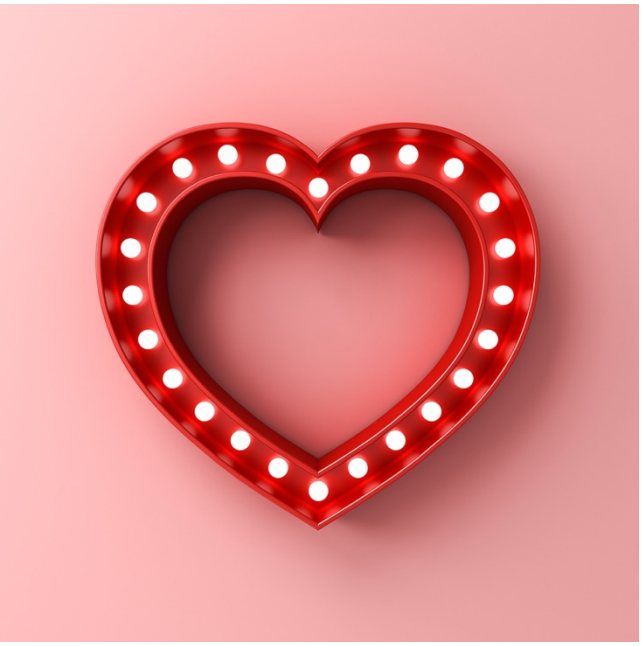
Project Report Group 3B

Speed Dating Matching Prediction with Machine Learning

University of California Irvine, MSBA, December 2022



Team Members:

Urvi Vaidya (vaidyau@uci.edu)

Nishant Parate (nparate@uci.edu)

Sem Leontev (sleontev@uci.edu)

Soo Lee (sool10@uci.edu)

Dandan Zhou (dandanz1@uci.edu)

Table of Contents

1.1 Executive Summary 3
1.2 Introduction of Business Idea, Importance, Data Source 3
2.1 Exploratory Data Analysis 4
2.2 Initial Data Visualization 4
2.3 Initial Data Preparation and Data Description 6
2.3 Principal Component Analysis & Modeling Results 8
3.1 Model Selection and Fine-Tuning Process 9
3.2 Logistic Regression Results 10
3.3 Decision Trees 11
3.4 Naïve Bayes 12
3.5 KNN ALGORITHM 12
4.1 Take Aways 13
5.1 Analysis Results and Recommendations 14
Appendix 14

1.1 Executive Summary

Matching Results Prediction in this Speed Dating project involves 84 different attributes and 6816 individual instances. The goal of this project is to predict the matching result of speed dating as "Yes" or "Not". The process of building this prediction model consists of exploratory data analysis which help us to understand the data set; data visualization which shows compared key instances and correlation among attributes; data preparation which allow us to drop missing values , refill the null values, and deal with outliers; principal components analysis which is one way to build a model with the most important features; the main building of models using different methods and datasets and fine-tuning it to find the best solution; and testing the model which helps us to optimize the model we choose.

Eventually, the result shows that Logistic Regression is the best model in this case with a 91% accuracy. We used Logistic Regression, Decision Tree Classifier, Naïve Bayes Classifier, and K Nearest Neighbors as our model options. The main focus of this project is on Logistic regression and Decision Trees, we have also experimented with naïve Bayes and KNN in order to see what the results would be. Meanwhile, we tested the models by using the initial not cleaned data (benchmark results), cleaned processed data, resampled initial data and data subset using extra trees classifier as a feature selection technique. The prediction accuracy ranged from 86% to 91% for Logistic Regression model based on the data subset using extra trees classifier.

1.2 Introduction of Business Idea, Importance, Data Source

Speed Dating Matching is a classic utilization of the models for prediction of categorical outcomes, which is a very important area of machine learning that has applications across fields. The Data set is provided by Columbia Business School professor Ray Fisman and Sheena Iyengar for their research about gender differences in mate selection evidence from a speed dating experiment. As part of their study the subjects were asked to fill a survey when they signed up for the Speed Dating Event and fill a scorecard on various attributes on each of their potential matches. At the end of the night subjects were matched with potential partners if both persons said yes. This experiment was conducted on various days with different groups of people and collated into one dataset.

We find this as an interesting topic to research since there are already serval apps that show potential matches based on individual needs, likes, and dislikes. While Speed Dating Matching is an interesting topic to research this project will help us learn how would the models work in real cases. It will also give us insight into human behaviour and understand what factors are important to people when it comes to choosing potential mates.

2.1 Exploratory Data Analysis

Exploratory Data Analysis primarily helps to set up an initial data table for the overall data to have an overview of the dataset. In this step, we will also perform data cleaning, so that we can have a workable dataset by checking the null values. We will also use different plots to visualize the data, which will help us to have some comparative initial key features interpretation based on the exploratory data analysis.

Excerpt of dataset table:

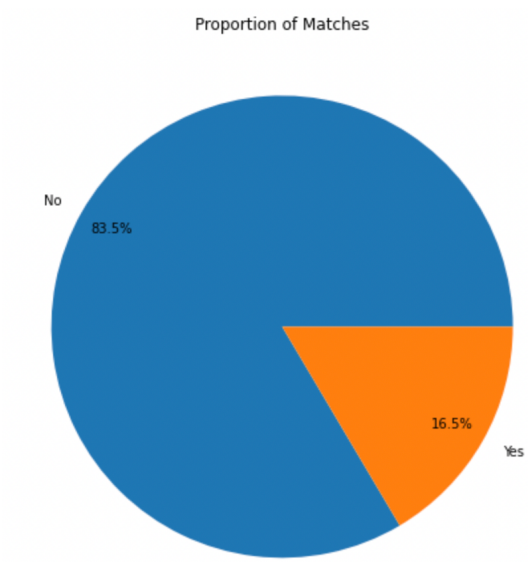
	match	gender	round	int_corr	samerace	age_o	pf_o_att	pf_o_sin	pf_o_int	pf_o_fun	pf_o_amb	pf_o_sha	dec_o	attr_o	sinc_o	intel_o	fun_o	amb_o
0	0	0	10	0.14	0	27.0	35.0	20.0	20.0	20.0	0.0	5.0	0	6.0	8.0	8.0	8.0	8.0
1	0	0	10	0.54	0	22.0	60.0	0.0	0.0	40.0	0.0	0.0	0	7.0	8.0	10.0	7.0	7.0
2	1	0	10	0.16	1	22.0	19.0	18.0	19.0	18.0	14.0	12.0	1	10.0	10.0	10.0	10.0	10.0
3	1	0	10	0.61	0	23.0	30.0	5.0	15.0	40.0	5.0	5.0	1	7.0	8.0	9.0	8.0	9.0
4	1	0	10	0.21	0	24.0	30.0	10.0	20.0	10.0	10.0	20.0	1	8.0	7.0	9.0	6.0	9.0

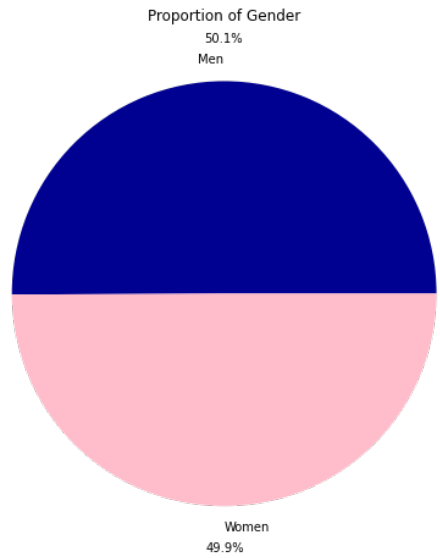
Exploratory data analysis created the initial table of data and provides the shape and number of datapoints, attributes, datatypes, and general description. Also, it helps with understanding how the variables interact with each other. It represents the first sight of the predictive capability of the variables and gives some insight about what algorithm can be more effective or appliable to this data set. There are 83 attributes and 1 target dependent variable for “match”. Thus, to predict whether the target dependent variable for “match”, we would have to check on the 83 listed attributes to find the most applicable models and tools. The table of speed dating matching results consists of 5,692 instances as “Not” and 1,124 instants results as “Yes”. Based on the existing matching results. We used visualization tools to compare the variables.

2.2 Initial Data Visualization

Visualized data charts are used to display the comparative key variables and can assist with initially interpreting the whole data sets. The first comparative variables tell us how many users successfully matched in this data set We set up one chart to find the proportion of matches that shows the results of existing “YES” or “No” matching results, while another chart shows the proportion of gender, the distribution of “Men” and “Women” among users. The first pie chart for the proportion of matching results gives researchers an overview of the numerical results for the proportion of matches results as “No” is 83.5%, while the answer for “Yes” is 16.5%. The second pie chart displays the proportion of users' gender for “Men” is 50.1% and for “Women” is 49.9%. Thus, this two pie charts' information combined can tell the result of matching in this data set that as men and women users are evenly distributed, the proportion of successfully matched instances is 16.5%.

