CED18I039
Paleti Krishnasai

CED18I039 - ASBD LAB Endsem

Dear PALETI KRISHNASAI,
Question: For the Given dataset, apply apt data pre-processing techniques to clean the data for further processing. Exploit the concepts discussed in Descriptive Statistics that relate to the data set to gain key insights from the data. Adopt a through exploratory data analytics approach, relating the various concepts and plots discussed in the course / tested in the lab assignments to gain key insights from the given data set. On the Pre-processing and EDA front adopt an exhaustive approach relating the maximum no of techniques / features under each set. Over the cleaned data set, apply the following algorithms.
Algorithm 1: Apriori
Algorithm 2: K-NN Classification or Regression
Algorithm 3: k-medoids Clustering
Dataset Name: (1) Tennis ATP Data set
Dataset Link: https://www.kaggle.com/gmadevs/atp-matches-dataset
General Instruction: You shall apply necessary pre-processing techniques like discretization, binning etc to make the dataset suitable for applying FIM algorithm. You may also make any valid assumptions required for the entire exercise and state them explicitly in your documents submitted. Submit a complete report describing the techniques employed, code snippets and corresponding output as done for your lab submissions or share the corresponding notebook link with all data present in the file do mention the dataset name in your answer script.
: Based on the type of the assigned dataset, you shall either consider the entire set of features (or) subset of features to generate frequent patterns and apply predictive analytics.

CED18I039
Paleti Krishnasai

EDA:

Dataset Legend:

tourney_id - Id of Tournament
tourney_name - Name of the Tournament
surface - Surface of the Court (Hard, Clay, Grass)
draw_size - Number of people in the tournament
tourney_level - Tournament level (Grand Slam, Finals, Masters, Tour Series, Challenger
tourney_date - Start date of tournament
match_num
winner_id
winner_seed - Seed of winner
winner_entry - How the winner entered the tournament
winner_name - Name of winner
winner_hand - Dominant hand of winner
winner_ht - height in cm
winner_ioc - Country of winner
winner_age - Age of winner
winner_rank
winner_rank_points
loser_id
loser_seed
loser_entry
loser_name
loser_hand
loser_ht
loser_ioc
loser_age
loser_rank
loser_rank_points
score - Final score
best_of - Best of X number of sets
round - Round (Round of 16, Quaterfinal, etc.)
minutes - Match length in minutes
w_ace - Number of aces for winner

CED18I039
Paleti Krishnasai

w_df	- Number of double faults for winner
w_svpt	- Number of service points played by winner
w_1stIn	- Number of first serves in for winner
w_1stWon	- Number of first serve points won for winner
w_2ndWon	- Number of second serve points won for winner
w_SvGms	- Number of service games played by winner
w_bpSaved	- Number of break points saved by winner
w_bpFaced	- Number of break points faced by winner
l_ace	
l_df	
l_svpt	
l_1stIn	
l_1stWon	
l_2ndWon	
l_SvGms	
l_bpSaved	
l_bpFaced	

ace = absolute number of aces
df = number of double faults
svpt = total serve points
1stin = 1st serve in
1st won = points won on 1st serve
2ndwon = points won on 2nd serve
SvGms = serve games
bpSaved = break point saved
bpFaced = break point faced

As we have all the separate 20 years worth of data, we will have to load it in a single dataframe by using the “pd.concat” function which will combine all of these in 1

[3]:

```
ltennis = pd.concat([
    pd.read_csv('atp_matches_2000.csv', usecols=cols),
    pd.read_csv('atp_matches_2001.csv', usecols=cols),
    pd.read_csv('atp_matches_2002.csv', usecols=cols),
    pd.read_csv('atp_matches_2003.csv', usecols=cols),
    pd.read_csv('atp_matches_2004.csv', usecols=cols),
    pd.read_csv('atp_matches_2005.csv', usecols=cols),
    pd.read_csv('atp_matches_2006.csv', usecols=cols),
    pd.read_csv('atp_matches_2007.csv', usecols=cols),
    pd.read_csv('atp_matches_2008.csv', usecols=cols),
    pd.read_csv('atp_matches_2009.csv', usecols=cols),
    pd.read_csv('atp_matches_2010.csv', usecols=cols),
    pd.read_csv('atp_matches_2011.csv', usecols=cols),
    pd.read_csv('atp_matches_2012.csv', usecols=cols),
    pd.read_csv('atp_matches_2013.csv', usecols=cols),
    pd.read_csv('atp_matches_2014.csv', usecols=cols),
    pd.read_csv('atp_matches_2015.csv', usecols=cols),
    pd.read_csv('atp_matches_2016.csv', usecols=cols),
    pd.read_csv('atp_matches_2017.csv', usecols=cols),

],ignore_index=True) #have to make sure that the index will not be duplicated

ltennis
```

