

به نام خدا

گزارش پروژه چهارم داده کاوی

عاطفه نادری

\*پیشاپیش ممنون که گزارش من رو میخوانی. یکم وقت برای قشنگ سازی گزارش نداشتم.\*

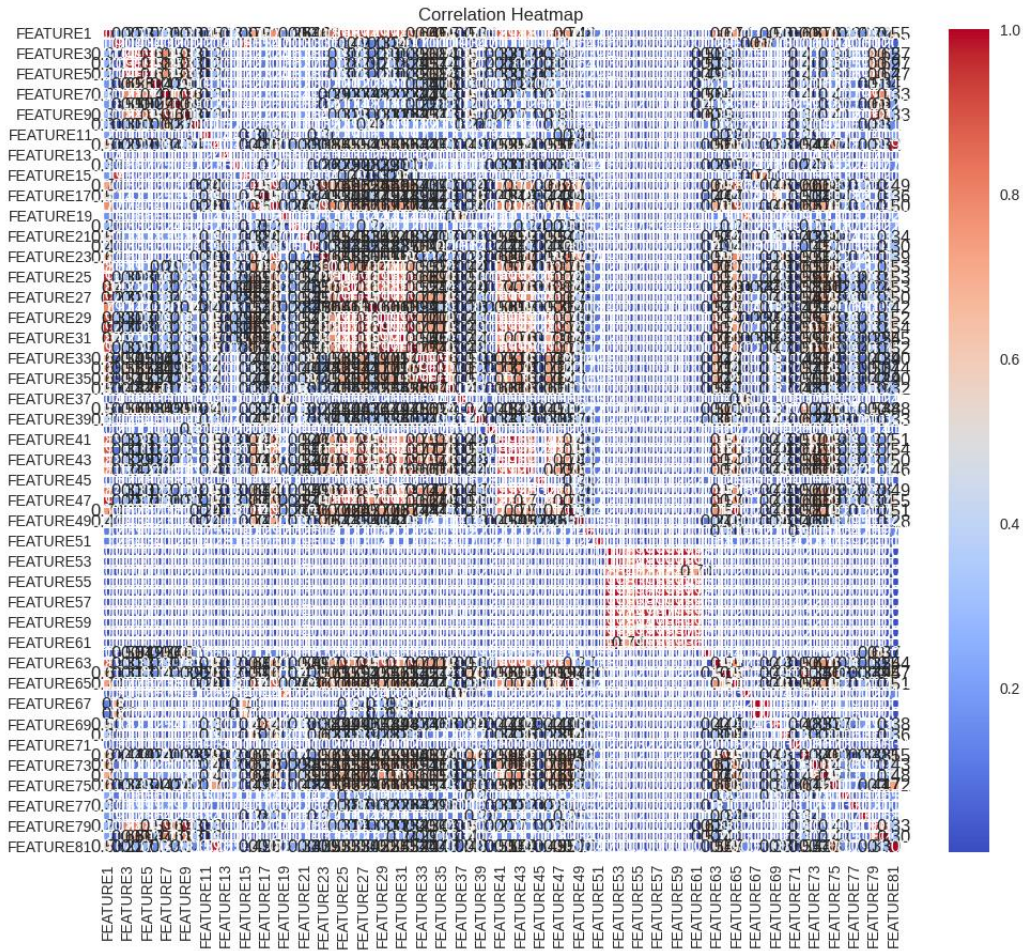
بیشتر عکس ها در کد قرار دارد.

## 1 . Preprocessing

ابتدا دو ستون class label و index را از فایل Data Set حذف میکنیم و روی ادامه آن که شامل ۸۱ ویژگی است با تابع z-score نرمالایز می کنیم. تا scale همه ستون ها در یک بازه قرار گیرند.

## 2 . feature analysis

ماتریس هم بستگی را بدست آوردیم. و معیار همبستگی pearson استفاده نمودیم. و heat map آن را نیز رسم کردیم. هم چنین ۲۰ ویژگی ای که بیشترین تاثیر را داشته اند نیز به ترتیب نزولی لیست نمودیم. همانطور که در heat map مشخص است. آن جاهایی که رنگ قرمز تری داره نشان دهنده correalation بالاتری است.



### 3. یادگیری Unsupervised

برای مورد الف و ب از کتابخانه از این [لینک](#) استفاده شده است. مقدار  $K=5$  قرار دادم. چند بار با معیار silhouette دنبال بهترین  $k$  بودم منتهای حدود ۵ ساعت روی کل دیتاست طول میکشید و چندین بار تا مراحل آخر رفت و قطع شد. با صحبت با ta محترم به همین مقدار اکتفا نمودم که برای  $k=5$  مقدار  $sil=0.3$  است.

```
✓ [46] # Import Special library to find feature importances after kmeans clustering
0s from kmeans_feature_imp import KMeansInterp
```

```
✓ [47] kms = KMeansInterp(
17s     n_clusters=5,
     ordered_feature_names=X.columns.tolist(),
     feature_importance_method='wcss_min', # or 'unsup2sup'
).fit(X.values)
```

cluster/\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from

که پس از import نمودن آن، ابتدا با k-means کلاستر بندی میکند و سپس بر اساس متد WCSS ویژگی‌ها را رتبه بندی میکند. Wcss اندازه گیری فشردگی (compactness) یا انسجام (cohesion) خوشه‌ها در یک الگوریتم خوشه بندی، به ویژه k-means است. نشان می دهد که نقاط داده در یک خوشه چقدر به مرکز آن خوشه نزدیک هستند.

این روش از تابع اصلی :

```
def get_feature_imp_wcss_min(self):
    labels = self.n_clusters
    centroids = self.cluster_centers_
    centroids = np.vectorize(lambda x: np.abs(x))(centroids)
    sorted_centroid_features_idx = centroids.argsort(axis=1)[:,:-1]

    cluster_feature_weights = {}
    for label, centroid in zip(range(labels), sorted_centroid_features_idx):
        ordered_cluster_feature_weights = centroids[label][sorted_centroid_features_idx[label]]
        ordered_cluster_features = [self.ordered_feature_names[feature] for feature in centroid]
        cluster_feature_weights[label] = list(zip(ordered_cluster_features,
                                                  ordered_cluster_feature_weights))

    return cluster_feature_weights
```

استفاده میکند.

نحوه کار بدین صورت است که از هر کلاستر مرکز آن را بدست می‌آورد. در واقع فاصله هر نقطه از مرکز برابر است با:


$$distance(C_j, p) = \sqrt{\sum_{i=1}^d (C_{ji} - p_i)^2}$$


بر اساس WCSS به موقع کلاسترینگ، برای به حداقل رساندن آن، هر نقطه داده به نزدیکترین مرکز اختصاص داده میشود.

$$WCSS(C_j) = \sum_{p_i=1 \in C_j}^{p_m} distance(C_j, p_i)^2$$

از آنجایی که هدف K-Means به حداقل رساندن مجموع مربع های درون خوشه ای (The Within-Cluster Sum of Squares) است و با فرض اینکه متریک فاصله استفاده شده، فاصله اقلیدسی است، می توانیم ابعاد و ویژگی هایی را که مسئول بیشترین مقدار WCSS هستند (مجموع مربع های هر نقطه داده) را پیدا کنیم. به این صورت که اون مرکز یک فاصله ای نسبت به باقی ویژگی ها دارد و بر اساس این فاصله ای که از هر ویژگی داره میتواند اهمیت رو نسبت به آن ویژگی نشان دهد. مثلا اگر فاصله center1 از دو ویژگی x1 و x2 یکسان باشد، یعنی این دو ویژگی اهمیت یکسانی روی کلاستر و مرکز center1 داشته اند.

در مرحله بعدی به ازای هر ویژگی و در هر کلاستر، یک سری وزن میدهد که نشان دهنده آن اهمیت آن ویژگی در کلاسترینگ است.

 kms.feature\_importances\_ # Features here are words

 {0: [('FEATURE24', 2.87429596004089),  
('FEATURE1', 2.866443490722764),  
('FEATURE18', 2.841860389428389),  
('FEATURE31', 2.7982806699813643),  
('FEATURE26', 2.789413949139161),  
('FEATURE42', 2.7818470257933274),  
('FEATURE47', 2.7519606373687755),  
('FEATURE44', 2.7510131996188614),  
('FEATURE30', 2.6584426334895457),  
('FEATURE16', 2.6489313437240294),  
('FEATURE25', 2.6232215909667165),  
('FEATURE41', 2.56457576830041),  
('FEATURE65', 2.560330624146655),  
('FEATURE48', 2.559637525184282),  
('FEATURE75', 2.5572933473813357),  
('FEATURE29', 2.513176889492789),  
.....

چون مجموع وزن ها در هر کلاستر برابر با ۱ نبود، ابتدا آن را نرمالایز نمودم تا بتوان مقایسه خوبی را نسبت به وزن ویژگی ها در هر کلاستر و با کلاستر دیگر داشت.

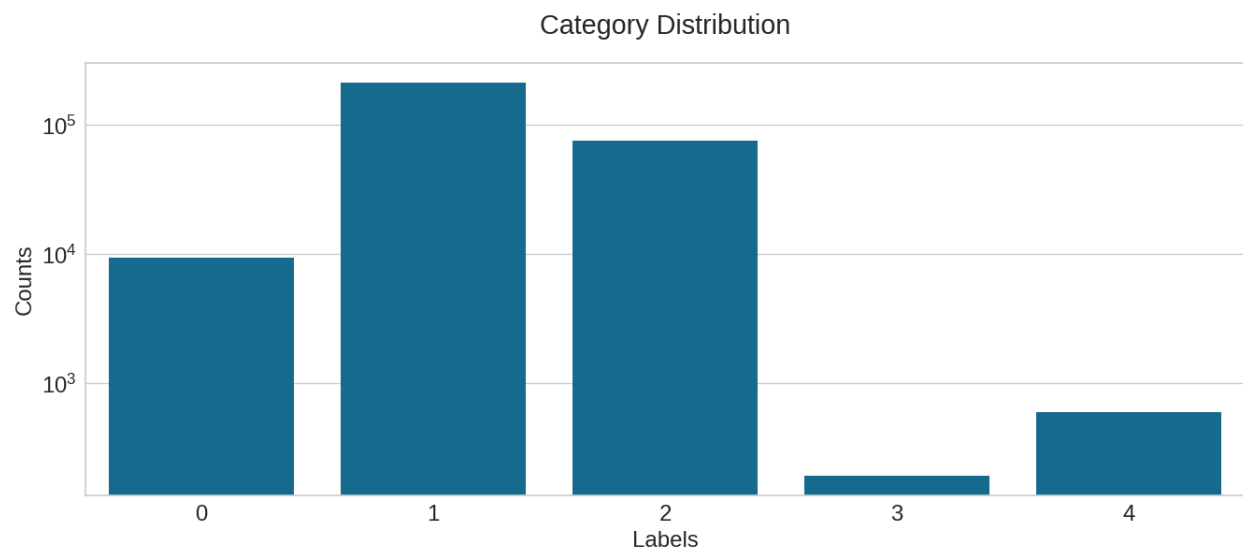
```
Normalized Data:
Key 0: [('FEATURE24', 0.02585059190733556), ('FEATURE1', 0.02585059190733556), ('FEATURE72', 0.03170169986096343), ('FEATURE64', 0.041796581019511565), ('FEATURE56', 0.09606179988167333), ('FEATURE43', 0.03298681147713984), ('FEATURE32', 0.02585059190733556)]
Key 1: [('FEATURE72', 0.03170169986096343), ('FEATURE64', 0.041796581019511565), ('FEATURE56', 0.09606179988167333), ('FEATURE43', 0.03298681147713984), ('FEATURE32', 0.02585059190733556)]
Key 2: [('FEATURE72', 0.041796581019511565), ('FEATURE64', 0.09606179988167333), ('FEATURE56', 0.03298681147713984), ('FEATURE43', 0.02585059190733556), ('FEATURE32', 0.03170169986096343)]
Key 3: [('FEATURE56', 0.09606179988167333), ('FEATURE43', 0.041796581019511565), ('FEATURE32', 0.03170169986096343), ('FEATURE24', 0.02585059190733556), ('FEATURE1', 0.02585059190733556)]
Key 4: [('FEATURE43', 0.03298681147713984), ('FEATURE32', 0.041796581019511565), ('FEATURE24', 0.02585059190733556), ('FEATURE1', 0.02585059190733556), ('FEATURE72', 0.03170169986096343)]
```

```
[ ] # Calculate the sum of values for each key in the normalized data
key_sums2 = {key: np.sum([val for _, val in sublist]) for key, sublist in key_data.items()}

# Print the sum of values for each key
print("Sum of Values for Each Key:")
for key, total_sum in key_sums2.items():
    print(f"Key {key}: {total_sum}")
```

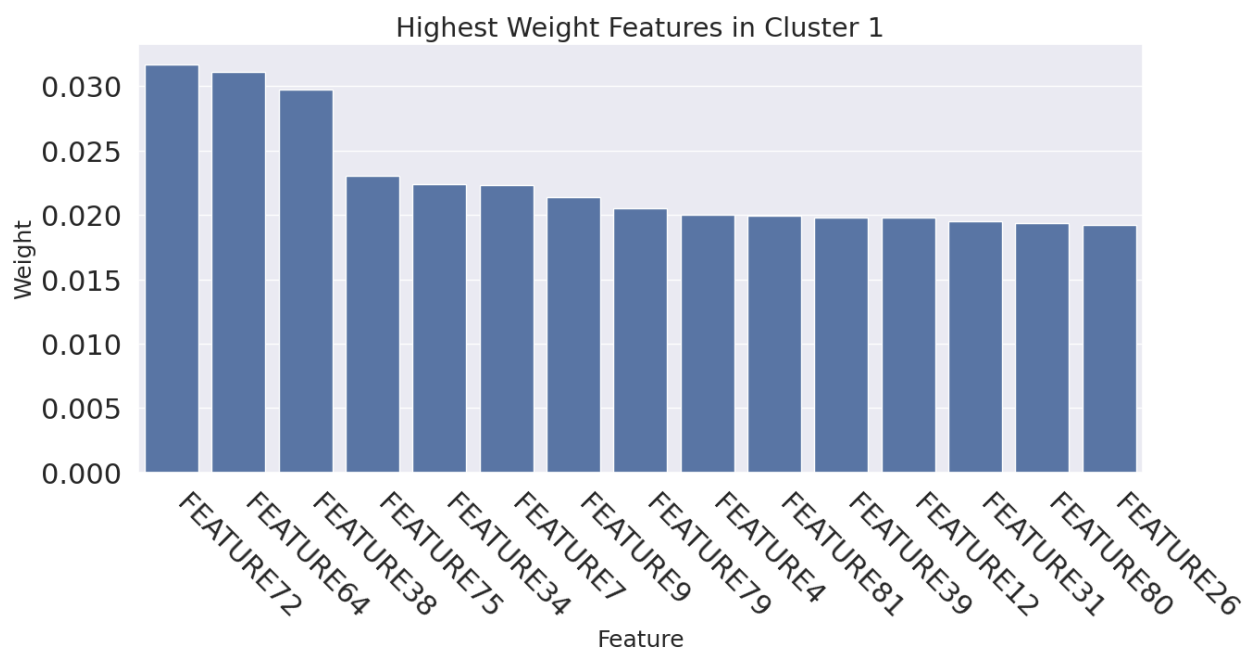
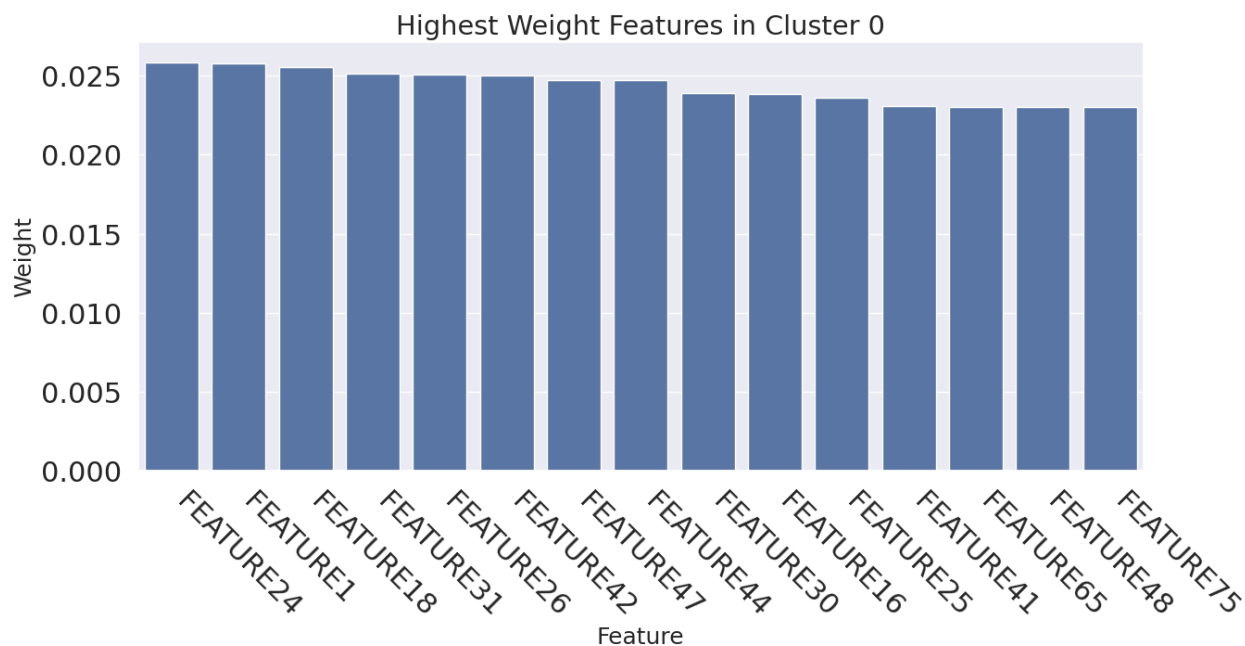
```
Sum of Values for Each Key:
Key 0: 1.0
Key 1: 1.0
Key 2: 1.0
Key 3: 1.0000000000000002
Key 4: 1.0
```

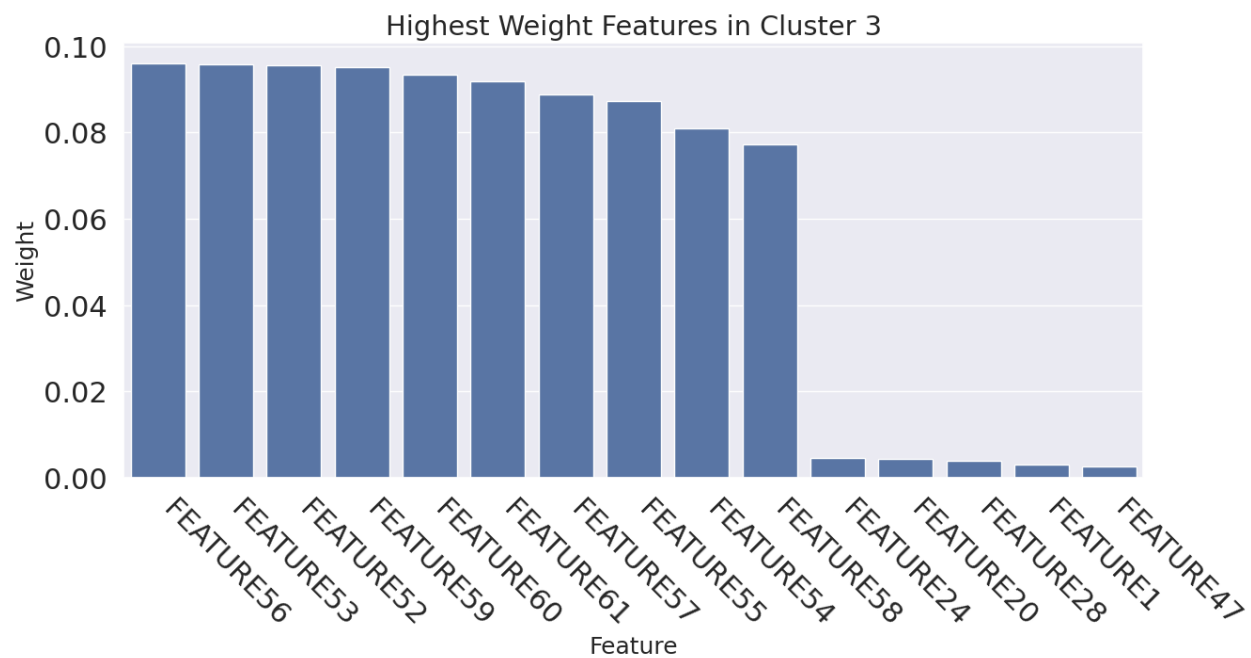
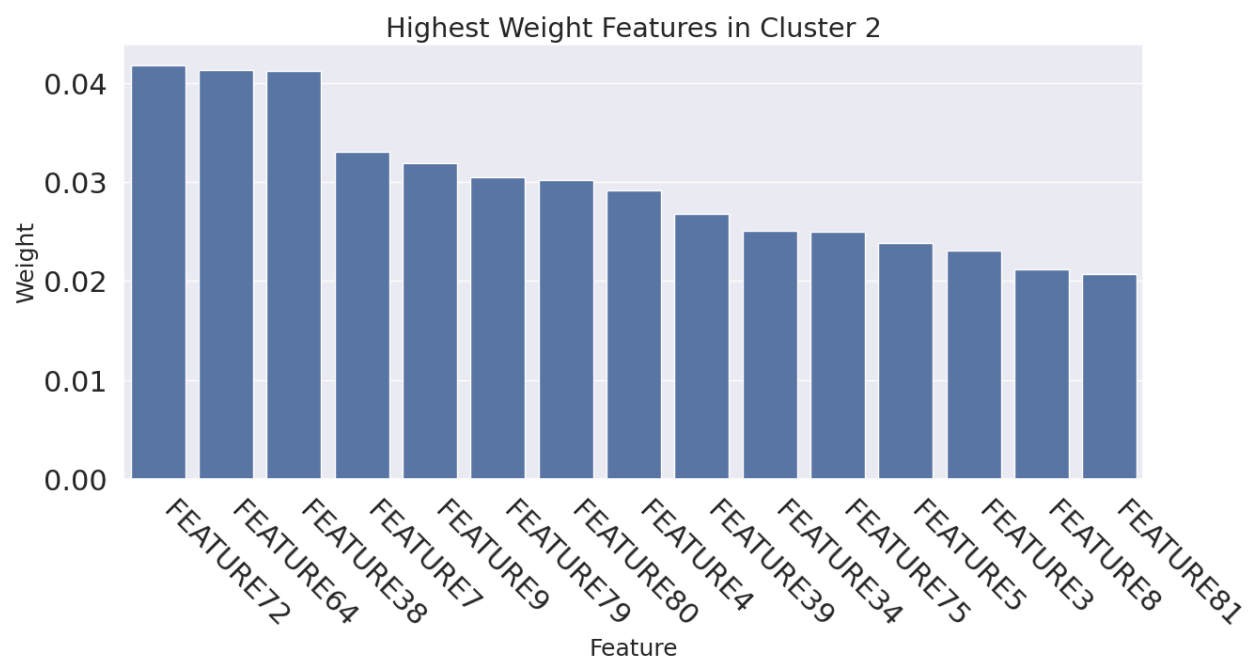
توزیع هر کلاستر :