Plots showing distribution:





Excerpt of correlation matrix:

Correlation Matrix

```
# Visualising the correlation matrix of the dataset.
corr = data.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	match	gender	round	int_corr	samerace	age_o	pf_o_att	pf_o_sin	pf_o_int	pf_o_fun	pf_o_amb	pf_o_sha	attr_o	sin
match	1.000000	-0.000652	-0.034069	0.033347	0.031892	-0.050860	0.016335	-0.034916	0.013011	0.047558	-0.012093	-0.047547	0.263340	0.163
gender	-0.000652	1.000000	0.024890	-0.001173	-0.001174	-0.069528	-0.383323	0.139373	0.134074	-0.046741	0.368028	0.150583	-0.139561	-0.061
round	-0.034069	0.024890	1.000000	0.022819	0.022935	-0.000200	-0.011370	0.055745	0.020862	-0.025532	-0.076370	-0.004830	-0.058595	-0.102
int_corr	0.033347	-0.001173	0.022819	1.000000	0.031667	0.089068	-0.096543	0.033675	0.087313	-0.010230	0.028258	0.032804	0.014636	0.010
samerace	0.031892	-0.001174	0.022935	0.031667	1.000000	0.005798	0.027053	-0.053328	0.021675	0.034161	-0.036442	-0.017872	0.054717	0.008
age_o	-0.050860	-0.069528	-0.000200	0.089068	0.005798	1.000000	0.043214	0.013538	0.043028	-0.106295	-0.080968	0.017055	0.031732	0.011
pf_o_att	0.016335	-0.383323	-0.011370	-0.096543	0.027053	0.043214	1.000000	-0.453503	-0.391015	-0.188001	-0.442790	-0.431124	0.038364	0.022
pf_o_sin	-0.034916	0.139373	0.055745	0.033675	-0.053328	0.013538	-0.453503	1.000000	-0.113372	-0.175884	0.018239	0.051986	0.006433	0.001
pf_o_int	0.013011	0.134074	0.020862	0.087313	0.021675	0.043028	-0.391015	-0.113372	1.000000	-0.145512	0.005143	-0.063549	-0.039815	-0.062
pf_o_fun	0.047558	-0.046741	-0.025532	-0.010230	0.034161	-0.106295	-0.188001	-0.175884	-0.145512	1.000000	-0.067439	-0.190938	-0.011196	0.008
pf_o_amb	-0.012093	0.368028	-0.076370	0.028258	-0.036442	-0.080968	-0.442790	0.018239	0.005143	-0.067439	1.000000	0.093683	-0.027334	0.033
pf_o_sha	-0.047547	0.150583	-0.004830	0.032804	-0.017872	0.017055	-0.431124	0.051986	-0.063549	-0.190938	0.093683	1.000000	-0.024278	-0.038
attr_o	0.263340	-0.139561	-0.058595	0.014636	0.054717	0.031732	0.038364	0.006433	-0.039815	-0.011196	-0.027334	-0.024278	1.000000	0.406
sinc_o	0.163802	-0.061666	-0.102779	0.010465	0.008616	0.011889	0.022149	0.001714	-0.062265	0.008116	0.033651	-0.038582	0.406243	1.000
intel_o	0.174907	0.031885	-0.093837	0.057577	0.008093	0.003458	0.004728	0.032547	-0.050237	-0.019185	0.062748	-0.053344	0.396611	0.651
fun_o	0.282160	-0.078059	-0.075235	0.015259	0.023773	0.003898	0.049657	0.007833	-0.072163	-0.017754	0.009978	-0.036848	0.583165	0.500
amb_o	0.135676	0.072588	-0.090093	0.032908	0.025649	-0.033049	-0.002634	0.026646	-0.043771	-0.052404	0.117006	-0.053821	0.356114	0.444
shar_o	0.273626	-0.052247	-0.056069	0.043244	0.071967	0.022409	0.038762	0.015545	-0.056604	-0.049603	0.009807	-0.010849	0.485761	0.401
like_o	0.309815	-0.090672	-0.078489	0.032527	0.042341	0.024624	0.039492	0.020860	-0.049203	-0.016041	-0.020299	-0.024828	0.666735	0.522
age	-0.049855	0.067321	0.007971	0.090800	0.007440	0.135742	-0.072150	0.030115	-0.006275	-0.030995	0.071806	0.055439	-0.056927	-0.004

The second initial data visualization table is Correlation Matrix. The correlation matrix of the dataset gives a table of correlation scores for each pair of attributes. We can see that none of the attributes are highly correlated to our target “Match” variable.

Excerpt from our data.describe():

