above chart has similar curve as both are corelated, we could clearly see there is a decrease in the use of Taxi in the year.
- After an increase from 2013 to 2015. There is a decrease in usage in the taxi by user.
- 2015 has more miles per hour compared to other years, show people start to use taxi only for long commute than short distance as it is expensive and other means are cheap than Taxi.
- Decrease in 2016 show people became comfortable with other service and use decreased.

iii. Stats on Fare & Total_cost:



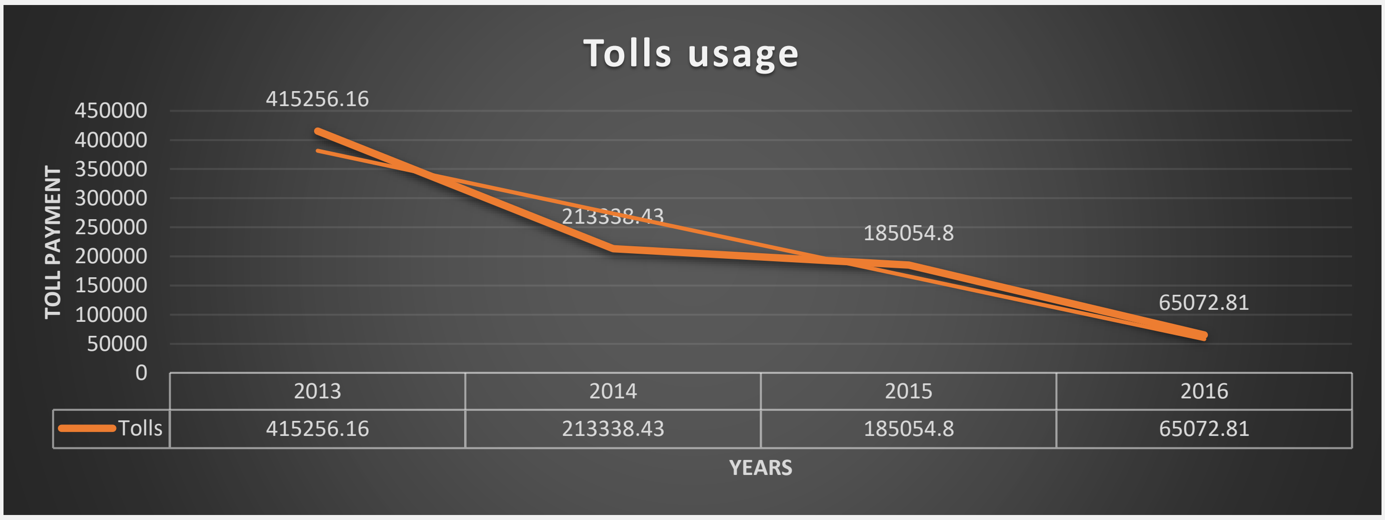
	2013	2014	2015	2016
Fare	315145120.7	378200443.9	348341834.8	275898312.2
Total_Cost	360883307.2	440225393.5	411368927.7	330305081.6

iv. Stats on Others (Tip & Extra, Tolls):



