Massive Data Analysis

Final Project Report

Ahmadreza Tavana: 98104852

Professor: Dr.Gholampour

Winter 2023

1. Introduction

This is the report of MDA project. The map road of this report is that I defined some ideas for the project and in each idea I implement some algorithms and my ideas on the data. At first we get familiar with the data and then try to collect information that we consider.

Many of the ideas are those which were recommended in the project documentation. The results, explanations of implementations and the plots are shown in this report and also the explanations of each steps are explained in both the notebook of the project and the report.

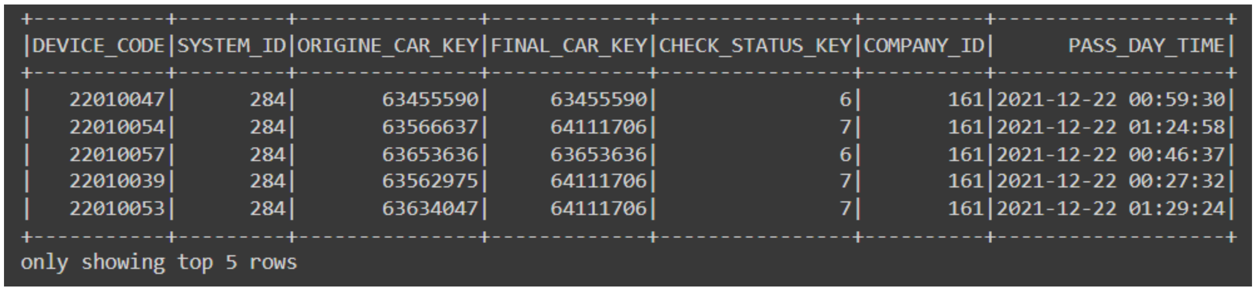
In some parts of this report, the written codes are attached for some ideas for explaining more clearly.

2. Ideas

2.1 Idea 1 (Getting Familiar with the Dataset)

This part is the first step of the project. In this notebook we try to figure out what is our data and what we can do with it. In the other word, in this notebook we try to get familiar with the data and generate new ideas for the next parts.

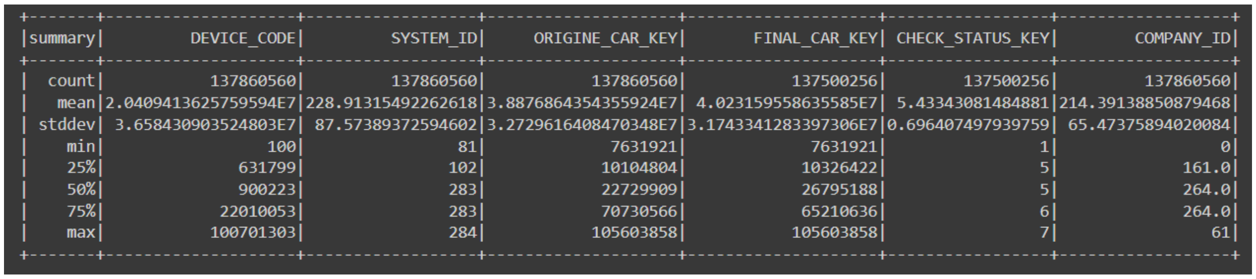
At first we take a look at the data to see what it is in real. The result is shown in below.



```
+-----------+---------+---------------+-------------+----------------+----------+-------------------+
|DEVICE_CODE|SYSTEM_ID|ORIGINE_CAR_KEY|FINAL_CAR_KEY|CHECK_STATUS_KEY|COMPANY_ID|      PASS_DAY_TIME|
+-----------+---------+---------------+-------------+----------------+----------+-------------------+
|   22010047|      284|       63455590|     63455590|               6|       161|2021-12-22 00:59:30|
|   22010054|      284|       63566637|     64111706|               7|       161|2021-12-22 01:24:58|
|   22010057|      284|       63653636|     63653636|               6|       161|2021-12-22 00:46:37|
|   22010039|      284|       63562975|     64111706|               7|       161|2021-12-22 00:27:32|
|   22010053|      284|       63634047|     64111706|               7|       161|2021-12-22 01:29:24|
+-----------+---------+---------------+-------------+----------------+----------+-------------------+
only showing top 5 rows
```

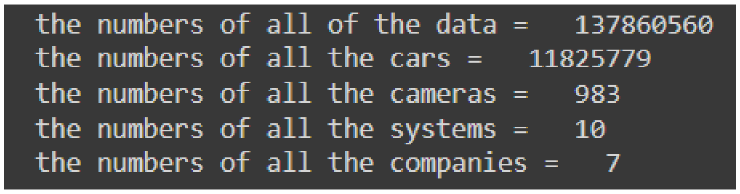
As we can see the data has 7 columns which contains: “DEVICE_CODE”, “SYSTEM_ID”, “ORIGINE_CAR_KEY”, “FINAL_CAR_KEY” etc.

Then we try to report the data by using summary command. The summary function shows many aspects of the dataframe such as mean, variance, SD, min, max, first quartile, third quartile and etc. The result of summarizing is shown in below.



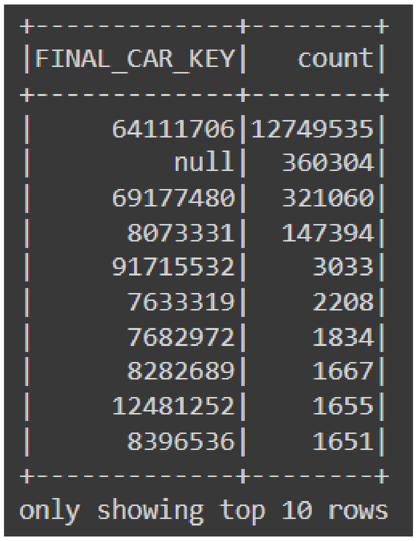
```
+-------+--------------------+------------------+--------------------+--------------------+-----------------+------------------+
|summary|         DEVICE_CODE|         SYSTEM_ID|     ORIGINE_CAR_KEY|       FINAL_CAR_KEY| CHECK_STATUS_KEY|        COMPANY_ID|
+-------+--------------------+------------------+--------------------+--------------------+-----------------+------------------+
|  count|           137860560|         137860560|           137860560|           137500256|        137500256|         137860560|
|   mean|2.0409413625759594E7|228.91315492262618|3.8876864354355924E7| 4.023159558635585E7| 5.43343081484881|214.39138850879468|
| stddev| 3.65843090352480 3E7| 87.5738937259460 2|3.2729616408470348E7|3.1743341283397306E7|0.696407497939759| 65.47375894020084|
|    min|                 100|                81|             7631921|             7631921|                1|                 0|
|    25%|              631799|               102|            10104804|            10326422|                5|             161.0|
|    50%|              900223|               283|            22729909|            26795188|                5|             264.0|
|    75%|            22010053|               283|            70730566|            65210636|                6|             264.0|
|    max|           100701303|               284|           105603858|           105603858|                7|                61|
+-------+--------------------+------------------+--------------------+--------------------+-----------------+------------------+
```

Then we try to count number of the all rows of data, number of cars, number of cameras, number of systems, number of companies in the below cell. The result is shown in below.



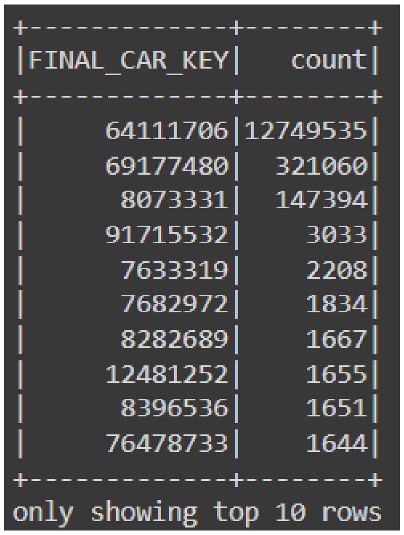
```
the numbers of all of the data =   137860560
the numbers of all the cars =   11825779
the numbers of all the cameras =   983
the numbers of all the systems =   10
the numbers of all the companies =   7
```

Now we try to find the most passed cars and see which car has the most passing through cameras. The result is shown in below.



```
+-------------+--------+
|FINAL_CAR_KEY|   count|
+-------------+--------+
|     64111706|12749535|
|         null|  360304|
|     69177480|  321060|
|      8073331|  147394|
|     91715532|    3033|
|      7633319|    2208|
|      7682972|    1834|
|      8282689|    1667|
|     12481252|    1655|
|      8396536|    1651|
+-------------+--------+
only showing top 10 rows
```

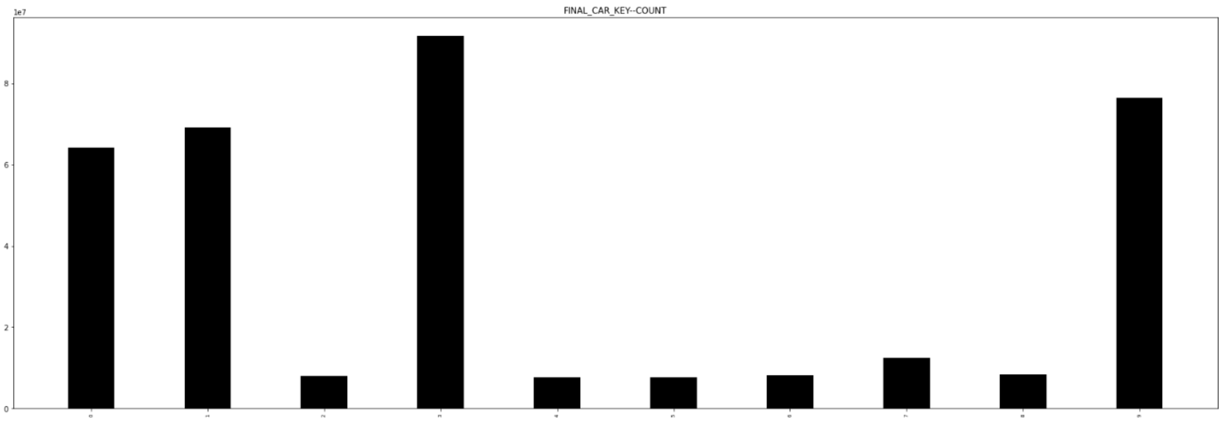
As we can see, we have some null cars in the data. We can conclude that the data may have some null values. In this part we try to delete the rows with null values in 'FINAL_CAR_KEY' in the dataset and then try to count the cars. the result is shown in below.



```
+-------------+--------+
|FINAL_CAR_KEY|   count|
+-------------+--------+
|     64111706|12749535|
|     69177480|  321060|
|      8073331|  147394|
|     91715532|    3033|
|      7633319|    2208|
|      7682972|    1834|
|      8282689|    1667|
|     12481252|    1655|
|      8396536|    1651|
|     76478733|    1644|
+-------------+--------+
only showing top 10 rows
```

As we can see the count of car "64111706" is too much larger than other cars. We can count it as an outlier but not considering that will not have huge effect on our work.