After concatenating all the datasets, we begin to study it.

shape of dataset:

53571 rows × 49 columns

Generate a profiling report with extensive EDA to finalize on what to explore in the dataset given.

ltennis_pp = pp.ProfileReport(ltennis)

ltennis_pp.to_file("ltennis_report.html")

Based on the profiling report, we can remove the columns with >50% missing values and the attributes which are deemed surplus or not necessary to the objective (KNN Regressor)

```
ltennis.drop(['winner_seed'],axis=1,inplace=True)
ltennis.drop(['winner_entry'],axis=1,inplace=True)
ltennis.drop(['winner_id'],axis=1,inplace=True)
ltennis.drop(['winner_name'],axis=1,inplace=True)
ltennis.drop(['winner_hand'],axis=1,inplace=True)
```

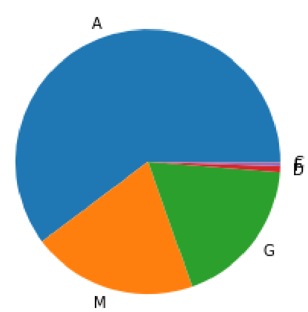
```
ltennis.drop(['winner_ht'],axis=1,inplace=True)
ltennis.drop(['winner_ioc'],axis=1,inplace=True)
ltennis.drop(['winner_age'],axis=1,inplace=True)
ltennis.drop(['winner_rank'],axis=1,inplace=True)
ltennis.drop(['winner_rank_points'],axis=1,inplace=True)


ltennis.drop(['loser_seed'],axis=1,inplace=True)
ltennis.drop(['loser_entry'],axis=1,inplace=True)
ltennis.drop(['loser_id'],axis=1,inplace=True)
ltennis.drop(['loser_name'],axis=1,inplace=True)
ltennis.drop(['loser_hand'],axis=1,inplace=True)
ltennis.drop(['loser_ht'],axis=1,inplace=True)
ltennis.drop(['loser_ioc'],axis=1,inplace=True)
ltennis.drop(['loser_age'],axis=1,inplace=True)
ltennis.drop(['loser_rank'],axis=1,inplace=True)
ltennis.drop(['loser_rank_points'],axis=1,inplace=True)


ltennis.drop(['tourney_id'],axis=1,inplace=True)
ltennis.drop(['tourney_date'],axis=1,inplace=True)
ltennis.drop(['tourney_name'],axis=1,inplace=True)
ltennis.drop(['match_num'],axis=1,inplace=True)
ltennis.drop(['score'],axis=1,inplace=True)
```

Tournament level:

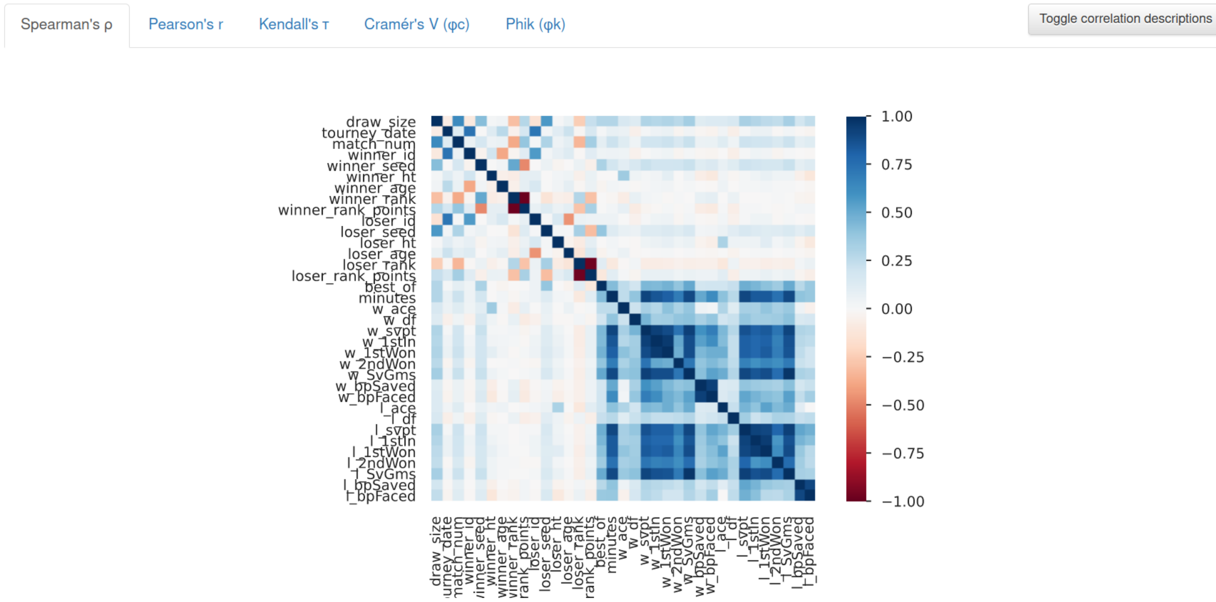
A	27977
M	9379
G	8480
D	351
F	215
C	30



(Reflected in the FIM generation)

CED18I039
Paleti Krishnasai

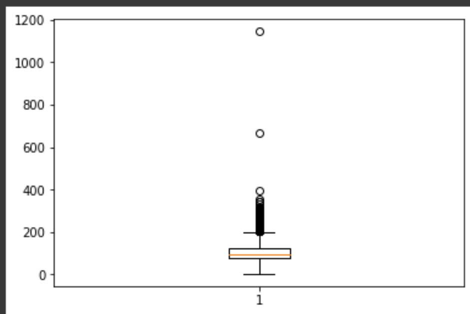
Analyzing Correlations:



As seen in the pic above, there are several measures of correlation to study the dataset from various angles.

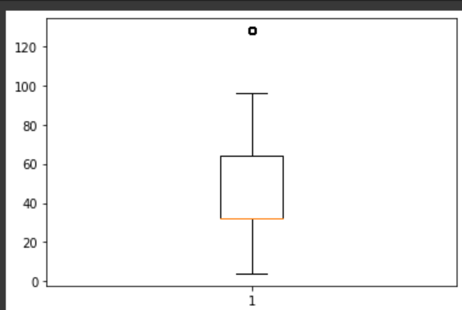
Here I have chosen “minutes” as my target feature in order to perform the regression. The attributes with less than ideal correlation are dropped from the dataset.

```
plt.boxplot(ltennis['minutes'].to_numpy())
plt.show()
```




```
# here we can see that there are outliers...but it is subect to statistical analysis as in real world matches such as
```

```
plt.boxplot(ltennis['draw_size'].to_numpy())
plt.show()
```



+ Code + Text

Here the minutes box plot shows many outliers, but they do have significant meaning as certain important matches have a high duration as well as higher number of stats (last 10 columns of the dataset).

(Detailed EDA will be uploaded as HTML file along with the report.)

CED18I039
Paleti Krishnasai

Check for missing values and fill/remove the missing value rows

`winner_seed` has 31266 (58.4%) missing values

`winner_entry` has 47361 (88.4%) missing values

`winner_ht` has 3570 (6.7%) missing values

`winner_rank` has 1139 (2.1%) missing values

`winner_rank_points` has 1139 (2.1%) missing values

`loser_seed` has 41607 (77.7%) missing values

`loser_entry` has 43144 (80.5%) missing values

Here the attributes above 50% missing values are removed from the dataset and the rows with null values are also removed as some of the attributes are of unique formatting and to convert them to numerical data is to manually go through each one by one (venue code, score : both these are very hard to encode due to their varying states and high cardinality). Another reason is that in real world matches, even the outliers have a greater meaning as only data doesn't tell the whole story (such as injuries, weather effects, extremely long rallies as to cover all these there needs to be an extremely comprehensive dataset encompassing all possible attributes which might come into play.

```
In [9]: ltennis.isnull().any()
```

```
Out[9]: surface          False
        draw_size        False
        tourney_level    False
        best_of          False
        round            False
        minutes          False
        w_ace            False
        w_df             False
        w_svpt           False
        w_1stIn          False
        w_1stWon         False
        w_2ndWon         False
        w_SvGms          False
        w_bpSaved        False
        w_bpFaced        False
        l_ace            False
        l_df             False
        l_svpt           False
        l_1stIn          False
        l_1stWon         False
        l_2ndWon         False
        l_SvGms          False
        l_bpSaved        False
        l_bpFaced        False
        dtype: bool
```