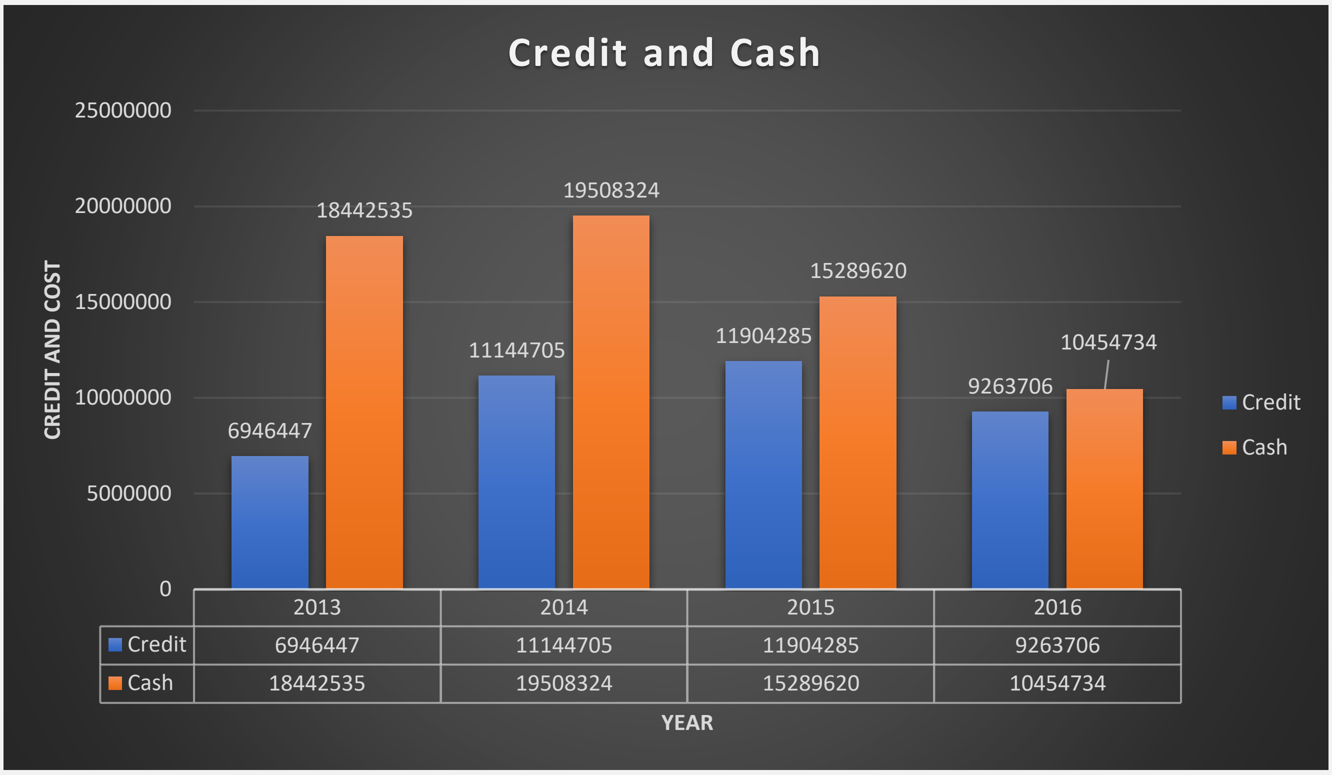
	2013	2014	2015	2016
Tips	22509979.09	5539812.76	38396765.4	32682360.98
Extra	22853833.22	26320656.58	24495585.29	20670333.87

- Fare and Total cost has same curve through out all the years shows, company didn't add any charges and people are making only essential extra charges and those are unavoidable.
- From the previous graph we can understand why there is a drop in total cost in 2016.
- As we saw alrEDAy Extra are because of mistake location or required and which are unavoidable human errors and Extra Curve also supports this fact.
- Tips curve is bit irregular, but we can see a huge raise. Show people like the Service.



	2013	2014	2015	2016
Tolls	415256.16	213338.43	185054.8	65072.81

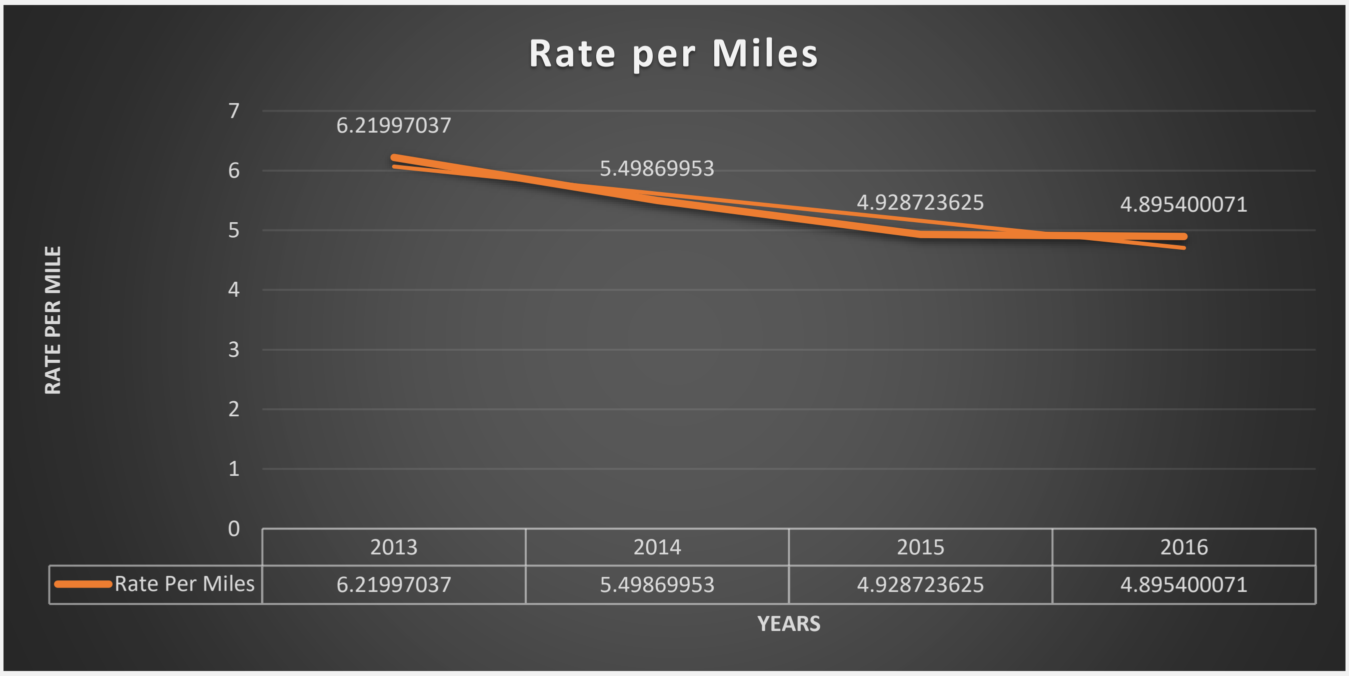
v. Payment Type:



	2013	2014	2015	2016
🟦 Credit	6946447	11144705	11904285	9263706
🟧 Cash	18442535	19508324	15289620	10454734

- We can clear observe that people preference moved from cash to Credit card other areas. It is preference evolved from both the customer and Business end as It is easy to transact.
- In the section current (2017) we can see a raise new method of payment through Mobile, Digital Cash. In upcoming year, we may see the use Digital cash more though it has too many restriction and less security like it is not traced in government record or it comes under taxation accounts.
- Digital cash is not very far, but it need time evolve into a predominant means of money exchange.

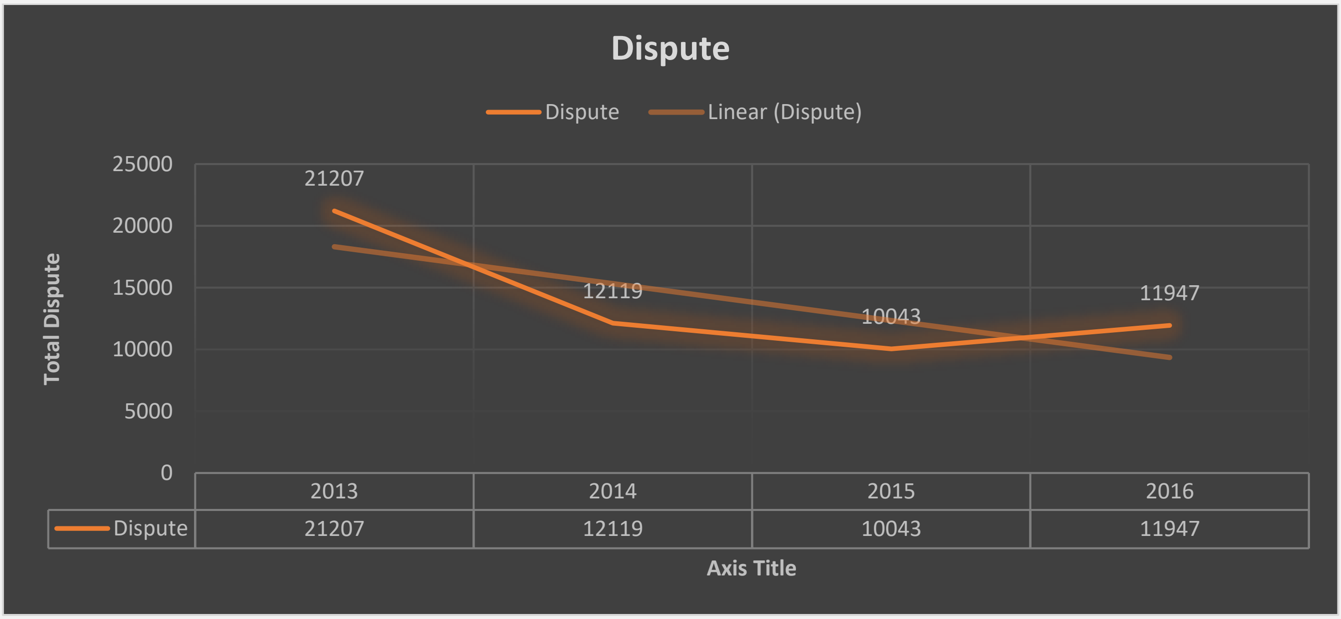
vi. Rate Per mile:



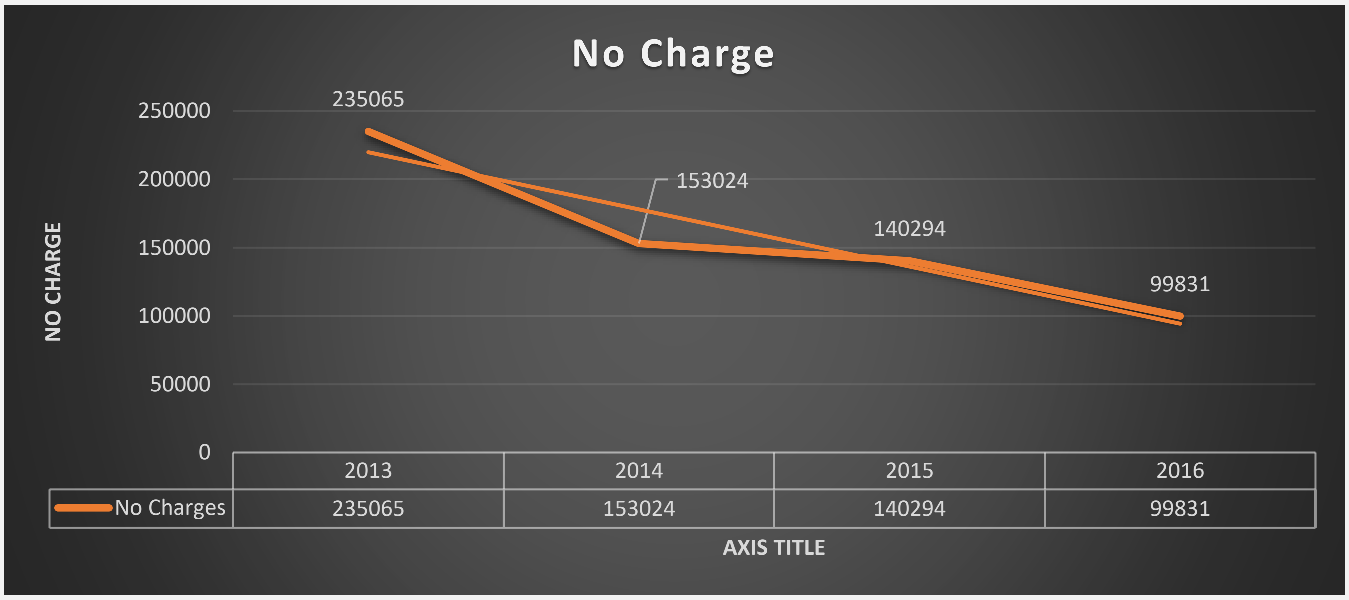
	2013	2014	2015	2016
Rate Per Miles	6.21997037	5.49869953	4.928723625	4.895400071

- The rate per miles has reduced over the area, we can see the curve going down.
- There few factors causing this, use of taxi has been reduced, people started using cheap means of transport. So, retain customer business has reduced a little.
- Over the year Business would gained all establishment cost, this is also may the reason for reduced cost.
- Previously we saw Year 2015 has more miles per hour, lower the transit & long distance gives more millage less Gas usage.

vii. Stats on Dispute and no charges:



	2013	2014	2015	2016
Dispute	21207	12119	10043	11947



	2013	2014	2015	2016
No Charges	235065	153024	140294	99831

- The Dispute curve didn’t vary much, some human error is unavoidable, so dispute is constant.
- The offer or no-charges has reduced over the year, shows the business has developed a stable market, Frequent user and Service became one of the essential.

9. Current (2017) Stats:

2017 Stats of the Dataset		
Seconds	**Hours**	**Days**
6178128941	1716146	71506.08333
Miles	**Trip_Total**	**Per Mile**
24189044.27	125703252.9	5.196701924
Fare	**Tips**	**Tip Percentage**
103828932.5	12703940.02	12.23545279
Tolls	**Extra**	**Extra Percentage**
37455.87	8340814.87	8.03322799
Credit card Count	**Cash Count**	**Usage Ratio**
3616793	3999456	47% :53%
NO Charge Count	**Dispute Count**	**No Charge Percentage**
37131	5768	0.48%

- Above show the stats of 2017 which doesn't have for whole year, so we can't fit it in trend curve.
- But there is no drastic change, cash payment is still more than card, Rate per mile has increased than 2016.
- 2017 seems to be similar year like 2016, so the curve may be still going a bit down and raise in any of the curve is not possible as we see similar stats.

10. Observations:

- To sum up, we observed a decrease in the use of Taxi service. After 2014 there is decline in the Use and Revenue. To have a deep insight, on the 2015 and 2016 can gives many perspective and facts. Our Scope restrict to do analysis on Trend.
- Researching on the factors that affect the business, I investigated factors like Cheap transport means, Climate, Crime, Parking Price in the city, new features in City. (Chicago is known for its crime and climate)
- When it comes to transport CTA provide cheap means and highly available service and increased the coverage. So, for shorter distance people preferred CTA Service. To support the points, we can see the curve in miles and hours chart. 2015 has more miles per hour.
- Climate and new features enhancement, Bike service (divvy bike in summer) which are far cheap and convenient than taxi. About climate winter was least when compared to previous years. Number of snow days is less than 10 in these year which made people to use CTA service.
- Crime is also comparatively less in these years these year, referred Chicago official site for crime stats. All these factors have cost the curve to go down.

- To over come these factor, Taxi firm are implementing mobile apps with digital cash to improve the user experience, Taxi Sharing feature, scheduled booking.

11. Conclusion:

Using the stats and data we can get an idea about preference of the people and usage, to have clearer picture and to validate our analysis and observation, analysis of CTA usage, Divy Bike sharing User, People preference to walk, Sentiment analysis on user feedback about the taxi service is need. All of these affects the usage of Cab. With all the analysis we can come with a good business strategy or plan to increase the usage and Stabilize the business for the taxi firms. When it comes to automated cab service we can scale up and down the service according to user preference, usage rate in area, location based. Also, Solution like scheduled booking service can be implemented to attract more user.

12. Appendix and Code:

References:

1. https://aws.amazon.com/emr
2. https://docs.aws.amazon.com/AmazonS3/latest/gsg/AmazonS3Basics.html
3. https://spark.apache.org/docs/0.9.0/python-programming-guide.html
4. https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Zaharia.pdf