```
data.describe()
```

	match	gender	round	int_corr	samerace	age_o	pf_o_att	pf_o_sin	pf_o_int	pf_o_fun	pf_o_amb	pf_o_s
count	6816.000000	6816.000000	6816.000000	6668.000000	6816.000000	6717.000000	6732.000000	6732.000000	6732.000000	6723.000000	6714.000000	6692.0000
mean	0.164906	0.500734	16.859155	0.195789	0.390258	26.289117	23.961000	17.298555	20.566772	17.355355	9.746534	11.2548
std	0.371123	0.500036	4.441200	0.304068	0.487844	3.582694	13.425856	7.700079	7.434063	6.646454	6.124239	6.6656
min	0.000000	0.000000	6.000000	-0.730000	0.000000	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000	14.000000	-0.020000	0.000000	24.000000	15.000000	10.000000	18.000000	14.000000	5.000000	5.0000
50%	0.000000	1.000000	18.000000	0.210000	0.000000	26.000000	20.000000	20.000000	20.000000	18.000000	10.000000	10.0000
75%	0.000000	1.000000	21.000000	0.430000	1.000000	28.000000	30.000000	20.000000	25.000000	20.000000	15.000000	15.0000
max	1.000000	1.000000	22.000000	0.910000	1.000000	55.000000	100.000000	60.000000	50.000000	50.000000	53.000000	30.0000

Based on the initial results, we can find there are outliers as the table has been displayed. An outlier does nothing but differs significantly from other data points in the given datasets. Outliers represent errors in measurement, bad data collection, or simply show variables not considered when collecting the data. For example, we can see from the above data that for several columns the mean value is very far from the min/max values which means there are significant outliers in the dataset. In supervised models, outliers can deceive the training process resulting in prolonged training times or leading to the development of less precise models.

We also visualized additional attributes that can be found in the attached file. These helped us gain some knowledge about the behavior of the participants and gave us information that we could analyze to gain insights for fine-tuning our models.

2.3 Initial Data Preparation and Data Description

We use data preparation to reduce the errors of missing values and outliers which would lead to a failure of the prediction. This step also consists of data cleaning, so that we can have a workable dataset.

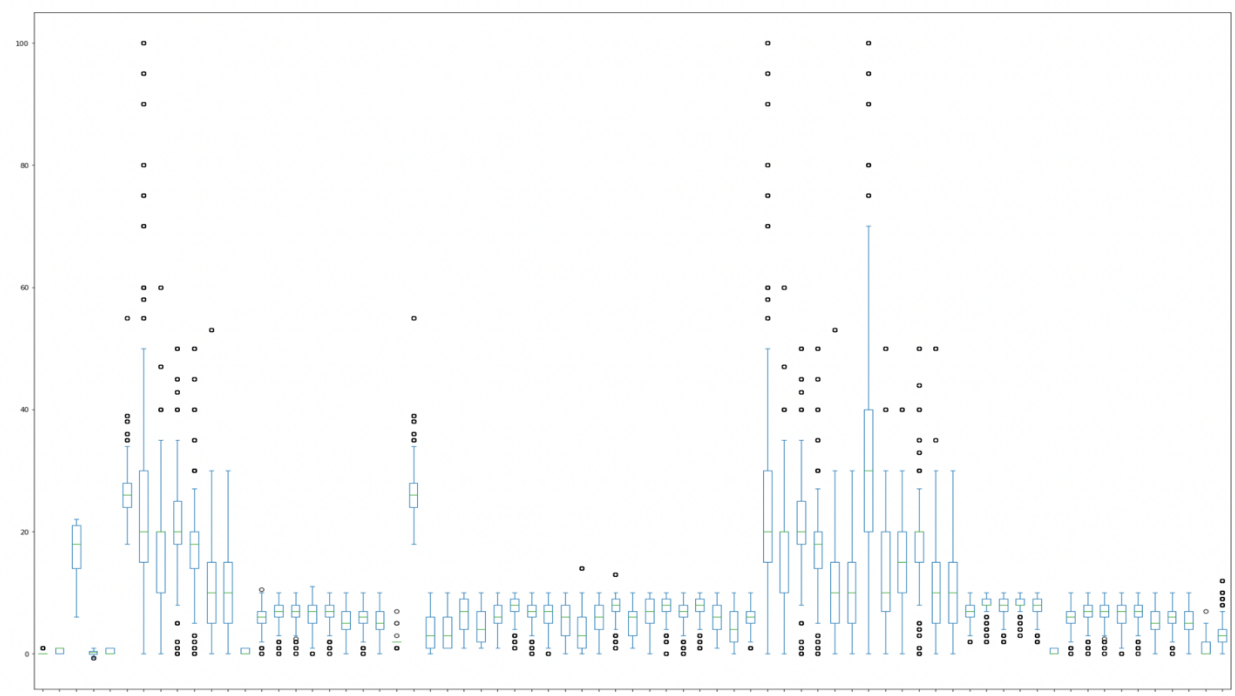
The first step involves checking and replacing null values. We first checked for null values and replaced them with the mean value of their column or just dropped them depending on the proportion of missing values in the columns. We started with looking at the proportion of missing values and we found that there were several columns with a large proportion of missing values which will make the data biased or skewed. Therefore, we set up the table and determine the correlation with the default target variable to see if these columns are highly correlated to the target columns. We saw that none of those columns with high missing values are individually highly correlated to our match column. As a result, we decided to drop columns with more than 20% missing values which are least correlated to the target column “Match”.

We also need to look at the object types to determine how to deal with them. There were originally 7 object type columns, of which 3 had a large number of unique values and too many categories in the columns may lead to overfitting of the data. We may perhaps at a later stage decide to do a deeper cleaning of the data and bin/categorize them in a certain way, but for now we do not have

the necessary knowledge or information about them, so for the purpose of our project we will drop them from our working data. The datatype selection and null value checking functions give us an overall result that this dataset includes 78 columns with null values and 7 object type columns. Based on the proportion of null values we decided to retain or replace those values for a better use. The next step is to replace all null values with the mean values of their columns.

To perform the ranking for category values and determine the efficiency of categorical data, we will be using one hot encoding. Because we cannot rank some types of categories as either better or worse than the others, therefore, assigning integers to them without knowing if those categories are effectively related to the prediction would not be correct. Although some machine learning algorithms can work directly with categorical data depending on implementation, such as a decision tree, most require any inputs or outputs variables to be a number, or numeric in value. This means that any categorical data must be mapped to integers. One hot encoding is useful for data that has no relationship to each other. Machine Learning algorithms treat the order of numbers as an attribute of significance. In other words, they will read a higher number as better or more important than a lower number. While this is helpful for some ordinal situations, some input data does not have any ranking for category values, and this can lead to issues with predictions and poor performance. That's when one hot encoding saves the data.

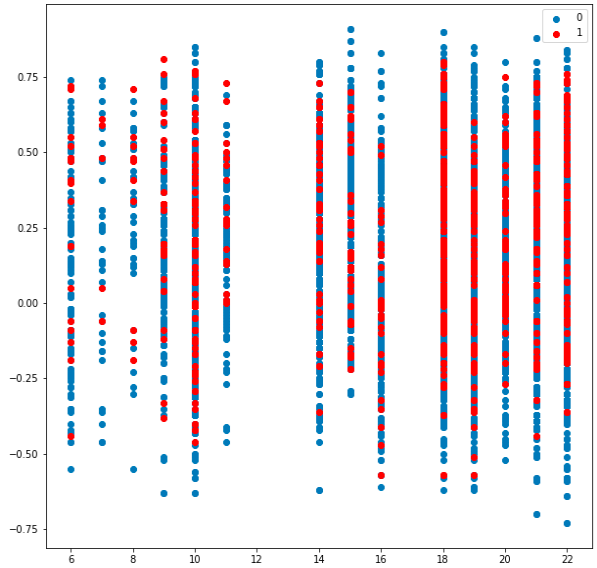
Plot visualizing outliers in our cleaned data:

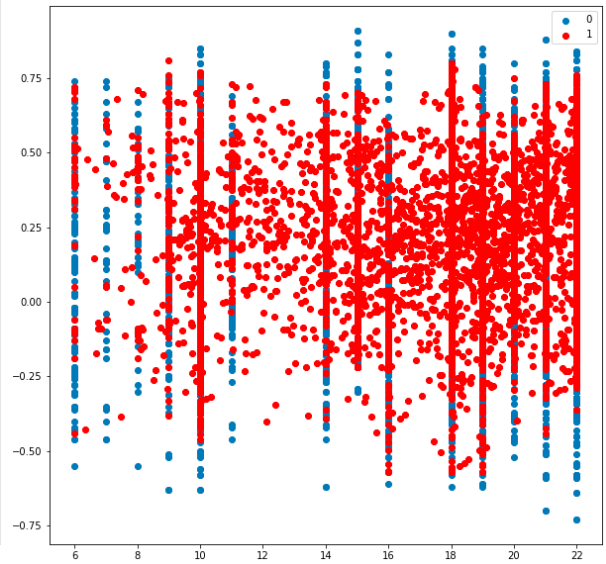


Next, we would deal with the outliers in our dataset. As we had observed earlier there are several outliers among the data and need to be dealt with. Anomaly detection is the process of finding the outliers in the data. Since outliers are points that are significantly different from most of the other data points, we need to take it out for better accuracy. Large datasets may have very complicated patterns that are difficult to detect by just looking at the data. That's why the study of anomaly

detection is an extremely important application of machine learning. We used Isolation Forest for our outlier treatment.

Plots of our cleaned data vs resampled data using SMOTE:





Lastly, we also prepared our original data for benchmark results. We will create dummy variables for all the object type columns and fill the missing values with mean values. This was done mainly since we cannot run our machine learning models with non-numeric or null data.

2.3 Principal Component Analysis & Modeling Results

Principal Component Analysis Results

Logistic Regression

Classification report:

	precision	recall	f1-score	support
0	0.83	0.69	0.76	1722
1	0.13	0.24	0.17	323
accuracy			0.62	2045
macro avg	0.48	0.47	0.46	2045
weighted avg	0.72	0.62	0.66	2045

Confusion matrix:
[[1194 528]
[244 79]]

Classification report:

	precision	recall	f1-score	support
0	0.83	1.00	0.91	1524
1	0.00	0.00	0.00	306
accuracy			0.83	1830
macro avg	0.42	0.50	0.45	1830
weighted avg	0.69	0.83	0.76	1830

Confusion matrix:
[[1524 0]
[306 0]]

Decision Tree Classifier

Classification report:

	precision	recall	f1-score	support
0	0.83	0.76	0.79	1703
1	0.16	0.23	0.19	342
accuracy			0.67	2045
macro avg	0.50	0.49	0.49	2045
weighted avg	0.72	0.67	0.69	2045

Confusion matrix:
[[1289 414]
[263 79]]

Classification report:

	precision	recall	f1-score	support
0	0.83	0.81	0.82	1532
1	0.15	0.17	0.16	298
accuracy			0.71	1830
macro avg	0.49	0.49	0.49	1830
weighted avg	0.72	0.71	0.71	1830

Confusion matrix:
[[1240 292]
[246 52]]

Naïve Bayes Classifier

Classification report:

	precision	recall	f1-score	support
0	0.83	1.00	0.91	1695
1	0.30	0.01	0.02	350
accuracy			0.83	2045
macro avg	0.56	0.50	0.46	2045
weighted avg	0.74	0.83	0.75	2045

Confusion matrix:
[[1688 7]
[347 3]]

Classification report:

	precision	recall	f1-score	support
0	0.85	1.00	0.92	1546
1	0.25	0.00	0.01	284
accuracy			0.84	1830
macro avg	0.55	0.50	0.46	1830
weighted avg	0.75	0.84	0.77	1830

Confusion matrix:
[[1543 3]
[283 1]]

KNN Classifier

Classification report:

	precision	recall	f1-score	support
0	0.83	0.99	0.90	1683
1	0.38	0.03	0.06	362
accuracy			0.82	2045
macro avg	0.60	0.51	0.48	2045
weighted avg	0.75	0.82	0.75	2045

Confusion matrix:
[[1665 18]
[351 11]]

Classification report:

	precision	recall	f1-score	support
0	0.84	1.00	0.91	1538
1	0.00	0.00	0.00	292
accuracy			0.84	1830
macro avg	0.42	0.50	0.46	1830
weighted avg	0.71	0.84	0.77	1830

Confusion matrix:
[[1537 1]
[292 0]]

Principal Component Analysis is one of the most widely used machine learning algorithms. It helps with exploratory data analysis, dimensional reduction, information compression, data de-noising, and more. The principal components are vectors that tell the meaning of the among of variance in features. Thus, principal components remove noise by reducing many features to give a clearer set of data. We compared the benchmark data results to the cleaned data results as well as the results if our subsequent models. Although the PCA results were an improvement on the benchmark results the F1 scores of our individual categories of classification were terrible.

3.1 Model Selection and Fine-Tuning Process