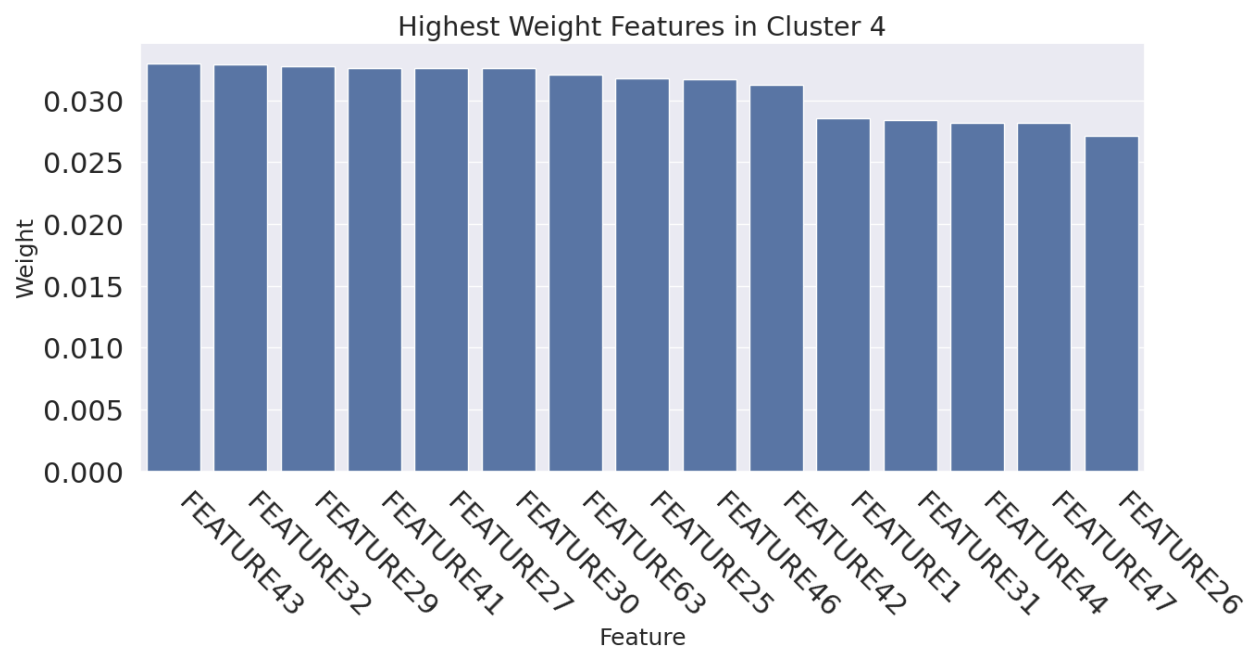
قسمت ۴. الف:

اهمیت هر ویژگی در هر کلاستر در تصاویر بعدی مشخص شده است:









تحلیل:

با توجه به قسمت سوم و correlation، ویژگی‌هایی که با هم بسیار corre بالایی داشتند در این رتبه بندی نیز در کنار هم یا با فاصله‌ای نزدیک کنار یکدیگر ظاهر شدند. برای مثال feature 48 و feature 65 وابستگی بسیار بالایی به یکدیگر داشتند.

ویژگی‌های موجود در این دیتاست اسم و مشخصاتی ندارند، بنابراین نمیتوان به طور خاص توجیه کرد که چرا این ویژگی در این کلاستر اهمیت بسیار بیشتری داشته و یا هر کلاستر نشان دهنده کدام داده‌ها هستند. ولی به طور کلی سه ویژگی‌ای که بیشترین تاثیر را در هر کلاستر داشته‌اند به طور خلاصه به صورت زیر است:

```

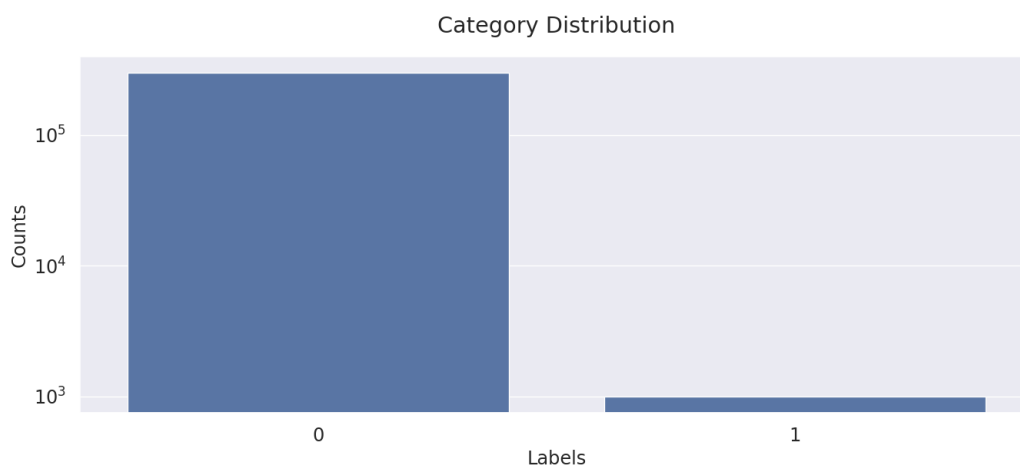
Normalized Data:
Key 0: [('FEATURE24', 0.02585059190733556), ('FEATURE1', 0.025779969054772787), ('FEATURE18', 0.02555887570592762), (
Key 1: [('FEATURE72', 0.03170169986096343), ('FEATURE64', 0.031128536134494473), ('FEATURE38', 0.029763296286535804),
Key 2: [('FEATURE72', 0.041796581019511565), ('FEATURE64', 0.04131049956229421), ('FEATURE38', 0.041198302760568566),
Key 3: [('FEATURE56', 0.09606179988167333), ('FEATURE53', 0.09590634625461346), ('FEATURE52', 0.09553035959927794), (
Key 4: [('FEATURE43', 0.03298681147713984), ('FEATURE32', 0.03287788752791686), ('FEATURE29', 0.03276984365895396), (
  
```



و هم چنین مقدار تاثیر گذاری ۱۵ ویژگی اول روی کلاستربندی هر خوشه یک گونه نیست. مثلاً در کلاستر سوم، وزن های ۳ ویژگی اول ۰.۰۹ است و باقی وزن های خیلی کمتری دارند. در صورتی که در دیگر کلاسترها بین ۱۵ ویژگی اول وزن ها متعادل تر است.

قسمت ۴. ب:

توزیع هر کلاستر (۰ و ۱) با توجه به class label :



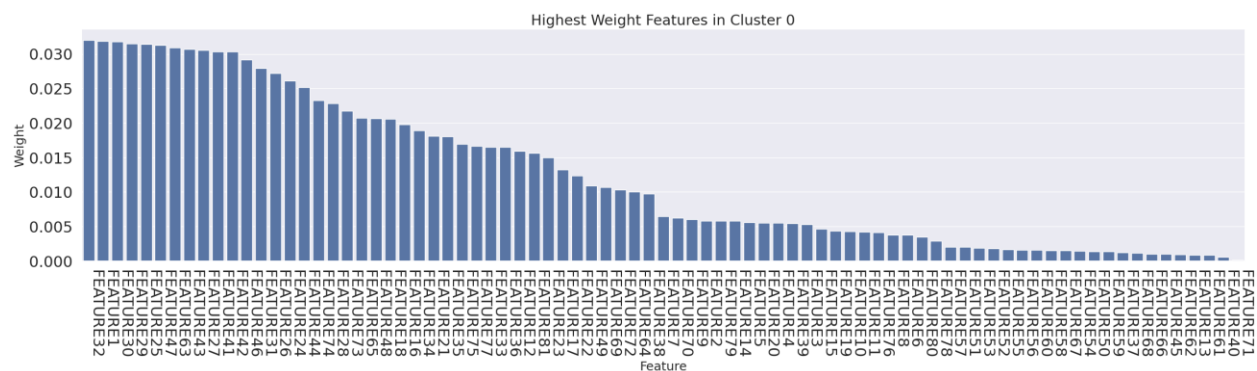
ابتدا داده های مختص هر class label را جدا میکنیم به df\_0 و df\_1. با توجه به آنکه df\_1 تعداد داده های بسیار کمی (۹۹۰) در برابر دیگر کلاستر دارد در مراحل بعدی این را لحاظ نمودیم و undersampling انجام دادیم. پس از آن مانند سوال قبلی از کتابخانه kmeans\_feature\_imp و تابع KMeansInterp مجدداً استفاده شده است. و مقدار کلاستر را نیز به تعداد یک قرار دادیم.

و فیچرها را با توجه به معیار wcss رتبه بندی کرده و نمایش دادیم. برای مثال برای کلاستر 0:

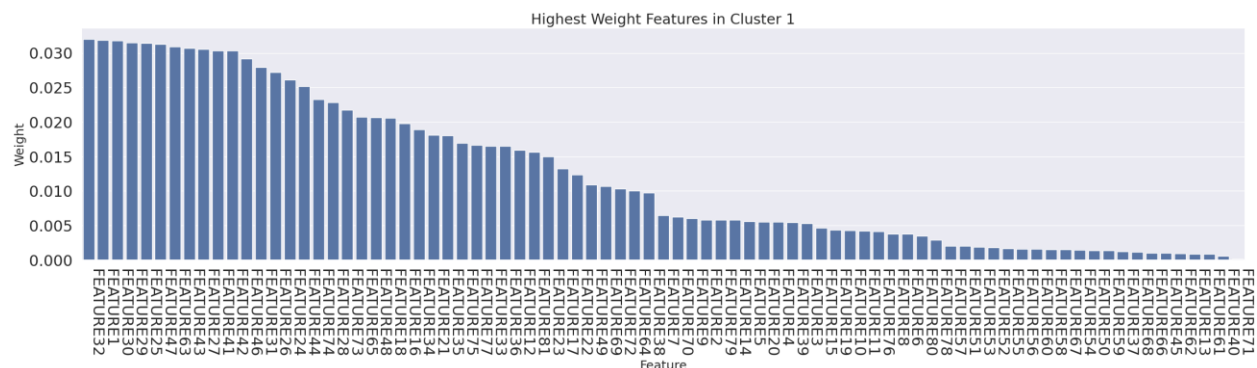
```
[109] kms.feature_importances_.items()
```

```
dict_items([(0, (('FEATURE32', 0.033570078829906556), ('FEATURE1', 0.033444601559284935), ('FEATURE30', 0.03329886939358991), ('FEATURE29', 0.03302786855254486), ('FEATURE25', 0.032930264914139286), ('FEATURE47', 0.032835430842097585), ('FEATURE63', 0.032419176129203184), ('FEATURE43', 0.032176151766242396), ('FEATURE27', 0.03200235666732202), ('FEATURE41', 0.031826804475993606), ('FEATURE42', 0.03180113106031426), ('FEATURE46', 0.03055220540001051), ('FEATURE31', 0.02927427355296631), ('FEATURE26', 0.02850068668042128), ('FEATURE24', 0.027383772712384267), ('FEATURE44', 0.02639362312738561), ('FEATURE74', 0.024418530026525796), ('FEATURE28', 0.02397160010342822), ('FEATURE73', 0.022773664047724355), ('FEATURE65', 0.02171684711423931), ('FEATURE48', 0.021672616982963643), ('FEATURE18', 0.02157064631134721), ('FEATURE16', 0.02072797320716182), ('FEATURE34', 0.019850539210732918), ('FEATURE21', 0.0190235782936367), ('FEATURE35', 0.0189070516969061), ('FEATURE75', 0.0177757094484832), ('FEATURE77', 0.017452038870327362), ('FEATURE33', 0.01734750454848294), ('FEATURE36', 0.017282791014783267), ('FEATURE12', 0.016725102246241977), ('FEATURE81', 0.016433095646711398), ('FEATURE23', 0.015736450885538166), ('FEATURE17', 0.013844148048070897), ('FEATURE22', 0.012958694590534227), ('FEATURE49', 0.011413209201220766), ('FEATURE69', 0.011221943268569018), ('FEATURE72', 0.01086306000797037), ('FEATURE64', 0.010502742997390476), ('FEATURE38', 0.01024921029381384), ('FEATURE7', 0.00675094068881507), ('FEATURE70', 0.006543739987968449), ('FEATURE9', 0.006311020690261163), ('FEATURE2', 0.006132945009289482), ('FEATURE79', 0.006081098145950345), ('FEATURE14', 0.006067946921187612), ('FEATURE5', 0.005894759784782977), ('FEATURE20', 0.005810911100603642), ('FEATURE4', 0.005802135714409494), ('FEATURE39', 0.0056857220696875975), ('FEATURE3', 0.005585013153435494), ('FEATURE15', 0.004877114623333007), ('FEATURE19', 0.0045529868566335655), ('FEATURE10', 0.00447283393067803), ('FEATURE11', 0.004409518772846153), ('FEATURE76', 0.00435503632258887), ('FEATURE8', 0.003936877321944241), ('FEATURE6', 0.00393118787445899), ('FEATURE80', 0.003671050173700116), ('FEATURE78', 0.003054048011917083), ('FEATURE57', 0.0021376185263197183), ('FEATURE51', 0.00209850298216168), ('FEATURE53', 0.0019611899525744944), ('FEATURE52', 0.0018778865729961792), ('FEATURE55', 0.0017615484688074177), ('FEATURE56', 0.0016759595706627311), ('FEATURE60', 0.0016511913410878845), ('FEATURE58', 0.0016185440832837833), ('FEATURE67', 0.0015725664376639238), ('FEATURE54', 0.0015497560331763703), ('FEATURE50', 0.001445137181356382), ('FEATURE59', 0.0014164640313411778), ('FEATURE37', 0.0012504923046833167), ('FEATURE68', 0.0012097266596360309), ('FEATURE66', 0.0010943404539682981), ('FEATURE45', 0.0010553059498838314), ('FEATURE62', 0.000998236211922173), ('FEATURE13', 0.0009285517326536324), ('FEATURE61', 0.0008921174718500201), ('FEATURE40', 0.0005918912530823929), ('FEATURE71', 0.00016685965352795663)]])
```

رتبه بندی ویژگی ها در کلاستر 0:



رتبه بندی ویژگی ها در کلاستر 1:



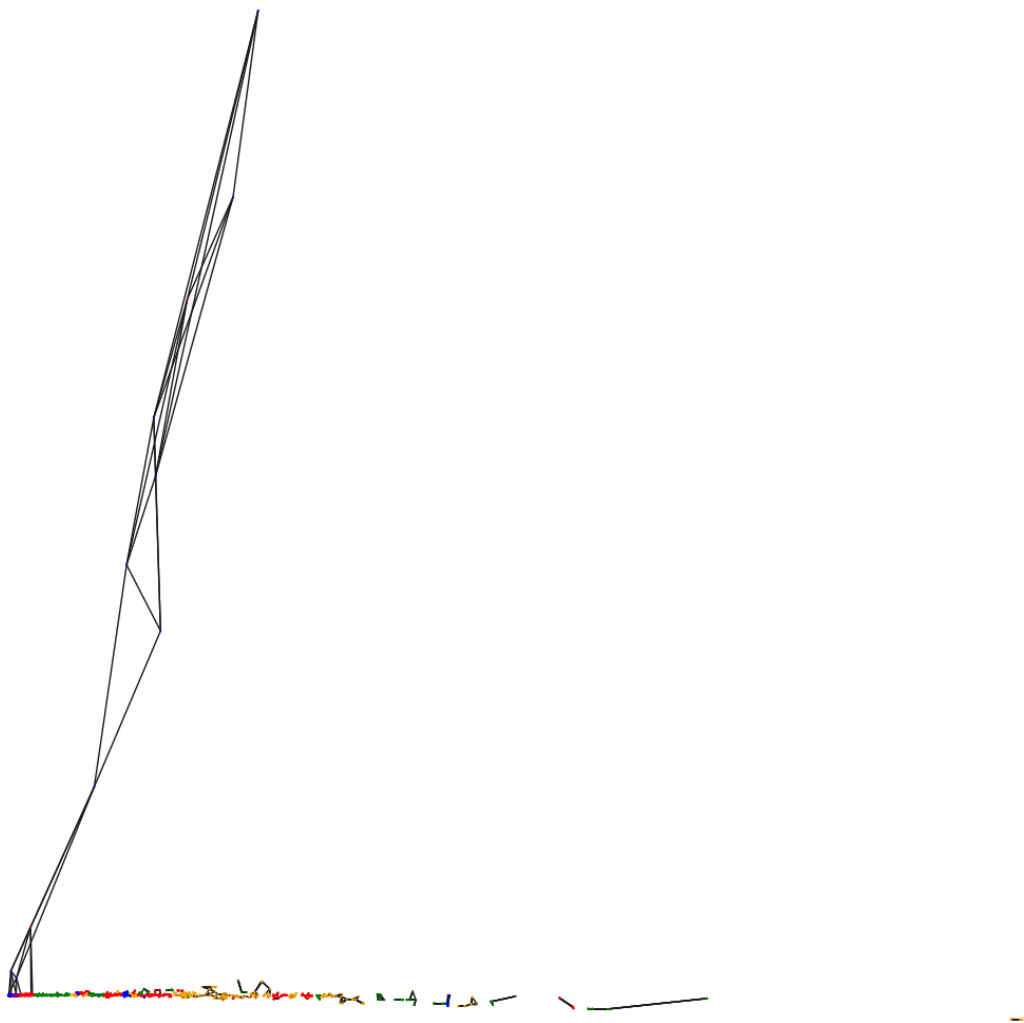
قسمت ۴.ج:

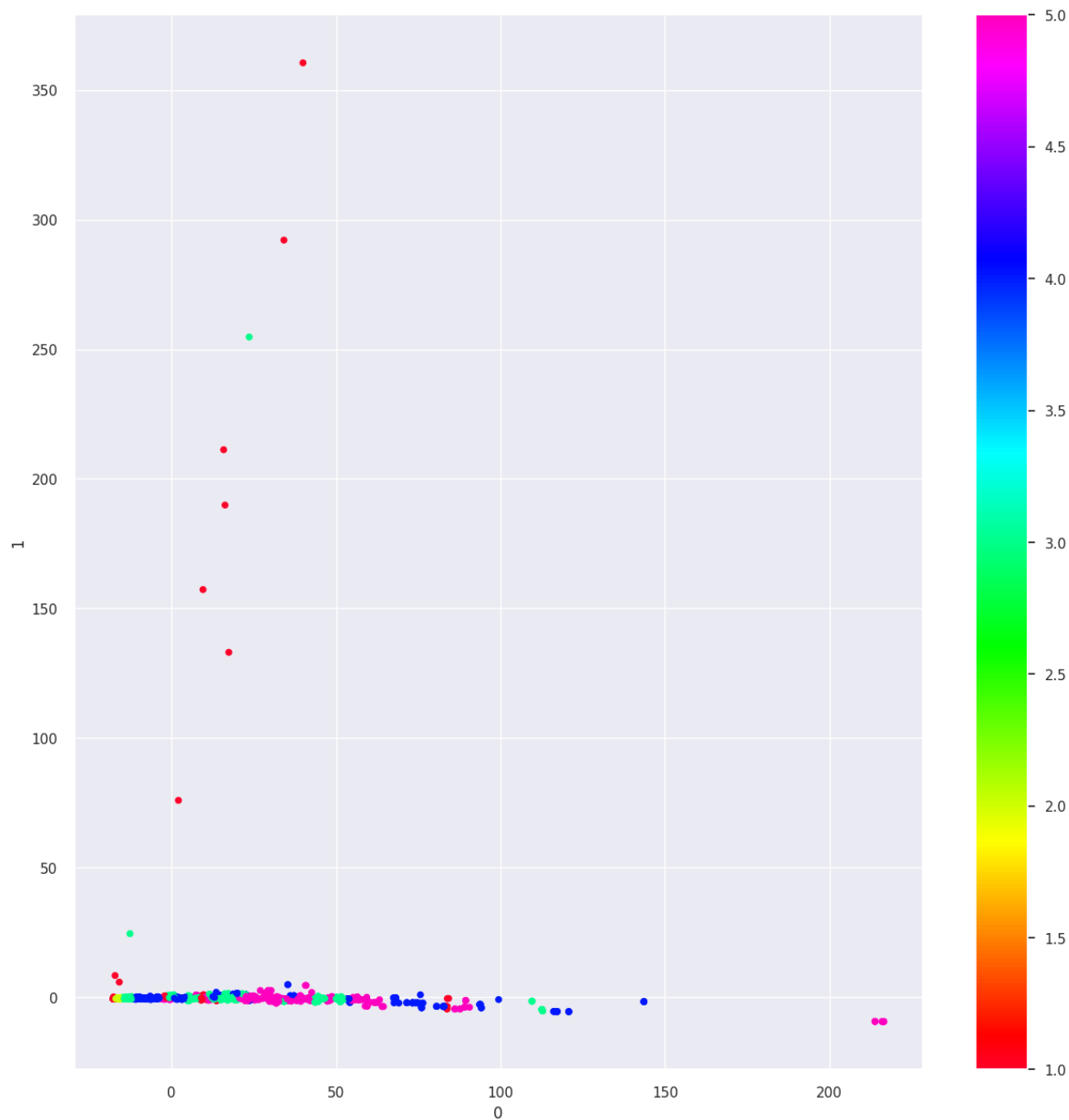
قبل از اجرای این الگوریتم، ابتدا یک undersampling روی دیتاست قرار میدهم هم به جهت آنکه داده های ما بسیار زیاد اند و زمان اجرای الگوریتم بسیار طولانی خواهد شد و نیز به دلیل جلوگیری از overfit و imbalance بودن دیتاست. از لیبل ۰ به اندازه دو برابر لیبل ۱ بر میداریم.

بعد از آن pca را روی دیتاست اجرا میکنیم تا دو بعدی شود. چون ورودی الگوریتم chameleon که از این لینک استفاده شده است، فقط دو بعدی میگیرد.

چند بار الگوریتم را با پارامترهای مختلف بررسی نمودم منتها جواب های sillhouet متفاوت بود. چون روی کل دیتاست هم اجرا نکردم. اما به ازای این پارامترهای مقدار sil برابر با ۰.۰۸ شد. درست است مقدار پایینی است و کلاسترینگ خوبی را نداشته است.

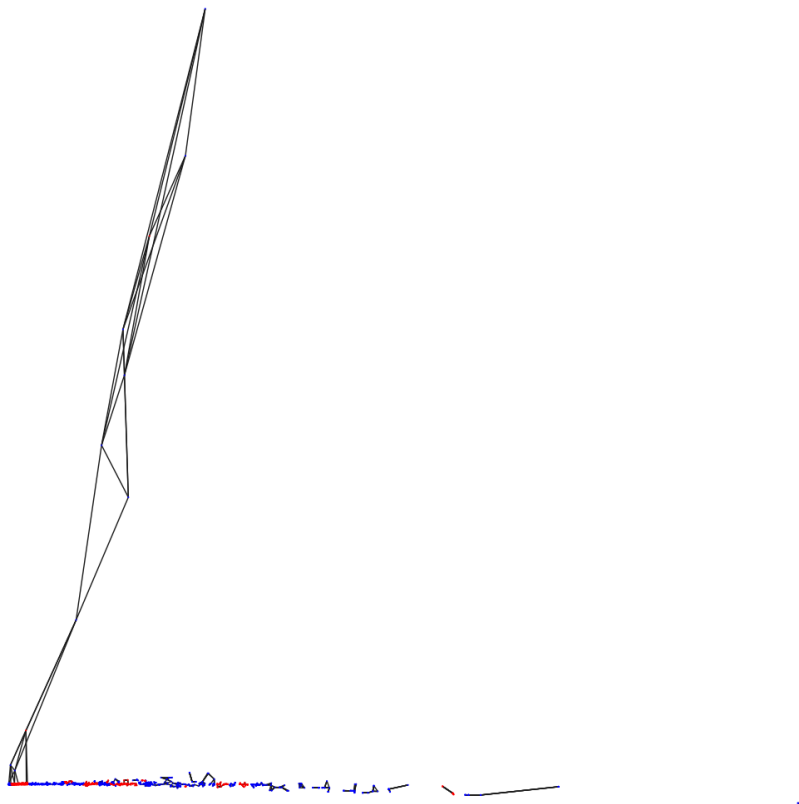
پس از اجرای الگوریتم chameleon روی دیتاست. با تعداد کلاستر برابر با ۵ و knn=4، خروجی گراف حاصل از آن به شرح زیر است:



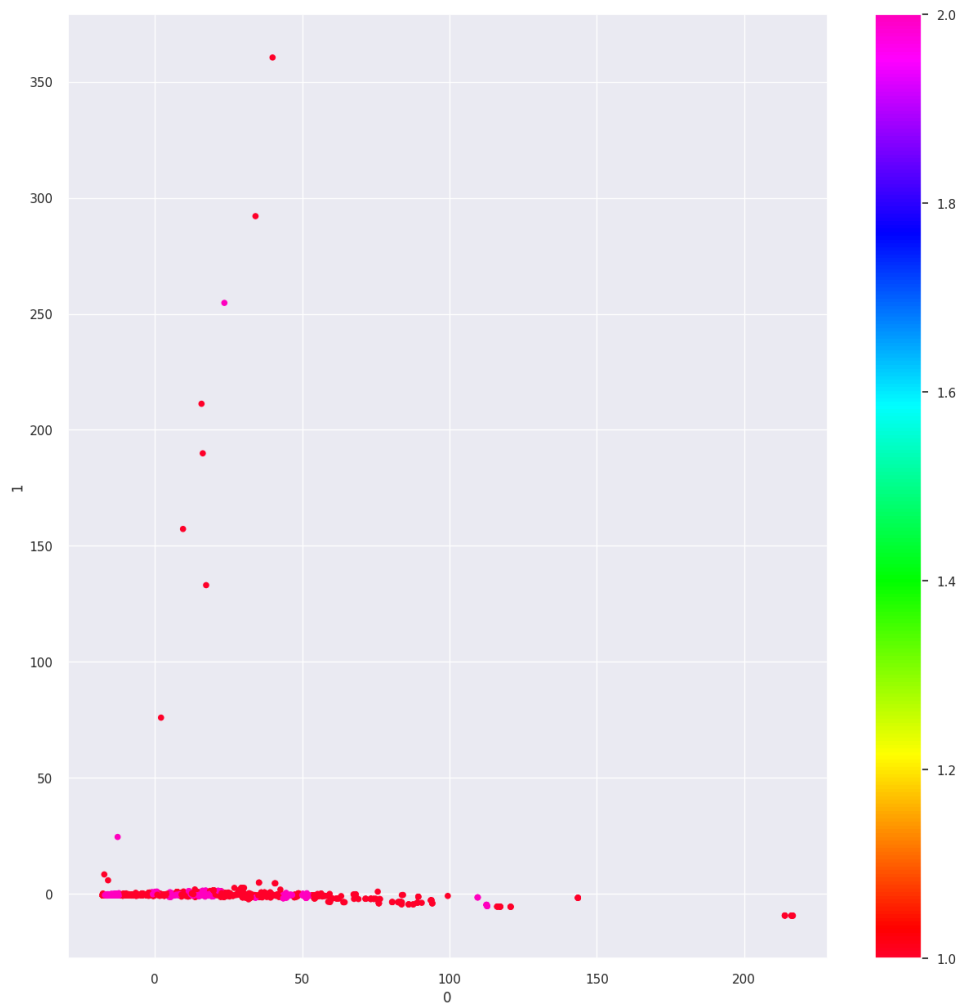


برای پارامترهای تعداد کلاستر برابر با ۲ و مقدار  $knn=4$  تصویر به این شکل شد:

شکل اول در واقع گراف کلی را نشان میدهد که هر نقطه به ۴ نقطه دیگر متصل است، و بنظر میرسد که یک سری داده که در خط افقی قرار دارند، چگالی زیادی دارند و نقاط کمی هستند که در راست مثلث قرار دارند و با قاعده مثلث فاصله زیادی دارند. شاید این نقاط outlier باشند.



در تصویر بعدی، که حاصل از کلاسترینگ الگوریتم chameleon است، نقاط صورتی نقاط دارای برچسب ۰ و نقاط قرمز برچسب ۱ هستند. و این تصویر همان تصویر بالایی است که هر نقطه هر راس را نشان میدهد. و احتمالاً دو کلاستری حاصل از قرار گیری نقاط قرمز و اتصال یالی بین آن ها با هم و اتصال نقاط صورتی با هم دیگر است. با توجه به آنکه نقاط صورتی و قرمز از هم گسیختگی و قابلیت تمییز آن ها کم است و این را در معیارها هم متوجه میشویم که مقادیر خوبی را نشان نمیدهد.



تصویر بعدی داده ها را به همراه labelد ای که الگوریتم chameleon به آن اختصاص داده است را نشان میدهد.

```
# Print the merged dataframe
merged_df
```

FEATURE6	FEATURE7	FEATURE8	FEATURE9	FEATURE10	...	FEATURE73	FEATURE74	FEATURE75	FEATURE76	FEATURE77	FEATURE78	FEATURE79	FEATURE80	FEATURE81	cluster
-0.220401	-0.683471	-0.329882	-0.603371	-0.533083	...	-0.274764	-0.390445	-0.501802	-0.296158	-0.034003	-0.878343	-0.596197	-0.542861	-0.572221	1
-0.220401	-0.683471	-0.329882	-0.603371	-0.275166	...	-0.282714	-0.390445	-0.501802	-0.296158	-0.034003	-0.878343	-0.596197	-0.542861	-0.438186	1
-0.220401	-0.683471	-0.329882	-0.700488	-0.533083	...	-0.258863	-0.378129	-0.501802	-0.296158	-0.034003	-0.878343	-0.514285	-0.542861	-0.706257	1
4.259352	3.863831	3.580990	4.349559	0.240670	...	15.332191	18.218258	21.807810	1.632692	43.044045	0.157306	4.631729	3.521426	22.347851	0
-0.220401	-0.380318	-0.329882	-0.506255	0.240670	...	-0.083950	-0.353498	-0.501802	-0.296158	-0.034003	1.192955	-0.596197	-0.542861	-0.572221	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
-0.220401	-0.607683	-0.329882	-0.506255	-0.533083	...	-0.282714	-0.390445	-0.501802	-0.296158	-0.034003	-0.878343	-0.596197	-0.542861	-0.706257	1
0.899537	-0.152953	-0.329882	-0.117790	-0.017248	...	-0.258863	-0.267290	0.123701	-0.296158	-0.034003	-0.878343	-0.145680	-0.016773	0.634099	0
0.899537	-0.152953	-0.329882	-0.117790	-0.275166	...	-0.139604	0.003651	0.123701	0.668267	-0.034002	0.157306	-0.048874	-0.112426	0.500063	0
-0.220401	-0.077164	-0.329882	-0.409139	1.014423	...	-0.219110	-0.316552	-0.293301	-0.296158	-0.034003	0.157306	-0.391416	-0.367499	-0.706257	0
-0.220401	-0.759260	-0.329882	-0.700488	-0.533083	...	-0.266813	-0.378129	-0.501802	0.668267	-0.034003	0.157306	-0.678109	-0.542861	-0.706257	1

پارامترهای مختلفی روی این دیتاست آزمایش شد اما با  $knn=4$  و تعداد کلاستر برابر با ۲ نتیجه بهتر از سایرین بود.



تحلیل ویژگی های مهم:

برای این قسمت از دو روش استفاده کردیم. اول با pca و روش دوم استفاده از الگوریتم supervised مانند random forest.

با pca

پس از آنکه خروجی الگوریتم cham را مشاهده نمودیم. یک سری label به ازای هر row ایجاد میکند و چون این label ها برای دیتاستی هست که دو بعدی بوده پس لازم است آن را دیتاست اصلی (قبل از pca) مرج کنیم.

```
# Print the merged dataframe
merged_df
```

FEATURE6	FEATURE7	FEATURE8	FEATURE9	FEATURE10	...	FEATURE73	FEATURE74	FEATURE75	FEATURE76	FEATURE77	FEATURE78	FEATURE79	FEATURE80	FEATURE81	cluster
-0.220401	-0.683471	-0.329882	-0.603371	-0.533083	...	-0.274764	-0.390445	-0.501802	-0.296158	-0.034003	-0.878343	-0.596197	-0.542861	-0.572221	1
-0.220401	-0.683471	-0.329882	-0.603371	-0.275166	...	-0.282714	-0.390445	-0.501802	-0.296158	-0.034003	-0.878343	-0.596197	-0.542861	-0.438186	1
-0.220401	-0.683471	-0.329882	-0.700488	-0.533083	...	-0.258863	-0.378129	-0.501802	-0.296158	-0.034003	-0.878343	-0.514285	-0.542861	-0.706257	1
4.259352	3.863831	3.580990	4.349559	0.240670	...	15.332191	18.218258	21.807810	1.632692	43.044045	0.157306	4.631729	3.521426	22.347851	0
-0.220401	-0.380318	-0.329882	-0.506255	0.240670	...	-0.083950	-0.353498	-0.501802	-0.296158	-0.034003	1.192955	-0.596197	-0.542861	-0.572221	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
-0.220401	-0.607683	-0.329882	-0.506255	-0.533083	...	-0.282714	-0.390445	-0.501802	-0.296158	-0.034003	-0.878343	-0.596197	-0.542861	-0.706257	1
0.899537	-0.152953	-0.329882	-0.117790	-0.017248	...	-0.258863	-0.267290	0.123701	-0.296158	-0.034003	-0.878343	-0.145680	-0.016773	0.634099	0
0.899537	-0.152953	-0.329882	-0.117790	-0.275166	...	-0.139604	0.003651	0.123701	0.668267	-0.034002	0.157306	-0.048874	-0.112426	0.500063	0
-0.220401	-0.077164	-0.329882	-0.409139	1.014423	...	-0.219110	-0.316552	-0.293301	-0.296158	-0.034003	0.157306	-0.391416	-0.367499	-0.706257	0
-0.220401	-0.759260	-0.329882	-0.700488	-0.533083	...	-0.266813	-0.378129	-0.501802	0.668267	-0.034003	0.157306	-0.678109	-0.542861	-0.706257	1

برای آنکه متوجه شویم که در هر کلاستر چه ویژگی هایی تاثیر گذار بوده اند، با توجه به آنکه کارکرد pca با واریانس است، یعنی آن ویژگی هایی که واریانس بالاتری دارند، نشان دهنده آن است که اطلاعات بیشتری دارند. پس می توانیم از pca برای مشخص نمودن اینکه چه ویژگی اهمیت بیشتری داشته است، استفاده نماییم. برای این کار ابتدا label های هر کلاستر را جدا می کنیم. (منظور این است که داده های هر کلاستر را در یک دیتافریم مجزا قرار می دهیم) و روی هر کدام از آن ها pca اجرا می کنیم تا متوجه شویم با توجه به pca و انتخاب pca1 (component 1) کدام ویژگی اولویت بالاتری (واریانس بالاتری و به تبع آن اهمیت بیشتری) داشته است.

در این تحلیل از Cumulative explained variance استفاده می شود که واریانس توضیح داده شده تجمعی معیاری است که در تجزیه و تحلیل مؤلفه اصلی (pca) برای تعیین کمیت مقدار واریانس در مجموعه داده اصلی که توسط هر مؤلفه اصلی و با ترکیبی از مؤلفه های اصلی چندگانه حفظ می شود، استفاده می شود.

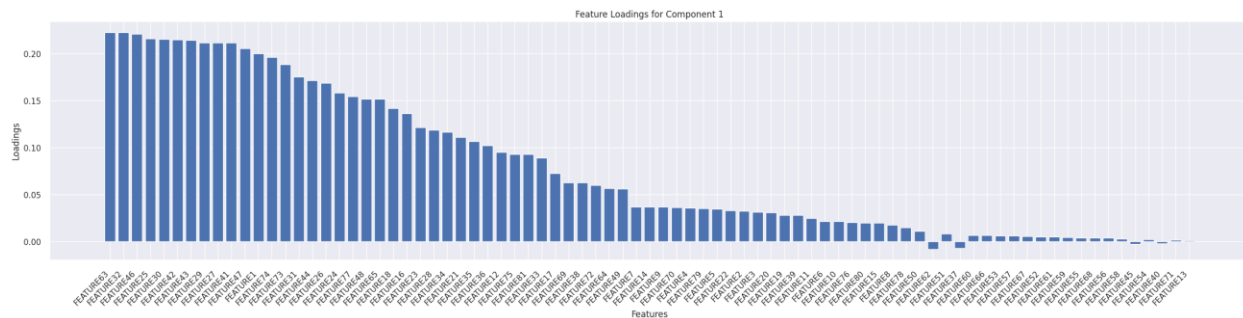
در آخر دو تا لیست خروجی می دهیم که ویژگی ها را رتبه بندی میکند. که در کد قرار دارد.

برای مثال:

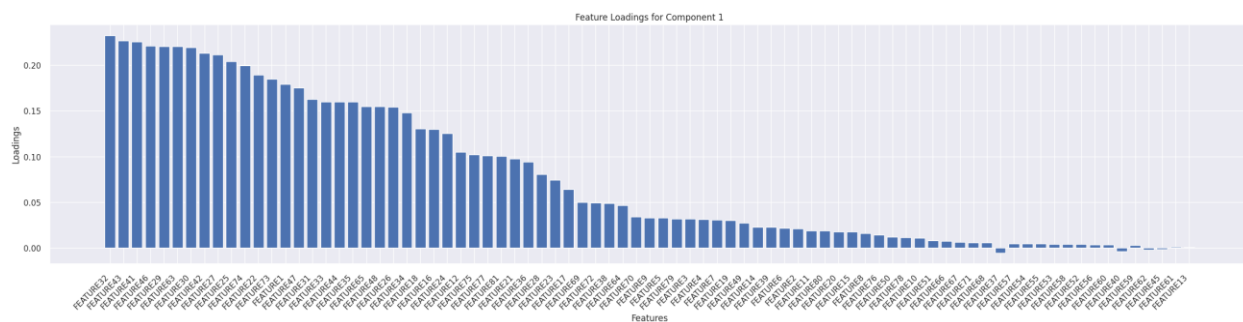
```
Component 1:
{'FEATURE63': 0.22302762454764535, 'FEATURE32': 0.22273705764716742, 'FEATURE46': 0.22128412842190062,
```

نمودارهای آن ها نیز به شرح زیر است:

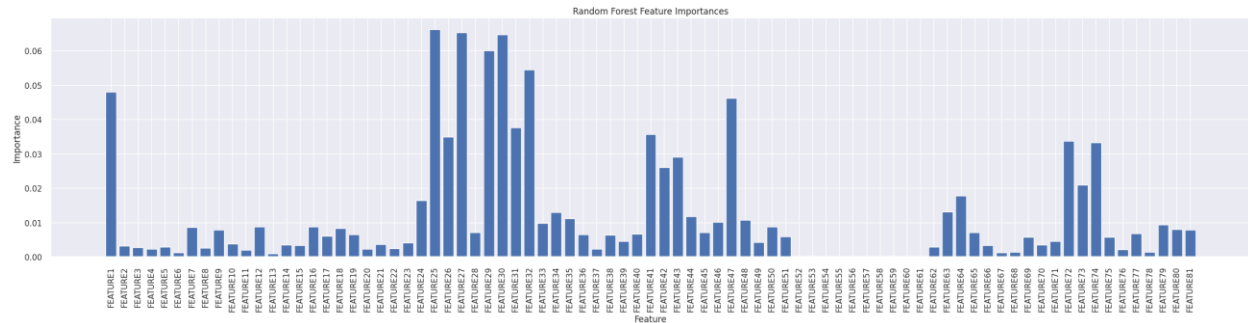
برای کلاستر 0 :



برای کلاستر ۱:



در قسمت بعدی برای تحلیل ویژگی ها از الگوریتم supervised مانند random forest استفاده نمودیم. در این روش ابتدا مانند روش قبل (pca) دیتاهای هر کلاستر را جدا نموده و سپس بر روی آن الگوریتم RF را اجرا نمودم و از ویژگی feature importance آن برای رتبه بندی فیچر ها استفاده نمودم. اما وزن هر feature را برابر با صفر نشان داد. مجددا کار را بر روی همه دیتاست و بدون جدا کردن لیبیل ها انجام دادم که نتیجه زیر را نشان داد:

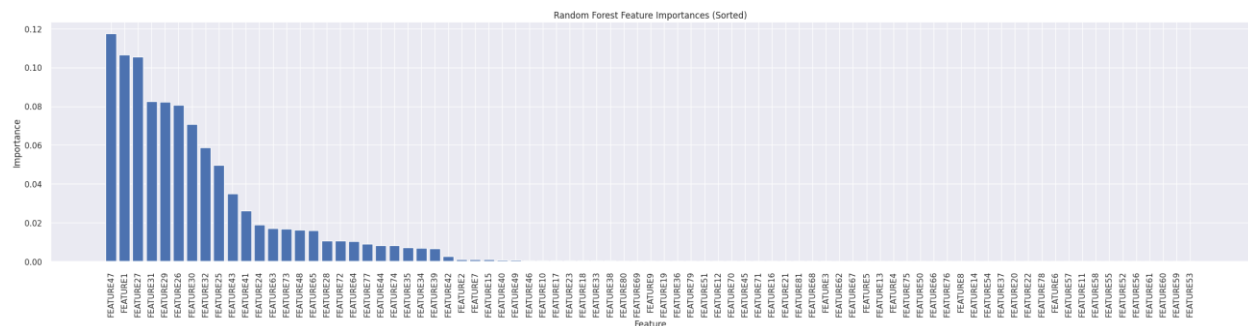


با توجه به هر دو تحلیل، مثلاً ویژگی 25, 27, 30 feature در هر دو نوع تحلیل اهمیت بالایی داشته است.

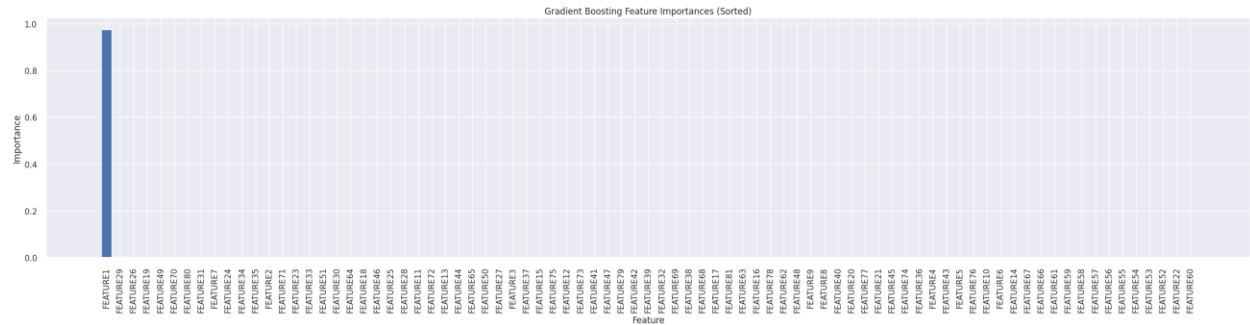
## Supervised:

در این قسمت با توجه به آنکه در قسمت های قبل میدانیم که دیتاست ما imbalance است، ۹۹۰ در برابر ۲۸۰۰۰۰. بنابراین در این نوع دیتاست undersampling بهتر است. ابتدا به صورت برابر از هر دو کلاس در undersampling خود قرار دادم. و با توجه به نتایج و دیتاست تصمیم گرفتم که دو برابر از داده های نوع 0 در sample خود قرار دهم که تا حد کمی به دیتاست اصلی شبیه شود. حتی cross validation نیز روی random forest انجام دادم ولی نتیجه زیاد تغییر نکرد بنابراین روی مدل های بعدی صرفاً بدون cross validation اجرا نمودم.

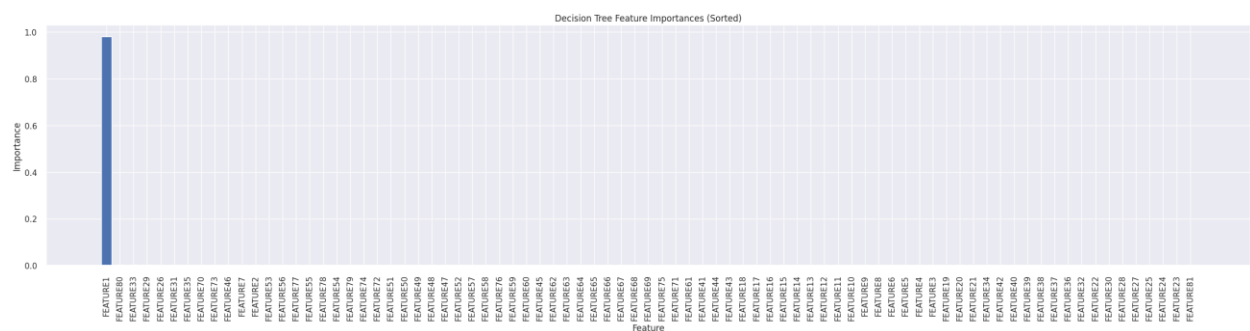
## Random forest:



## Gradient Boosting:



## Decision Tree:

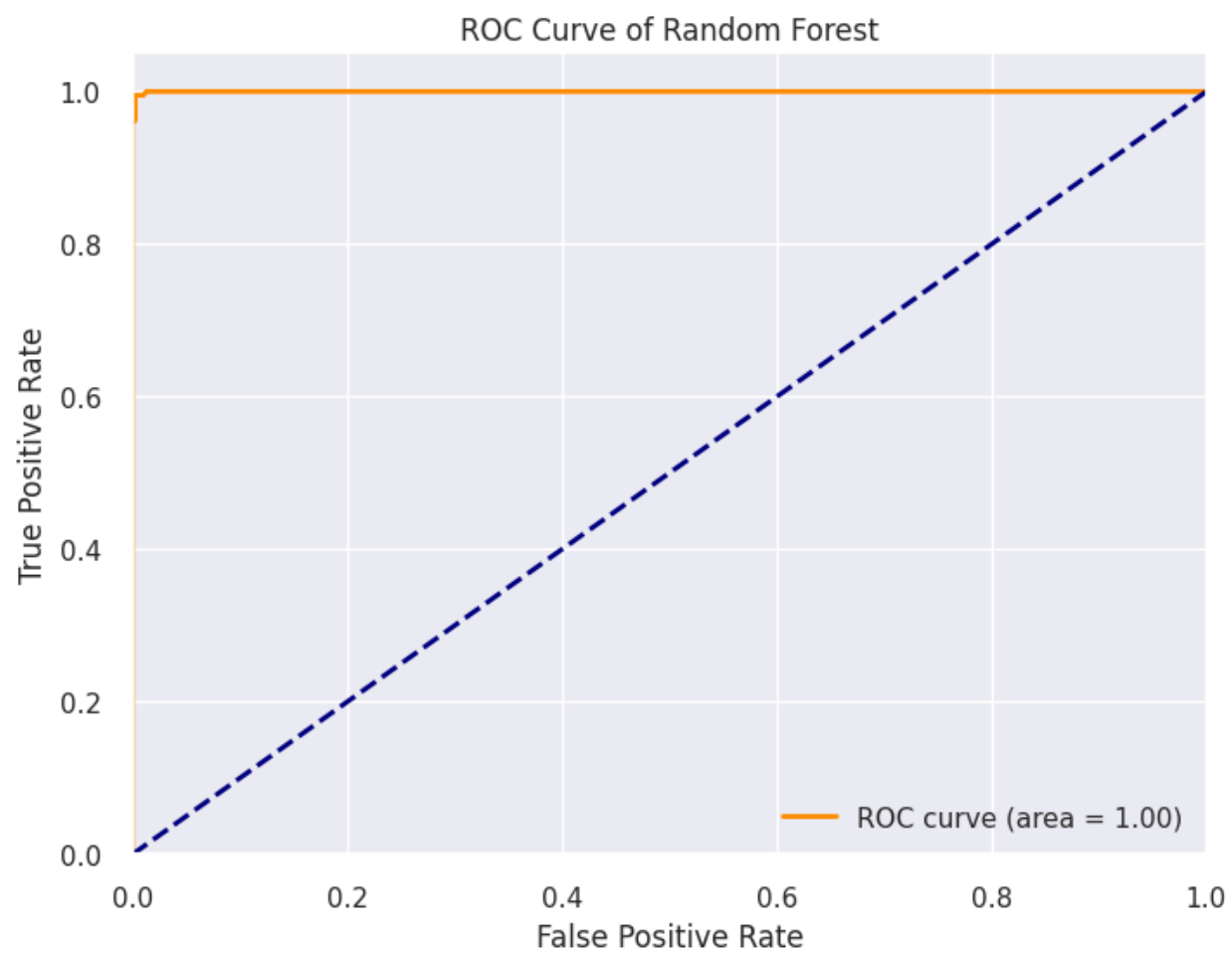


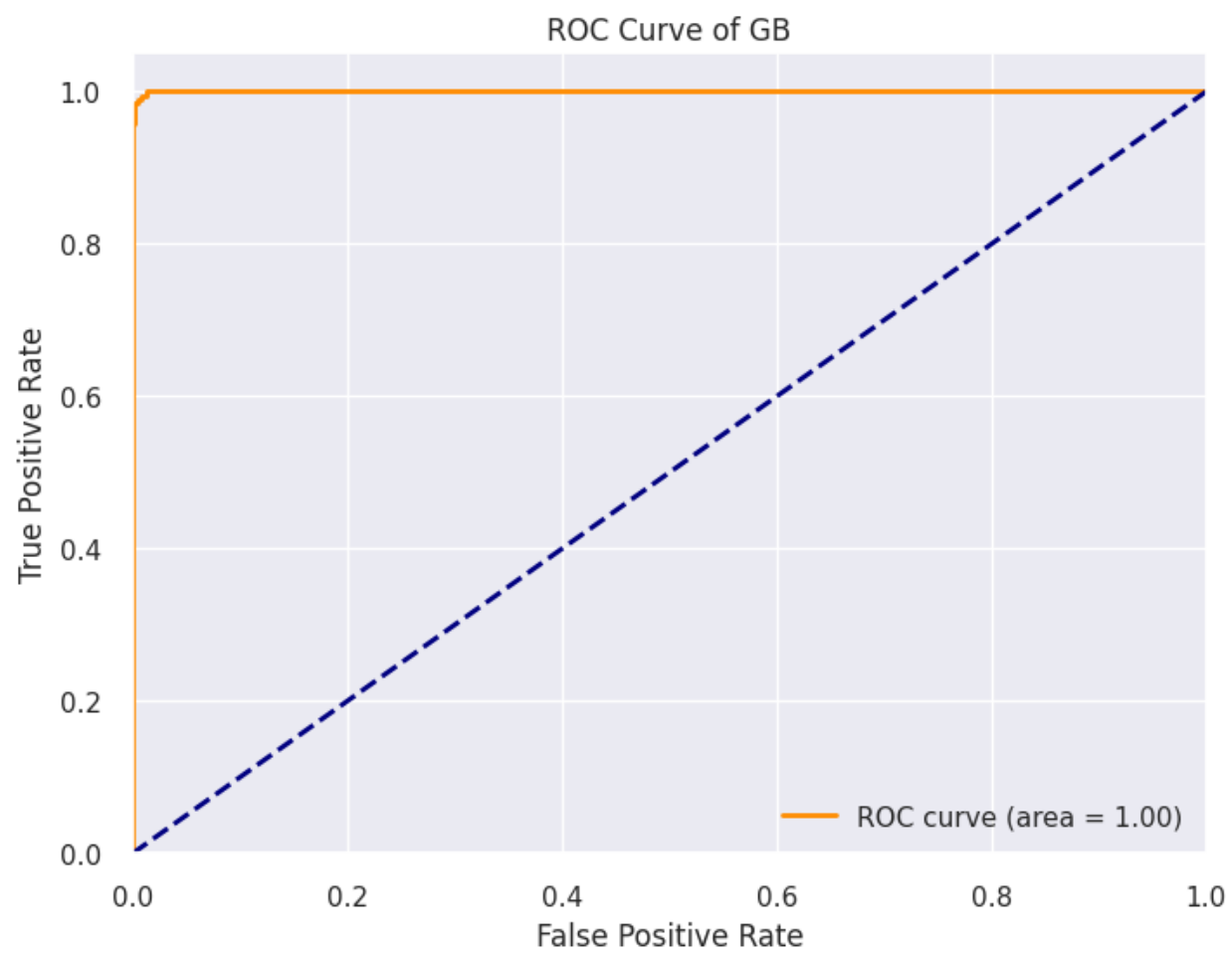
تحلیل:

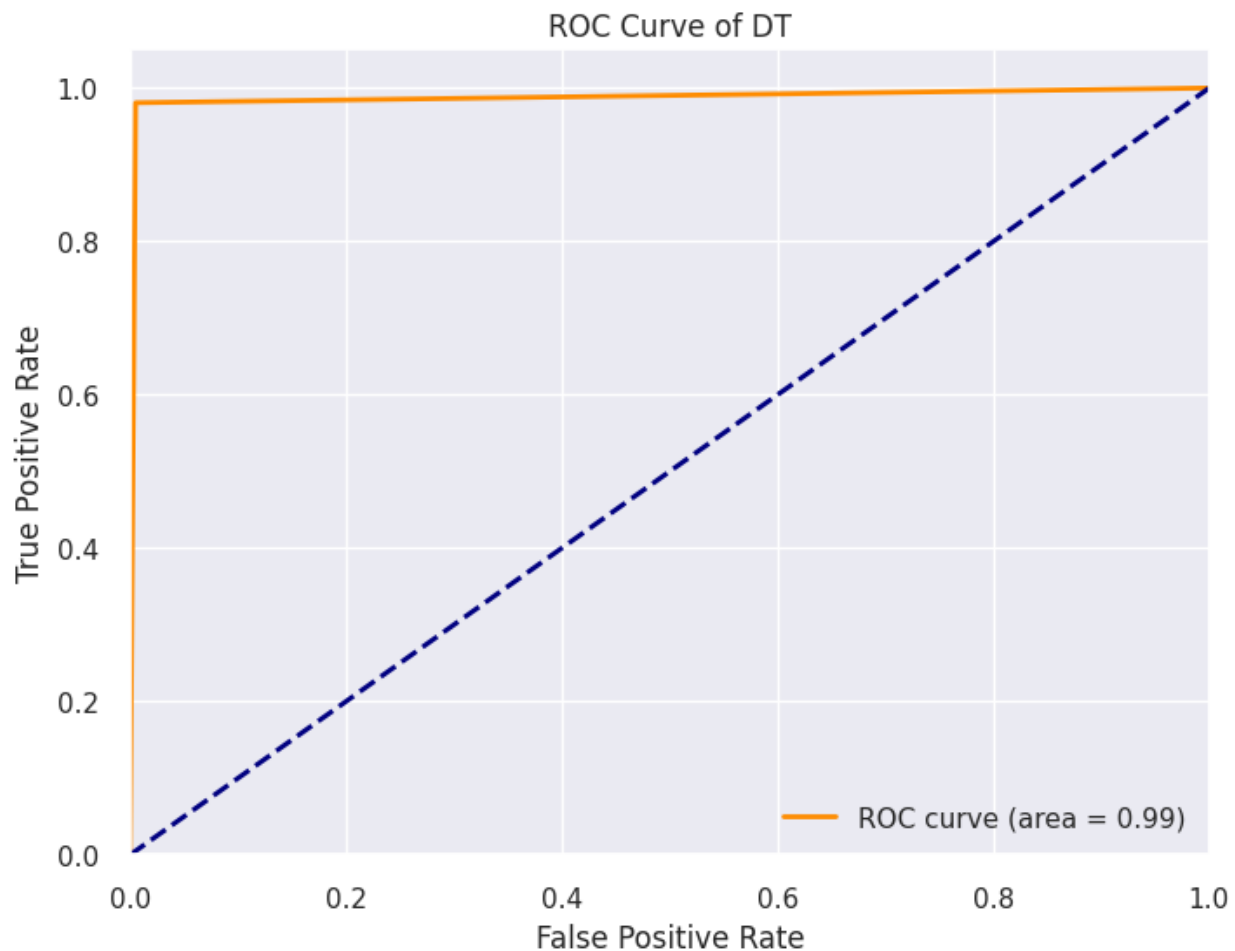
وزن هر کدام از فیچر ها در این classification متفاوت است. اما feature 1 در همه ضریب بالایی را داشته است پس نشان میدهد که feature مهمی است. و نقش بسزایی در clustering داشته است. هم چنین feature 1 در قسمت سوم از پروژه با feature 47 هم بستگی زیادی داشته است. و همان طور هم که در نمودار نشان داده شده در random forest نیز مشاهده میکنیم این دو تا در کنار هم و در اولویت بالایی قرار دارند. هم چنین برای مثال feature27 در خروجی random forest از اهمیت نسبتا بالایی برخوردار است و ما این ویژگی را نیز در قسمت ۴.۱ الف نیز مشاهده مینماییم که در کلاسترهای مختلف (مخصوصا ۴) نیز اهمیت نسبتا بالایی داشته است.

4. ارزیابی

Roc برای Random Forest:







هر سه الگوریتم از نظر ویژگی های ارزیابی خوب هستند اما با اختلاف کمی بنظر random forest بهتر است. چون دیتاست ما imbalance بود دقت بالای ۹۹ درصد میدهد. و overfit است. نمودار roc هم نشان دهنده این است که در هر دو الگوریتم این نتایج خوب است و چون بالای خط ۵۰ درصد یعنی تصادفی نیست. اما roc برای random forest و GB بهتر است (۱ در برابر ۰.۹۹).

A	B	C	D	E	F	G
	accuracy	Precision	Recall	F1-score	Support	RMSE
Random Forest	0.994949495	class 0: 0.9974 class 1: 0.9951	class 0: 0.9974 class 1: 0.9951	class 0: 0.9974 class 1: 0.9951	class 0: 386 class 1: 208	0.058
GradientBoosting	0.9915	class 0: 0.9922 class 1: 0.9903	class 0: 0.9948 class 1: 0.9855	class 0: 0.9935 class 1: 0.98795	class 0: 386 class 1: 208	0.0917
Decision Tree	0.989	class 0: 0.9896 class 1: 0.9902	class 0: 0.9948 class 1: 0.9807	class 0: 0.9922 class 1: 0.9855	class 0: 386 class 1: 208	0.1



با توجه به جدول بالا هر سه خوب هستند اما random forest قدرت خودش رو در rmse نشان میده که بهتر از بقیه عمل میکنه. چون مقادیر کمتر بهتر است. در بقیه معیارها نیز با اختلاف هر چند کم بهتر است.

8		silhouette	calinski_harabasz_score	davies_bouldin_score	distortion_score
9	k-means 4.a	0.57	1476.234	0.9275	407984.161
10	k-means 4.b		29115.91	1	19337199.6
11	chameleon	0.053	106.0698	3.692568	nan

امتیاز Silhouette: میزان جداسازی خوشه‌ها را اندازه‌گیری می‌کند. مقادیر نزدیک به ۱ نشان دهنده خوشه‌های به خوبی از هم جدا شده‌اند، در حالی که مقادیر نزدیک به ۰ نشان دهنده خوشه‌های همپوشانی هستند.

امتیاز Calinski-Harabasz: نسبت واریانس بین خوشه‌ای به واریانس درون خوشه‌ای را ارزیابی می‌کند. مقادیر بالاتر نشان دهنده خوشه‌های با تعریف بهتر است.

امتیاز Davies-Bouldin: میانگین شباهت بین هر خوشه و مشابه‌ترین خوشه را اندازه‌گیری می‌کند. مقادیر پایین‌تر نشان دهنده خوشه‌بندی بهتر است.

Distortion Score: مجموع فاصله‌های مجذور بین نقاط داده و خوشه‌های اختصاص داده شده را نشان می‌دهد. مقادیر پایین‌تر نشان دهنده خوشه‌های فشرده‌تر است.

تحلیل:

با توجه به این معیارها، به نظر می‌رسد k-means با ۵ خوشه (قسمت الف سوال ۴) در چندین معیار عملکرد نسبتاً خوبی دارد. تا بقیه موارد. در Distortion Score الگوریتم k-means عملکرد بهتری داشته است. مقایسه cham با بقیه جالب نیست چون روی کل دیتا این الگوریتم اجرا نشده است.

۷. نتیجه‌گیری:

الف:

بله وجود دارند. همانطور که در خلال گزارش توضیح داده ام، ویژگی ای مثل feature 1 در همه حالات تاثیر گذار بوده است. و هم چنین ویژگی هایی که با او هم correlate بالایی داشته است، یعنی feature 47 که در نزدیکی آن آمده است. این ویژگی در supervised اهمیت بسیار بالایی دارد در unsupervised نیز جزء ویژگی های پر اهمیت است. درست است که اولین نیست ولی اهمیت خودش را نشان میدهد و جز ۱۰ تا ویژگی اول است.

ب:

اگر از نظر میزان دقت و بقیه پارامترها در نظر بگیریم. چون که ما k-means را روی کل دیتاست قرار دادیم و بقیه معیارها را روی آن سنجیدیم، اما برای مثال classification ابتدا undersampling انجام دادیم شاید نتوانیم مقایسه درست و حسابی داشته باشیم. اما بنظرم از لحاظ وزن دهی الگوریتم های کلاسترینگ بیشتر به همه ویژگی ها اهمیت داده بودند اما supervised ها مانند DT و GB با اختلاف بسیاری به یکی از ویژگی ها بایاس شده بودند و به آن، اهمیت دادند. و در سه الگوریتمی که برای supervised بررسی نمودم، random forest رتبه بندی منطقی تری داشت.

از لحاظ متریک ها و معیارهای اندازه گیری، منطقا و با توجه به جداول میتوانیم بگوییم supervised خروجی بهتری را نشان داده است.

ج)

۱. با توجه به جداول و معیارها classification بهتر از clustering عمل کرده است. البته همانطور که اشاره شد، ما الگوریتم های classification را روی دیتاستی اجرا کردیم که undersampling بود و زیاد نمیتوان به نتایج اکتفا نمود. چون علی رغم اینکه cross validation و undersampling وزن دار را قرار دادم، اما مجددا نتایج تفاوت خاصی نکرد چون واقعا دیتاست imbalance بود و حجم یک لیبل که ۱ بود نسبت به دیگری بسیار بسیار کم بود. و احتمال overfit بالا میرود.

در مورد خود هر کدام از روش های clustering (چون تفاوت زیادی بین آن ها بود. در classification تفاوت بالا نبود و همه مثل هم بودند) برای الگوریتم cham نسبت به k-means مجددا روی دیتاست یکتایی اجرا نشده است. اما با توجه به پارامترهایی که به الگوریتم داده شد بنظرم برای این نوع دیتاست k-means جواب بهتری میدهد (با توجه به موردالف و مقدار silhouette )