Code and sample data:

```
%pyspark

import gc

#rEDAing the datafrom S3

data = sc.textFile("s3://bhasheyambigdata/Taxi_Trips.csv")

data.take(2)


# printing sample data

[u'Trip ID,Taxi ID,Trip Start Timestamp,Trip End Timestamp,Trip Seconds,Trip Miles,Pickup Census Tract,Dropoff Census Tract,Pickup Community Area,Dropoff Community Area,Fare,Tips,Tolls,Extras,Trip Total,Payment Type,Company,Pickup Centroid Latitude,Pickup Centroid Longitude,Pickup Centroid Location,Dropoff Centroid Latitude,Dropoff Centroid Longitude,Dropoff Centroid  Location,Community Areas',


u'0600bad12fd21f47978933fca9ff6f7053add238,44e35d9801892913e340f343010f26bd3cc4b5c1daaf93e296f046aa60786c40e1e8bba1859f8f25cd54d642b656da1013604a3a09e6420dca1ea033ec93c147,07/18/2017 12:15:00 PM,07/18/2017 12:15:00 PM,294,0.64,17031081600,17031081800,8,8,$5.50,$2.00,$0.00,$0.00,$8.00,Credit Card,,41.892072635,-87.628874157,POINT (-87.6288741572 41.8920726347),41.89321636,-87.63784421,POINT (-87.6378442095 41.8932163595),37',
#Spliting the data in the memory

data1 = data.map(lambda a: a.split(","))

Data =data1.filter(lambda lines: lines[0] != "Trip ID" )


# count the number of instance

total = Data.count()

print "Total number of Instance  used in the analysis :",total


Total number of Instance  used in the analysis : 112860054


import gc

data1.unpersist()

del data1

gc.collect()

# deleting the reference with all columns and releasing the memory


#Removing the column with less significants to have optimal memory

Taxi_data = Data.map(lambda
lines:(lines[2],lines[4],lines[5],lines[8],lines[9],lines[10],lines[11],lines[12],lines[13],lines[14],lines[15],lines[16]))

Taxi_data.take(5)


#Removing the data reference to keep the memory clean and optimized

import gc

Data.unpersist()

del Data

gc.collect()


#cleaning the data removed empty rows

Taxi_data = Taxi_data.filter(lambda x: x[1] is not u'')

Taxi_data = Taxi_data.filter(lambda x: x[2] is not u'')

Taxi_data = Taxi_data.filter(lambda x: x[5] is not u'')

Taxi_data = Taxi_data.filter(lambda x: x[6] is not u'')

Taxi_data = Taxi_data.filter(lambda x: x[7] is not u'')
Taxi_data = Taxi_data.filter(lambda x: x[8] is not u'')

Taxi_data = Taxi_data.filter(lambda x: x[9] is not u'')



# following function is used to find the sum the total of the column with integer values.

# column number is given as input, so it will filter the column and sum all the values also clean the memory once the calculation is done.

import gc

def findint(rdd,num,display,display1,tempstatcollector):

   temp = rdd.map(lambda lines: (lines[num],int(lines[num])))

   counting = temp.values().sum()

   print display,counting

   print display1,counting / 3600

   tempstatcollector.append(counting / 3600)

   temp.unpersist()

   del temp

   gc.collect()

   return tempstatcollector


# following function is used to find the sum the total of the column with float values.

# column number is given as input, so it will filter the column and sum all the values also clean the memory once the calculation is done.

import gc

def findfloat(rdd,num,display,tempstatcollector):

   temp = rdd.map(lambda lines: (lines[num],float(lines[num][1:4])))

   counting = temp.values().sum()

   print display,counting

   tempstatcollector.append(counting)

   temp.unpersist()

   del temp

   gc.collect()

   return tempstatcollector


# following function is used to find the sum the total of the column with float values.
# column number is given as input, so it will filter the column and sum all the values also clean the memory once the calculation is done.

import gc

def findfloat0(rdd,num,display,tempstatcollector):

   temp = rdd.map(lambda lines: (lines[num],float(lines[num])))

   counting = temp.values().sum()

   print display,counting

   tempstatcollector.append(counting)

   temp.unpersist()

   del temp

   gc.collect()

   return tempstatcollector


# following function is used to find the count of the column values.

# column number is given as input, so it will filter the column and count all the values also clean the memory once the calculation is done.

import gc

def countvalue(rdd,num,tempstatcollector):

   tempans = rdd.map(lambda lines: (lines[num],1))

   ans = tempans.reduceByKey(lambda x,y:x+y)

   coll = ans.collect()

   tempstatcollector.append(coll)

   print coll

   tempans.unpersist()

   del tempans

   gc.collect()

   return tempstatcollector


#function which consolidata all the stats by calling three functions defined to perform those.

def stats(rdd,year,tempstatcollector):

        findint(rdd,1,"In "+year+" number of seconds taxi service used","In "+year+" number of hours taxi service used",tempstatcollector)

        tempstatcollector = findfloat(rdd,2,"In "+year+" miles travelled by users",tempstatcollector)

        tempstatcollector = findfloat(rdd,5,"In "+year+" fare spent by all user",tempstatcollector)
        tempstatcollector = findfloat(rdd,6,"In "+year+"  tips given by all user",tempstatcollector)

        tempstatcollector = findfloat(rdd,7,"In "+year+"  tolls paid by all user",tempstatcollector)

        tempstatcollector = findfloat(rdd,8,"In "+year+"  Extra paid by all user",tempstatcollector)

        tempstatcollector = findfloat(rdd,9,"In "+year+"  Trip_total_includes paid by all user",tempstatcollector)

        tempstatcollector= countvalue(rdd,10,tempstatcollector)

        return tempstatcollector
```

```
Total:

In total number of seconds taxi service used 83345858073 second

In total number of hours taxi service used 23151627 hours

In total miles travelled by users 313221448.78

In total fare spent by all user 1366630691.7

In total  tips given by all user 140897324.4

In total  tolls paid by all user 884848.4

In total  Extra paid by all user 91158753.2

In total  Trip_total_includes paid by all user 1597501966.0

[(u'Mobile', 396), (u'Unknown', 227613), (u'Prcard', 14163), (u'Cash', 67694669), (u'Pcard', 29700), (u'Credit Card', 42875936), (u'No Charge', 665345), (u'Way2ride', 23), (u'Dispute', 61084)]
```

```
# spliting the dataset into years

year = Taxi_data.map(lambda lines: (lines[0][6:10],lines))


#To find the trend for the years calculating stats for different years

import gc

loop = ["2013","2014","2015","2016","2017"]

for l in loop:

        tempyear = year.filter(lambda lines: lines[0] == l)

        tempyear = tempyear.map(lambda lines:lines[1])

        print "number of Instance in " +l + " :",tempyear.count()

        #calculate Stat

        print stats(tempyear, l,[l])

        tempyear.unpersist()

        del tempyear

        gc.collect()
```