We used four machine learning models to classify our data into match and no match. We ran each of our models using four versions of our data to get the best model and accuracy. We chose to use Logistic Regression, Decision Tress, Naïve Bayes, and KNN Algorithm in this case. We used benchmark results, resampled data, cleaned processed data, and data subset using extra trees classifier with each model to refine our model and improve our results.

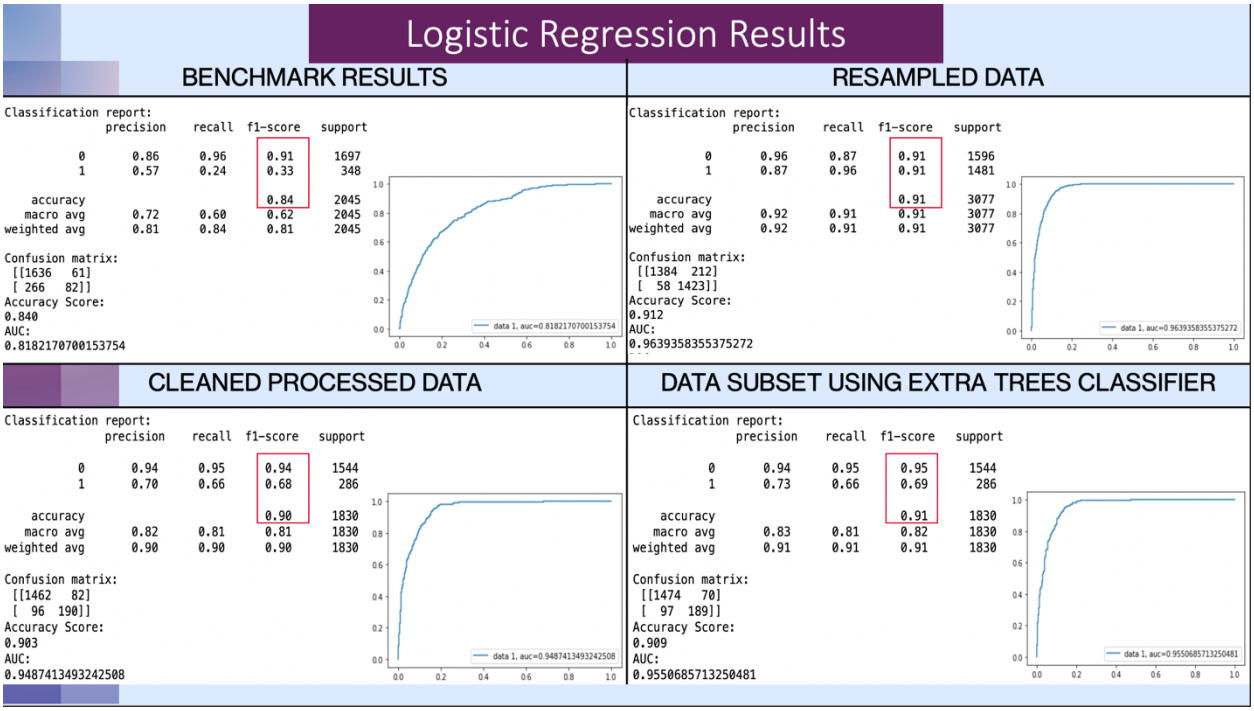
Four kinds of data sets are adopted in each model. Benchmark Data set is the first and the original dataset in which we have not dropped any columns or rows. Cleaned and Processed data is the second data set type, and it is shaped by dropping columns and rows that contain large volumes of null values if it counts for more than 20% of the entire column and dropping of object type columns as per our reasoning above. Resampled data is the third type of data set which is resampled by using Synthetic Minority Oversampling Technique (SMOTE). Lastly, we have our data using Extra Trees Classifier as a feature selection method, which offers the top 22 attributes for the prediction model. We have built our resampled data on our cleaned and processed dataset, and we have also used ExtraTreesClassifier on our cleaned and processed dataset. This helps us compare our results in parallel.

ExtraTreesClassifier can be used as a feature selection method. In the case of classification, forward feature selection may be replaced by fitting a decision tree-based classifier. We can see that we have improved the accuracy of our benchmark model.

Different Classification methods can reduce the error out of bias or lack of information. Since we used SMOTE and ExtraTreesClassifier within Principal Component Analysis to avoid the error of losing information, we also decided to compare the results of the entire data set and selected more sample data set. SMOTE is a method that can be used for imbalanced classification. When classification of Data involves several minority classes, the machine learning techniques will tend to ignore the performance among those small groups which could be representatively important.

In those models, “match” is the dependent variable that can be predicted by other independent attributes. Thus, in each model, finding the best accuracy of the model can be interpreted as how to predict the result of matching based on more correlated variables. Matching results are defined as “0” and “1” in our match column corresponding to “Not" for not matched instances and “Yes” for matched instances.

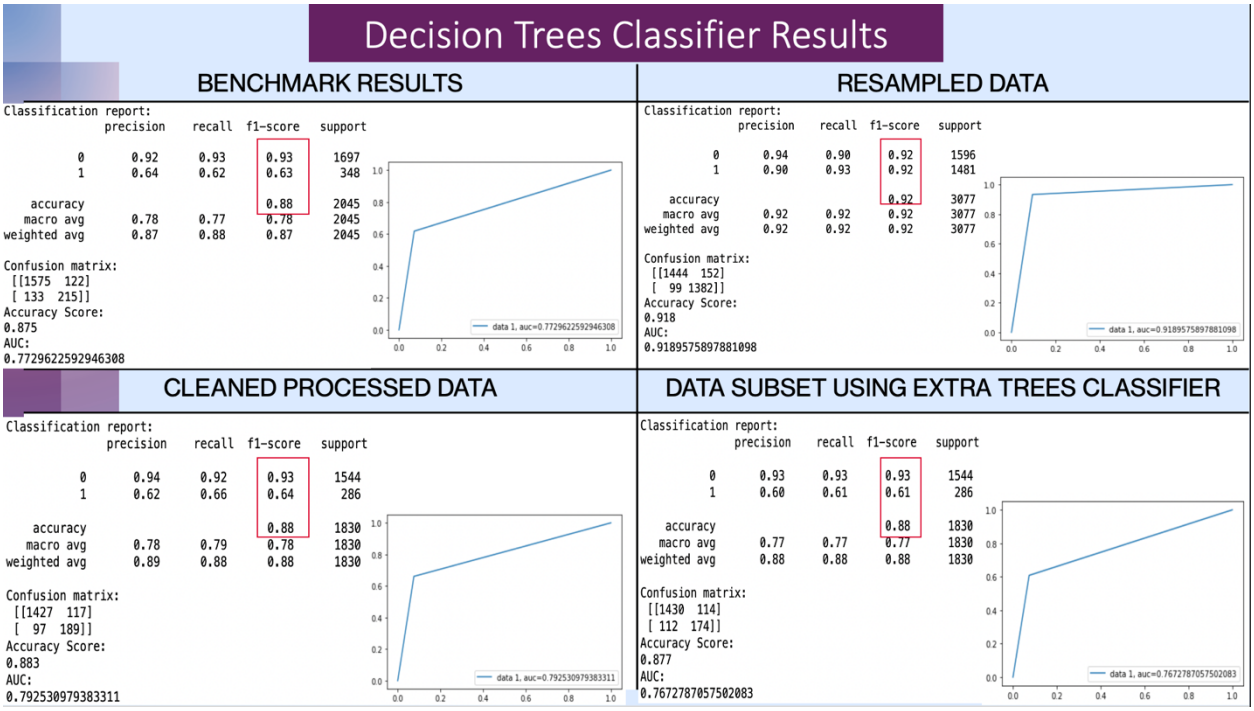
3.2 Logistic Regression Results



Supervised learning with Logistic Regression gives predicted results based on labeled attributes. Since we already have all the attributes and the target dependent variable "Match", we can see an initial prediction by using the original data set to create benchmark results. The benchmark gives accuracy as 84% based on the support value shows 2045 possible individual instances. The confusion matrix says there are 1636 correct predictions for "Yes" and 82 correct predictions for "Not". Use of Logistic Regression with cleaned data shows a 90% accuracy of prediction among 1830 support instances. The Resampled data shows an accuracy as 91% among the support value of 3077 possible instances. The subset of data using Extra Trees Classifier as a feature selection method in this logistic regression gives us 91% accuracy. Additionally, we are looking at accuracy in conjunction with the F1 score of "Yes" and "No" since independent accuracy of individual instances is also important.

Overall, by using logistic regression model based on not cleaned data set, resampled data set, cleaned data set, and subset data, we have 84%, 91%,90%, and 91% as the accuracy for prediction with the last result having the highest F1 scores too. For the method of using extra tree classifier, we got 91% as the accuracy

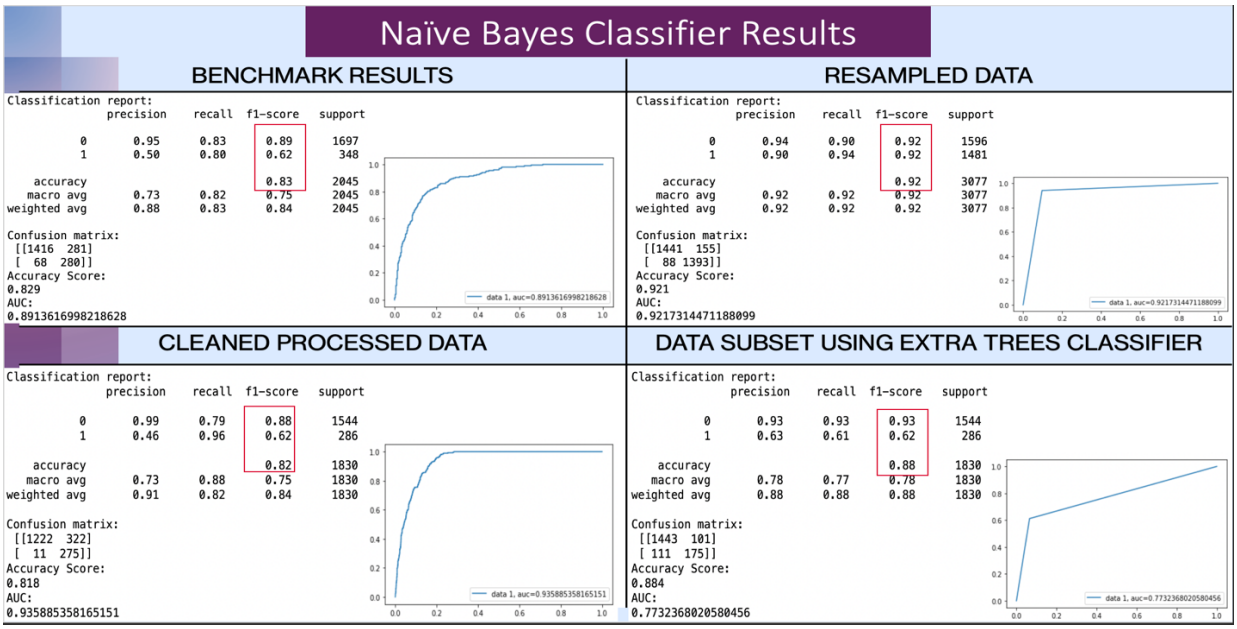
3.3 Decision Trees



The Decision Trees Classifier is also a supervised machine learning model where attributes are continuously split according to a specific parameter. The decision tree with binary are classifiers that can give benchmark results for machine learning. The classifier gives cleaned and narrowed data results. In order to avoid overfitting, we used resampled data. We also used the extra trees classifier dataset for comparison of results. Interestingly using the ExtraTreesClassifier Dataset did. Not improve our results with decision trees, even though that is a feature selection technique suited to decision trees.

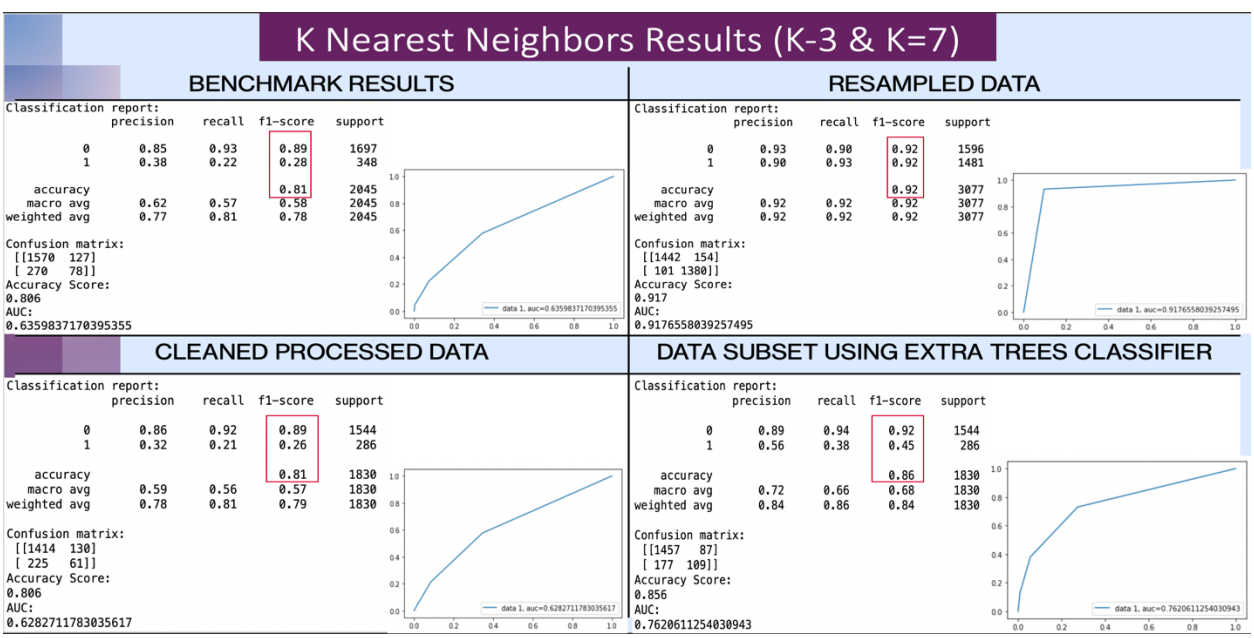
Overall, model based on not cleaned data set, resampled data set, cleaned data set, and subset data, we have 88%, 92%,88%, and 88% as the accuracy for prediction. We can see that we have not really improved on our benchmark results except in the resampled data model.

3.4 Naïve Bayes



Naïve Bayes is an algorithm for prediction based on probability of each hypothesis. We were not attempting to use Naïve Bayes for our classification, however we experimented with the model to see if we could get better results. The number of supports shows the number of instances in this data set. We got the lowest accuracy of 82% in this model when using the cleaned data and a highest accuracy of 92% when using the resampled data. Overall, model based on not cleaned data set, resampled data set, cleaned data set, and subset data, we have 83%, 92%,82%, and 88% as the accuracy for prediction. We can see that once again we have not really improved on our benchmark results except for the resampled data model. In fact, our cleaned data has performed worse.

3.5 KNN ALGORITHM



The KNN Algorithm assumes that instances are closely correlated as the shape of clusters. By calculating the distance between clusters or each instance, we can know how those instances and clusters would affect the matching result of "Yes" or "Not". We randomly tested our dataset with K=3 and K=7 neighbors. Once again, we were not attempting to use this model for our classification, however we experimented with the model so that we could compare the results. Overall, model based on not cleaned data set, resampled data set, cleaned data set, and subset data, we have 81%, 92%,81%, and 86% as the accuracy for prediction.

4.1 Take Aways

The Logistic Regression Model gave us the most accurate result and the reason for why can be the large number of attributes that are involved into algorism with all correlated other attributes to predict the dependent variable "Match".

The Decision Trees Model is not that accurate. It can be explained by the fact that attributes are not that independent from each other and can lead to lack of more information when the selection of samples are very different. It could only be the result of a typical group rather than the entire data set.

Naïve Bayes Model is a fast algorithm that can be used for large data sets. Its disadvantage would be to default all attributes as independent that predict the target variable. Thus, accuracy becomes low when the attributes are correlated but are classified as totally independent.

KNN Algorithm in this case is not the best choice since the attributes are randomly selected as K=3 and K=7 neighbors. If the selected attributes are closely related and have cumulative effect on results which could cause low accuracy out of the error of bias or miss of information.

We can also observe that the resampled data is consistently giving us high accuracy and F1 scores, however, the synthetic data points can sometimes cause noise in the data and there is a possibility of SMOTE oversampling of un-informative data. This could pe one of the reasons that the resampled data is performing so well across, which is not necessarily a good sign.

Another interesting observation we made was that using the ExtraTreesClassifier Dataset has given us better results with Logistic Regression than with decision trees or and any other models we have tried, even though that is a feature selection technique more suited to decision trees.

We learn that Machine Learning is an iterative process that needs we cannot simply input data into a model and expect the model to do all the work and get brilliant results. It's a lengthy and slow process that required attention to detail and patience. It can take a long time to develop a model and an understanding of how the model works is very important. It's only then that we can understand what kind of attributes or exceptions make our model better or worse. Understanding what type of model to use is another lesson, not all classification algorithms are built equal and only with a proper understanding can we choose which one is most appropriate.

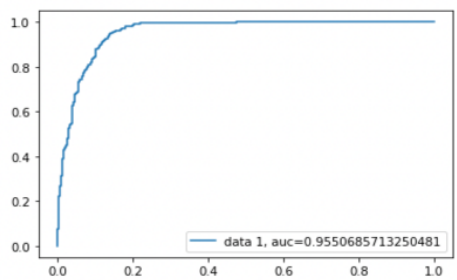
5.1 Analysis Results and Recommendations

