به نام خدا

درس آنالیز داده های حجیم

پروژه پایانی درس

تحلیل دیتای ترافیکی شهر تهران

استاد:دکترغلامپور

دانشجو:سجاد هاشم بیکی

پاییز 1401

-توضیح کلی در رابطه با روند پروژه

در این پروژه قصد داریم داده های ترافیکی شهر تهران را با استفاده از مفاهیم بیان شده در درس تحلیل وبررسی کنیم.

در متن پروژه ایده هایی مطرح شده است که بخش اصلی پروژه من مربوط به پیاده سازی این ایده ها میباشد.همچنین سعی شده برخی ایده های مطرح شده در طی جلسات کلاس درس،پیاده سازی شوند.

مانند کلاسترینگ وکاهش ابعاد.

4 بخش پروژه مربوط به ایده های مطرح شده در متن پروژه میباشد.

- CF on cameras
- SVD decomposition and Utility matrix
- Pixie
- Frequent-items

4 بخش دیگر پروژه مربوط به ایده های خارج از متن پروژه میباشند.

- CF on cars
- Car cosine similarity
- Clustering
- PCA on cameras

قالب کلی گزارش بدین صورت است که در هربخش،ابتدا یک توضیح مربوط به الگوریتم استفاده شده و نحوه پیاده سازی آن داده میشود وسپس قطعه کدهای مربوط به هر زیرقسمت بررسی میشود.

در نهایت خروجی هر قسمت بررسی و نتایج آن تفسیر میشود.

۱-اماده سازی و پیش پردازش داده ها

ابتدا کتابخانه های مربوط به پای اسپارک را ایمپورت میکنیم ویک Session میسازیم.

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import functions
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField, StringType, IntegerType,TimestampType
from pyspark.sql.functions import col,current_timestamp,to_date,hour,dayofweek
spark = SparkSession \
    .builder \
    .appName("Spark_Processor") \
    .master("local[*]") \
    .getOrCreate()

sc=spark.sparkContext
```

دیتای مربوط به پروژه را ازگوگل درایو میخوانیم،از حالت فشرده درمیاوریم وسپس در یک دیتافریم ذخیره میکنیم.

```
#import require libraries
import numpy as np
import matplotlib.pyplot as plt
import random
import pandas as pd
import seaborn as sns
from scipy.linalg import eigh
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

```
# import data from google drive
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
# !unzip "/content/drive/My Drive/MDA2022ProjectData.zip" -d "/content/drive/My Drive/MDA2022ProjectData"
```

```
# read data and creat dataframe
df=spark.read.csv('/content/drive/My Drive/MDA2022ProjectData/MDA2022ProjectData.csv',header=True)
```

کد مربوط به unzip کردن دیتا فقط یکبار ران شده و برای دفعات بعد کامنت شده است.

دیتا فریم بدست امده بدین صورت میباشد:

```
#  showing top 5 rows of df
df.show(5)
```

```
+-----------+---------+---------------+-------------+----------------+----------+-------------------+
|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|
+-----------+---------+---------------+-------------+----------------+----------+-------------------+
```

ستون های بلااستفاده را حذف میکنیم.

```
#drop useless columns
df = df[['DEVICE_CODE','FINAL_CAR_KEY','PASS_DAY_TIME']]
df.show(5)
```

```
+-----------+-------------+-------------------+
|DEVICE_CODE|FINAL_CAR_KEY|      PASS_DAY_TIME|
+-----------+-------------+-------------------+
|   22010054|     64111706|2021-12-22 01:24:18|
|   22010040|     64111706|2021-12-22 01:40:29|
|   22010079|     63348486|2021-12-22 01:14:32|
|   22010053|     64111706|2021-12-22 00:32:33|
|   22010044|     64111706|2021-12-22 01:25:06|
+-----------+-------------+-------------------+
only showing top 5 rows
```

با استفاده از توابع hour و dayofweek ساعت و روز هفته را به دیتا فریم اضافه میکنیم.

```
#add columns that show time and day of the week
# Sunday:1 to Saturday:7
df_1 = df.withColumn('DAY_OF_WEEK',dayofweek(df.PASS_DAY_TIME))
df_2 = df_1.withColumn('HOUR', hour(df_1.PASS_DAY_TIME))
```

```
df_2.show(5)
```

```
+-----------+-------------+-------------------+-----------+----+
|DEVICE_CODE|FINAL_CAR_KEY|      PASS_DAY_TIME|DAY_OF_WEEK|HOUR|
+-----------+-------------+-------------------+-----------+----+
|   22010054|     64111706|2021-12-22 01:24:18|          4|   1|
|   22010040|     64111706|2021-12-22 01:40:29|          4|   1|
|   22010079|     63348486|2021-12-22 01:14:32|          4|   1|
|   22010053|     64111706|2021-12-22 00:32:33|          4|   0|
|   22010044|     64111706|2021-12-22 01:25:06|          4|   1|
+-----------+-------------+-------------------+-----------+----+
```

- پیاده سازی CF روی دوربین ها

میتوان این روش را روی خودروها و دوربین ها پیاده سازی کرد.ما در این بخش به دوربین ها می پردازیم و در بخش بعدی شباهت خودروها را با این روش محاسبه میکنیم.

ابتدا به کد های مربوط به این بخش میپردازیم و نتایج آن را تحلیل میکنیم.

در ابتدا بایستی بردار 7*24 را برای هر دوربین بدست بیاوریم.که در واقع نشان میدهد هر دوربین در یک روز مشخص و ساعت مشخص چه تعداد تردد را ثبت کرده است.

برای اینکار ابتدا یک ماتریس صفر با ابعاد (962*7*24) میسازیم . سپس د مرحله بعد،درایه های ماتریس را پر میکنیم.که 962 تعداد دوربین ها میباشد.

```
#creat  vector  7*24
vec_7_24= np.zeros(shape=(df_2.select(df_2.DEVICE_CODE).distinct().count(),7,24))
```

```
vec_7_24.shape
```

```
(962, 7, 24)
```

برای اینکار ابتدا دیتافریم زیر را تشکیل میدهیم.که تاریخ را براساس روزهای هفته دارد.روز یکشنبه به عدد یک و شنبه به عدد هفت کد شده است.برای ساعت ها نیز همین طور عمل کردیم.برای مثال،ردیف دوم متعلق به روز چهارشنبه و ساعت اول(یک بامداد) میباشد.در اینجا صرفا ساعت ها را در نظر گرفتیم و لزومی به در نظر گرفتن دقیقه وجود ندارد.

DEVICE_CODE	FINAL_CAR_KEY	PASS_DAY_TIME	DAY_OF_WEEK	HOUR
22010059	63315887	2021-12-22 00:51:16	4	0
100701235	47807908	2021-12-22 01:07:42	4	1
635607	81726547	2021-12-22 00:05:34	4	0
10015301	36037995	2021-12-22 00:51:14	4	0
22010044	64111706	2021-12-22 01:30:33	4	1

حال با استفاده از دستور groupby یک دیتا فریم میسازیم که نشان میدهد هر دوربین در یک ساعت و روز مشخص چه تعداد تردد ثبت کرده است.دیتا فریم بدست امده به صورت زیر میباشد.برای مثال سطر اول نشان میدهد دوربین 200502 در روز چهارشبه و ساعت 12 شب،تعداد 9 تردد را ثبت کرده است.

DEVICE_CODE	DAY_OF_WEEK	HOUR	count
200502	4	0	9
100701100	4	3	3
22010079	4	3	10
900242	4	4	4
22010117	4	5	12

حال با استفاده از مقادیر دیتافریم بالا،میتوان ماتریس اولیه که دارای مقادیر صفر بود(vec_7_24) را پر کرد و سپس CF را پیاده سازی میکنیم.

برای اینکار ابتدا به هر دوربین یک ایندکس از صفر نسبت میدهیم. سپس روی دیتافریم بالا حلقه for میزنیم و یک به یک درایه های مربوط به ماتریس(vec_7_24) را پر میکنیم.

قسمت ایندکس گذاری برای هر دوربین با کد زیر انجام شده است.

```
#indexing the cameras
device_dict = {}
for i, devicecode in enumerate(device):
    device_dict[i] = devicecode
device_dict = {j: i for i, j in device_dict.items()}
```

دیکشنری device_dict کد هر دوربین را به همراه ایندکس آن دوربین نشان میدهد.

حال روش CF را روی دوربین ها پیاده میکنیم.در اینجا هر دوربین را میتوان مانند یک کاربر در نظر گرفت.متناظر با مسئله سیستم توصیه گر برای ویدیو،در اینجا هر دوربین یک کاربر است و تعداد تردد های ثبت شده،در روز و ساعت مشخص،همان ratingها هستند.

در اینجا از معیار Pearson Correlation Coefficient برای محاسبه میزان شباهت میان دوربین ها استفاده کردیم. که به راحتی با استفاده از دستور corr قابل محاسبه میباشد.

```
#calculating correlation by corr() methode
user_item_df = pd.DataFrame(user_item_matrix)
cor_df = user_item_df.T.corr()
cor_df
```

دیتافریم زیر هم بستگی میان دوربین ها را نشان میدهد.

	0	1	2	3	4	5	6	7	8	9	...	952	953
0	1.000000	0.146597	0.247875	0.256629	0.693542	0.515575	0.238373	0.205426	0.319099	0.137944	...	0.231572	0.123755
1	0.146597	1.000000	0.165116	-0.041933	0.199462	0.261108	0.246065	0.240740	0.247921	-0.045295	...	0.091927	0.136391
2	0.247875	0.165116	1.000000	0.124076	0.441665	0.329383	0.729441	0.353954	0.423679	0.153221	...	0.529873	-0.026496
3	0.256629	-0.041933	0.124076	1.000000	0.402062	0.079395	0.087244	0.191559	-0.045562	0.166624	...	0.029985	0.140912
4	0.693542	0.199462	0.441665	0.402062	1.000000	0.547321	0.499879	0.422596	0.352065	0.311290	...	0.383071	0.112795
...	...	...	...	...	...	...	...	...	...	...	...	...	...
957	-0.155145	0.105951	-0.116954	-0.083323	-0.209890	-0.119040	-0.141591	-0.101542	-0.158500	-0.009992	...	-0.140500	-0.027353
958	0.182194	-0.019399	-0.049101	-0.002873	0.101665	0.078453	-0.040661	0.103989	-0.073638	0.066311	...	-0.065804	-0.027353
959	0.014743	-0.036263	-0.055956	-0.048663	-0.020001	-0.038817	-0.052424	-0.044864	-0.057531	-0.038128	...	-0.029074	-0.012085
960	0.075320	0.046761	0.011752	-0.069026	0.103789	0.075225	0.155654	0.065171	0.066129	-0.054083	...	0.052386	-0.017142
961	0.036661	-0.036263	0.029699	-0.048663	0.163336	0.144882	0.059060	0.227562	0.067450	0.130437	...	0.168943	-0.012085

962 rows × 962 columns

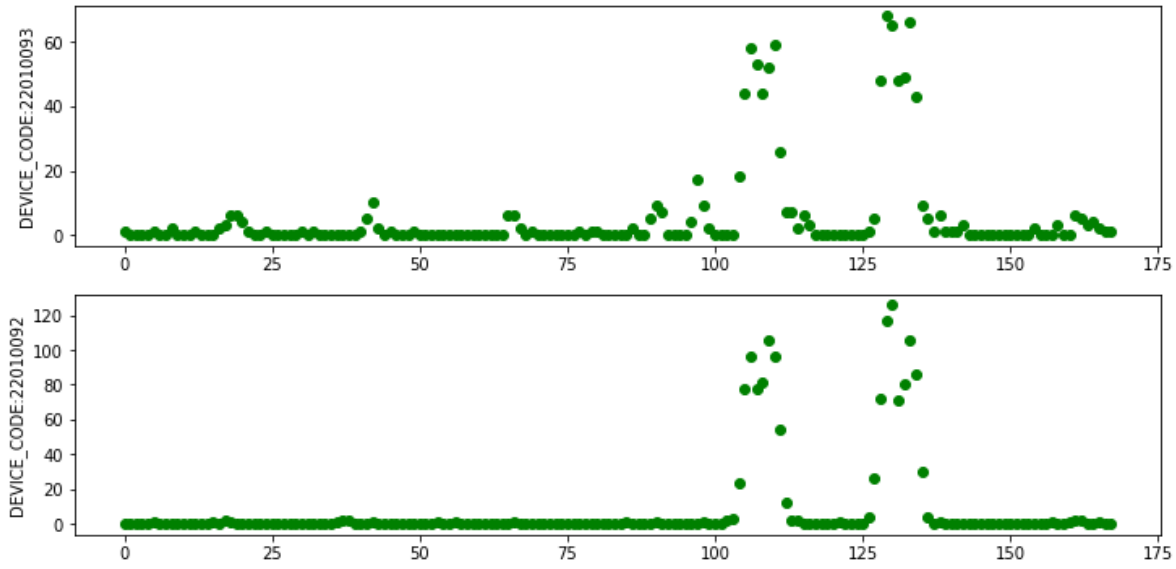
حال باتوجه به داشتن میزان هم بستگی میان دوربین ها و همچنین ایندکس گذاری که برای هر دوربین انجام داده بودیم،میتوان میزان شباهت میان هر دو دوربین را مشاهده کرد.

در ادامه صدتا از بیشترین میزان هم بستگی ها را از ماتریس هم بستگی بدست امده استخراج میکنیم و نمایش میدهیم.

	Camera 1	Camera 2
0	100700864	100700804
1	22010099	22010094
2	900243	900207
3	900277	900222
4	22010087	22010094
...	...	...
95	900273	900246
96	900225	900222
97	900225	900207
98	631634	631633
99	22010093	22010092

100 rows × 2 columns

برای نمایش بهتر دوتا از دوبین های مشابه که در قسمت قبل پیدا کردیم(سطر 99 در دیتا فریم بالا)، بردار متناظر با انها را پلات میکنیم.



همانطور که مشاهده میشود،دو دوربین تردد مشابه ای را در طول ساعت های مختلف روزهای هفته داشته اند(در طول168 ساعت، که محور افقی نشان دهنده ان است).با توجه به نمودار،در دو محدوده زمانی بیشترین تردد را داشتیم.محدوده اول حدود106امین ساعت از هفته میباشد(در ابتدا ایندکس یکشنبه را یک درنظر گرفتیم و شنبه را هفت).

که در واقع پنجشنبه حوالی ساعت 9 صبح تا 11 صبح را در برمیگیرد.محدوده دوم 131امین ساعت از هفته میباشد که حدود ساعت 11 صبح جمعه خواهد بود.که باتوجه به اخر هفته بودن،افزایش چشکگیر این میزان تردد در دو روز منطقی بنظر میرسد.احتمالا این منطقه جایی است که مردم برای گذراندن اخر هفته خود به انجا میروند.

یا بطور مثال میتوان اینگونه تعبیر کرد که این دو دوربین از لحاظ موقعیت جغرافیای در دوسر یک خیابان قرار دارند.

دلیل اینکه دقیقا میزان ترافیک ثبت شده توسط این دو دوربین یکسان نیست ،میتواند به دلیل کوچه ها و مسیرهای فرعی این خیابان باشد.یعنی دقیقا همان تعداد خودرو که از یک سر خیابان وارد میشود در همان ساعت همان تعداد خارج نمیشود.البته میتواند این موضوع به دلیل پارک کردن خودروها در ان خیابان هم باشد.

تعبیرهای مختلفی میتوان داشت که برای بررسی صحت آنها باید فاکتور های دیگر هم مانند دقیقه و حتی ثانیه را در زمان ثبت خودرو،در نظر گرفت.

- پیاده سازی CF روی خودروها

در قسمت قبل روش CF را بر روی دوربین ها پیاده سازی کردیم و ازاین طریق دوربین های مشابه را پیدا کردیم.

در این قسمت همان کار را این بار بر روی خودروها انجام میدهیم و نتایج انها را تحلیل میکنیم.

مانند قسمت قبل یک ماتریس صفر(matrix) میسازیم.ابعاد این ماتریس برابر با تعداد خودروها(مسلما منحصربه فرد) و تعداد دوربین ها میباشد(path).در قطعه کد زیر نیز این موضوع مشخص میباشد.

```
path = device
list_temp=df_2.select('FINAL_CAR_KEY').distinct().collect()
cars =  [i.FINAL_CAR_KEY for i in list_temp]
matrix = np.zeros((len(cars),len(path)))
```

حال مسیر هر خودرو(لیست دوربین هایی که از آنها گذر کرده) را بدست میاوریم و در دیتافریم زیر ذخیر میکنیم.

```
+-------------+-------------------------+
|FINAL_CAR_KEY|collect_list(DEVICE_CODE)|
+-------------+-------------------------+
|         null|     [631830, 900247, ...|
|     10000048|                 [900183]|
|     10000061|         [900256, 900256]|
|     10000126|               [22010087]|
|    100001297|                 [900269]|
+-------------+-------------------------+
```

پس از تشکیل دیتافریم،مانند کاری که برای دوربین ها کردیم،خودروها را ایندکس گذاری میکنیم وسپس به سراغ پرکردن درایه های ماتریس(matrix) میرویم.

هر سطر بیانگر یک خودرو است که در دیکشنری مربوطه ایندکس انها ذخیره شده است.هر درایه از سطر matrix ، یک یا صفر است. یک به معنای وجود آن دوربین مشخص در مسیر خودرو و صفر به معنای عدم وجود میباشد.

قطعه کد مربوط به ایندکس گذاری خودروها:

```
#indexing the cars
car_dict = {}
for i, carkey in enumerate(cars):
  car_dict[i] = carkey
car_dict = {j: i for i, j in car_dict.items()}
```

قطعه کد مربوط به پرکرن درایه ها با یک(وجود دوربین مذکور در مسیر خودرو):

```
for j,m in df_4_list:
  for i in m:
    idx = device.index(i)
    matrix[car_dict[j]][idx] = 1
matrix
```

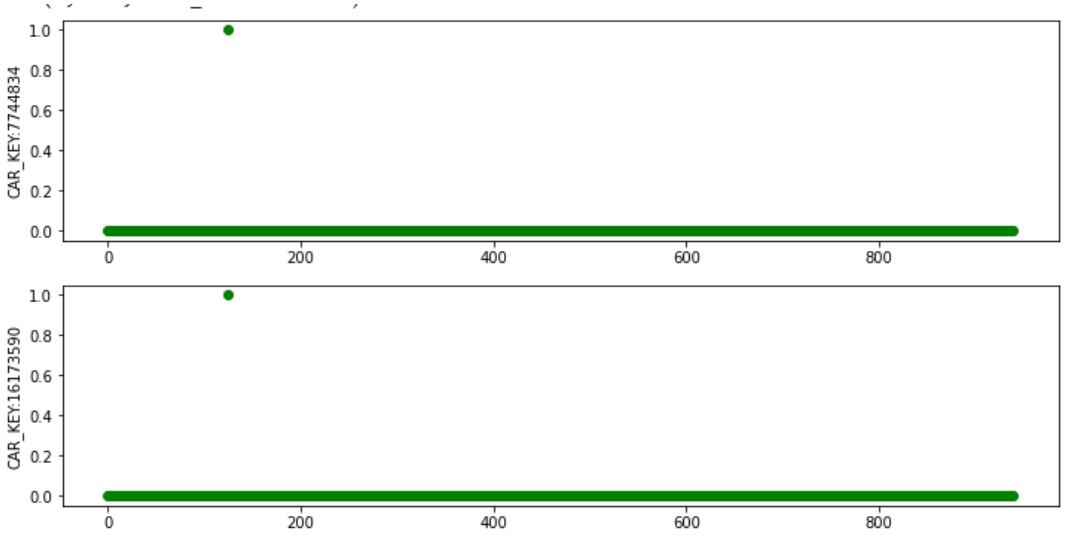
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

مانند قسمت قبل با محاسبه هم بستگی ها با استفاده از دستور()corr،میزان شباهت میان خودروها را میسنجیم و در ادامه ، خودروها با بیشترین شباهت را گزارش میکنیم.

در اینجا تعدادی از خودرو ها که بیشترین شباهت را دارند در دیتافریم نمایش داده ایم.

	Car 1	Car 2
0	7771041	8460449
1	12155308	77950331
2	14662708	85894067
3	9243443	8858359
4	9818356	22431192
...	...	...
95	12183087	75111529
96	10007867	8543672
97	9476609	32964106

حال همان نموداری که برای دوربین های مشابه رسم کردیم، برای دو خودروی مشابه که به دلخواه انتخاب شده اند رسم میکنیم.مشاهده می شود که این دو خودرو توسط دوربین یکسانی رویت شده اند که یعنی مسیر مشابهی را طی کرده اند.



- یافتن خودرو های مشابه با استفاده از فاصله کسینوسی

در قسمت قبل خودروهای مشابه را با استفاده از روش CF پیدا و مشاهده کردیم.حال در این قسمت با استفاده از مفهوم دیگری به اسم فاصله کسینوسی ،بردار مسیر خودروها و زاویه آن ها با یک مسیر مشخص در نظر میگیریم و سپس خودروهای مشابه را پیدا میکنیم.

بردار مسیر خودرو،در واقع بردار شامل تمام دوربین هایی است که آن خودرو در یک روز مشخص از انها رد شده و توسط ان دوربین ها ثبت شده است.(مولفه ی value = [Device Code List])

خودروهای مشابه،خودروهایی هستند که بردارهای مسیر شبیه بهم دارند یعنی از مسیرهای تقریبا یکسانی در یک روز مشخص گذر کرده اند.

دیتافریم تشکیل شده بدین صورت میباشد که پنج سطرابتدایی آن را نشان داده ایم.

```
+-----------+-------------+-------------------+----------+
|DEVICE_CODE|FINAL_CAR_KEY|      PASS_DAY_TIME|      date|
+-----------+-------------+-------------------+----------+
|   22010059|     63315887|2021-12-22 00:51:16|2021-12-22|
|  100701235|     47807908|2021-12-22 01:07:42|2021-12-22|
|     635607|     81726547|2021-12-22 00:05:34|2021-12-22|
|   10015301|     36037995|2021-12-22 00:51:14|2021-12-22|
|   22010044|     64111706|2021-12-22 01:30:33|2021-12-22|
+-----------+-------------+-------------------+----------+
only showing top 5 rows
```

حال با استفاده از دستور map زوج های مورد نظر خود را به صورت rdd میسازیم.

```
df_a_rdd=df_a.rdd.map(lambda x: ((x[1],str(x[3])),x[0]))  # (key=(plate,date),value=DEVICE_CODE)
```

این rdd به صورت ((key=(plate ,date),value =Device_ Code)) است.چند نمونه اول بدین شکل است:

```
[(('63315887', '2021-12-22'), '22010059'),
 (('47807908', '2021-12-22'), '100701235'),
 (('81726547', '2021-12-22'), '635607'),
 (('36037995', '2021-12-22'), '10015301'),
 (('64111706', '2021-12-22'), '22010044')]
```

حال با استفاده از دستور groupbykey() مسیر خوردروها را با مفهومی که در ابتدا بیان شد،بدست میاوریم.

زوج های بدست امده به صورت `(key=(plate,date),value = [Device Code List])` هستند.در اینجا چند نمونه نمایش داده شده است.

```
[(('84772646', '2021-12-22'), ['100700820']),
 (('84755468', '2021-12-22'), ['100701096', '22010095']),
 (('83441842', '2021-12-22'), ['900241']),
 (('7657081', '2021-12-22'), ['900149']),
 (('29484171', '2021-12-22'), ['202601'])]
```

یک مسیر فرضی انتخاب میکنیم.بدین صورت که چند دوربین را به صورت رندوم انتخاب کرده و آن را بردار مسیر فرضی در نطر میگیریم.اینکار با استفاده از کد زیر انجام میشود.در اینجا طول مسیر را به دلخواه برابر با چهار در نظرگرفتیم.

```
ind = np.random.randint(0,len(uniq_cam),4) #creating random index for choosing random camera
hypo_path = np.array(uniq_cam)[(ind).astype(int)] #creating a hypothetical path
```

حال کسینوس زاویه میان دو بردار مسیر فرضی و مسیر خودروها را محاسبه میکنیم.

هر چه مقدار کسینوس به یک نزدیک تر باشد به معنای زاویه کمتر میان ان دو بردار و شباهت بیشتر آنهاست.مقادیر به دست امده در rdd_cosine_similarity ذخیره شده اند.که در ادامه به صورت مرتب شده براساس میزان شباهت و بطور نزولی نمایش میدهیم.

همانطور که گفته شد، در اینجا یک مسیر فرضی شامل 4 دوربین را در نظر گرفتیم.مقادیر به دست آمده متناظربا مسیر فرضی ،به صورت زیر است:

```
[(0.5, (('94177400', '2021-12-23'), ['900155'])),
 (0.5, (('92618743', '2021-12-22'), ['900155'])),
 (0.5, (('88011827', '2021-12-24'), ['900155'])),
 (0.5, (('8540129', '2021-12-24'), ['900155'])),
 (0.5, (('63349828', '2021-12-23'), ['900155'])),

[(0.354, (('7889187', '2021-12-24'), ['900155', '22010031'])),
 (0.354, (('7674021', '2021-12-25'), ['900155', '631359'])),
 (0.354, (('29252273', '2021-12-25'), ['900203', '900155'])),
 (0.354, (('65084017', '2021-12-28'), ['900222', '900155'])),
 (0.354, (('35817374', '2022-01-04'), ['900155', '631829']))]
```

مولفه های هر زوج در تصویر بالا بدین صورت میباشد:

```
(key =cos(theta), value = ((plate,date),[Device Code List]))
```

مقدار اول در زوج بالا،میزان شباهت مسیر خودرو به مسیر فرضی است.مقدار دوم شامل روز،پلاک خودرو و مسیر طی شده توسط خودرو در آن روز میباشد.

نتایج بالا به ازای مسیر فرضی زیر به دست امده است.

```
hypo_path
```

```
array([['900155'],
       ['212001'],
       ['22010073'],
       ['22009915']], dtype='<U9')
```

- روش Pixie برای شباهت خودروها و دوربین ها

در ابتدا بایستی ارتباطات میان هرخودرو و دوربین را بدست بیاوریم.به این معنی که باید مشخص کنیم هر خودرو چندبار از یک دوربین رد شده است یا اگر از طرف دیگر نگاه کنیم،آن دوربین چندبار خودرو مورد نظر را رویت کرده است.

برای اینکار ابتدا یک rdd شامل زوج های میسازیم که مولفه اول هر زوج، دوربین و خودرو و مولفه دوم عدد یک است.سپس با استفاده از دستور reduceByKey به rdd نهایی که تعداد دفعات عبور خودرو از یک دوربین مشخص را نشان میدهد،میرسیم.

قطعه کد این دومرحله بدین صورت است:

```
#creat  KV_car_camera : (key=(FINAL_CAR_KEY,DEVICE_CODE),1)
KV_car_camera = df.rdd.map(lambda x: ((x[1],x[0]),1))
reduced_car_camera=KV_car_camera.reduceByKey(lambda x,y: x+y)
```

حال با استفاده از rdd ساخته شده (reduced_car_camera) ،گراف دو بخشی را میسازیم.

باید نشان دهیم هر خودرو از چه دوربین هایی گذر کرده و تعداد دفعات گذر از هر دوربین چقدر بوده است. و همچنین باید نشان دهیم هر دوربین چه خودروهایی را دیده و تعداد دفعات رویت یک خودرو مشخص چقدر میباشد.

با استفاده از map زوج های مورد نظر خود را میسازیم و سپس دولیست میسازیم.یکی نشان دهنده اتصالات خودرو به دوربین ها و دیگری بالعکس(اتصالات دوربین به خودروها).

قطعه کد زیر گراف دو بخشی ما را میسازد:

```
camera_car_rdd=reduced_car_camera.map(lambda x:(x[0][1],(x[0][0],x[1]))).groupByKey()
camera_car_list=camera_car_rdd.mapValues(dict).collect()

car_camera_rdd=reduced_car_camera.map(lambda x:(x[0][0],(x[0][1],x[1]))).groupByKey()
car_camera_list=car_camera_rdd.mapValues(dict).collect()
```

حال به سراغ پیاده سازی pixie میرویم.

روند کلی در الگوریتم pixie بدین صورت است که (مثلا میخواهیم دوربین های مشابه را پیدا کنیم) یک دوربین را در نظر میگیریم(دراینجا به صورت رندوم آن دوربین انتخاب شده است.)،از آن دوربین با احتمال متناسب با وزن یال های متصل به آن دوربین،به یک خودرو میرویم.حال از این خودرو و باز هم با احتمالی متناسب با وزن یال های متصل به آن،به یک دوربین میرویم.با یک احتمال(در ورودی تابع a میباشد)به همان جای اولیه برمیگردیم.همچنین یک ترشولد در نظر میگیریم،اگر دفعات رویت شدن دوربین از آن مقدار بیشتر باشد،آن دوربین را مشابه با همان دوربین اول در نظر میگیریم.

تابع پیاده سازی الگوریتم بدین صورت میباشد:

```
#a:probability , lim:threshold ,iter:number of steps
def Pixie_sim(pin,board,query,a,lim,iter):
  pin_node=query
  visit_pin_s={}
  for i in range(0,iter):
      s=pin[pin_node]
      r=board[random.choices(list(s.keys()),list(s.values()))[0]]
      pin_node=random.choices(list(r.keys()),list(r.values()))[0]
      if pin_node in visit_pin_s:
        visit_pin_s[pin_node]=visit_pin_s[pin_node]+1
      else:
        visit_pin_s[pin_node]=1
      if random.random()<a:
        pin_node=query
  sim_item={}
  for pin_node in visit_pin_s:
    if visit_pin_s[pin_node]>lim:
      sim_item[pin_node]=visit_pin_s[pin_node]
  return sim_item
```

برای یک خودرو خاص ،احتمال 0.55،ترشولد 20 و تعداد گام 6000 به خروجی زیر میرسیم:

```
#similarity on cars
car_camera_dict=dict(car_camera_list)
camera_car_dict=dict(camera_car_list)
cars=list(car_camera_dict.keys())
sim_cars=Pixie_sim(car_camera_dict,camera_car_dict,random.choices(cars)[0],0.55,20,6000)
print(str(random.choices(cars)[0]))
sim_cars
```

```
31112426
{'20332201': 212,
 '88682899': 191,
 '64111706': 1998,
 '95686695': 201,
 '8589324': 209,
 '16993854': 205,
 '16694581': 225,
 '12521319': 193,
 '14004295': 194,
 '31112426': 203,
 '9833773': 191,
 '17268428': 212,
 '82807364': 192,
 '14067234': 196,
 '101329458': 197}
```

خودروهای مشابه:

```
sim_cars
```

```
31112426
{'20332201': 212,
 '88682899': 191,
 '64111706': 1998,
 '95686695': 201,
 '8589324': 209,
 '16993854': 205,
 '16694581': 225,
 '12521319': 193,
 '14004295': 194,
 '31112426': 203,
 '9833773': 191,
 '17268428': 212,
 '82807364': 192,
 '14067234': 196,
 '101329458': 197}
```

در این قسمت یک دوربین را در نظر میگیریم و دوربین های مشابه را بدست میاوریم.خروجی کد به ازای احتمال 0.55،ترشولد 80 و تعداد گام 6000 بدین صورت میباشد:

```
#similarity on cameras
cameras=list(camera_car_dict.keys())
sim_cam=Pixie_sim(camera_car_dict,car_camera_dict,random.choices(cameras)[0],0.55,80,6000)
print(str(random.choices(cameras)[0]))
sim_cam
```

```
211701
{'211701': 2116, '22010054': 95, '22010040': 110, '114': 84, '22010053': 142}
```

دوربین های مشابه:

```
sim_cam
```

```
211701
{'211701': 2116, '22010054': 95, '22010040': 110, '114': 84, '22010053': 142}
```

- تشکیل ماتریس Utility و تجزیه به مقادیر تکین(SVD)

در این قسمت تجزیه SVD ماتریس Utility را بدست میاوریم.با استفاده از کتابخانه scipy و قسمت جبرخطی آن،تجزیه را محاسبه میکنیم.

```
from scipy.linalg import svd as scipy_svd
matrix_df_normalise = matrix_df - np.asarray([(np.mean(matrix_df, 1))]).T
Utility = matrix_df_normalise.T / np.sqrt(matrix_df.shape[0] - 1)
U, S, V = scipy_svd(Utility)
pd.DataFrame(np.diag(S))
```

قسمتی از ماتریس قطری که مقادیر تکین بر روی قطر اصلی آن قرار دارند بدین شکل میباشد.

	0	1	2	3	4	5	6	7	8	9	...	930	931	932	933	934
0	0.532425	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
1	0.000000	0.395324	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
2	0.000000	0.000000	0.337711	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.000000	0.000000	0.000000	0.318462	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
4	0.000000	0.000000	0.000000	0.000000	0.318165	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
935	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
936	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
937	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
938	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

- خوشه بندی دوربین ها(روش K-mean)

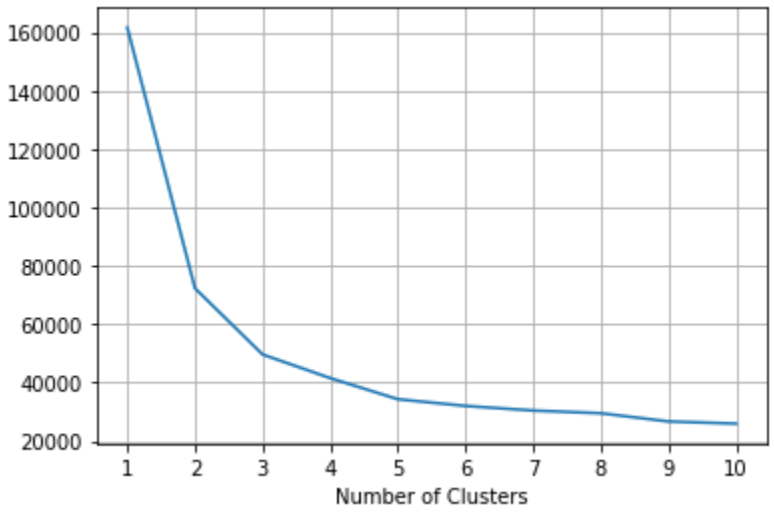
در این قسمت با استفاده از بردارهای ساخته شده برای هر دوربین (بردار24*7 که در ابتدا ساختیم)،انها را با روش k-means خوشه بندی میکنیم.

همانطور که در شکل زیر مشخص است، خوشه بندی را با استفاده از کتابخانه sklearn انجام میدهیم.قبل از train کردن ابتدا دیتا را استاندارد میکنیم.بدین صورت که میانگین را صفر و واریانس را یک میکنیم.

در ابتدا باید تعداد کلاسترها را تعیین کنیم.که در ادامه با توجه به میزان خطا اینکار انجام میشود.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
feature_matrix = user_item_matrix
standard_scaler= StandardScaler()
scaled_feature_matrix = standard_scaler.fit_transform(feature_matrix)
e = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(scaled_feature_matrix)
    e.append(kmeans.inertia_)
```

با توجه به نمودار خطا برحسب تعداد کلاسترها،یک کلاستر ده تایی مناسب بنظر میرسد.



با توجه به میزان خطا تعداد کلاسترها را مشخص کردیم.حال مشخص میکنیم هر دوربین متعلق به کدام کلاستر میباشد.

```
labels = kmeans.labels_.tolist()
np.array(labels)
```

```
np.array(labels)
```

```
array([6, 1, 1, 1, 5, 6, 6, 1, 6, 1, 1, 6, 1, 7, 2, 6, 5, 7, 1, 4, 1, 1,
       1, 1, 6, 6, 7, 4, 6, 3, 1, 4, 1, 4, 1, 1, 1, 4, 1, 1, 9, 1, 1, 6,
       5, 1, 1, 5, 6, 3, 4, 9, 1, 8, 4, 1, 1, 4, 7, 1, 6, 1, 6, 1, 7, 1,
       1, 1, 1, 1, 1, 1, 1, 5, 6, 1, 1, 4, 1, 1, 1, 4, 1, 5, 7, 1, 6, 4,
       6, 1, 7, 6, 1, 1, 6, 6, 5, 4, 1, 6, 1, 2, 7, 1, 6, 1, 1, 1, 5, 1,
       1, 1, 5, 4, 6, 1, 8, 3, 5, 1, 1, 6, 6, 1, 7, 1, 1, 4, 4, 1, 7, 1,
       8, 7, 5, 1, 1, 1, 1, 1, 1, 1, 6, 1, 1, 7, 4, 6, 4, 4, 7, 5, 1, 7,
       6, 1, 1, 1, 4, 2, 4, 1, 1, 4, 4, 3, 1, 1, 4, 7, 6, 6, 1, 1, 6, 1,
       7, 7, 7, 1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 6, 7, 3, 1, 1, 6, 1, 6, 1,
       6, 8, 6, 1, 1, 1, 8, 7, 1, 1, 1, 4, 4, 1, 6, 4, 1, 1, 1, 3, 8, 6,
       1, 1, 1, 1, 5, 6, 7, 8, 1, 1, 1, 1, 1, 1, 6, 1, 1, 0, 1, 5, 1, 1,
       1, 4, 4, 7, 6, 1, 6, 6, 4, 1, 1, 8, 4, 4, 7, 1, 9, 1, 1, 1, 6, 1,
       1, 4, 6, 4, 4, 1, 6, 4, 1, 1, 2, 6, 1, 4, 7, 1, 6, 6, 6, 1, 1, 1,
       1, 5, 4, 1, 4, 6, 1, 8, 1, 1, 1, 1, 1, 1, 1, 1, 1, 6, 4, 6, 1, 6,
       6, 1, 6, 4, 1, 7, 9, 1, 3, 1, 1, 6, 6, 6, 5, 1, 6, 1, 7, 9, 0, 1,
       1, 4, 8, 1, 1, 6, 1, 9, 1, 1, 6, 7, 1, 5, 6, 1, 1, 1, 5, 1, 5, 7,
       4, 4, 1, 4, 7, 1, 1, 1, 1, 6, 8, 4, 1, 6, 6, 1, 1, 2, 1, 1, 6, 5,
       6, 4, 4, 5, 2, 4, 6, 1, 1, 3, 1, 6, 7, 6, 8, 4, 1, 1, 6, 6, 1, 1,
       1, 1, 1, 8, 1, 6, 1, 1, 1, 1, 1, 8, 5, 8, 8, 4, 4, 1, 6, 1, 6, 1,
       7, 6, 7, 1, 6, 1, 1, 6, 8, 6, 1, 1, 1, 6, 7, 4, 7, 6, 1, 1, 1, 5,
       1, 1, 6, 4, 8, 4, 7, 1, 1, 5, 7, 4, 1, 1, 5, 1, 1, 1, 7, 4, 1, 1,
       1, 6, 6, 4, 7, 1, 1, 1, 7, 6, 9, 8, 7, 1, 3, 8, 7, 7, 6, 1, 3, 7,
       1, 9, 1, 6, 5, 4, 6, 5, 6, 6, 1, 6, 1, 5, 1, 6, 1, 6, 1, 1, 5, 1,
       1, 4, 6, 4, 5, 4, 1, 7, 4, 1, 6, 9, 7, 7, 4, 1, 6, 8, 6, 4, 1, 4,
```

باتوجه به لیبل های بدست امده و دیکشنری مربوط به دوربین ها که برای ایندکس گذاری ساخته بودیم،مشخص میکنیم هر دوربین با کد مشخص،مربوط به کدام یک از ده کلاستر میباشد.

برای مثال تعدادی از دوربین هایی که در کلاستر چهارم قرار گرفته اند بصورت زیر میباشند.

	Label 4
0	232
1	155
2	200902
3	209103
4	631346
...	...
67	100701269
68	100701262

اگر موقعیت های مکانی دوربین ها را داشتیم میتوانستیم بررسی کنیم دوربین هایی که در یک منطقه هستند در یک کلاستر قرار میگرند یا خیر.

این طور به نظر میرسد که دوربین های یک منطقه احتمالا ترافیک یکسانی را مشاهده و ثبت میکنند پس بردار هفتگی_ساعتی(همان بردار 7*24 که در ابتدا ساختیم) انها شبیه خواهد بود و در نهایت در یک کلاستر قرار میگیرند.

برای مثال در اینجا دوربین های کلاستر چهارم نمایش داده شده است که احتمالا از لحاظ جغرافیایی نیز هم خوشه بودن انها توجیه خواهد شد.

- یافتن مسیرهای پرتردد

در این قسمت میخواهیم با استفاده از الگوریتم A-priori مسیرهای پرتردد را پیدا کنیم.

ابتدا یک rdd به صورت زیر درست میکنیم:

key= (plate, date), value= [list of device codes]

قطعه کد زیر rdd مورد نظر را میسازد.چند نمونه از آن در ادامه نشان داده شده است.

```
#add column 'date' to dataframe
df_a=df.withColumn("date",to_date("PASS_DAY_TIME"))
df_a.show(3)
```

```
+-----------+-------------+-------------------+----------+
|DEVICE_CODE|FINAL_CAR_KEY|      PASS_DAY_TIME|      date|
+-----------+-------------+-------------------+----------+
|   22010059|     63315887|2021-12-22 00:51:16|2021-12-22|
|  100701235|     47807908|2021-12-22 01:07:42|2021-12-22|
|     635607|     81726547|2021-12-22 00:05:34|2021-12-22|
+-----------+-------------+-------------------+----------+
only showing top 3 rows
```

```
#creat rdd (key=(plate,date),value=DEVICE_CODE)
df_a_rdd=df_a.rdd.map(lambda x: ((x[1],str(x[3])),x[0]))
```

```
#groupby the rdd and creat the new rdd with new format (key=(plate,date),value=[list of device codes])
rdd_a_1=df_a_rdd.groupByKey().mapValues(set).mapValues(list)
```

```
rdd_a_1.take(5)
```

```
[(('84772646', '2021-12-22'), ['100700820']),
 (('84755468', '2021-12-22'), ['100701096', '22010095']),
 (('83441842', '2021-12-22'), ['900241']),
 (('7657081', '2021-12-22'), ['900149']),
 (('29484171', '2021-12-22'), ['202601'])]
```

با توجه به خروجی کد،مثلا خودرو با شماره پلاک 84755468 در تاریخ 12/22 از دو دوربین با کدهای 100701096 و 22010095 عبور کرده است.

با استفاده از rdd ساخته شده به سراغ پیاده سازی الگوریتم A-priori میرویم.

تعبیر مدل item-basket برای خودروها و دوربین ها:

در اینجا هر خودرو مانند یک خریدار میباشد که در سبد خرید خود تعدادی کالا خریداری کرده است.**کالاها همان دوربین ها هستند** که هر خودرو از انها عبور کرده است.با توجه به rdd ساخته شده،**سبد خرید** همان [list of device codes] میباشد.

مسیرهای پرتردد بدین گونه تعریف میشود:مجموعه های یک یا چندعضوی از دوربین ها که خودروهای زیادی از انها عبور کرده اند.

حال باتوجه به این مفهوم، الگوریتم A-priori را بر روی basket ها پیاده میکنیم.

ابتدا لیست سبدها را از rdd ساخته شده استخراج میکنیم.قطعه کد زیر اینکار را انجام میدهد.

```
#creat baskets of cameras
rdd_of_baskets=rdd_a_1.map(lambda x:x[1])
```

چند نمونه از سبدها نمایش داده شده است:

```
rdd_of_baskets.take(5)
```

```
[['100700820'], ['100701096', '22010095'], ['900241'], ['900149'], ['202601']]
```

یک rdd از ایتم ها میسازیم.حال برای هرایتم(دوربین) یک زوج (key=item ,value=1) میسازیم و تعداد هر ایتم را میشماریم.

```
#creat rdd of items
rdd_of_items = rdd_of_baskets.flatMap(lambda x:x)

#creat (key=item, value=1) for each item
KV_of_items=rdd_of_items.map(lambda x :(x,1))

#count each item with reduceByKey (key = item, value =number of the item)
number_of_item = KV_of_items.reduceByKey(lambda x, y: x + y)
```

تعدادی از ایتم ها همراه با ساپورت بدین صورت میباشد(ایتم ها براساس ساپورت به صورت نزولی مرتب شده اند):

```
#sort the items by their support
number_of_item_sorted_list=sorted(number_of_item.collect(),key=lambda x: x[1],reverse=True)
number_of_item_sorted_list
```

```
[('900212', 258396),
 ('900244', 250597),
 ('100700853', 214275),
 ('900269', 207356),
 ('900222', 182983),
 ('631634', 173842),
 ('900101', 163240),
 ('900142', 151506),
 ('100700841', 143324),
 ('631633', 142285),
 ('900268', 139104),
 ('900225', 138451),
 ('900155', 135421),
 ('900246', 132204),
 ('900236', 131893),
 ('900164', 122542),
 ('900191', 119686),
 ('100700804', 110369),
 ('631357', 109736),
 ('900234', 108221),
 ('100700864', 100147),
```

برای مثال ایتم 900222 در 182983 سبد موجود بوده است.

با توجه به اینکه میانگین ساپورت ها حدود 12هزار میباشد،ایتم هایی که 3 برابر میانگین خریداری شده اند احتمالا کاندید های خوبی باشند.پس ساپورت ترشولد را 35 هزار در نظر میگریم.

محاسبه میانگین ساپورت ها:

```
support=number_of_item.map(lambda x:x[1])
```

```
support.sum()/support.count()
```

```
12469.257900101937
```

ساپورت ترشولد را برابر با35000 در نظر میگیریم و ایتم های پرتکرار با بدست میاوریم.کد مربوطه بدین صورت میباشد:

```
#find frequant items
support_threshold= 35000
frequent_items=number_of_item.filter(lambda x :x[1]>=support_threshold)
```

ایتم های پرتکرار بدین صورت میباشند که به صورت نزولی براساس ساپورت مرتب شده اند:

```
#sort the frequant items by their support
frequent_items_sorted_list=sorted(frequent_items.collect(),key=lambda x: x[1],reverse=True)
frequent_items_sorted_list
```

```
[('900212', 258396),
 ('900244', 250597),
 ('100700853', 214275),
 ('900269', 207356),
 ('900222', 182983),
 ('631634', 173842),
 ('900101', 163240),
 ('900142', 151506),
 ('100700841', 143324),
 ('631633', 142285),
 ('900268', 139104),
 ('900225', 138451),
 ('900155', 135421),
 ('900246', 132204),
 ('900236', 131893),
 ('900164', 122542),
 ('900191', 119686),
 ('100700804', 110369),
 ('631357', 109736),
 ('900234', 108221),
```

پس از بدست اوردن ایتم های پرتکرار حال به سراغ محاسبه دوتایی های پرتکرار میرویم.

بدین صورت که زیرمجموعه های دوتایی را از ایتم های پرتکرار میسازیم(candidate pairs).ساختن زیر مجموعه ها با تابع findsubsets انجام میدهیم. یک مجموعه را میگیرد و زیر مجموعه های n عضوی را به ما میدهد.

```
#this function creat subsets of given set
def findsubsets(sett, n):
    return list(itertools.combinations(sett, n))
```

برای محاسبه ساپورت دوتایی های بدست امده،تعداد سبدهایی که دوتایی مورد نظر را دارد را محاسبه میکنیم.در واقع بایستی آن دوتایی مورد نظر زیر مجموعه آن سبد باشد.چک کردن زیر مجموعه و محاسبه ساپورت ایتم ست ها را با تابع support_calc انجام میدهیم.

```
#this function calculate support of given itemset
def support_calc(itemset):
    support=0
    for i in range(basket_numbers):
        if set(itemset).issubset(set(baskets_list[i])):
            support=support+1

    return support
```

با توجه به توابع بیان شده،ساپورت دوتایی ها را بدست میاوریم و سپس با توجه به ترشولد انتخابی،دوتایی های پرتکرار را پیدا میکنیم.

تعدادی از دوتایی های کاندید شده بدین صورت میباشند:

```
#find candidate pairs
candidate_pairs = findsubsets(frequent_items_whithout_support,2)
```

```
candidate_pairs
```

```
[('900212', '900244'),
 ('900212', '100700853'),
 ('900212', '900269'),
 ('900212', '900222'),
 ('900212', '631634'),
 ('900212', '900101'),
 ('900212', '900142'),
 ('900212', '100700841'),
 ('900212', '631633'),
 ('900212', '900268'),
 ('900212', '900225'),
 ('900212', '900155'),
 ('900212', '900246'),
 ('900212', '900236'),
 ('900212', '900164'),
 ('900212', '900191'),
 ('900212', '100700804'),
 ('900212', '631357'),
 ('900212', '900234'),
 ('900212', '100700864'),
 ('900212', '900218')
```

حال با استفاده از تابع support_calc، ساپورت کاندیداهای دوتایی را بدست میاوریم.

```
#calculate support of candidate pairs
candidate_pairs_KV=[]
for i in range(len(candidate_pairs)):
    supp=support_calc(candidate_pairs[i])
    candidate_pairs_KV.append((candidate_pairs[i],supp))
```

تعدادی از کاندیدهای دوتایی به همراه ساپورت بدین صورت میباشد:

```
candidate_pairs_KV
```

```
(('900244', '100'), 455),
(('900244', '22010079'), 161),
(('900244', '900258'), 567),
(('900244', '900199'), 496),
(('900244', '117'), 114),
(('900244', '900228'), 290),
(('900244', '900151'), 240),
(('900244', '100701119'), 68),
(('900244', '103001'), 146),
(('100700853', '900269'), 1894),
(('100700853', '900222'), 1315),
(('100700853', '631634'), 3363),
(('100700853', '900101'), 2225),
(('100700853', '900142'), 8251),
(('100700853', '100700841'), 980),
(('100700853', '631633'), 1830),
(('100700853', '900268'), 2729),
(('100700853', '900225'), 1538),
(('100700853', '900155'), 843),
(('100700853', '900246'), 792),
(('100700853', '900236'), 1696),
(('100700853', '900164'), 361),
(('100700853', '900191'), 550),
```

حال یک rdd شامل کاندید های دوتایی میسازیم بدین صورت:

```
#creat rdd of candidate pairs (key = candidate pairs, value = support )
candidate_pairs_rdd=sc.parallelize(candidate_pairs_KV)
```

برای انتخاب ترشولد مناسب ،میانگین ساپورت های کاندیدهای دوتایی را بدست میاوریم:

```
#calculate the average support
candidate_pairs_supports = candidate_pairs_rdd.map(lambda x :x[1])
candidate_pairs_supports.sum() / candidate_pairs_supports.count()
```

```
489.2156862745098
```

میانگین حدود 490 میباشد.پس ترشولد را چیزی حدود 1500 قرار میدهیم.

بدست اوردن دوتایی های پرتکرار:

```
frequent_pairs = candidate_pairs_rdd.filter(lambda x:x[1]>support_threshold)
```

در نهایت دوتایی های پرتکرار بدین صورت میباشند(براساس ساپورت نزولی مرتب شده اند) :

```
#sort the candidate pairs by their support
frequent_pairs_sorted_list=sorted(frequent_pairs.collect(),key=lambda x: x[1],reverse=True)
frequent_pairs_sorted_list
```

تعدادی از دوتایی های پرتکرار:

```
frequent_pairs_sorted_list[0:15]
```

```
[(('900212', '900244'), 19198),
 (('900212', '900142'), 9430),
 (('100700853', '900142'), 8251),
 (('900244', '900142'), 6938),
 (('900269', '900225'), 6908),
 (('900244', '631633'), 6838),
 (('900212', '100700853'), 6707),
 (('900212', '631633'), 6006),
 (('900244', '100700853'), 5488),
 (('100700864', '900185'), 5335),
 (('631634', '900207'), 5085),
 (('900268', '900225'), 5042),
 (('900222', '900155'), 4734),
 (('900101', '100700841'), 4558),
 (('900244', '631634'), 4446)]
```

به عنوان نمونه زوج ('900212', '900244') شامل دو دوربین است که 19198خودرو از از مسیر مربوط به این دو دوربین عبور کرده اند،در واقع محدوده جغرافیایی این دو دوربین پرترافیک است و اگر لوکیشن دوربین ها را داشتیم میتوانستیم خیابان های پرتردد شهر تهران را ازین طریق بیابیم.

حال باتوجه به دوتایی های پرتکرار به دست امده،سه تایی های کاندید را بدست میاوریم و تمام مراحلی که برای به دست اوردن دوتایی های پرتکرار انجام داده بودیم را،برای سه تایی های کاندید نیز انجام میدهیم و در نهایت،سه تایی های پرتکرار را به دست میاوریم.

**** قبل از بدست اوردن سه تایی های پرتکرار، کاندیدها را فیلتر میکنیم،هر کاندید،سه عضو دارد،بنابراین سه زیر مجموعه دوعضوی دارد.هر کاندید در صورتی میتواند پرتکرار باشد، که هر سه زیرمجموعه دو عضوی ان پرتکرار باشد.(دوتایی های پرتکرار در قسمت قبل بدست امد). پس از اعمال این شرط ،تعداد کاندیدهای سه تایی به شدت کاهش میابد. ****

تکه کدهای مربوط به پیداکردن سه تایی های پرتکرار بدین صورت میباشد:

در این کد،ایتم هایی که در ساخت زوج های پرتکرار نقش داشته اند را جدا میکنیم.

```
#creat list of items that each item is in a frequant pair(at least)
frequent_pairs_splited=[]
for  i  in range(len(frequent_pairs_sorted_list)):
      frequent_pairs_splited.append(frequent_pairs_sorted_list[i][0][0])
      frequent_pairs_splited.append(frequent_pairs_sorted_list[i][0][1])
```

در اینجا ایتم های انتخاب شده در کد قبلی را داخل یک set میریزیم تا ایتم های منحصر به فرد به دست بیاید.سپس با استفاده ازین ایتم ها،زیر مجموعه های سه تایی درست میکنیم و کاندید های سه تایی را میسازیم.

```
#we will creat triplex subsets from member of this list
unique_frequent_pairs_splited= list(set(frequent_pairs_splited))
```

```
#creat candidate triples with the member of frequent pairs
candidate_triples = findsubsets(unique_frequent_pairs_splited,3)
```

تعدادی از کاندیدهای سه تایی بدین شکل میباشد:

```
candidate_triples
```

```
[('900155', '900236', '900227'),
 ('900155', '900236', '900222'),
 ('900155', '900236', '900255'),
 ('900155', '900236', '22009977'),
 ('900155', '900236', '900215'),
 ('900155', '900236', '900142'),
 ('900155', '900236', '231'),
 ('900155', '900236', '203902'),
 ('900155', '900236', '900233'),
 ('900155', '900236', '100700804'),
 ('900155', '900236', '631633'),
 ('900155', '900236', '900226'),
 ('900155', '900236', '100700841'),
 ('900155', '900236', '900273'),
 ('900155', '900236', '900266'),
 ('900155', '900236', '631634'),
 ('900155', '900236', '22009971'),
 ('900155', '900236', '206602'),
 ('900155', '900236', '900101'),
```

در کد زیر شرطی که در قسمت بالا به رنگ قرمز گفته شده بود چک میشود و کاندیدهای نهایی سه تایی بدست میاید.(اگر یک سه تایی بخواهد پرتکرار باشد،باید همه زیرمجموعه ای دوتایی اش،سه زیر محموعه دوتایی دارد،پرتکرار باشند.)

```
#all binary subsets of a frequant triple should be frequant
#so we check this condition here and find final candidates
final_candidate_triples=[]
for i in tqdm(range(len(candidate_triples))):
    subs_of_candidate=findsubsets(candidate_triples[i],2)
    flag=0
    for j in range(len(frequent_pairs_whithout_support)):
        if subs_of_candidate[0]==frequent_pairs_whithout_support[j]:
            flag=flag+1
        if subs_of_candidate[1]==frequent_pairs_whithout_support[j]:
            flag=flag+1
        if subs_of_candidate[2]==frequent_pairs_whithout_support[j]:
            flag=flag+1
    if flag==3:
        final_candidate_triples.append(candidate_triples[i])
```

تعدادی از کاندیدهای سه تایی نهایی بدین صورت میباشد:

```
final_candidate_triples[0:5]
```

```
[('900222', '631634', '900225'),
 ('900222', '900246', '631829'),
 ('631634', '900207', '900202'),
 ('900101', '900268', '900259'),
 ('900212', '900268', '900225')]
```

و در نهایت با اعمال فیلتر روی کاندیداها،سه تایی های پرتکرار را بدست میاوریم.

```
#find frequant triples from final candidates

frequant_triples_list=[]
for i in tqdm(range(len(final_candidate_triples))):
    supp=support_calc(final_candidate_triples[i])
    if supp>=support_threshold:
        frequant_triples_list.append((final_candidate_triples[i],supp))
```

در نهایت سه تایی های پرتکرار بدین صورت میباشد(براساس ساپورت، نزولی مرتب شده اند):

```
frequent_triples_sorted_list=sorted(frequant_triples_list,key=lambda x: x[1],reverse=True)
frequent_triples_sorted_list
```

```
[(('900268', '900225', '900259'), 247),
 (('900222', '900246', '631829'), 170),
 (('631634', '900207', '900202'), 139),
 (('900212', '900269', '900259'), 133),
 (('900101', '900268', '900259'), 120),
 (('900212', '900268', '900225'), 109),
 (('900212', '900268', '900259'), 106),
 (('900222', '631634', '900225'), 91),
 (('900212', '900225', '900259'), 73)]
```

برای مثال سه تایی ('900268', '900225', '900259') شامل سه دوربین 900268 و900225 و900259 میباشد که به احتمال زیاد در نقطه ای از شهر قرار دارند که تردد زیادی در ان محدوده وجود دارد.اگر لوکیشن دوربین ها را در اختیار داشتیم میتوانستیم نقاط پرترافیک مختلف شهر را بدین صورت پیدا کنیم.

- پیاده سازی PCA روی دوربین ها

در این قسمت میخواهیم روی بردار های ساخته شده برای هر دوربین کاهش بعد با روش PCA انجام دهیم و سپس نتیجه را به صورت ویژوال با استفاده از کلاسترینگ مشاهده کنیم.

بردارهای هر دوربین به صورت 168 بعدی(7*24) هستند.

برای محاسبه مولفه های اساسی بردارها را استاندارد میکنیم و سپس ماتریس کواریانس را بدست میاوریم.

با توجه به الگوریتم PCA میتوان ثابت کرد جهتی که بیشترین واریانس را به ما میدهد،بردار ویژه ماتریس کواریانس متناظر با بزرگترین مقدار ویژه آن خواهد بود.جهت بعدی دومین بردار ویژه ماتریس کواریانس میباشد.و همین روند برای جهت های بعدی نیز برقرار است.

بنابراین،مولفه های اساسی اول و دوم ،بردار ویژه های ماتریس کواریانس هستند.

پس از محاسبه بردارهای ویژه،بردار دوربین ها را روی این دوبردار پروجکت میکنیم ود واقع کاهش بعد در این مرحله انجام میشود.

سپس با استفاده از الگوریتم k-means روی داده های دو بعدی بدست امده،کلاسترینگ انجام میدهیم.

تعداد کلاسترها را مانند دو بخش قبل که بر روی داده های 168 بعدی انجام دادیم،برابر با 10 در نظر میگریم.البته در اینجا هم با توجه به میزان خطا،باز هم به عدد ده رسیدیم،چرا که دو مولفه اساسی اول بخش عمده اطلاعات و پراکندگی داده ها را در بردارند و از ابتدا هم همین انتظار را داشتیم که نمودار خطا شبیه قبل شود.

در نهایت با توجه به کلاستر هر دوربین،لیبل آن را از 0تا9 در نظر میگریم و سپس بردارهای دو بعدی را با توجه به رنگ لیبل ها رسم میکنیم.

قطعه کدهای PCA و کلاسترینگ بدین شکل میباشد:

استاندارد سازی،محاسبه ماتریس کوواریانس،بدست اوردن مقادیر و بردارهای ویژه ماتریس کوواریانس

```
#we standardized data  here,the mean of each dimension will be 0 and the std will be 1
data_standardized = StandardScaler().fit_transform(data)
```

```
#calculate the covariance matrix
covMatrix = np.matmul(data_standardized.T ,data_standardized)

# calculate the eigendecomposition of the covariance matrix
values, vector = eigh(covMatrix,eigvals=(covMatrix.shape[0]-2,covMatrix.shape[0]-1))
```

کاهش ابعاد داده ها براساس دو مولفه اساسی اول:

```
# applying the PCA
# we project each vector on the first two principal components
projectedData = np.matmul(vector, data_standardized.T)
```

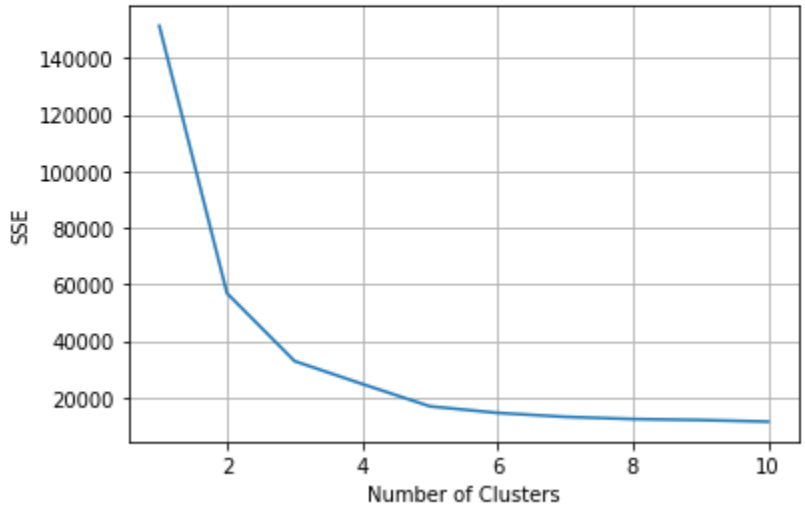
پیاده سازی الگوریتم k-means:

```
kmeans_kwargs = {"init":"random","n_init":10,"max_iter":250,"random_state":52}
e = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i,**kmeans_kwargs)
    kmeans.fit(projectedData.T)
    e.append(kmeans.inertia_)
```

با توجه به نمودار خطا،تعداد کلاسترها را برابر با ده در نظر میگیریم:

```
plt.plot(range(1, 11), e)
plt.grid()
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
```

```
Text(0, 0.5, 'SSE')
```



بدست اوردن لیبل دوربین ها و تشکیل دیتافریم:

```
#labels of cameras
labels = kmeans.labels_.tolist()
```

```
#creat dataframe of data after dimension reduction
reducedData = np.vstack((projectedData, labels)).T #Stack with labels
reducedData = pd.DataFrame(reducedData, columns = ['pca_1', 'pca_2', 'label'])
```

با توجه به لیبل هر دوربین و مولفه های اساسی،دیتافریم داده ها بعد از کاهش بعد بدین شکل میباشد:

قبل از کاهش بعد،هر دوربین معادل با بردار 168 بعدی بود و الان یک بردار 2 بعدی

	pca_1	pca_2	label
0	0.254625	-0.691969	2.0
1	-0.228706	5.460891	8.0
2	-0.233602	5.332592	8.0
3	-0.198528	5.482310	8.0
4	-0.008721	4.906783	8.0
...	...	...	...
976	-0.057246	3.972431	8.0
977	-0.128027	3.896782	8.0
978	-0.182643	5.526233	8.0
979	-0.251029	5.477000	8.0
980	-0.168403	5.509682	8.0

هر سطر نشان دهنده یک دوربین است،که با توجه به دیکشنری ساخته شده برای دوربین ها میتوان کد دوربین ها را براساس ایندکس به دست اورد.

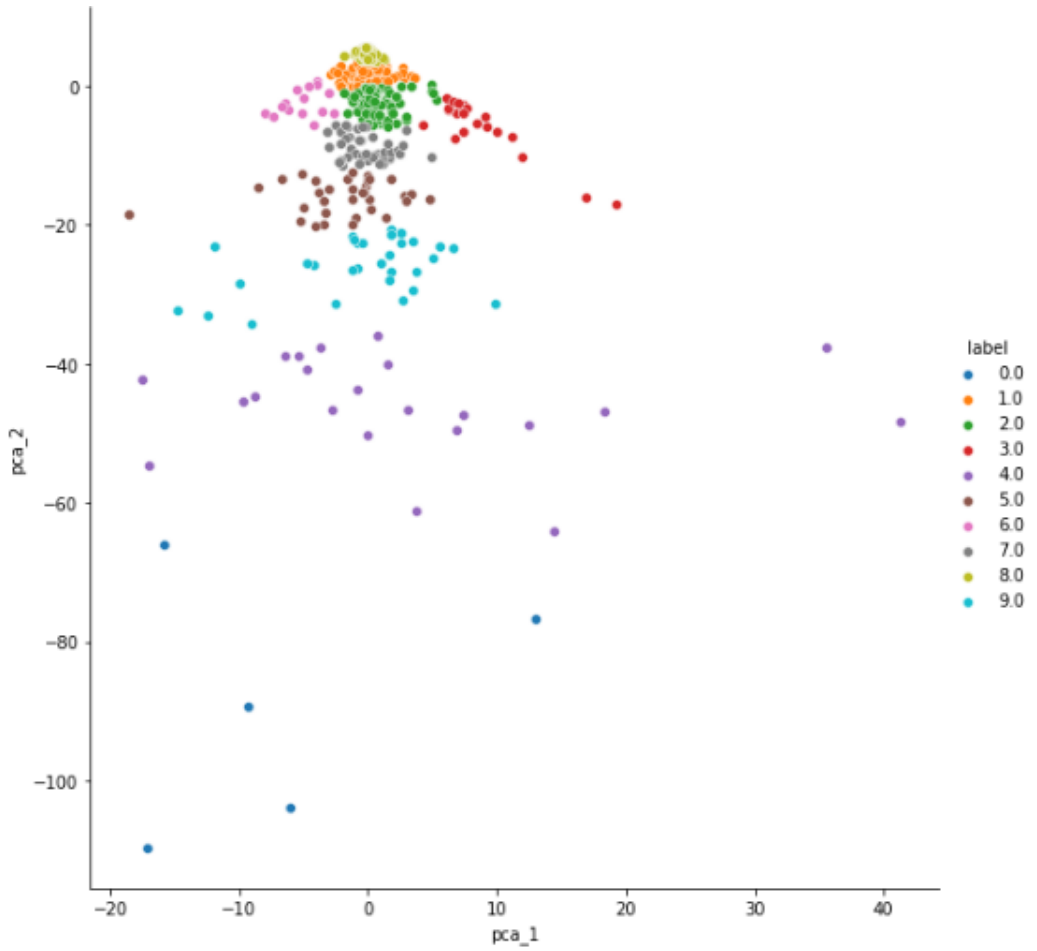
کد مربوط به پلات کردن داده های دوبعدی:

```
#plot the reducedData with clustering labels
sns.FacetGrid(reducedData, hue = 'label', size = 8) \
  .map(sns.scatterplot, 'pca_1', 'pca_2').add_legend()
```

نتیجه کلاسترینگ روی داده های دوبعدی به صورت زیر میباشد که به وضوح میتوان تفاوت دوربین ها را تنها با دو مولفه ی اساسی اول مشاهده کرد.

محور افقی:مولفه اساسی اول(pc1)

محور عمودی:مولفه اساسی دوم (pc2)



با توجه به تراکم موجود در قسمت بالایی نمودار میتوان گفت،تعداد زیادی از دوربین ها ترافیک های مشابهی را ثبت کرده اند،بدین گونه که تعداد دوربین های موجود در قسمت های پرتردد شهر به نسبت مکان های کم تردد بیشتر بوده و در نتیجه دوربین هایی که در نواحی پرتردد قرار دارند از لحاظ ترافیک ثبت شده به یکدیگر شبیه ترند و خوشه های انها به یکدیگر نزدیکتر است،به گونه ای که حتی میتوان برخی خوشه ها را (خوشه های قسمت بالایی نمودار) یک خوشه در نظر گرفت.

برای مثال دوربین های مربوط به کلاستر 6 (صورتی رنگ) بدین صورت میباشند:

	Label 6
0	22010139
1	22010112
2	100701297
3	22010125
4	100701298
5	22010111
6	107301
7	22010110
8	22010122
9	205202
10	22010057
11	22010134
12	100701156
13	137
14	100701293

قطعه کدهای مربوط به بدست اوردن کد هر دوربین براساس ایندکس ذخیره شده در دیکشنری و لیبل به دست امده از کلاسترینگ:

```
#find device code from dictionary
def key_find(arg):
    for key, value in device_dict.items():
         if arg == value:
             return key
```

```
#find correspending labels
for i in range(len(labels)):
  if labels[i]==0:
    label0.append(key_find(i))
  elif labels[i]==1:
    label1.append(key_find(i))
  elif labels[i]==2:
    label2.append(key_find(i))
  elif labels[i]==3:
    label3.append(key_find(i))
  elif labels[i]==4:
    label4.append(key_find(i))
  elif labels[i]==5:
    label5.append(key_find(i))
  elif labels[i]==6:
    label6.append(key_find(i))
  elif labels[i]==7:
    label7.append(key_find(i))
  elif labels[i]==8:
    label8.append(key_find(i))
  elif labels[i]==9:
    label9.append(key_find(i))
```