Sample data

```
[(u'03/08/2013 06:30:00 PM', u'240', u'0.87', u'8', u'', u'$5.05', u'$0.00', u'$0.00', u'$1.00', u'$6.05', u'Cash', u''),

 (u'03/05/2013 01:45:00 PM', u'720', u'2.21', u'8', u'', u'$8.45', u'$2.00', u'$0.00', u'$0.00', u'$10.45', u'Credit Card', u''),

 (u'07/07/2013 01:00:00 PM', u'660', u'0.3', u'7', u'8', u'$9.45', u'$2.00', u'$0.00', u'$0.00', u'$11.45', u'Credit Card', u'Choice Taxi Association'),

 (u'08/27/2013 07:45:00 AM', u'2040', u'13.3', u'8', u'56', u'$30.05', u'$0.00', u'$0.00', u'$2.00', u'$32.05', u'Cash', u'Dispatch Taxi Affiliation'),

 (u'05/03/2013 09:15:00 PM', u'900', u'6.5', u'4', u'8', u'$16.05', u'$0.00', u'$0.00', u'$1.00', u'$17.05', u'Cash', u'')]


 number of Instance in 2013 : 25742055

In 2013 number of seconds taxi service used 19002718380

In 2013 number of hours taxi service used 5278532

In 2013 miles travelled by users 58020100.69

In 2013 fare spent by all user 315145120.65

In 2013  tips given by all user 22509979.09

In 2013  tolls paid by all user 415256.16

In 2013  Extra paid by all user 22853833.22

In 2013  Trip_total_includes paid by all user 360883307.18

[(u'Unknown', 88297), (u'Prcard', 531), (u'Cash', 18442535), (u'Pcard', 7973), (u'Credit Card', 6946447), (u'No Charge', 235065), (u'Dispute', 21207)]

['2013', 5278532, 58020100.690000184, 315145120.6500039, 22509979.09000001, 415256.16, 22853833.220000006, 360883307.18000436, [(u'Unknown', 88297), (u'Prcard', 531), (u'Cash', 18442535), (u'Pcard', 7973), (u'Credit Card', 6946447), (u'No Charge', 235065), (u'Dispute', 21207)]]

PythonRDD[104] at RDD at PythonRDD.scala:48

63

number of Instance in 2014 : 30869551

In 2014 number of seconds taxi service used 22627797060

In 2014 number of hours taxi service used 6285499

In 2014 miles travelled by users 80059910.72

In 2014 fare spent by all user 378200443.93

In 2014  tips given by all user 35539812.76

In 2014  tolls paid by all user 213338.43

In 2014  Extra paid by all user 26320656.58

In 2014  Trip_total_includes paid by all user 440225393.45
```

```
[(u'Unknown', 42934), (u'Prcard', 1755), (u'Cash', 19508324), (u'Pcard', 6690), (u'Credit Card', 11144705), (u'No Charge', 153024), (u'Dispute', 12119)]

['2014', 6285499, 80059910.72000179, 378200443.93000364, 35539812.760000095, 213338.42999999988, 26320656.580000013, 440225393.45000505, [(u'Unknown', 42934), (u'Prcard', 1755), (u'Cash', 19508324), (u'Pcard', 6690), (u'Credit Card', 11144705), (u'No Charge', 153024), (u'Dispute', 12119)]]

PythonRDD[126] at RDD at PythonRDD.scala:48

63

number of Instance in 2015 : 27394395

In 2015 number of seconds taxi service used 20292329880

In 2015 number of hours taxi service used 5636758

In 2015 miles travelled by users 83463581.86

In 2015 fare spent by all user 348341834.82

In 2015  tips given by all user 38396765.4

In 2015  tolls paid by all user 185054.8

In 2015  Extra paid by all user 24495585.29

In 2015  Trip_total_includes paid by all user 411368927.7

[(u'Unknown', 41459), (u'Prcard', 2298), (u'Cash', 15289620), (u'Pcard', 6396), (u'Credit Card', 11904285), (u'No Charge', 140294), (u'Dispute', 10043)]

['2015', 5636758, 83463581.86000165, 348341834.82000244, 38396765.40000013, 185054.79999999993, 24495585.29000002, 411368927.70000637, [(u'Unknown', 41459), (u'Prcard', 2298), (u'Cash', 15289620), (u'Pcard', 6396), (u'Credit Card', 11904285), (u'No Charge', 140294), (u'Dispute', 10043)]]

PythonRDD[148] at RDD at PythonRDD.scala:48

63

number of Instance in 2016 : 19874721

In 2016 number of seconds taxi service used 15244226746

In 2016 number of hours taxi service used 4234507

In 2016 miles travelled by users 67472540.93

In 2016 fare spent by all user 275898312.21

In 2016  tips given by all user 32682360.98

In 2016  tolls paid by all user 65072.81

In 2016  Extra paid by all user 20670333.87

In 2016  Trip_total_includes paid by all user 330305081.63

[(u'Unknown', 35291), (u'Prcard', 3945), (u'Cash', 10454734), (u'Pcard', 5256), (u'Credit Card', 9263706), (u'No Charge', 99831), (u'Way2ride', 11), (u'Dispute', 11947)]
```

```
['2016', 4234507, 67472540.93000062, 275898312.21002454, 32682360.980000004, 65072.81000000005, 20670333.86999999, 330305081.6300224, [(u'Unknown', 35291), (u'Prcard', 3945), (u'Cash', 10454734), (u'Pcard', 5256), (u'Credit Card', 9263706), (u'No Charge', 99831), (u'Way2ride', 11), (u'Dispute', 11947)]]

PythonRDD[170] at RDD at PythonRDD.scala:48

63

number of Instance in 2017 : 7688207

In 2017 number of seconds taxi service used 6178128941

In 2017 number of hours taxi service used 1716146

In 2017 miles travelled by users 24189044.27

In 2017 fare spent by all user 103828932.53

In 2017  tips given by all user 12703940.02

In 2017  tolls paid by all user 37455.87

In 2017  Extra paid by all user 8340814.87

In 2017  Trip_total_includes paid by all user 125703252.9

[(u'Mobile', 396), (u'Unknown', 19632), (u'Prcard', 5634), (u'Cash', 3999456), (u'Pcard', 3385), (u'Credit Card', 3616793), (u'No Charge', 37131), (u'Way2ride', 12), (u'Dispute', 5768)]

['2017', 1716146, 24189044.269999534, 103828932.52999182, 12703940.019999927, 37455.87000000007, 8340814.870000004, 125703252.900007, [(u'Mobile', 396), (u'Unknown', 19632), (u'Prcard', 5634), (u'Cash', 3999456), (u'Pcard', 3385), (u'Credit Card', 3616793), (u'No Charge', 37131), (u'Way2ride', 12), (u'Dispute', 5768)]]

PythonRDD[192] at RDD at PythonRDD.scala:48

63

FINISHED
```

```
#spliting the Dataset according to AM and PM

timesplit = Taxi_data.map(lambda lines: (lines[0][20:22],lines))

timesplit.take(5)

import gc

timeam = timesplit.filter(lambda lines : lines[0] == "AM")

timepm = timesplit.filter(lambda lines : lines[0] == "PM")

timeam.take(5)

timesplit.unpersist()

del timesplit

gc.collect()

# spliting the dataset into hours for both the AM and Pm dataset

timeam = timeam.map(lambda lines: lines[1])

timepm = timepm.map(lambda lines: lines[1])
```

```
timeam = timeam.map(lambda lines: (lines[0][11:13],lines))

timepm = timepm.map(lambda lines: (lines[0][11:13],lines))

timepm.take(5)

[(u'12', (u'07/18/2017 12:15:00 PM', u'294', u'0.64', u'8', u'8', u'$5.50', u'$2.00', u'$0.00', u'$0.00', u'$8.00', u'Credit
Card', u'')),

(u'11', (u'07/19/2017 11:15:00 PM', u'180', u'0.6', u'8', u'32', u'$4.50', u'$2.00', u'$0.00', u'$0.00', u'$6.50', u'Credit
Card', u'Taxi Affiliation Services'))

,(u'01', (u'07/11/2017 01:15:00 PM', u'1980', u'17.8', u'28', u'76', u'$43.75', u'$6.50', u'$0.00', u'$0.00', u'$50.25',
u'Credit Card', u'Choice Taxi Association')),

(u'04', (u'07/02/2017 04:15:00 PM', u'600', u'1.7', u'32', u'8', u'$8.50', u'$0.00', u'$0.00', u'$1.00', u'$9.50', u'Cash',
u'Taxi Affiliation Services')), (u'03',

(u'07/04/2017 03:30:00 PM', u'300', u'0', u'32', u'8', u'$5.50', u'$0.00', u'$0.00', u'$0.00', u'$5.50', u'Credit Card',
u'Blue Ribbon Taxi Association Inc.'))]


#Calculating the stat of different hours see busy hours.

import gc

check = ["01","02","03","04","05","06","07","08","09","10","11","12"]

for l in check:

        tempyear = timeam.filter(lambda lines: lines[0] == l)

        tempyear = tempyear.map(lambda lines:lines[1])

        print "number of Instance in " +l + " AM:",tempyear.count()

        #calculate Stat

        print hourstat(tempyear, l+"AM",[l+"AM"])

        tempyear.unpersist()

        del tempyear

        gc.collect(

for l in check:

        tempyear = timepm.filter(lambda lines: lines[0] == l)

        tempyear = tempyear.map(lambda lines:lines[1])

        print "number of Instance in " +l  + " PM:",tempyear.count()

        #calculate Stat

        print hourstat(tempyear, l+"PM" ,[l+"PM"])

        tempyear.unpersist()

        del tempyear

        gc.collect()
```