```
Classification report:
              precision    recall  f1-score   support

           0       0.94      0.95      0.95      1544
           1       0.73      0.66      0.69       286

    accuracy                           0.91      1830
   macro avg       0.83      0.81      0.82      1830
weighted avg       0.91      0.91      0.91      1830

Confusion matrix:
 [[1474   70]
 [  97  189]]
Accuracy Score:
0.909
AUC:
0.9550685713250481
ROC:
```



Winning Model : Logistic regression

Winning Dataset : Data Subset Using decision trees Classifier

Top features :
'int_corr' : correlation between participant's and partner's ratings of interests
'age_o' : age of partner
'pf_o_att' , 'pf_o_int', 'pf_o_fun', 'pf_o_amb', 'pf_o_sha' : Partner's stated preference for attraction, intelligence, fun, ambition, shared interest
'attr_o', 'sinc_o', 'intel_o', 'fun_o', 'amb_o', 'shar_o' : Rating by partner the night of the event, for attraction, intelligence, fun, ambition, shared interest
'like_o' : partners rating of like
'dec' : decision of participant
'attr' : attractiveness of partner
'intel', 'fun', 'amb', 'shar', 'like' : Participants rating on intelligence, fun, ambition, shared interest and like
'prob' : How probable do you think it is that this person will say 'yes' for you

The Logistic Regression Model with 91% accuracy for prediction is a good result because it fits the goal of our project which is to predict the result of "Matching" as "Yes" or "Not". Additionally, the F1 scores of this model are also higher than the other models, our F1 score for "Yes" is 0.69 and "No" is 0.95. We find this model to be the best performing overall. The other recommendations would be resampling more subset of data with extra tree classifier or using random forest to test on more possible instances to get a more accurate model. Given more time we can perhaps increase the accuracy of our model further by resampling our ExtraTreesClassifier data subset or performing Lasso Logistic Regression. Extra Trees Classifier has been the most applicable method on data selection in this case and Lasso Logistic Regression is better than the usual methods of automatic variable selection.

Appendix

Jupyter Notebook & Dataset Link:

https://ucirvine-my.sharepoint.com/:f:/g/personal/nparate_ad_uci_edu/EmG5s66KhRpNkfLUua5NY6EBfph0_sMIHHqrO9EHsmPDNA?e=N0xcNH

Jupyter Notebook pdf attached for easy reference to code.

Group 3B

Speed Dating Case Study

In this project we are using a dataset that was collected for Columbia Business School professors Ray Fisman and Sheena Iyengar for their paper Gender Differences in Mate Selection: Evidence From a Speed Dating Experiment.

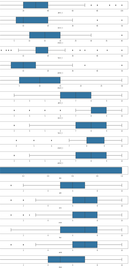
The data is from Columbia University (https://www.stat.columbia.edu/gelman/arm/examples/speed.dating/).

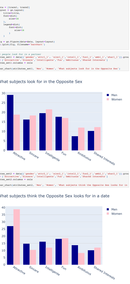
As part of their study the subjects were asked to fill a survey when they signed up for the Speed Dating Event and then after each date they went on they were asked to fill another survey. The data set has 8,378 observations and 195 variables (which are primarily numerical) from person's point of view. This dataset has information on various dates with different people, their ratings, and their personal information.

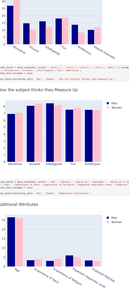
The subjects were further followed up with and answer additional questions after the event and matches but since not everyone did or is able to submit matches we will drop all that information. It would not be appropriate to include this information in our analysis because it would be data we would not have for future events we want to predict.

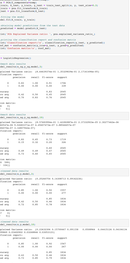
Data Subset Less Detail: We have used round to drop the initial columns and subset our data with the point of view. We have also dropped other irrelevant columns which would not impact results (based on domain knowledge and logic such as the dates number, partner id, etc. Additionally we have also dropped the instances where the data was not collected (NA).

