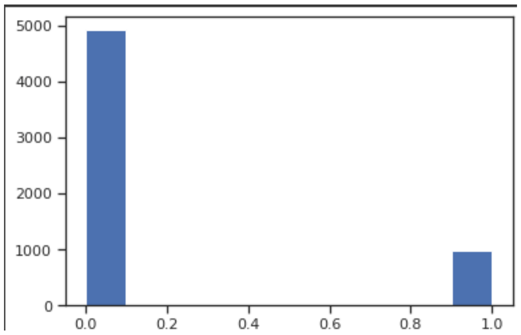
Problem Formulation

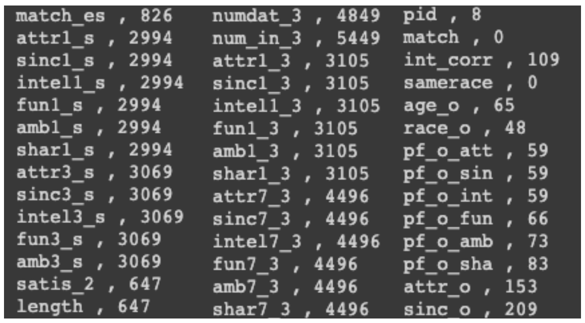
This dataset is from a speed dating experiment and includes data such as demographics, dating habits, goals, lifestyle, questionnaire filled by participants, rating of each other, and finally we are given binary match column indicating whether the session was successful or not. The goal of this project is to predict whether a session will be successful (1) or not (0) based on interests and preferences recorded through various features in the dataset. The input will be variations of the dataset (preprocessing) and output will be the probability of a successful match (float). In order to better understand the data, I will first conduct some exploratory analysis and look for potential challenges and ways to improve them.

Exploratory Data Analysis

An initial peak into the data shows that we have 5909 rows and 192 columns. First, I wanted to check the distribution of the target column to see if most session end in a match or not. It seems that it is highly unbalanced with significantly higher number of non-matches (0) than matches (1).

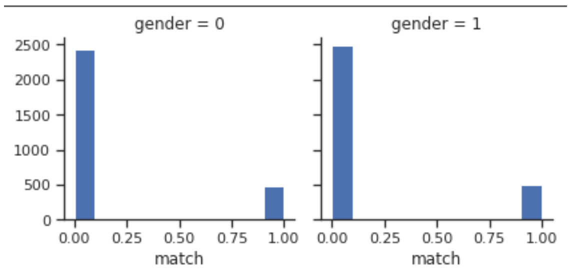


Then I wanted to check to see number of missing values for each column. Many columns seem to have high number of missing values indicating that this should be dealt with previous to prediction.

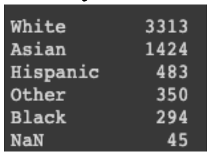


```
match_es , 826    numdat_3 , 4849  pid , 8
attr1_s , 2994    num_in_3 , 5449  match , 0
sinc1_s , 2994    attr1_3 , 3105   int_corr , 109
intel1_s , 2994   sinc1_3 , 3105   samerace , 0
fun1_s , 2994     intel1_3 , 3105  age_o , 65
amb1_s , 2994     fun1_3 , 3105    race_o , 48
shar1_s , 2994    amb1_3 , 3105    pf_o_att , 59
attr3_s , 3069    shar1_3 , 3105   pf_o_sin , 59
sinc3_s , 3069    attr7_3 , 4496   pf_o_int , 59
intel3_s , 3069   sinc7_3 , 4496   pf_o_fun , 66
fun3_s , 3069     intel7_3 , 4496  pf_o_amb , 73
amb3_s , 3069     fun7_3 , 4496    pf_o_sha , 83
satis_2 , 647     amb7_3 , 4496    attr_o , 153
length , 647      shar7_3 , 4496   sinc_o , 209
```

Since many columns have missing values and we already have a fairly large number of columns (features), I wanted to check the significance of these columns to see if any can be removed or manipulated. First, I wanted to see if match rate is associated with age of the potential partner. No particular difference in match rate between male and females was observed, however I did note that the data is pretty balanced gender-wise.

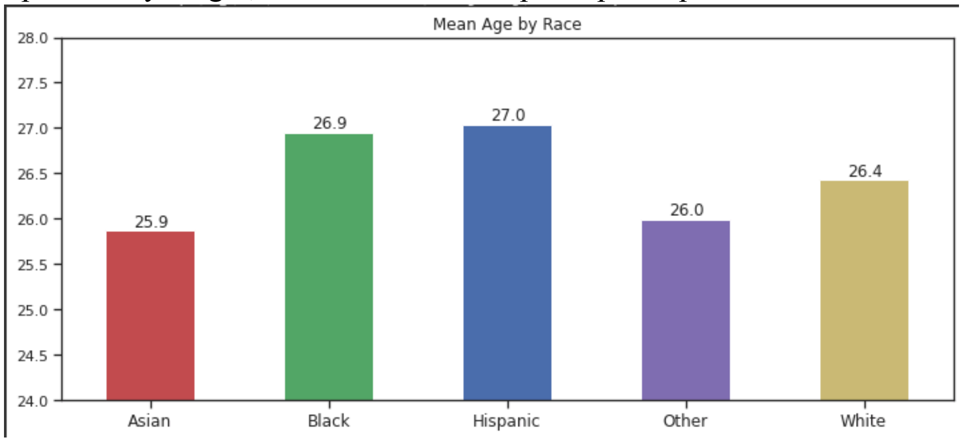


I decided to take a quick look to see the race composition of the experiment and saw that participants were mostly Caucasian followed by Asian.

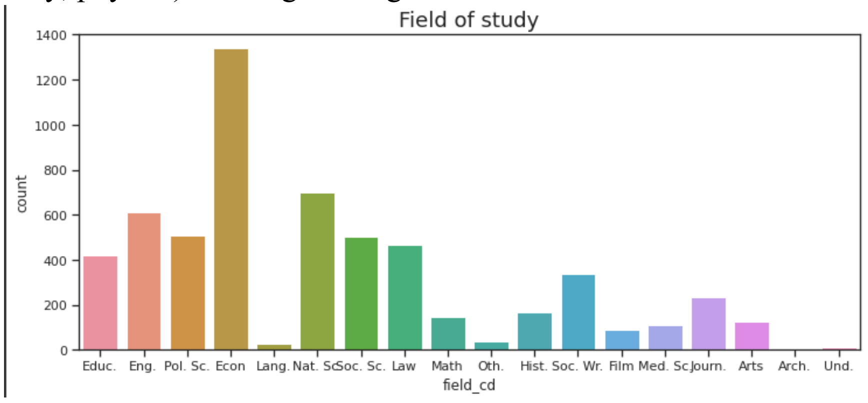


```
White       3313
Asian       1424
Hispanic     483
Other        350
Black        294
NaN           45
```

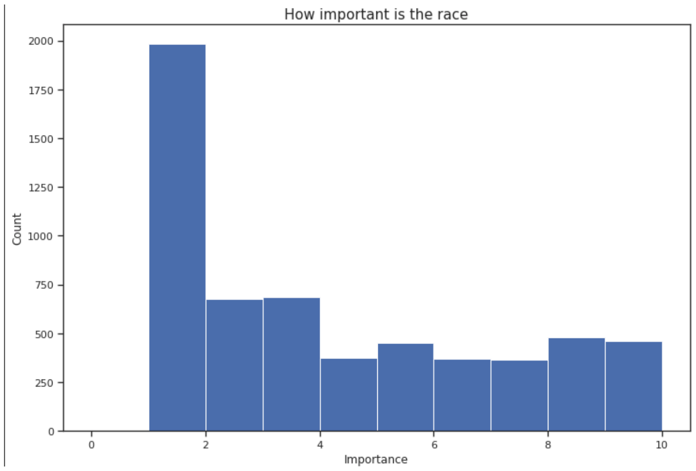
A further look into race and age shows the mean age for every race. Here we can see that the Asian participants are younger, while black and Hispanic participants are older on average.



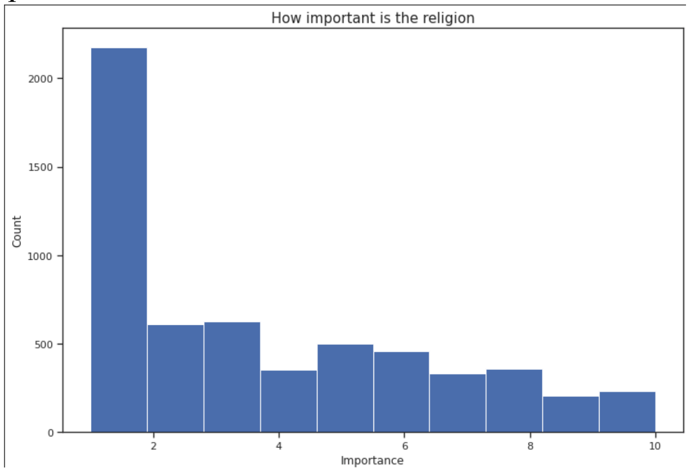
I would assume that one's career interest would be a significant factor in possibility of a match. Looking at fields of study, a great majority of business/finance/econ, followed by natural sciences (chemistry, physics) and Engineering.



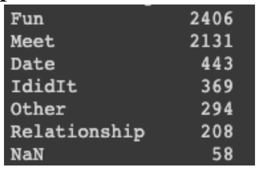
I wanted to see if race was a significant factor regarding a successful match. I used the ‘imprace’ column which asks the participants “How important is it to you (on a scale of 1-10) that a person you date be of the same racial/ethnic background?” the graph below show that most people do not think race is important.



I also wanted to check to see how important religion is. Similar to race it looks like most people do not think race is important.

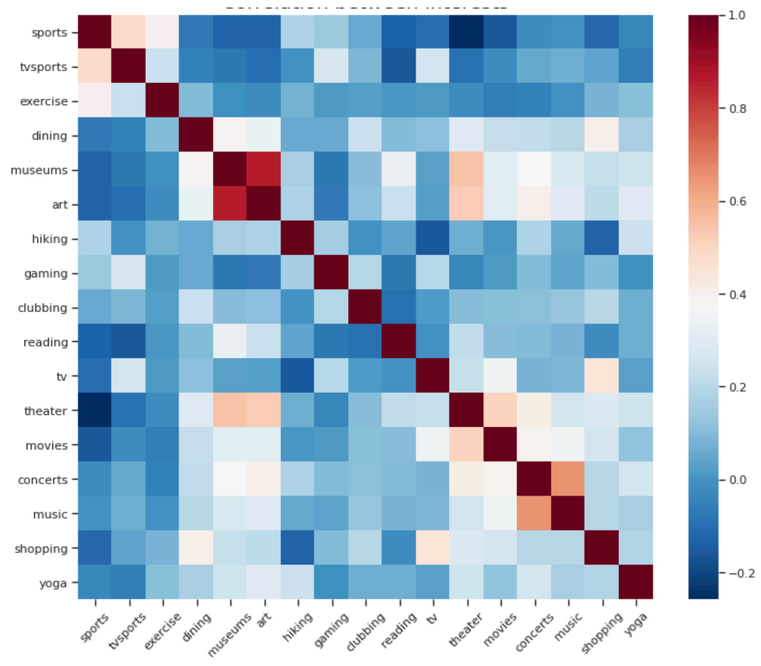


I wanted to look into people’s goals of out this speed date session. Most of the people declare to be there to have fun and meet new people.



```
Fun             2406
Meet            2131
Date             443
IdidIt           369
Other            294
Relationship     208
NaN               58
```

Looking into people’s interests: There is not much going on here, I notice that not surprisingly liking art comes with liking museums, liking music with liking concerts, and liking sports with liking watching sport.



Problem Formulation

There two major problems to improve upon, first is dealing with missing values as well as unbalanced data. For every chosen model, I picked certain hyperparameters as well as preprocessing steps to compare them between different types of searches. AKA the same model with the same hyperparameters (each time) was ran with three different searches to observe performance score. After best parameters are obtained for each model, I will use those to run our 5 models.

1. **MODELS**: XGBClassifier, Random Forest, Linear Regression, SVC, and MLP.
2. **SEARCHES**: Grid, Bayes, and Random search.
3. **PREPROCESSING OPTIONS**: For every model I first ran the hyperparameters using the preprocessing pipeline (called Pipeline 1 in Jupyter code) provided to us as code (I considered this baseline for my models). The following are different preprocessing techniques applied to the final pipeline (keep in mind that sometimes I tried these one by one and sometimes I combined a few to observe performance).
    a. I extended the numeric imputer strategy to include mean, median, most frequent and constant to see which works best.
    b. I also decided to perform SMOTE to account for the unbalanced data. For SMOTE I used SMOTEENN from the imblearn library which conducts over sampling using SMOTE and cleaning using ENN ("Combine over- and under-sampling using SMOTE and Edited Nearest Neighbours"). I used different parameter searches to look for best SMOTEEN strategies for the model.
    c. Results are cross validated.
    d. I also applied different Imputers such as MinMaxScaler(), OrdinalEncoder() to compare to StandardScale().
    e. I also applied TruncatedSVD() as an alternative to PCA(). TruncatedSVD() is a dimensionality reduction algorithm using truncated SVD and performs

dimensionality reduction by means of truncated singular value decomposition (SVD).

Results and Observations

XGBClassifier

- **Model hyperparameters**: I chose n_estimators since this is the number of trees I want to build before taking maximum voting. I chose max_depth because deeper trees can model more complex relationships by adding more nodes (though it may overfit). I chose min_child_weight because a smaller weight allows the algorithm to crease "children" that correspond to fewer samples, thus can allow for more complex trees (may overfit). The last two parameters can be used to control the complexity of the model and there represents a good trade-off between bias and variance (I used the same concept for my random forest model). I chose sub_sample and cosample_bytree because it helps control the sampling of the dataset so instead of using whole training set every time, a tree is built on slightly variable data each time, decreasing the chances of overfitting to a single sample. I chose eta and learning_rate because is controls the learning rate. This makes the model less prone to overfitting to have a lower eta and learning rate.
- **Preprocessing Methods**: Methods explained above
- **Model performance**: This model performed best with Grid search, though it took the longest as well. The fast search was Bayes and then Random. The following are a list of best parameters obtained using grid search. SMOTE actually slightly decreased the performance of the model and adding more parameters up to a certain point did not make. Significant different. For example, I added more parameters such as eta, gamma and learning rate however results were consistent and did not improve significantly. Model in general ran quite faster with grid search when SMOTE was implemented. This model also resulted my highest score on the public leader board (without using SMOTE). In order the model performed best with grid, bayes and then random search. The following best parameters were obtained using grid search. An interesting note is that my best XGBClassifier Grid search found the best Imputer strategy to be 'most_frequenct' which resulted in my highest performance. I also noticed that using RobustScaler instead of StandardsScaler improve the score. I used gridsearch to also search for best SMOTEEN strategies for the model and found that the 'minority' resampling performed best here. Overall, TruncatedSVD() decrease model performance significantly.
    - best score {'my_classifier__max_depth': 5, 'my_classifier__n_estimators': 100, 'preprocessor__num__imputer__strategy': 'mean'}
    - best score {'my_classifier__colsample_bytree': 1.0, 'my_classifier__gamma': 0.5, 'my_classifier__max_depth': 5, 'my_classifier__min_child_weight': 1, 'my_classifier__n_estimators': 100, 'my_classifier__subsample': 0.8}

Random Forest

- **Model hyperparameters:** I tried model hyperparameters such as n_estimators, max_depth, and max_features. I chose n_estimators since this is the number of trees I want to build before taking maximum voting so intuitively having higher number of trees would improve the performance (though at a cost of time!). I chose max_depth because it indicates the depth of each tree and its default is set to none, I wanted to try different variation since as you increase max_depth you tend to increase variance resulting in a decrease in bias, so to decrease my variance I set different values for max_depth. Then I used min_sample_leaf cause as you increase it you decrease variance and increase bias, this is why I decreased max_depth and increased min_sample_leaf (to increase regularization). I chose max_feature because this is the number of features that are considered on a per split level, this will make it a "true" random forest as features are randomly selected from the entire features.

- **Preprocessing Methods**: Methods explained above
- **Model performance**: Random forest had the best results with Bayes search. It also wa the fastest using Bayes and slowest using Grid search. Overall this model did not perform as well as XGBClassifier (even with its best parameters). Below are the best parameters selected for random forest using Bayes since that was its highest performance. One noticeable factor was that the numeric imputer strategy picked 'mean' instead of 'median'.
    - best score {'my_classifier__max_depth': 10, 'my_classifier__max_features': 'auto', 'my_classifier__min_samples_leaf': 4, 'my_classifier__min_samples_split': 2, 'my_classifier__n_estimators': 700,

Linear Regression

- **Model hyperparameters**: I chose the parameter C to inverse the regularization parameter which helps retain strength modification by regularization. For small values of C, we increase the regularization strength which will create simple models which underfit the data. For big values of C, we low the power of regularization which implies the model is allowed to increase its complexity, and therefore, overfit the data. I chose penalty (L1, L2) because it helps create a good fitting model.
- **Preprocessing Methods**: Methods explained above
- **Model performance**: Logistic regression performed best with Bayes search. Though it did not perform as well as XGBClassifier and Random Forest. The wait times were much shorter for all three searches compared to other models (though that may be due to having less parameters compared to other models). I also noticed that SMOTE made logistic regression performance slightly better. Also replacing StandardScaler() with RobustScaler() slightly improved the results with Bayes search for logistic regression. Below are the best parameters selected for Logistic Regression using Bayes since that was its highest performance
    - best score {'my_classifier__C': 0.01, 'my_classifier__penalty': 'l2', 'preprocessor__num__imputer__strategy': 'mean'}

C-Support Vector Classification (SVC)

- **Model hyperparameters**: since SVC tries to find the best hyperplane to separate the different classes by maximizing the distance between points and hyperplane I decided to use kernel as one of the hyperparameters because it specifies the type of kernel to be used(for example for a linear kernel it means the decision boundary is straight line (though this is not common in practice). I also chose gamma which is a parameter for nonlinear hyperplanes (most likely our data); The higher the gamma value, it tries to exactly fit the training dataset (may lead to overfitting). I also added the parameter C which is the penalty parameter for the error term and it controls the trade-off between smooth decision boundary and classifying the training points exactly therefore increasing C values may lead to overfitting.

- **Preprocessing Methods**: Methods explained above
- **Model performance**: This model performed very similarly to logistic regression in terms of scoring. For this model, Grid search took the longest, and the followed by Bayes search and then random search. Interestingly this model took more time than all versions of my XGBClassifier. Also, this model has the best performance with Bayes search with the following parameters.
    - best score OrderedDict([('my_svc__C', 0.0012602593949011189), ('my_svc__degree', 8), ('my_svc__gamma', 2.285959941576884), ('my_svc__kernel', 'poly')])

Multi-Layer Perceptron Classifier (MLP)

- **Model hyperparameters**: I chose hidden_layer as one of the parameters because the represent layers between the input and output layer; increasing layers has the potential to improve accuracy as smaller values may cause underfitting. I chose the activation parameter because they were used to introduce non-linearity to models, allowing them to learn nonlinear prediction boundaries as our data more likely represents this shape rather than linear predictions. I chose the learning rate parameter because it defines how quickly a network. Updates its parameters. A low learning rate slows the learning process down but soothes the converging, larger rates speed up the process but may not converge. I chose alpha because different alphas yield different decision functions, this is a parameter for regularization that decreases chances of overfitting by constraining the size of the weights. I used solver because it represents the weight used for optimization.
- **Preprocessing Methods**: Methods explained above.
- **Model performance**: The first factor that I noticed that the long wait times for this model using the different searches. Grid search took the longest out of the three. The SMOTE function made the model unbearably slow so I had to give up on that ☹. MLP performed similarilry in terms of score to the other models, however when I added the preprocessing step TruncatedSVD the model performed dropped significantly. Below are the best parameter found using MLP and grid search (best score).
    - best score {'my_classifier__activation': 'relu', 'my_classifier__alpha': 0.05, 'my_classifier__learning_rate': 'constant', 'my_classifier__solver': 'sgd'}