Encode Object data types:

```
In [10]: ltennis.shape
```

```
Out[10]: (46432, 24)
```

```
In [11]: # encoding objects
ltennis['surface'] = ltennis['surface'].astype('category')
ltennis['tourney_level'] = ltennis['tourney_level'].astype('category')
ltennis['round'] = ltennis['round'].astype('category')

ltennis['surface'] = ltennis['surface'].cat.codes
ltennis['tourney_level'] = ltennis['tourney_level'].cat.codes
ltennis['round'] = ltennis['round'].cat.codes
```

CED18I039
Paleti Krishnasai

Model Building: [KNN Regressor]

To predict the amount of time a match has been played when certain attributes are given.

```
In [13]: df_train_y = ltennis.iloc[:,5].values
         df_train_y

Out[13]: array([162.,  86.,  64., ..., 164.,  73.,  60.])

In [14]: ind=[]
         for i in range(24):
             if i != 5:
                 ind.append(i)

In [15]: df_train_x = ltennis.iloc[:,ind].values
         df_train_x

Out[15]: array([[ 1., 32.,  0., ..., 17.,  4.,  4.],
                [ 1., 32.,  0., ..., 10.,  4.,  9.],
                [ 1., 32.,  0., ...,  8.,  6., 10.],
                ...,
                [ 3.,  4.,  2., ..., 19.,  7., 11.],
                [ 3.,  4.,  2., ..., 10.,  8., 10.],
                [ 3.,  4.,  2., ...,  9.,  2.,  6.]])

In [16]: from sklearn.model_selection import train_test_split

In [17]: tennis_train_x, tennis_test_x, tennis_train_y, tennis_test_y = train_test_split(df_train_x, df_train_y, test_size = 0

In [18]: from sklearn.neighbors import KNeighborsRegressor

In [19]: neigh = KNeighborsRegressor(n_neighbors=3)

In [21]: neigh.fit(tennis_train_x, tennis_train_y)

Out[21]: KNeighborsRegressor(n_neighbors=3)

In [22]: pred_y=neigh.predict(tennis_test_x)

In [24]: pred_y

Out[24]: array([109.33333333,  90.        ,  93.        , ..., 154.33333333,
                165.        , 122.        ])

In [25]: from sklearn.metrics import r2_score,mean_squared_error
         r2_score(tennis_test_y,pred_y)

Out[25]: 0.8754717283979411
```

R2 score = 0.875

CED18I039
Paleti Krishnasai

Apriori FIM

```
#based on the profiling report, we can remove the columns with >50% missing values.
ltennis.drop(['winner_seed'],axis=1,inplace=True)
ltennis.drop(['winner_entry'],axis=1,inplace=True)


ltennis.drop(['loser_seed'],axis=1,inplace=True)
ltennis.drop(['loser_entry'],axis=1,inplace=True)
```

ltennis = ltennis.dropna()
#filling with statistical values will create biased data in this case as it is real world match data which can swing in either direction.
#dataset format gives an index scalar error without manually altering the dataset to convert certain objects to numerical data (like scores which cannot be sliced uniformly...we need to manually do 1 row at a time.)

In [8]:
```
ltennis=ltennis.applymap(str)
```

In [9]:
```
from mlxtend.frequent_patterns import apriori, association_rules
import csv
```

In [10]:
```
# Obtaining the individual transactions
l = ltennis.values.tolist()
```

In [11]:
```
#One Hot Encoding
from mlxtend.preprocessing import TransactionEncoder
ohe = TransactionEncoder()
ohe_rec = ohe.fit(l).transform(l)
df = pd.DataFrame(ohe_rec, columns=ohe.columns_)
df
#df[cat] = le.fit_transform(df[cat].astype(str))
```

Transforming data for the Frequent ItemSet mining.