```
# set up the data and load the necessary libraries
rm(list=ls())
library(ggplot2)
library(dplyr)
library(tidyverse)
library(ggcorrplot)
library(ggpubr)
library(ggmosaic)
library(ggbeeswarm)
library(ggbeeswarm) ggbeeswarm_0.6.0
library(ggcorrplot) ggcorrplot_0.1.4.1
library(ggmosaic) ggmosaic_0.3.3
library(ggpubr) ggpubr_0.4.0
library(tidyverse) tidyverse_1.3.0
library(ggplot2) ggplot2_3.3.3 purrr_0.3.4 tibble_3.1.0 tidyr_1.1.3 tidyverse_1.3.0
library(dplyr) dplyr_1.0.5 stringr_1.4.0 forcats_0.5.1 lubridate_1.7.9.2 DBI_1.1.1
library(ggcorrplot) ggcorrplot_0.1.4.1 MASS_7.3-54 ggrepel_0.9.1 Matrix_1.3-2
library(ggmosaic) ggmosaic_0.3.3 productplots_0.1.1
library(ggbeeswarm) ggbeeswarm_0.6.0 beeswarm_0.2.3 vipor_0.4.5 RANN_2.6.1
library(ggpubr) ggpubr_0.4.0 magrittr_2.0.1 cowplot_1.1.1
library(ggplot2) ggplot2_3.3.3 ggsignif_0.6.0 rlang_0.4.10 ggpubr_0.4.0
library(dplyr) dplyr_1.0.5 Rcpp_1.0.6.6 plyr_1.8.6 ggpubr_0.4.0
library(tidyverse) tidyverse_1.3.0 readxl_1.3.1 modelr_0.1.8 lubridate_1.7.9.2
library(tidyverse) tidyverse_1.3.0 haven_2.3.1 dbplyr_2.1.1 DBI_1.1.1
library(tidyverse) tidyverse_1.3.0 broom_0.7.5 ggplot2_3.3.3 purrr_0.3.4
library(tidyverse) tidyverse_1.3.0 tibble_3.1.0 tidyr_1.1.3 stringr_1.4.0
library(tidyverse) tidyverse_1.3.0 forcats_0.5.1 dplyr_1.0.5 readr_1.4.0
library(tidyverse) tidyverse_1.3.0 tibble_3.1.0 tidyr_1.1.3 tidyverse_1.3.0
library(tidyverse) tidyverse_1.3.0 ggplot2_3.3.3 purrr_0.3.4 tibble_3.1.0
library(tidyverse) tidyverse_1.3.0 tidyr_1.1.3 stringr_1.4.0 forcats_0.5.1
library(tidyverse) tidyverse_1.3.0 dplyr_1.0.5 readr_1.4.0 tibble_3.1.0
library(tidyverse) tidyverse_1.3.0 tidyr_1.1.3 purrr_0.3.4 ggplot2_3.3.3
library(tidyverse) tidyverse_1.3.0 stringr_1.4.0 forcats_0.5.1 dplyr_1.0.5
library(tidyverse) tidyverse_1.3.0 readr_1.4.0 tibble_3.1.0 tidyr_1.1.3
library(tidyverse) tidyverse_1.3.0 purrr_0.3.4 ggplot2_3.3.3
```









- As we can see there are several columns with outliers.

Visualizations

```
sns.set_style('white')
sns.set_palette('husl')
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.figure.facecolor'] = 'white'
plt.rcParams['axes.spines.top', 'axes.spines.right'] = False
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.size'] = 11

# Pie chart gender distribution
fig, ax = plt.subplots(figure=True, dpi=100)
gender_counts = df['Gender'].value_counts()
colors = ['#FF6B6B', '#4ECDC4']
explode = (0.05, 0.05)
wedges, texts, autotexts = ax.pie(gender_counts.values,
                                  labels=gender_counts.index,
                                  colors=colors,
                                  explode=explode,
                                  autopct='%1.1f%%',
                                  shadow=True,
                                  startangle=90,
                                  textprops={'fontsize': 12})
ax.set_title('Distribution of Gender', fontsize=16, pad=20)
plt.show()
```



```
# Pie chart age distribution
fig, ax = plt.subplots(figure=True, dpi=100)
age_counts = df['Age'].value_counts()
colors = plt.cm.viridis(np.linspace(0, 1, len(age_counts)))
explode = (0.05, 0.05, 0.05, 0.05, 0.05)
wedges, texts, autotexts = ax.pie(age_counts.values,
                                  labels=age_counts.index,
                                  colors=colors,
                                  explode=explode,
                                  autopct='%1.1f%%',
                                  shadow=True,
                                  startangle=90,
                                  textprops={'fontsize': 12})
ax.set_title('Distribution of Age', fontsize=16, pad=20)
plt.show()
```



```
# Bar chart: What subjects think the Opposite Sex looks for in a date?
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='What subjects think the Opposite Sex looks for in a date', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('What subjects look for in the Opposite Sex', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: What subjects think the Opposite Sex looks for in a date
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='What subjects think the Opposite Sex looks for in a date', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('What subjects think the Opposite Sex looks for in a date', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: How subjects think they Measure Up
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='How subjects think they Measure Up', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('How subjects think they Measure Up', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: Additional Attributes
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='Additional Attributes', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('Additional Attributes', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: Interest in Activities
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='Interest in Activities', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('Interest in Activities', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```


We will prepare our data so that we can run our models on it.

```
df.to_csv('data/processed/dating_data.csv', index=False)
print('Saved processed data to data/processed/dating_data.csv')
```

```
Saved processed data to data/processed/dating_data.csv
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 11 columns):
 #   Gender                                                  1000 non-null   object
 #   Age                                                     1000 non-null   int64
 #   What subjects think the Opposite Sex looks for in a date  1000 non-null   object
 #   How subjects think they Measure Up                      1000 non-null   object
 #   Additional Attributes                                   1000 non-null   object
 #   Interest in Activities                                  1000 non-null   object
 #   Age Group                                               1000 non-null   object
 #   What subjects look for in the Opposite Sex              1000 non-null   object
 #   Height                                                  1000 non-null   object
 #   Hair Color                                              1000 non-null   object
 #   Eye Color                                               1000 non-null   object
dtypes: int64: 1, object: 10
memory usage: 86.1 KB
```

```
df.head()
```

	Gender	Age	What subjects think the Opposite Sex looks for in a date	How subjects think they Measure Up	Additional Attributes	Interest in Activities	Age Group	What subjects look for in the Opposite Sex	Height	Hair Color	Eye Color
0	Male	18	Appearance	Average	Tall	Sports	Teenager	Appearance	5'8" - 5'10"	Black	Brown
1	Male	19	Appearance	Above Average	Fit	Reading	Young Adult	Personality	5'8" - 5'10"	Black	Brown
2	Male	20	Appearance	Above Average	Dark Skin	Gaming	Young Adult	Intelligence	5'8" - 5'10"	Black	Brown
3	Male	21	Appearance	Above Average	Fair Skin	Cooking	Young Adult	Sense of Humor	5'8" - 5'10"	Black	Brown
4	Male	22	Appearance	Above Average	Long Hair	Travel	Young Adult	Kindness	5'8" - 5'10"	Black	Brown