Now we try to plot the histogram of the most passed car in a relative way and see the result in the below. The histogram shows number of cars that total cameras recorded at least a single time.



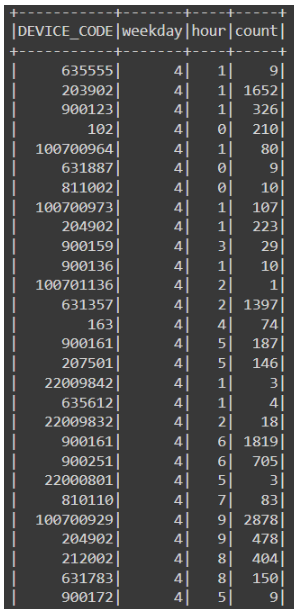
We can have a sense from the count of the cars in this histogram relatively. As we have 4 cars which are recorded too much rather than other cars, we can count them as outliers but as I said it doesn't have huge effect in the result of the work.

2.2 Idea 2 (Plotting Histogram by Using Clustering)

In this part at first we cluster cameras to figure out pattern of passing cars in each streets and roads. Then we plot the histogram of each cluster cameras.

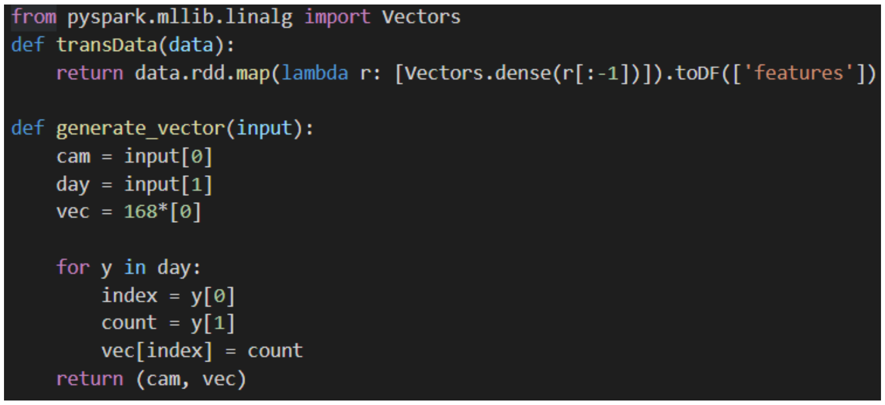
At first we do some preprocessing on the data and make vectors and data frame for clustering and plotting histograms.

Then we add week and hour to dataframe and obtain the day and hour and key of each passing car for each week through a camera by groupBy command. The result is show in below.



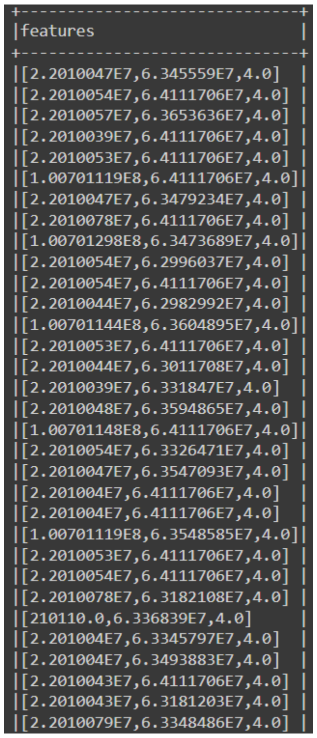
```
+-----------+-------+----+-----+
|DEVICE_CODE|weekday|hour|count|
+-----------+-------+----+-----+
|     635555|      4|   1|    9|
|     203902|      4|   1| 1652|
|     900123|      4|   1|  326|
|        102|      4|   0|  210|
|  100700964|      4|   1|   80|
|     631887|      4|   0|    9|
|     811002|      4|   0|   10|
|  100700973|      4|   1|  107|
|     204902|      4|   1|  223|
|     900159|      4|   3|   29|
|     900136|      4|   1|   10|
|  100701136|      4|   2|    1|
|     631357|      4|   2| 1397|
|        163|      4|   4|   74|
|     900161|      4|   5|  187|
|     207501|      4|   5|  146|
|   22009842|      4|   1|    3|
|     635612|      4|   1|    4|
|   22009832|      4|   2|   18|
|     900161|      4|   6| 1819|
|     900251|      4|   6|  705|
|   22000801|      4|   5|    3|
|     810110|      4|   7|   83|
|  100700929|      4|   9| 2878|
|     204902|      4|   9|  478|
|     212002|      4|   8|  404|
|     631783|      4|   8|  150|
|     900172|      4|   5|    9|
```

The we add camera indices to the dataframe for clustering in the next parts. The, we define some functions for obtaining features and converting dataframe columns to vectors. The functions are defined in below and the results are shown after them.



```python
from pyspark.mllib.linalg import Vectors
def transData(data):
    return data.rdd.map(lambda r: [Vectors.dense(r[:-1])]).toDF(['features'])

def generate_vector(input):
    cam = input[0]
    day = input[1]
    vec = 168*[0]

    for y in day:
        index = y[0]
        count = y[1]
        vec[index] = count
    return (cam, vec)
```

The output of features which obtained from the above functions is show in below.



```
+------------------------------+
|features                      |
+------------------------------+
|[2.2010047E7,6.345559E7,4.0]  |
|[2.2010054E7,6.4111706E7,4.0] |
|[2.2010057E7,6.3653636E7,4.0] |
|[2.2010039E7,6.4111706E7,4.0] |
|[2.2010053E7,6.4111706E7,4.0] |
|[1.00701119E8,6.4111706E7,4.0]|
|[2.2010047E7,6.3479234E7,4.0] |
|[2.2010078E7,6.4111706E7,4.0] |
|[1.00701298E8,6.3473689E7,4.0]|
|[2.2010054E7,6.2996037E7,4.0] |
|[2.2010054E7,6.4111706E7,4.0] |
|[2.2010044E7,6.2982992E7,4.0] |
|[1.00701144E8,6.3604895E7,4.0]|
|[2.2010053E7,6.4111706E7,4.0] |
|[2.2010044E7,6.3011708E7,4.0] |
|[2.2010039E7,6.331847E7,4.0]  |
|[2.2010048E7,6.3594865E7,4.0] |
|[1.00701148E8,6.4111706E7,4.0]|
|[2.2010054E7,6.3326471E7,4.0] |
|[2.2010047E7,6.3547093E7,4.0] |
|[2.201004E7,6.4111706E7,4.0]  |
|[2.201004E7,6.4111706E7,4.0]  |
|[1.00701119E8,6.3548585E7,4.0]|
|[2.2010053E7,6.4111706E7,4.0] |
|[2.2010054E7,6.4111706E7,4.0] |
|[2.2010078E7,6.3182108E7,4.0] |
|[210110.0,6.336839E7,4.0]     |
|[2.201004E7,6.3345797E7,4.0]  |
|[2.201004E7,6.3493883E7,4.0]  |
|[2.2010043E7,6.4111706E7,4.0] |
|[2.2010043E7,6.3181203E7,4.0] |
|[2.2010079E7,6.3348486E7,4.0] |
```

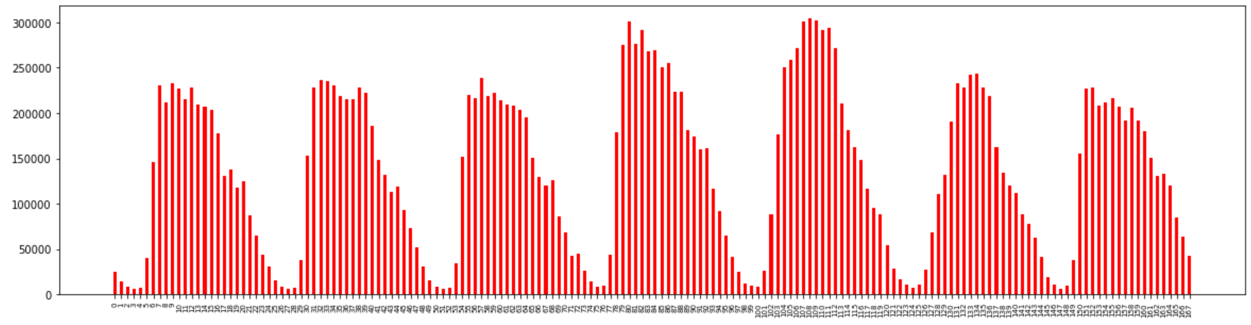
Now we implement a LDA clustering to cluster different cameras and then plot histograms of record of passing cars through each camera.

We fix number of cluster centers equal to 6 cluster and deploy the clustering on each camera.

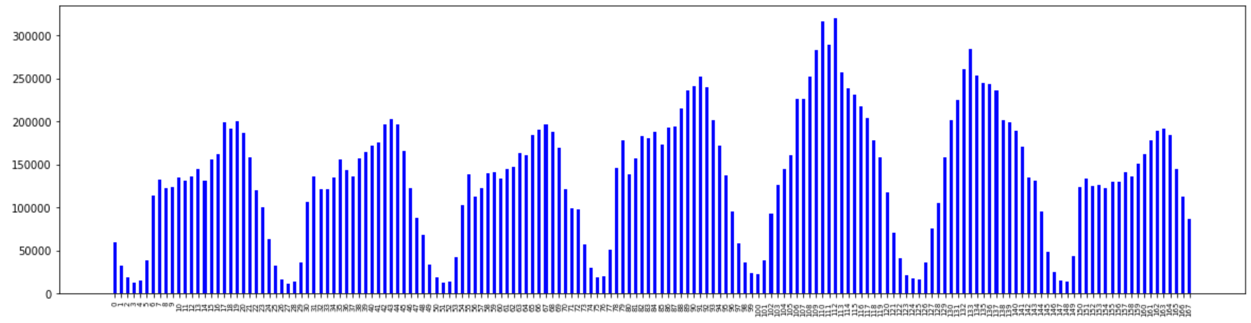
Our clustering shows different types of streets in the city. In the other words, we plot number of passing cars to recognize how much each street get crowded and analyze number of cars in each street in specific hours.

With this approach, we can have a sense of number of passing cars from near cameras, because we deployed a clustering on cameras so if center of a clustering has a specific histogram, we can conclude that the pattern and histogram of near cameras so we can say that near cameras to each clustering centers, has approximately a same histogram.

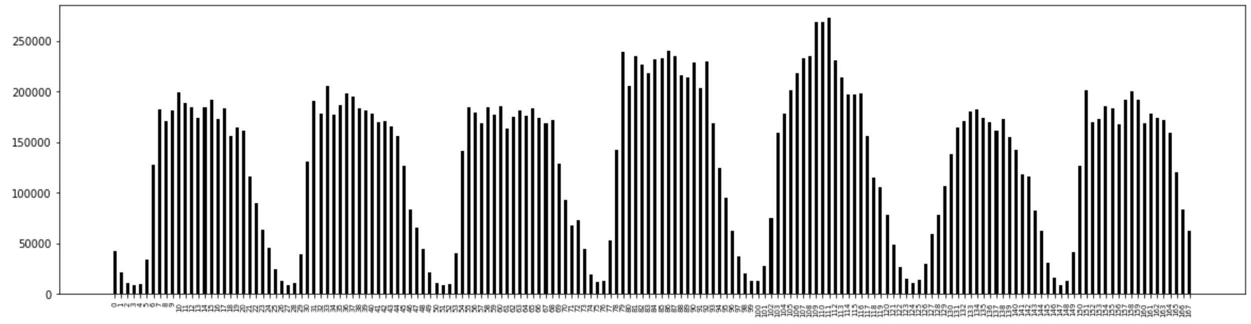
Now, we plot histogram of each camera in different days of week. We differ each camera of the streets with different colors which specify each camera.



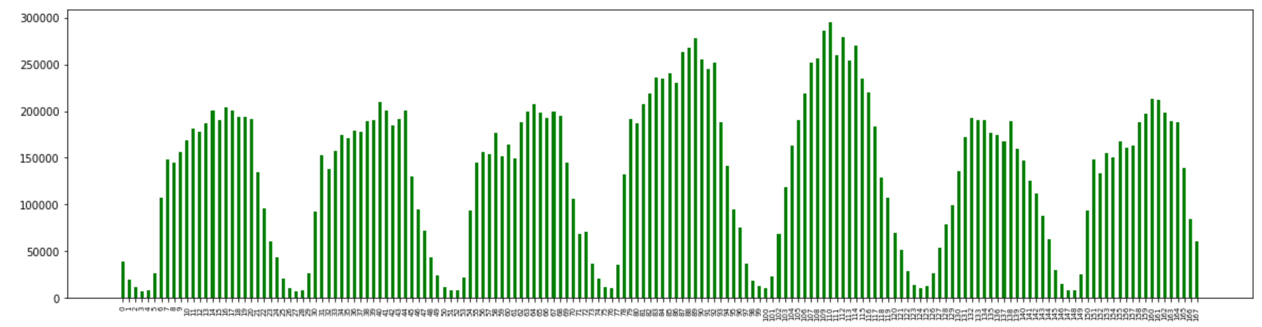
Cluster Center 1



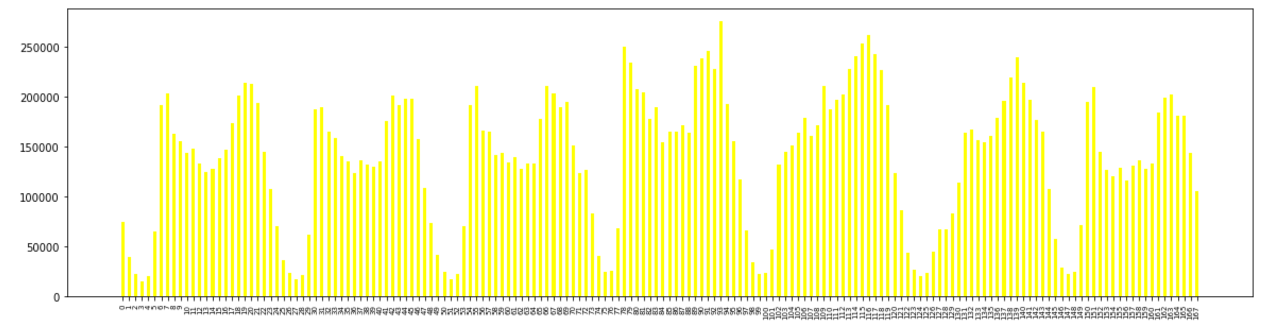
Cluster Center 2



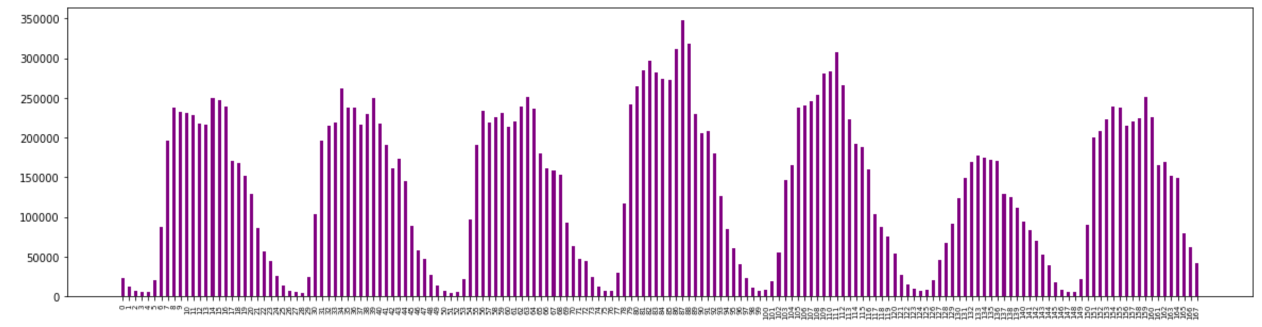
Cluster Center 3



Cluster Center 4



Cluster Center 5



Cluster Center 6

These 6 clusters, shows pattern of passing cars through a specific camera in each day of week. Actually we differ the pattern of recording cameras to these 6 group. It means that these 6 groups representing pattern of recording all of the cameras in the city and any camera in the city belongs to one of these clusters. With analyzing these clusters, we can predict type of each street. It means that we can predict whether a street has entertaining environment of mostly crowded in normal day by pattern of recording of each camera.

In the cluster center 1 cameras, at the start of the streets are not crowded but suddenly, it gets too crowded fast. We can conclude that this street maybe is one of the main streets of the city which at 05:00 AM and 06:00 AM starts to get crowded. It means many people cross these streets for going to work or university or school etc.

In the cluster center 2 cameras, the different of passing cars in start of the day and the middle of the day is low in count. It shows that streets which has these kinds of cameras, are not such important streets. Also the amount of passing cars is lower than other ones.

In the cluster center 3 cameras, the different of passing cars in start of the day and the middle of the day is low even lower than cluster center 2 but in 5th and 6th day of the week, the amount of passing cars get increased. It shows that these streets are entertaining streets that in the last days of the week they get crowded and people cross these streets for chilling out.

In the cluster center 4 cameras, the pattern is like cluster center 2 but it seems that these streets are a little bit more important that those ones and intercept with some more important streets or highways.

In the cluster center 5 cameras, the crossing cars are too much in both starting of day and starting of night or evening. It seems that these streets are highways from Tehran to Karaj or something like that because people come to Tehran at start of each day and come back to Karaj in evening so streets get crowded in these streets in these hours in days.

In the cluster center 6 cameras, as it seems, Friday is not crowded that much and the number of cars passing through these streets is too low. Also as other streets, the middle days of the week, streets are more crowded than other days. These streets are probably, streets which contain main offices of the city which are reclusion.

2.3 Idea 3 (Item-Basckets and Frequent-Items)

In this part we try to find the most frequent itemsets as what we did in the homeworks in this term. We use A-priori algorithm in this part and try to find the most frequent Itemsets.

We map the data with key of ('Final_CAR_KEY','PASS_DAY_TIME') and value of ('DEVICE_CODE') which shows the pathway of the cars. Now we have a RDD which has rows that each row contains pathway of each date with license plate of each passed car.

Then we group_by the above RDD by its key for each car in every date to have all pathway that each car passed in a list. Notice that in this part we use set instead of list to not having repeated pathways. This approach base on the usage of this part does not have huge effect on the whole of the problem.

After some other pre-prcoessing we use A-priori function for finding the most frequent item stets.

At first some other functions are written and I explain them in the following and then I write the original function for A-Priori function.

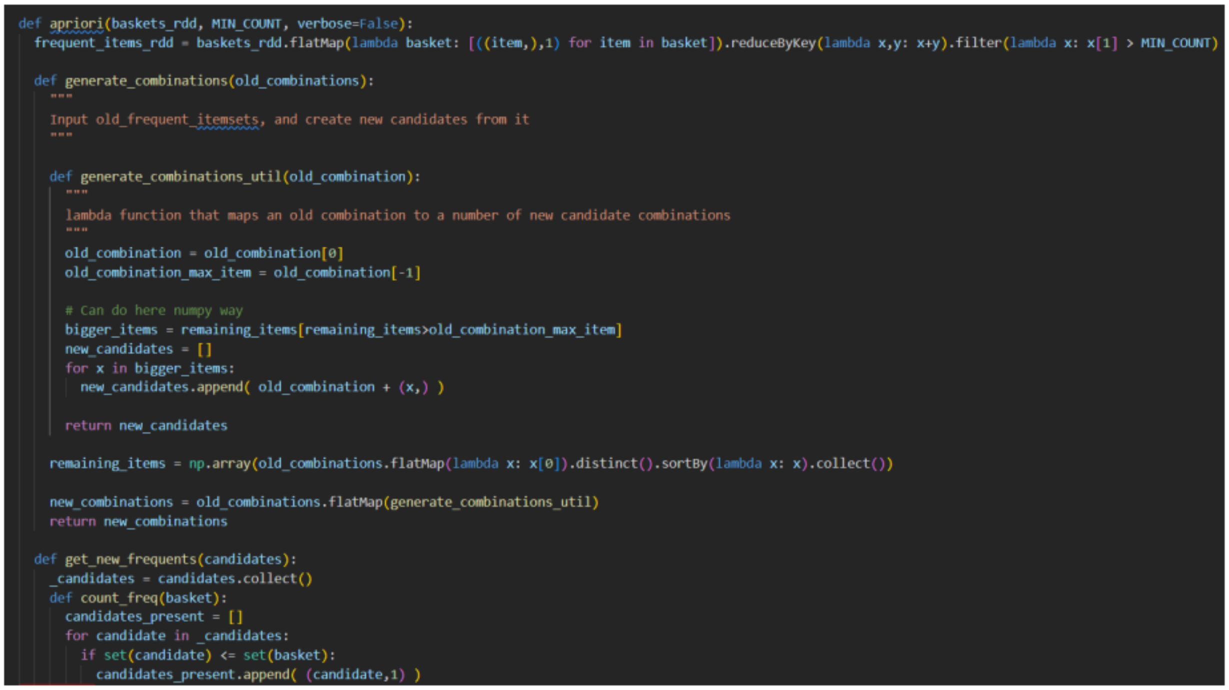
generate_combinations

This function gets frequent itemsets of its previous step and generates candidate of the next step. At first it gets the frequent remaining items from whole of the frequent itemssets and then sort them. Then for each of frequent itemset, it gets the largest element and then compare to remaining items of the collection and if that item be larger than that one, it adds it to the end of that collection and generates a new candidate. In this function it doesn’t need to add smaller elements to the collections.

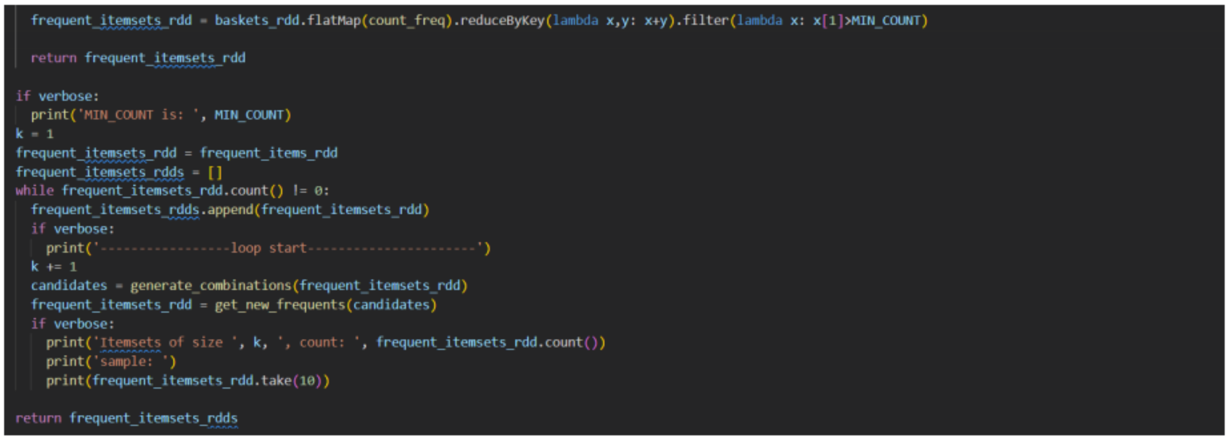
For example, suppose [b,c,d] collection which was frequent in the previous step and the remaining collection of frequent is [a,b,c,d,z] (these elements are sorted). From these remaining elements, z is the only element which is larger than other elements so a new candidate is [b,c,d,z]. But [a,b,c,d] is not a candidate. Assume that it is frequent. We know that all subsets of a frequent set are also frequent. Now if [a,b,c] was frequent, so it would be available in the frequent list and create its combination in checking [a,b,c] but if it isn't frequent, so [a,b,c,d] wouldn't be frequent and was not created correctly.

get_new_frequents

This function gets all the baskets as input and gets the most frequents itemsets of the next step. In this function we get help from a new function which is defined as "count_freq" which gets a basket as input and for each candidate of that step, if that candidate exists on that basket, it creates an output as (candidate,1). Finally, with counting all candidates, we can get most frequent itemsets.

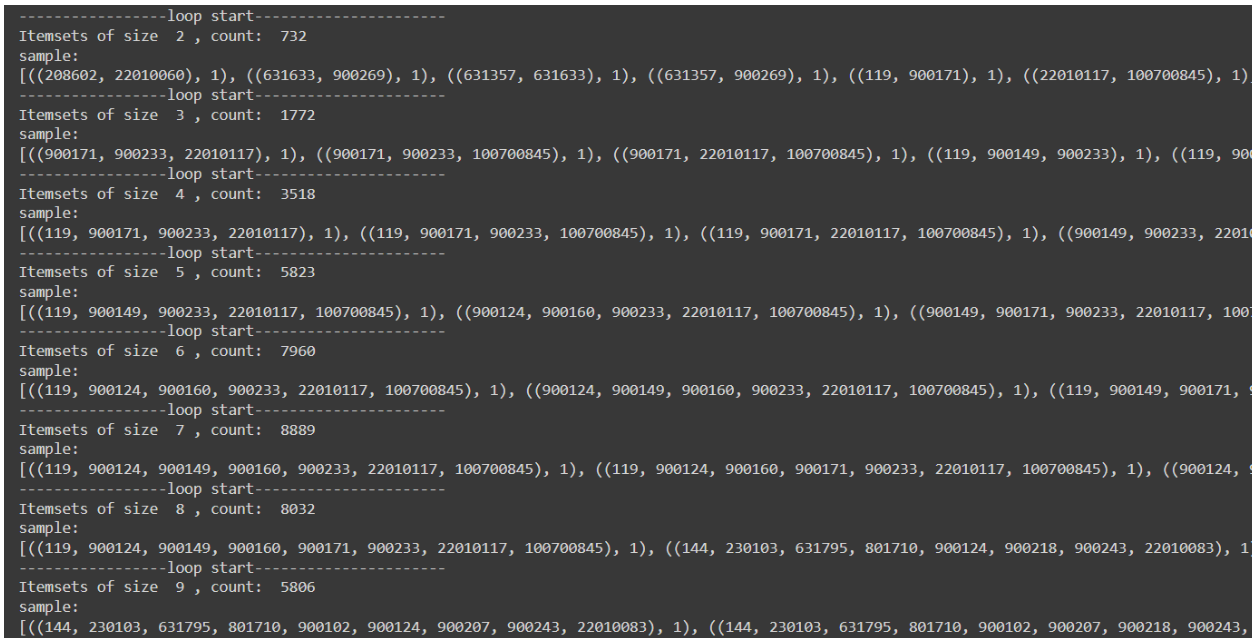


```python
def apriori(baskets_rdd, MIN_COUNT, verbose=False):
  frequent_items_rdd = baskets_rdd.flatMap(lambda basket: [((item,),1) for item in basket]).reduceByKey(lambda x,y: x+y).filter(lambda x: x[1] > MIN_COUNT)

  def generate_combinations(old_combinations):
    """
    Input old_frequent_itemsets, and create new candidates from it
    """

    def generate_combinations_util(old_combination):
      """
      lambda function that maps an old combination to a number of new candidate combinations
      """
      old_combination = old_combination[0]
      old_combination_max_item = old_combination[-1]

      # Can do here numpy way
      bigger_items = remaining_items[remaining_items>old_combination_max_item]
      new_candidates = []
      for x in bigger_items:
        new_candidates.append( old_combination + (x,) )

      return new_candidates

    remaining_items = np.array(old_combinations.flatMap(lambda x: x[0]).distinct().sortBy(lambda x: x).collect())

    new_combinations = old_combinations.flatMap(generate_combinations_util)
    return new_combinations

  def get_new_frequents(candidates):
    _candidates = candidates.collect()
    def count_freq(basket):
      candidates_present = []
      for candidate in _candidates:
        if set(candidate) <= set(basket):
          candidates_present.append( (candidate,1) )
```



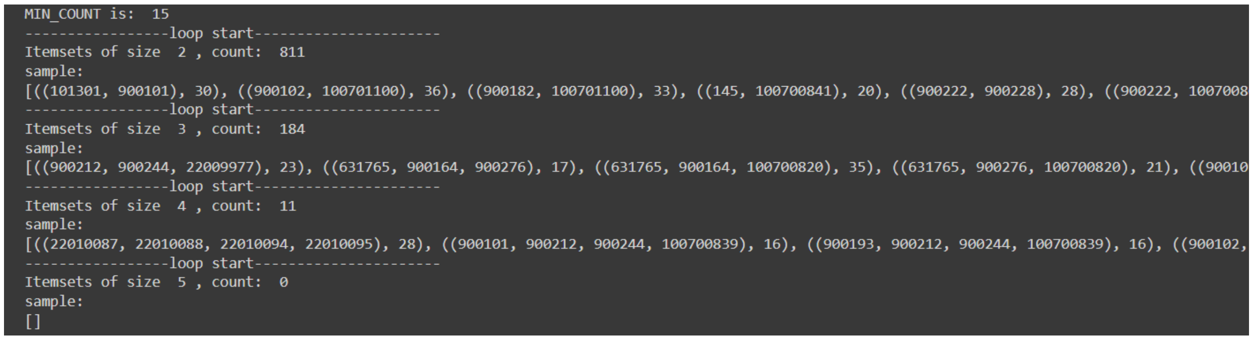
```python
  frequent_itemsets_rdd = baskets_rdd.flatMap(count_freq).reduceByKey(lambda x,y: x+y).filter(lambda x: x[1]>MIN_COUNT)

  return frequent_itemsets_rdd

if verbose:
  print('MIN_COUNT is: ', MIN_COUNT)
k = 1
frequent_itemsets_rdd = frequent_items_rdd
frequent_itemsets_rdds = []
while frequent_itemsets_rdd.count() != 0:
  frequent_itemsets_rdds.append(frequent_itemsets_rdd)
  if verbose:
    print('----------------loop start--------------------')
  k += 1
  candidates = generate_combinations(frequent_itemsets_rdd)
  frequent_itemsets_rdd = get_new_frequents(candidates)
  if verbose:
    print('Itemsets of size ', k, ', count: ', frequent_itemsets_rdd.count())
    print('sample: ')
    print(frequent_itemsets_rdd.take(10))

return frequent_itemsets_rdds
```

The result of these functions is shown in below as an input.



```
-----------------loop start---------------------
Itemsets of size  2 , count:  732
sample:
[((208602, 22010060), 1), ((631633, 900269), 1), ((631357, 631633), 1), ((631357, 900269), 1), ((119, 900171), 1), ((22010117, 100700845), 1)
-----------------loop start---------------------
Itemsets of size  3 , count:  1772
sample:
[((900171, 900233, 22010117), 1), ((900171, 900233, 100700845), 1), ((900171, 22010117, 100700845), 1), ((119, 900149, 900233), 1), ((119, 90
-----------------loop start---------------------
Itemsets of size  4 , count:  3518
sample:
[((119, 900171, 900233, 22010117), 1), ((119, 900171, 900233, 100700845), 1), ((119, 900171, 22010117, 100700845), 1), ((900149, 900233, 2201
-----------------loop start---------------------
Itemsets of size  5 , count:  5823
sample:
[((119, 900149, 900233, 22010117, 100700845), 1), ((900124, 900160, 900233, 22010117, 100700845), 1), ((900149, 900171, 900233, 22010117, 100
-----------------loop start---------------------
Itemsets of size  6 , count:  7960
sample:
[((119, 900124, 900160, 900233, 22010117, 100700845), 1), ((900124, 900149, 900160, 900233, 22010117, 100700845), 1), ((119, 900149, 900171,
-----------------loop start---------------------
Itemsets of size  7 , count:  8889
sample:
[((119, 900124, 900149, 900160, 900233, 22010117, 100700845), 1), ((119, 900124, 900160, 900171, 900233, 22010117, 100700845), 1), ((900124,
-----------------loop start---------------------
Itemsets of size  8 , count:  8032
sample:
[((119, 900124, 900149, 900160, 900171, 900233, 22010117, 100700845), 1), ((144, 230103, 631795, 801710, 900124, 900218, 900243, 22010083), 1
-----------------loop start---------------------
Itemsets of size  9 , count:  5806
sample:
[((144, 230103, 631795, 801710, 900102, 900124, 900207, 900243, 22010083), 1), ((144, 230103, 631795, 801710, 900102, 900207, 900218, 900243,
```

Now we see the result of A-Priori algorithm for a sample is as below.



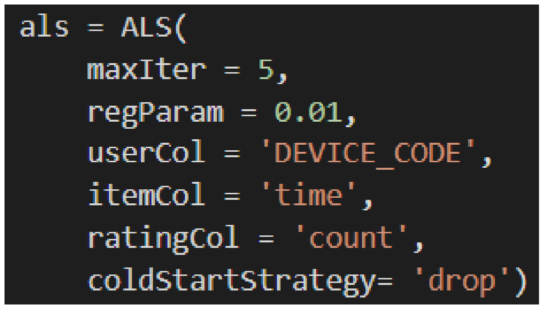
```
MIN_COUNT is:  15
-----------------loop start---------------------
Itemsets of size  2 , count:  811
sample:
[((101301, 900101), 30), ((900102, 100701100), 36), ((900182, 100701100), 33), ((145, 100700841), 20), ((900222, 900228), 28), ((900222, 1007008
-----------------loop start---------------------
Itemsets of size  3 , count:  184
sample:
[((900212, 900244, 22009977), 23), ((631765, 900164, 900276), 17), ((631765, 900164, 100700820), 35), ((631765, 900276, 100700820), 21), ((90010
-----------------loop start---------------------
Itemsets of size  4 , count:  11
sample:
[((22010087, 22010088, 22010094, 22010095), 28), ((900101, 900212, 900244, 100700839), 16), ((900193, 900212, 900244, 100700839), 16), ((900102,
-----------------loop start---------------------
Itemsets of size  5 , count:  0
sample:
[]
```

2.4 Idea 4 (Collaborative Filtering using ALS)

In this part we use ALS algorithm to implement collaborative filtering to obtain a recommender system for the cameras.

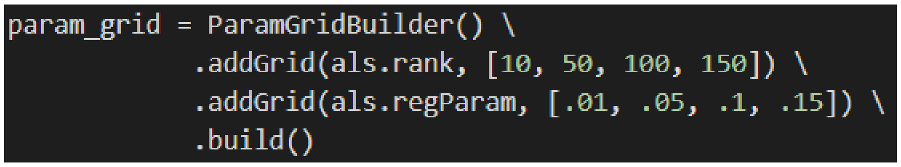
At first we obtain the number of pass ways for each cameras and cars and then we separate the data to train and test data to learn ALS model.

The architecture of the used ALS model is shown in below.



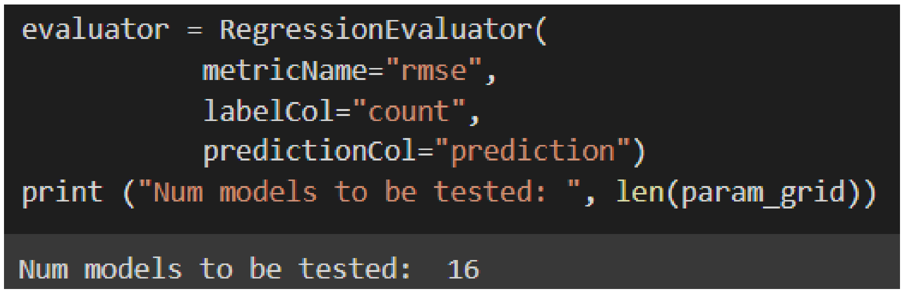
```
als = ALS(
    maxIter = 5,
    regParam = 0.01,
    userCol = 'DEVICE_CODE',
    itemCol = 'time',
    ratingCol = 'count',
    coldStartStrategy= 'drop')
```

Now we will define the tuning parameter using param_grid function.

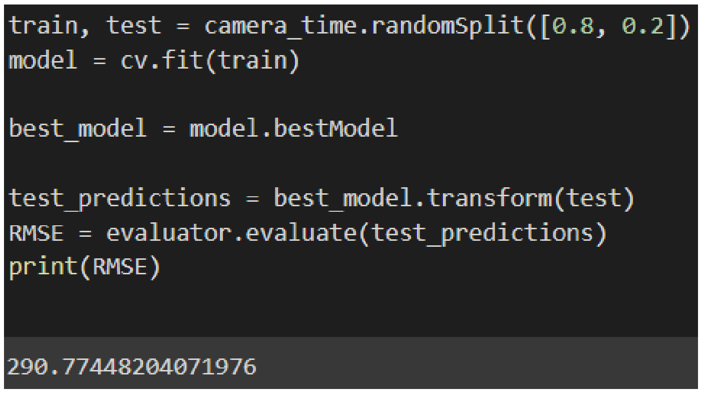


```
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 100, 150]) \
            .addGrid(als.regParam, [.01, .05, .1, .15]) \
            .build()
```

In this part we define the evaluator, select rmse as metricName in evaluator.

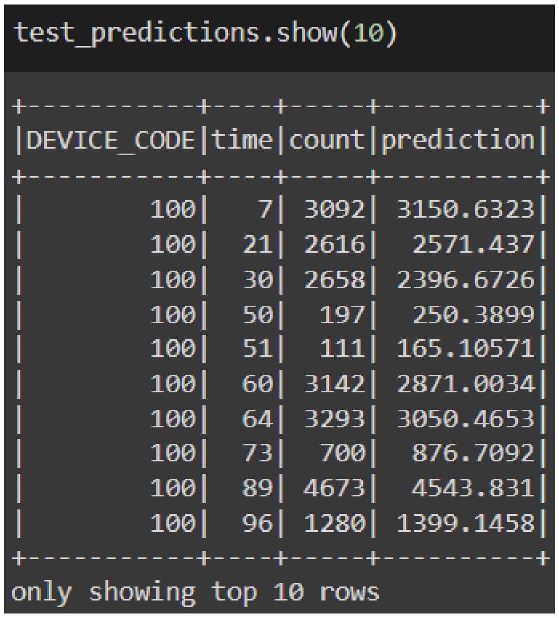


```
evaluator = RegressionEvaluator(
           metricName="rmse",
           labelCol="count",
           predictionCol="prediction")
print ("Num models to be tested: ", len(param_grid))

Num models to be tested:  16
```



```
train, test = camera_time.randomSplit([0.8, 0.2])
model = cv.fit(train)

best_model = model.bestModel

test_predictions = best_model.transform(test)
RMSE = evaluator.evaluate(test_predictions)
print(RMSE)

290.77448204071976
```

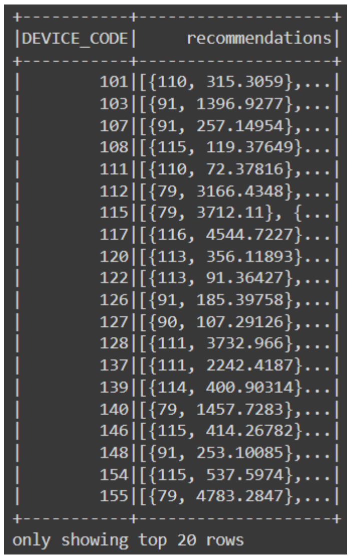
Now we see the result of the prediction test in the below. Then we analyze the result and figure out whether our model performed well or not.



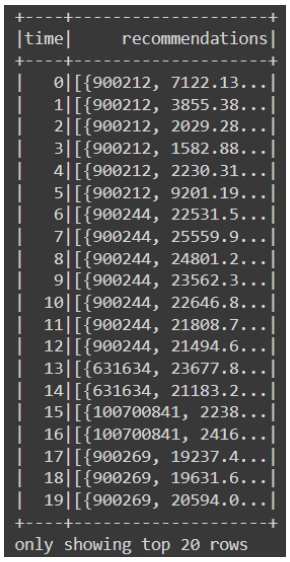
```
test_predictions.show(10)

+-----------+----+-----+----------+
|DEVICE_CODE|time|count|prediction|
+-----------+----+-----+----------+
|        100|   7| 3092| 3150.6323|
|        100|  21| 2616|  2571.437|
|        100|  30| 2658| 2396.6726|
|        100|  50|  197|  250.3899|
|        100|  51|  111| 165.10571|
|        100|  60| 3142| 2871.0034|
|        100|  64| 3293| 3050.4653|
|        100|  73|  700|  876.7092|
|        100|  89| 4673|  4543.831|
|        100|  96| 1280| 1399.1458|
+-----------+----+-----+----------+
only showing top 10 rows
```

As we can see, the prediction and the count columns are almost near to each other and it shows that the model perform almost well to predict the count of cars that cameras recorded. Now, we make recommendations based on our ALS model.

recommendForAllUsers(n) function in ALS takes n recommendations. Now we can predict the palte of each car which has the most probability. Also we can predict the time of cameras which has the most probability to record a plate of a car. We did that for the first 10 assumptions and the result is shown.



```
+-----------+--------------------+
|DEVICE_CODE|     recommendations|
+-----------+--------------------+
|        101|[{110, 315.3059},...|
|        103|[{91, 1396.9277},...|
|        107|[{91, 257.14954},...|
|        108|[{115, 119.37649}...|
|        111|[{110, 72.37816},...|
|        112|[{79, 3166.4348},...|
|        115|[{79, 3712.11}, {...|
|        117|[{116, 4544.7227}...|
|        120|[{113, 356.11893}...|
|        122|[{113, 91.36427},...|
|        126|[{91, 185.39758},...|
|        127|[{90, 107.29126},...|
|        128|[{111, 3732.966},...|
|        137|[{111, 2242.4187}...|
|        139|[{114, 400.90314}...|
|        140|[{79, 1457.7283},...|
|        146|[{115, 414.26782}...|
|        148|[{91, 253.10085},...|
|        154|[{115, 537.5974},...|
|        155|[{79, 4783.2847},...|
+-----------+--------------------+
only showing top 20 rows
```

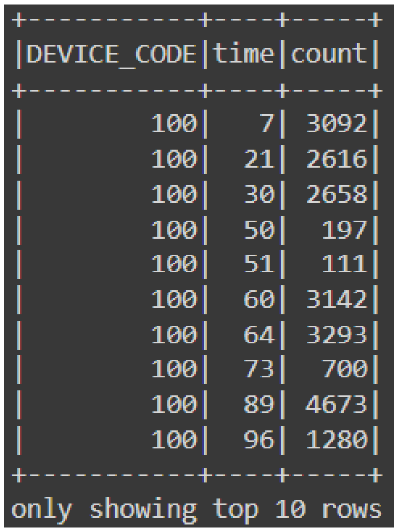


```
+----+--------------------+
|time|     recommendations|
+----+--------------------+
|   0|[{900212, 7122.13...|
|   1|[{900212, 3855.38...|
|   2|[{900212, 2029.28...|
|   3|[{900212, 1582.88...|
|   4|[{900212, 2230.31...|
|   5|[{900212, 9201.19...|
|   6|[{900244, 22531.5...|
|   7|[{900244, 25559.9...|
|   8|[{900244, 24801.2...|
|   9|[{900244, 23562.3...|
|  10|[{900244, 22646.8...|
|  11|[{900244, 21808.7...|
|  12|[{900244, 21494.6...|
|  13|[{631634, 23677.8...|
|  14|[{631634, 21183.2...|
|  15|[{100700841, 2238...|
|  16|[{100700841, 2416...|
|  17|[{900269, 19237.4...|
|  18|[{900269, 19631.6...|
|  19|[{900269, 20594.0...|
+----+--------------------+
only showing top 20 rows
```

As we can see our model has this ability to work as Recommender System. As a matter of fact, our recommender system can estimate number of cars which will be recorded by each camera for specific days and hours. The application of this recommender system is that we can decide that which camera can be turned off for saving energy or we can decide to keep which cameras always on or we can decide to use specific cameras in specific roads for specific hours.

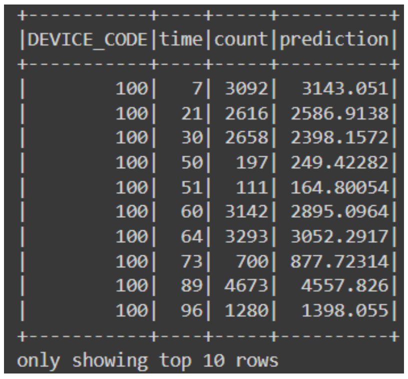
Also another useful application of this is recommending roads in navigating applications like Google Maps, Balad, Neshaan or other ones. The base of this recommendation is that if a camera record car passing a lot, it can alert that cars don't pass from that road and recommend other roads base on their traffic.

Now, we test the algorithm for some camera to see the result. The result can be seen in below.



```
+-----------+----+-----+
|DEVICE_CODE|time|count|
+-----------+----+-----+
|        100|   7| 3092|
|        100|  21| 2616|
|        100|  30| 2658|
|        100|  50|  197|
|        100|  51|  111|
|        100|  60| 3142|
|        100|  64| 3293|
|        100|  73|  700|
|        100|  89| 4673|
|        100|  96| 1280|
+-----------+----+-----+
only showing top 10 rows
```

Now for these cameras we predict the number of count of seen cars. The result can be seen in below.



```
+-----------+----+-----+----------+
|DEVICE_CODE|time|count|prediction|
+-----------+----+-----+----------+
|        100|   7| 3092|  3143.051|
|        100|  21| 2616| 2586.9138|
|        100|  30| 2658| 2398.1572|
|        100|  50|  197| 249.42282|
|        100|  51|  111| 164.80054|
|        100|  60| 3142| 2895.0964|
|        100|  64| 3293| 3052.2917|
|        100|  73|  700| 877.72314|
|        100|  89| 4673|  4557.826|
|        100|  96| 1280|  1398.055|
+-----------+----+-----+----------+
only showing top 10 rows
```

2.5 Idea 5 (Pixie)

In this part we use Page Rank algorithm to find the most similar cars and cameras. In the other words, we use pixie to find similarities of cars and cameras in this part.

For using this algorithm, we make a 2 parts graph which one side of that are cars and the other side are cameras. Then we consider an edge between each car and the camera which recorded that car that the weight of each edge is equal to number of passing that car which was recorded by that camera. The we save all the edges in a list to obtain exit edge of each vertex.

Then we implement Pixie algorithm. This algorithm can be started from either a camera or a car. We explain just one side of that is this report but the implementation for both sides are written in the notebook.

With starting from a primary camera, we select a camera accidentally and then we we select list of all cars which are recorded by that camera and do this process for cars again. We set a counter for each camera and with passing through that camera on the graph, we add a number to that counter.

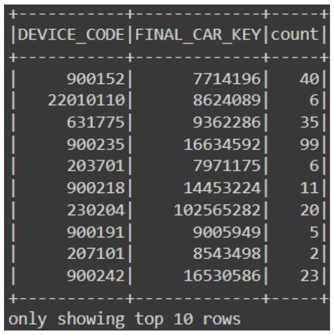
After we pass all of the list of cameras and cars on the graph we print list of counters as output.

An important note of this algorithm is that we turn back to the previous vertex with a specific probability. In this part I put this probability equal to 0.1.

Now the process of this part is explained in following of this part.

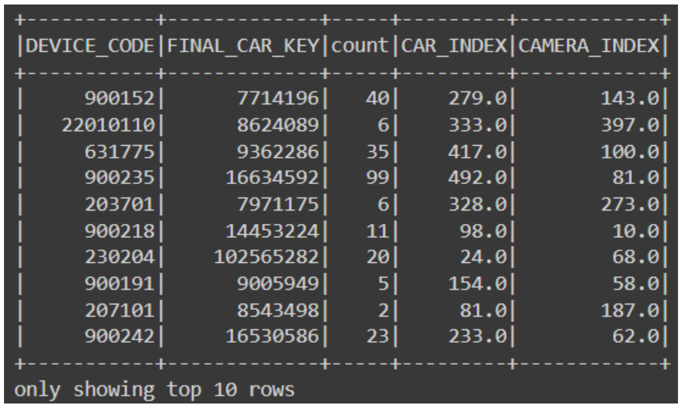
For the first step, we just keep the most passed cars to have better analysis through the data.

Then, we make the dataframe shows each vertex our graph. The result can be seen in below.



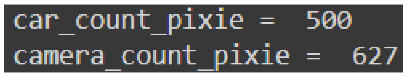
```
+-----------+-------------+-----+
|DEVICE_CODE|FINAL_CAR_KEY|count|
+-----------+-------------+-----+
|     900152|      7714196|   40|
|   22010110|      8624089|    6|
|     631775|      9362286|   35|
|     900235|     16634592|   99|
|     203701|      7971175|    6|
|     900218|     14453224|   11|
|     230204|    102565282|   20|
|     900191|      9005949|    5|
|     207101|      8543498|    2|
|     900242|     16530586|   23|
+-----------+-------------+-----+
only showing top 10 rows
```

Then we add index to cars and cameras. The result is show in below.



```
+-----------+------------+-----+---------+------------+
|DEVICE_CODE|FINAL_CAR_KEY|count|CAR_INDEX|CAMERA_INDEX|
+-----------+------------+-----+---------+------------+
|     900152|     7714196|   40|    279.0|       143.0|
|   22010110|     8624089|    6|    333.0|       397.0|
|     631775|     9362286|   35|    417.0|       100.0|
|     900235|    16634592|   99|    492.0|        81.0|
|     203701|     7971175|    6|    328.0|       273.0|
|     900218|    14453224|   11|     98.0|        10.0|
|     230204|   102565282|   20|     24.0|        68.0|
|     900191|     9005949|    5|    154.0|        58.0|
|     207101|     8543498|    2|     81.0|       187.0|
|     900242|    16530586|   23|    233.0|        62.0|
+-----------+------------+-----+---------+------------+
only showing top 10 rows
```

Now we make 2 RDDs which obtain exit vertex of each car and camera. The results are shown in below. (Note: The car and camera heads are shown in the notebook and because the results are too long, I refuse to take them in this report.)



```
car_count_pixie =  500
camera_count_pixie =  627
```

Then with using Pixie algorithm, we can find the most similar camera to each camera. Now in the next cell we select query of camera and query of car. In the below "p" shows probability of getting back to the first head and "iter" shows steps of the algorithm. In this part I define 3 functions as "pixie", "cars_pixie" and "select_k" that are in the notebook.

The application of this algorithm is that we can find near cameras. It’s because of that if a camera record passing of a specific car, its nearest cameras will record that car with more possibility.

Be careful that other factors like similarities of cameras are effective in this algorithm. It means that similarity of cameras is not only base on their physical distance.

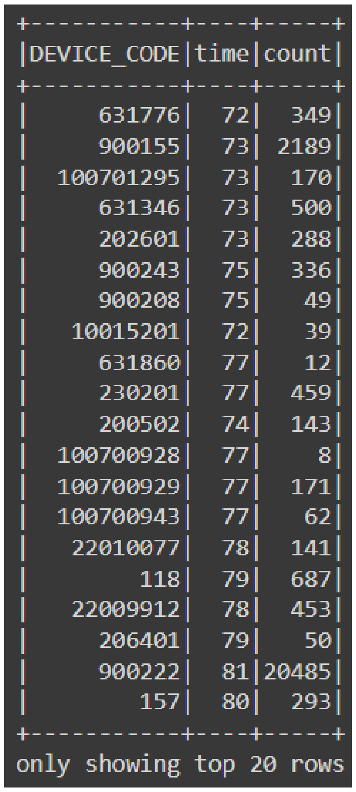
Generally, we can find the most similar cameras in this algorithm. If we draw the graph that each vertex be the cameras we can reach to map of the city approximately.

The other side of this algorithm is that this algorithm can find the nearest cars too exactly as the same as cameras. For example, if we mark an anomaly of a specific car, we can find other cars which have the same action on that city and penalize them. Also we can find best citizens in the city and encourage them because we can find any cars that are similar to each other and doesn’t matter whether that action is fine or not.

2.6 Idea 6 (HITS (Hubs and Authority))

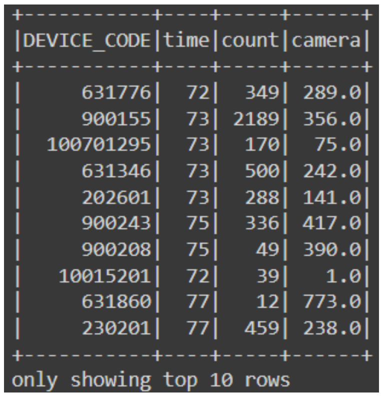
In this part the cameras are hubs and the dates are authorities and then by using SVD we implement HITS algorithm.

At first we calculate number of recording of cameras in specific hours of the week. The code of this part is shown in below.



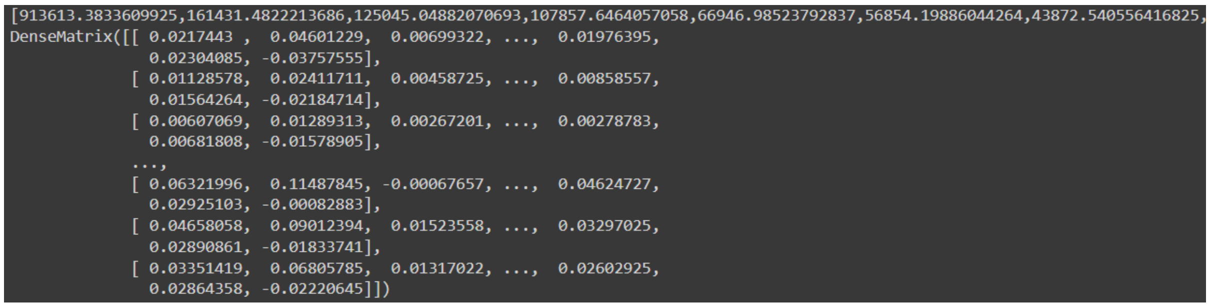
```
+-----------+----+-----+
|DEVICE_CODE|time|count|
+-----------+----+-----+
|     631776|  72|  349|
|     900155|  73| 2189|
|  100701295|  73|  170|
|     631346|  73|  500|
|     202601|  73|  288|
|     900243|  75|  336|
|     900208|  75|   49|
|   10015201|  72|   39|
|     631860|  77|   12|
|     230201|  77|  459|
|     200502|  74|  143|
|  100700928|  77|    8|
|  100700929|  77|  171|
|  100700943|  77|   62|
|   22010077|  78|  141|
|        118|  79|  687|
|   22009912|  78|  453|
|     206401|  79|   50|
|     900222|  81|20485|
|        157|  80|  293|
+-----------+----+-----+
only showing top 20 rows
```

Then we obtain indices of cameras. In this part we get a matrix which each elements of that shows the number of repeat of recording of a camera in a specific time. The claimed matrix is sparse.



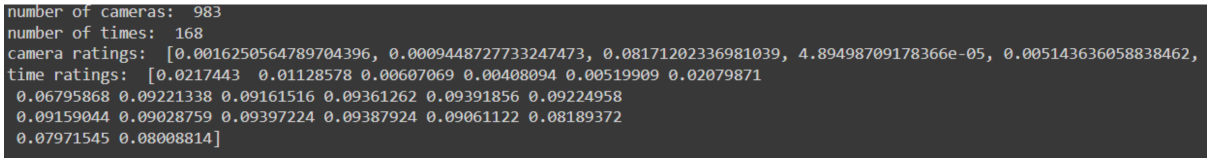
```
+-----------+----+-----+------+
|DEVICE_CODE|time|count|camera|
+-----------+----+-----+------+
|     631776|  72|  349| 289.0|
|     900155|  73| 2189| 356.0|
|  100701295|  73|  170|  75.0|
|     631346|  73|  500| 242.0|
|     202601|  73|  288| 141.0|
|     900243|  75|  336| 417.0|
|     900208|  75|   49| 390.0|
|   10015201|  72|   39|   1.0|
|     631860|  77|   12| 773.0|
|     230201|  77|  459| 238.0|
+-----------+----+-----+------+
only showing top 10 rows
```

Then we obtain SVD decomposition with using "computeSVD" function. The result is shown in below.



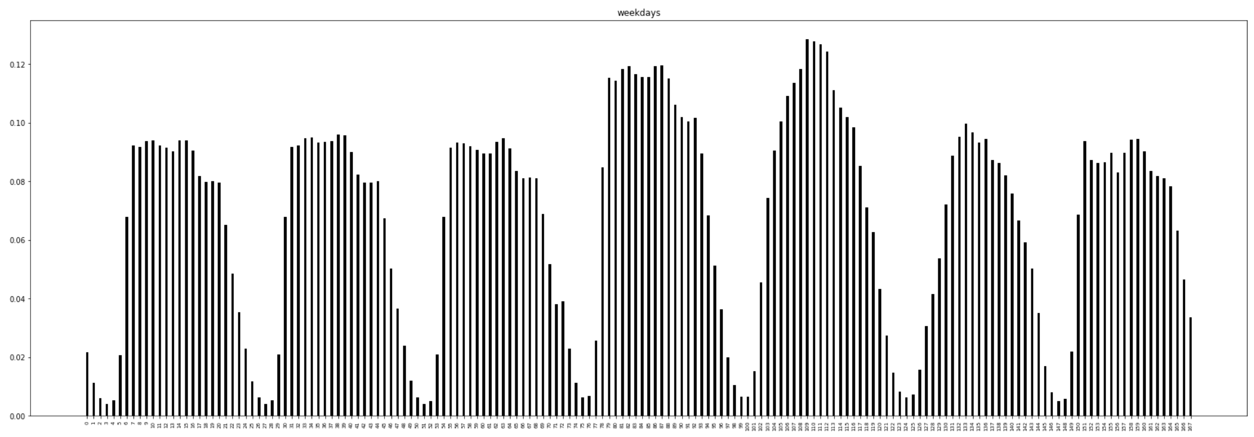
```
[913613.3833609925,161431.4822213686,125045.04882070693,107857.6464057058,66946.98523792837,56854.19886044264,43872.540556416825,
DenseMatrix([[ 0.0217443 ,  0.04601229,  0.00699322, ...,  0.01976395,
               0.02304085, -0.03757555],
             [ 0.01128578,  0.02411711,  0.00458725, ...,  0.00858557,
               0.01564264, -0.02184714],
             [ 0.00607069,  0.01289313,  0.00267201, ...,  0.00278783,
               0.00681808, -0.01578905],
             ...,
             [ 0.06321996,  0.11487845, -0.00067657, ...,  0.04624727,
               0.02925103, -0.00082883],
             [ 0.04658058,  0.09012394,  0.01523558, ...,  0.03297025,
               0.02890861, -0.01833741],
             [ 0.03351419,  0.06805785,  0.01317022, ...,  0.02602925,
               0.02864358, -0.02220645]])
```

Now we obtain value of hubs and authorities. The result is shown in below.



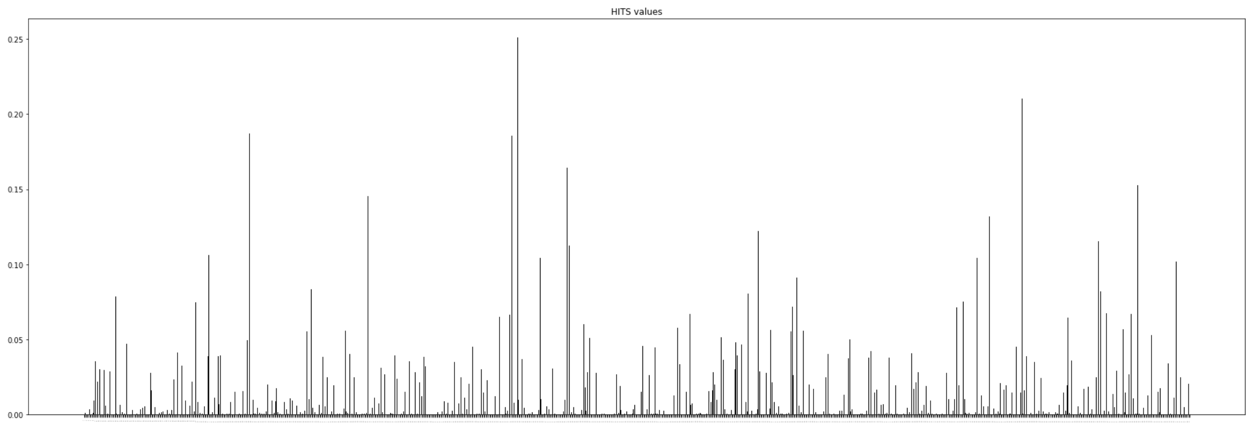
```
number of cameras:  983
number of times:  168
camera ratings:  [0.0016250564789704396, 0.0009448727733247473, 0.08171202336981039, 4.89498709178366e-05, 0.005143636058838462,
time ratings:  [0.0217443  0.01128578 0.00607069 0.00408094 0.00519909 0.02079871
 0.06795868 0.09221338 0.09161516 0.09361262 0.09391856 0.09224958
 0.09159044 0.09028759 0.09397224 0.09387924 0.09061122 0.08189372
 0.07971545 0.08008814]
```

Now we plot the histogram of hours of the week which shows which time of the week has the most amount of cameras recordings. The result is shown in below.



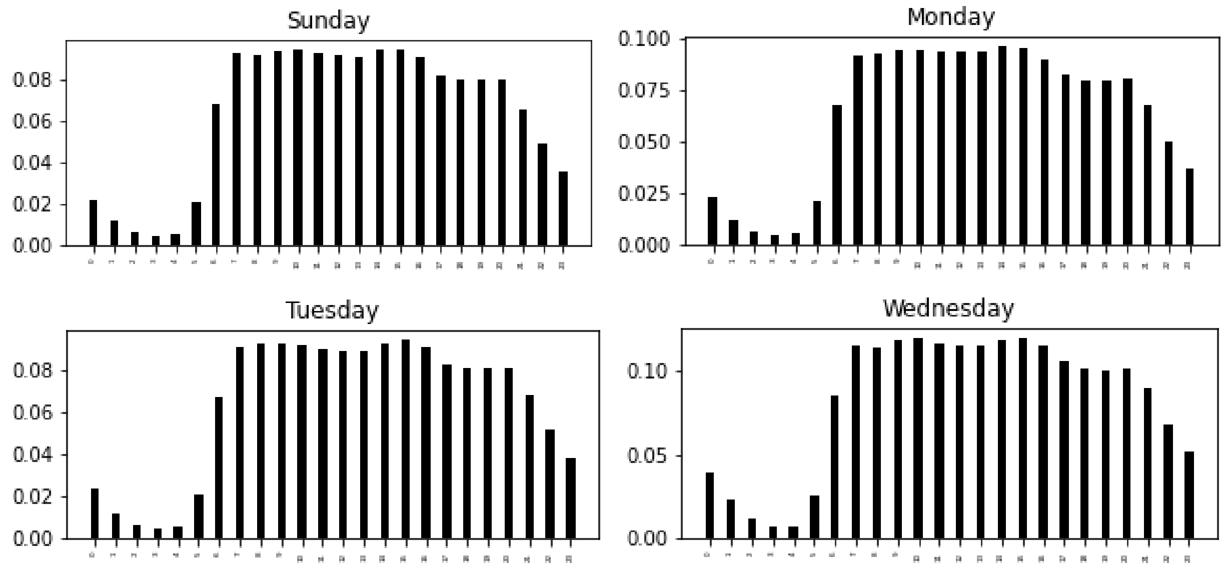
As it can be seen, the fifth day of the week has the most recording of the cameras of passing cars. We can conclude that, that day might be the busiest and populated day.

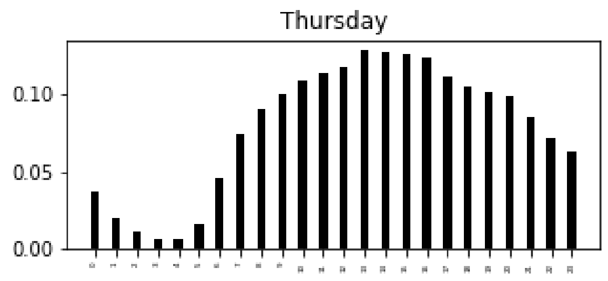
Now, we show HITS value of cameras in below.

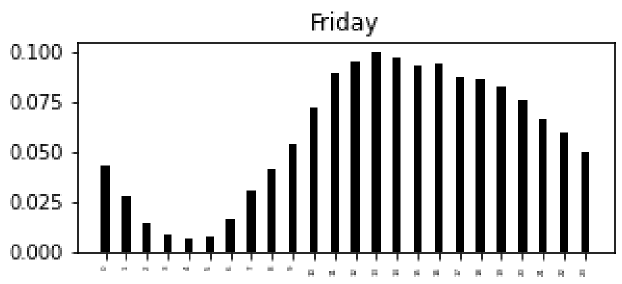


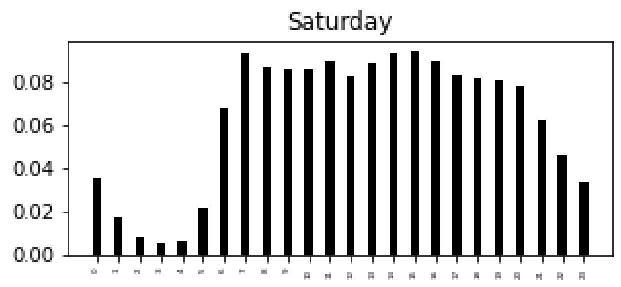
As we can see we can conclude the amount of probability of activity of each camera in specific hours which can shows that each road can be more crowded in each day of the week. But as I said it's just a possibility that shows the activity of recording of each camera and cannot conclude us to valid results.

Now we can plot the histogram of the activity of each camera in specific hours of each day. The plots of this part are shown in below.









As we can see, the amount of passing of cars in starting hours of the days are too low and it's too crowded in the middle of the day and night. The above histograms properly make sense with what is happening in real life because at the start of a data means, 00:00 people are still out of their houses and the streets are still crowded but at 03:00 AM streets become less crowded and at 05:00 AM that people starts to get out of their how for going to work, streets start to become more crowded and this process continues 09:00 PM and then streets start to become less crowded again because people go home at these hours. The behavior of people is different in different days. For example, morning of Fridays is less crowded because most of people in Iran are chilling or sleeping in their houses.

Conclusion and Other Ideas

In this project, we deployed 6 ideas which some of them were recommended by project document and some of them were new.

We could deploy more ideas for this data but unfortunately because of lack of time we couldn’t deploy them.

1. One of these ideas are finding anomaly cars by checking difference between time of their passing through near cameras and if this time is informally fast, we can count them as anomaly and can consume penalty for them.

2. The other one is following paths of cars by checking all history of their passing through different cameras in different times. With this approach we can find out that which streets each car passed during a specific time.

3. One of the interesting ideas is to check time of activities of each camera to figure out in which times their activities were a lot and in which times not. With this analyzing we can check activities of cameras and for example set a program to turn on and off cameras when their activities are more or less than normal mode.

4. Another interesting Ideas is controlling car traffic by checking amount passing cars through cameras and find most crowded streets and also find patterns of passing cars through each camera in each day of week.

5. The other idea is to find pattern of people behavior in days of week and vacations and find their most interesting streets for advertising and marketing and find busiest streets for putting billboards.