42062 rows × 20460 columns

```
import time
#Applying apriori algo with min_suport = 0.2 and confidence 0.2
start = time.process_time()
freq_items=apriori(df, min_support=0.2, use_colnames=True)
time_taken=time.process_time() - start
print(f'Time taken for mining the dataset is {time_taken}')
freq_items
```

```
Time taken for mining the dataset is 29.403258499000003
```

	support	itemsets
0	0.413746	(0.0)
1	0.603205	(1.0)
2	0.488232	(10.0)
3	0.406590	(11.0)
4	0.343445	(12.0)
...	...	...
643	0.212472	(A, 3.0, 8.0, 32.0, R)
644	0.224454	(9.0, A, 3.0, 32.0, R)
645	0.202011	(A, Hard, 3.0, 32.0, R)
646	0.201203	(R32, A, 3.0, 32.0, R)
647	0.207265	(A, 4.0, 3.0, 5.0, R)

648 rows × 2 columns

#Forming association rules using FIS

start = time.process_time()
rules = association_rules(freq_items, metric="confidence", min_threshold=0.2)
time_taken=time.process_time() - start
print(f'Time taken for forming the association rules with the above frequent itemset is {time_taken}')
rules

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(0.0)	(1.0)	0.413746	0.603205	0.278399	0.672872	1.115496	0.028825	1.212968
1	(1.0)	(0.0)	0.603205	0.413746	0.278399	0.461532	1.115496	0.028825	1.088744
2	(10.0)	(0.0)	0.488232	0.413746	0.207218	0.424425	1.025811	0.005214	1.018554
3	(0.0)	(10.0)	0.413746	0.488232	0.207218	0.500833	1.025811	0.005214	1.025245
4	(0.0)	(2.0)	0.413746	0.659574	0.281703	0.680860	1.032272	0.008807	1.066697
...	...	...	...	...	...	...	...	...	...
4829	(A)	(4.0, R, 5.0, 3.0)	0.593481	0.338691	0.207265	0.349237	1.031139	0.006259	1.016206
4830	(4.0)	(A, 5.0, R, 3.0)	0.620964	0.338572	0.207265	0.333780	0.985848	-0.002975	0.992808
4831	(3.0)	(A, 4.0, R, 5.0)	0.916932	0.207384	0.207265	0.226042	1.089968	0.017108	1.024107
4832	(5.0)	(A, 4.0, R, 3.0)	0.659574	0.373259	0.207265	0.314241	0.841887	-0.038926	0.913939
4833	(R)	(A, 4.0, 5.0, 3.0)	0.984761	0.210332	0.207265	0.210473	1.000668	0.000138	1.000178

4834 rows × 9 columns

Now testing with different support and confidence values

In [17]:
```
#Applying apriori algo with min_suport = 0.5 and confidence 0.13
start = time.process_time()
freq_items=apriori(df, min_support=0.5, use_colnames=True)
time_taken=time.process_time() - start
print(f'Time taken for mining the dataset is {time_taken}')
freq_items
```

```
Time taken for mining the dataset is 22.35745227000001
```

Out[17]:

	support	itemsets
0	0.603205	(1.0)
1	0.659574	(2.0)
2	0.916932	(3.0)
3	0.620964	(4.0)
4	0.659574	(5.0)
5	0.526223	(6.0)
6	0.514360	(9.0)
7	0.593481	(A)
8	0.525177	(Hard)
9	0.984761	(R)
10	0.569730	(1.0, 3.0)

18]:
```python
#Forming association rules using FIS
start = time.process_time()
rules = association_rules(freq_items, metric="confidence", min_threshold=0.13)
time_taken=time.process_time() - start
print(f'Time taken for forming the association rules with the above frequent itemset is {time_taken}')
rules
```

```
Time taken for forming the association rules with the above frequent itemset is 0.009409328000003825
```

18]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(1.0)	(3.0)	0.603205	0.916932	0.569730	0.944506	1.030072	0.016633	1.496873
1	(3.0)	(1.0)	0.916932	0.603205	0.569730	0.621344	1.030072	0.016633	1.047905
2	(1.0)	(R)	0.603205	0.984761	0.594955	0.986324	1.001587	0.000943	1.114277
3	(R)	(1.0)	0.984761	0.603205	0.594955	0.604162	1.001587	0.000943	1.002419
4	(2.0)	(3.0)	0.659574	0.916932	0.617921	0.936849	1.021721	0.013137	1.315383
5	(3.0)	(2.0)	0.916932	0.659574	0.617921	0.673901	1.021721	0.013137	1.043933
6	(R)	(2.0)	0.984761	0.659574	0.649161	0.659207	0.999443	-0.000362	0.998922
7	(2.0)	(R)	0.659574	0.984761	0.649161	0.984212	0.999443	-0.000362	0.965267
8	(4.0)	(3.0)	0.620964	0.916932	0.573463	0.923504	1.007167	0.004081	1.085911
9	(3.0)	(4.0)	0.916932	0.620964	0.573463	0.625415	1.007167	0.004081	1.011881
10	(5.0)	(3.0)	0.659574	0.916932	0.576506	0.874058	0.953242	-0.028278	0.659574
11	(3.0)	(5.0)	0.916932	0.659574	0.576506	0.628734	0.953242	0.028278	0.916932

k-medoids Clustering

This is a variant of K-means where the medoid vector is taken instead of the mean vector.

The basic process is the same with only “tourney_name” included apart from the attributes used in the KNN dataframe.

This is done to increase the chances of clustering as its an unsupervised algorithm and its experimental to find the optimal value of K.

Only 25000 values are taken as the system was not able to support more and kept crashing.(including google collab)

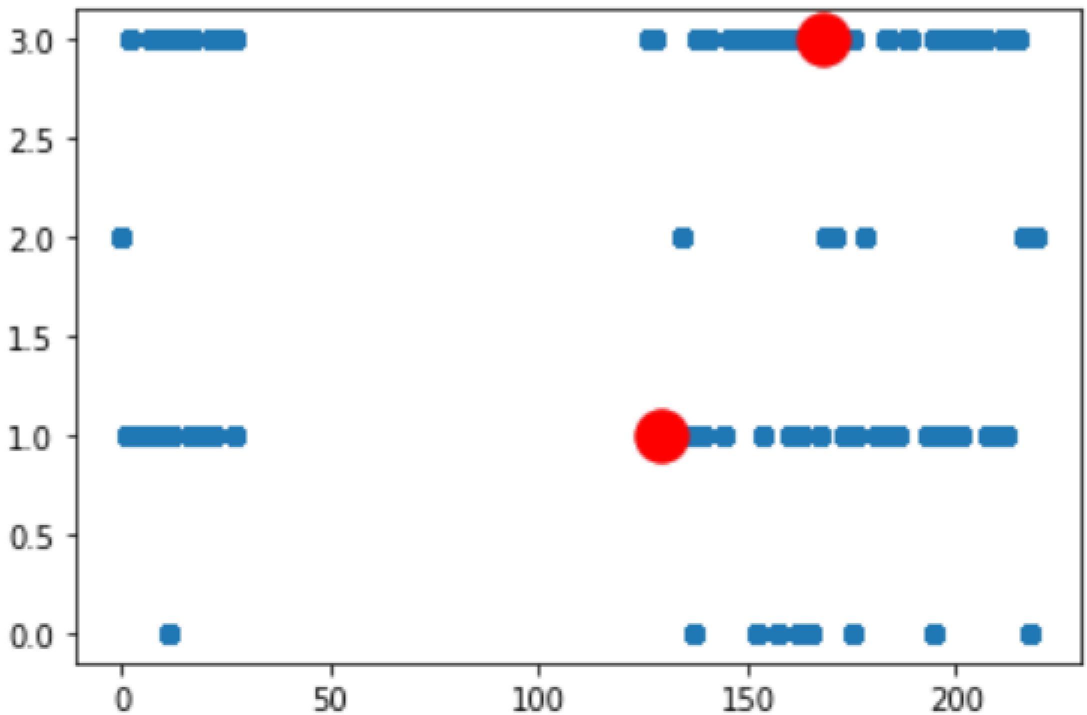
```
In [16]: new_data = ltennis.iloc[:25000,:].values
         # new_data = ltennis
```

```
In [13]: # !pip install scikit-learn-extra
```

```
In [14]: from sklearn_extra.cluster import KMedoids
```

```
In [17]: X = new_data
         kmedoids = KMedoids(n_clusters=2, random_state=0).fit(X)
         Kmed_pred=kmedoids.fit_predict(X)

         plt.scatter(X[:,0], X[:,1])
         plt.scatter(kmedoids.cluster_centers_[:, 0], kmedoids.cluster_centers_[:, 1], s=300, c='red')
         plt.show()
```



<u>Other cluster values have been tested, but the system kept crashing due to high dataset load. (will upload the code file separately.</u>

[Please find the attached HTML file for extensive EDA analysis]