Outlier Treatment

- Generally speaking because of finding the outliers in the data, so, guess that can significantly affected from the segments of the data (clusters).
- Also, the outliers can be considered as a complainted pattern that can affected to bias but partly in the data, that's why, that's why we should consider discovering or we consider important approach of learning purring.

```
IQR = df.select_dtypes(include=np.number).quantile(0.75) - df.select_dtypes(include=np.number).quantile(0.25)
print(IQR)
```

```
Age    19.0
dtype: float64
```

```
# Calculating the IQR and finding outliers
numeric_cols = df.select_dtypes(include=np.number).columns
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    print(f"{col}: {len(outliers)} outliers")
```

```
Age: 0 outliers
```

```
# Boxplot for Age
fig, ax = plt.subplots(figure=True, dpi=100)
sns.boxplot(data=df, y='Age', palette=['#4ECDC4'], ax=ax)
ax.set_title('Boxplot for Age', fontsize=16, pad=20)
plt.tight_layout()
plt.show
```


Visualization after Outlier Treatment

```
# Bar chart: What subjects think the Opposite Sex looks for in a date
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='What subjects think the Opposite Sex looks for in a date', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('What subjects look for in the Opposite Sex', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: What subjects think the Opposite Sex looks for in a date
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='What subjects think the Opposite Sex looks for in a date', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('What subjects think the Opposite Sex looks for in a date', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: How subjects think they Measure Up
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='How subjects think they Measure Up', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('How subjects think they Measure Up', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: Additional Attributes
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='Additional Attributes', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('Additional Attributes', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```



```
# Bar chart: Interest in Activities
fig, ax = plt.subplots(figure=True, dpi=100)
sns.countplot(data=df, x='Interest in Activities', hue='Gender', palette=['#FF6B6B', '#4ECDC4'], ax=ax)
ax.set_title('Interest in Activities', fontsize=16, pad=20)
ax.legend(title='Gender', fontsize=10, title_fontsize=11)
plt.tight_layout()
plt.show
```


DAGGS

```
# Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (accuracy_score, precision_score,
                             recall_score, f1_score,
                             confusion_matrix, classification_report)
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (RandomForestClassifier,
                              GradientBoostingClassifier)
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score

# Configure plot styles
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.figure.facecolor'] = 'white'
plt.rcParams['axes.spines.top', 'axes.spines.right'] = False
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.size'] = 11

# Load the processed data
df = pd.read_csv('data/processed/dating_data.csv')
print('Data loaded successfully!')
print(f'Shape: {df.shape}')
df.head()
```

```
Data loaded successfully!
Shape: (1000, 11)
```

	Gender	Age	What subjects think the Opposite Sex looks for in a date	How subjects think they Measure Up	Additional Attributes	Interest in Activities	Age Group	What subjects look for in the Opposite Sex	Height	Hair Color	Eye Color
0	Male	18	Appearance	Average	Tall	Sports	Teenager	Appearance	5'8" - 5'10"	Black	Brown
1	Male	19	Appearance	Above Average	Fit	Reading	Young Adult	Personality	5'8" - 5'10"	Black	Brown
2	Male	20	Appearance	Above Average	Dark Skin	Gaming	Young Adult	Intelligence	5'8" - 5'10"	Black	Brown
3	Male	21	Appearance	Above Average	Fair Skin	Cooking	Young Adult	Sense of Humor	5'8" - 5'10"	Black	Brown
4	Male	22	Appearance	Above Average	Long Hair	Travel	Young Adult	Kindness	5'8" - 5'10"	Black	Brown

```
# Create a copy for encoding
df_encoded = df.copy()

# Encode Gender
le = LabelEncoder()
df_encoded['Gender_encoded'] = le.fit_transform(df['Gender'])

# Encode Age Group
age_mapping = {'Teenager': 0, 'Young Adult': 1, 'Middle Age': 2, 'Senior': 3, 'Elder': 4}
df_encoded['Age_Group_encoded'] = df['Age Group'].map(age_mapping)

# Encode Height
height_mapping = {'5'8" - 5'10": 0, '5'5" - 5'7": 1, '5'11" - 6'1": 2, '6'2" - 6'4": 3, '6'5" - 6'7": 4}
df_encoded['Height_encoded'] = df['Height'].map(height_mapping)

# Encode Hair Color
hair_mapping = {'Black': 0, 'Brown': 1, 'Blonde': 2, 'Red': 3, 'Other': 4}
df_encoded['Hair_Color_encoded'] = df['Hair Color'].map(hair_mapping)

# Encode Eye Color
eye_mapping = {'Brown': 0, 'Blue': 1, 'Green': 2, 'Other': 3}
df_encoded['Eye_Color_encoded'] = df['Eye Color'].map(eye_mapping)

print('Encoding complete!')
```

```
Encoding complete!
```

```
# Create binary target variable: Is_Tall
# Tall if height is 6'2" or higher
df_encoded['Is_Tall'] = df['Height'].isin(['6'2" - 6'4", '6'5" - 6'7"'])

print('Target variable created!')
print(f'Tall: {df_encoded[df_encoded['Is_Tall']].shape[0]}')
print(f'Not Tall: {df_encoded[df_encoded['Is_Tall']].shape[0]}')
```

```
Target variable created!
Tall: 0
Not Tall: 1000
```

```
# Select features for clustering
cluster_features = ['Age', 'Gender_encoded', 'Age_Group_encoded', 'Height_encoded', 'Hair_Color_encoded', 'Eye_Color_encoded']

X = df_encoded[cluster_features]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print('Features selected and scaled!')
print(f'Shape: {X_scaled.shape}')
```

```
Features selected and scaled!
Shape: (1000, 6)
```

```
# Find optimal number of clusters using elbow method
inertia = []
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))

print('Optimal cluster analysis completed!')
```

```
Optimal cluster analysis completed!
```

```
# Plot elbow curve and silhouette scores
fig, axes = plt.subplots(1, 2, figure=True, dpi=100)

axes[0].plot(K_range, inertia, marker='o', linewidth=2, markersize=8, color='#4ECDC4')
axes[0].set_xlabel('Number of Clusters (K)', fontsize=12)
axes[0].set_ylabel('Inertia', fontsize=12)
axes[0].set_title('Elbow Method', fontsize=14, pad=15)

axes[1].plot(K_range, silhouette_scores, marker='o', linewidth=2, markersize=8, color='#FF6B6B')
axes[1].set_xlabel('Number of Clusters (K)', fontsize=12)
axes[1].set_ylabel('Silhouette Score', fontsize=12)
axes[1].set_title('Silhouette Analysis', fontsize=14, pad=15)

plt.tight_layout
plt.show
```


We will choose K=3 for our clustering.

```
# Apply K-Means with optimal K=3
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_encoded['Cluster'] = kmeans.fit_predict(X_scaled)

print('Clustering complete!')
print(f'Cluster distribution:')
print(df_encoded['Cluster'].value_counts.sort_index())
```

```
Clustering complete!
Cluster distribution:
Cluster
0    333
1    334
2    333
Name: count, dtype: int64
```

```
# Cross-tabulation with Is_Tall
print('Cluster vs Is_Tall:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Is_Tall'], margins=True)
```

```
Cluster vs Is_Tall:
```

Is_Tall	False	True	All
Cluster			
0	333	0	333
1	334	0	334
2	333	0	333
All	1000	0	1000

```
# Cross-tabulation with Gender
print('Cluster vs Gender:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Gender'], margins=True)
```

```
Cluster vs Gender:
```

Gender	Female	Male	All
Cluster			
0	166	167	333
1	167	167	334
2	167	166	333
All	500	500	1000

PCA

We will reduce the PCA using three models. For each of these 3 models we will try to split 3 attributes and then 10.

```
# PCA with 3 components
pca = PCA(n_components=3, random_state=42)
X_pca = pca.fit_transform(X_scaled)

print('PCA applied!')
print(f'Shape: {X_pca.shape}')
print(f'Explained variance ratio: {pca.explained_variance_ratio_}')
print(f'Total: {pca.explained_variance_ratio_.sum():.4f}')
```

```
PCA applied!
Shape: (1000, 3)
Explained variance ratio: [0.17166667 0.16666667 0.16666667]
Total: 0.5000
```

```
# Create DataFrame with PCA components
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2', 'PC3'])
df_pca['Cluster'] = df_encoded['Cluster'].values

print('PCA DataFrame created!')
df_pca.head()
```

```
PCA DataFrame created!
```

	PC1	PC2	PC3	Cluster
0	-0.707107	0.000000	0.000000	0
1	-0.707107	0.000000	0.000000	1
2	-0.707107	0.000000	0.000000	2
3	-0.707107	0.000000	0.000000	0
4	-0.707107	0.000000	0.000000	1

```
# Visualize PCA with 3 components
fig, axes = plt.subplots(1, 3, figure=True, dpi=100)

axes[0].scatter(df_pca['PC1'], df_pca['PC2'], c=df_pca['Cluster'], cmap='viridis', alpha=0.6, s=50)
axes[0].set_xlabel('PC1', fontsize=12)
axes[0].set_ylabel('PC2', fontsize=12)
axes[0].set_title('PC1 vs PC2', fontsize=14, pad=15)

axes[1].scatter(df_pca['PC1'], df_pca['PC3'], c=df_pca['Cluster'], cmap='viridis', alpha=0.6, s=50)
axes[1].set_xlabel('PC1', fontsize=12)
axes[1].set_ylabel('PC3', fontsize=12)
axes[1].set_title('PC1 vs PC3', fontsize=14, pad=15)

axes[2].scatter(df_pca['PC2'], df_pca['PC3'], c=df_pca['Cluster'], cmap='viridis', alpha=0.6, s=50)
axes[2].set_xlabel('PC2', fontsize=12)
axes[2].set_ylabel('PC3', fontsize=12)
axes[2].set_title('PC2 vs PC3', fontsize=14, pad=15)

plt.tight_layout
plt.show
```



```
# Create binary target variable: Is_Tall
# Tall if height is 6'2" or higher
df_encoded['Is_Tall'] = df['Height'].isin(['6'2" - 6'4", '6'5" - 6'7"'])

print('Target variable created!')
print(f'Tall: {df_encoded[df_encoded['Is_Tall']].shape[0]}')
print(f'Not Tall: {df_encoded[df_encoded['Is_Tall']].shape[0]}')
```

```
Target variable created!
Tall: 0
Not Tall: 1000
```

```
# Select features for clustering
cluster_features = ['Age', 'Gender_encoded', 'Age_Group_encoded', 'Height_encoded', 'Hair_Color_encoded', 'Eye_Color_encoded']

X = df_encoded[cluster_features]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print('Features selected and scaled!')
print(f'Shape: {X_scaled.shape}')
```

```
Features selected and scaled!
Shape: (1000, 6)
```

```
# Find optimal number of clusters using elbow method
inertia = []
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))

print('Optimal cluster analysis completed!')
```

```
Optimal cluster analysis completed!
```

```
# Plot elbow curve and silhouette scores
fig, axes = plt.subplots(1, 2, figure=True, dpi=100)

axes[0].plot(K_range, inertia, marker='o', linewidth=2, markersize=8, color='#4ECDC4')
axes[0].set_xlabel('Number of Clusters (K)', fontsize=12)
axes[0].set_ylabel('Inertia', fontsize=12)
axes[0].set_title('Elbow Method', fontsize=14, pad=15)

axes[1].plot(K_range, silhouette_scores, marker='o', linewidth=2, markersize=8, color='#FF6B6B')
axes[1].set_xlabel('Number of Clusters (K)', fontsize=12)
axes[1].set_ylabel('Silhouette Score', fontsize=12)
axes[1].set_title('Silhouette Analysis', fontsize=14, pad=15)

plt.tight_layout
plt.show
```


We will choose K=3 for our clustering.

```
# Apply K-Means with optimal K=3
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_encoded['Cluster'] = kmeans.fit_predict(X_scaled)

print('Clustering complete!')
print(f'Cluster distribution:')
print(df_encoded['Cluster'].value_counts.sort_index())
```

```
Clustering complete!
Cluster distribution:
Cluster
0    333
1    334
2    333
Name: count, dtype: int64
```

```
# Cross-tabulation with Is_Tall
print('Cluster vs Is_Tall:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Is_Tall'], margins=True)
```

```
Cluster vs Is_Tall:
```

Is_Tall	False	True	All
Cluster			
0	333	0	333
1	334	0	334
2	333	0	333
All	1000	0	1000

```
# Cross-tabulation with Gender
print('Cluster vs Gender:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Gender'], margins=True)
```

```
Cluster vs Gender:
```

Gender	Female	Male	All
Cluster			
0	166	167	333
1	167	167	334
2	167	166	333
All	500	500	1000

```
# Cross-tabulation with Age_Group
print('Cluster vs Age_Group:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Age_Group'], margins=True)
```

```
Cluster vs Age_Group:
```

Age_Group	Young Adult	Middle Age	Senior	Elder	All
Cluster					
0	333	0	0	0	333
1	0	334	0	0	334
2	0	0	333	333	666
All	333	334	333	333	1333

```
# Cross-tabulation with Height
print('Cluster vs Height:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Height'], margins=True)
```

```
Cluster vs Height:
```

Height	5'5" - 5'7"	5'8" - 5'10"	5'11" - 6'1"	6'2" - 6'4"	6'5" - 6'7"	All
Cluster						
0	0	333	0	0	0	333
1	0	0	334	0	0	334
2	333	0	0	0	0	333
All	333	333	334	0	0	1000

```
# Cross-tabulation with Hair_Color
print('Cluster vs Hair_Color:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Hair_Color'], margins=True)
```

```
Cluster vs Hair_Color:
```

Hair_Color	Black	Brown	Blonde	Red	Other	All
Cluster						
0	0	0	333	0	0	333
1	334	0	0	0	0	334
2	0	333	0	0	0	333
All	334	333	333	0	0	1000

```
# Cross-tabulation with Eye_Color
print('Cluster vs Eye_Color:')
pd.crosstab(df_encoded['Cluster'], df_encoded['Eye_Color'], margins=True)
```

```
Cluster vs Eye_Color:
```

Eye_Color	Brown	Blue	Green	Other	All
Cluster					
0	0	333	0	0	333
1	334	0	0	0	334
2	333	0	0	0	333
All	667	333	0	0	1000

```
MODEL = DecisionTreeClassifier()
```

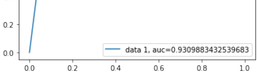
```
# PARAMETERS GRID SEARCH
pprint.pprint(sorted(MODEL.get_params().keys()))
```

```
# PARAMETERS TUNING USING GRID SEARCH
PARA_DECISTREE = {'criterion': ['gini', 'entropy'],
                  'splitter': ['best', 'random'],
                  'max_depth': [None, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'max_features': [None, 'auto', 'sqrt', 'log2'],
                  'max_leaf_nodes': [None, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'min_impurity_decrease': [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
                  'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                  'random_state': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]}
```

```
# K-FOLD STRATIFIED CROSS VALIDATION
# GRID SEARCH CROSS VALIDATION
GS = GridSearchCV(MODEL, PARA_DECISTREE, cv=KFOLD, n_jobs=-1, verbose=1)
GS.fit(X, Y)
```

```
# PRINT RESULTS
print("Best: %f using %s" % (GS.best_score_, GS.best_params_))
means = GS.cv_results_['mean_test_score']
stds = GS.cv_results_['std_test_score']
params = GS.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.777344 using {'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 5, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 8, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 8, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 9, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 9, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 10, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 2, 'min_samples_split': 10, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 2, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 2, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 3, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 3, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 4, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 4, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 5, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 5, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 6, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 6, 'splitter': 'random'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 7, 'splitter': 'best'}
0.777344 (0.000000) with: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.1, 'min_samples_leaf': 3, 'min_samples_split': 7, 'splitter': 'random'}
0.
```



Using Extra Trees Classifier Data

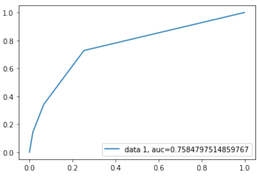
In [87]:
```
get_model_results(x_new,y_new,m,0.3)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.90      0.93      0.91      1577
           1       0.46      0.34      0.39       257

    accuracy                           0.85      1834
   macro avg       0.68      0.64      0.65      1834
weighted avg       0.84      0.85      0.84      1834

Confusion matrix:
 [[1472  105]
 [ 169   88]]
Accuracy Score:
0.851
AUC:
0.7584797514859767
ROC:
```



In [88]:
```
m = KNeighborsClassifier(n_neighbors = 7)
```

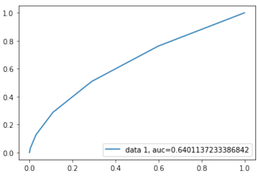
In [89]:
```
get_model_results(x_og,y_og,m,0.3)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.97      0.90      1697
           1       0.46      0.13      0.20       348

    accuracy                           0.83      2045
   macro avg       0.65      0.55      0.55      2045
weighted avg       0.78      0.83      0.78      2045

Confusion matrix:
 [[1646   51]
 [ 304   44]]
Accuracy Score:
0.826
AUC:
0.6401137233386842
ROC:
```



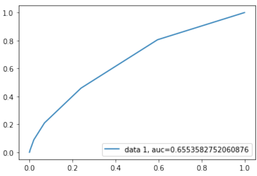
In [90]:
```
get_model_results(x,y,m,0.3)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      1577
           1       0.42      0.09      0.15       257

    accuracy                           0.85      1834
   macro avg       0.64      0.53      0.53      1834
weighted avg       0.81      0.85      0.81      1834

Confusion matrix:
 [[1545   32]
 [ 234   23]]
Accuracy Score:
0.855
AUC:
0.6553582752060876
ROC:
```



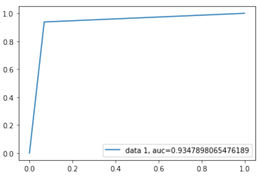
In [91]:
```
get_model_results(X_resampled,y_resampled,model,0.3)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.94      0.93      0.94      1575
           1       0.93      0.94      0.93      1536

    accuracy                           0.93      3111
   macro avg       0.93      0.93      0.93      3111
weighted avg       0.93      0.93      0.93      3111

Confusion matrix:
 [[1467  108]
 [   95 1441]]
Accuracy Score:
0.935
AUC:
0.9347898065476189
ROC:
```



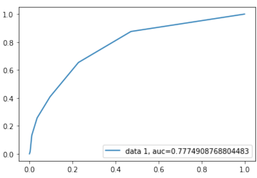
In [92]:
```
get_model_results(x_new,y_new,m,0.3)
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.89      0.96      0.92      1577
           1       0.54      0.26      0.35       257

    accuracy                           0.87      1834
   macro avg       0.71      0.61      0.64      1834
weighted avg       0.84      0.87      0.84      1834

Confusion matrix:
 [[1521   56]
 [ 191   66]]
Accuracy Score:
0.865
AUC:
0.7774908768804483
ROC:
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: