Firstly, I want to see if I need to keep a variable for prediction or not.

**ID** – Unique identifier, I do not want to analyze this info. Perhaps there might be info within the ID numbers, but I want to eliminate this initially out of logic and import it later if I can see any potential in it. I do want to use the information to understand that the lower the number, the earlier the applicant is. And I later want to see if earlier applicants are more likely compared to late applicants.

**Office\_PIN** – If I consider this as a category, I want to check if different offices have different applicant success rates. So, I am going to consider their success rates (ratio of successes per total applicants in that office) for each PIN and if they differ by a large margin, then I can safely assume that Office\_PIN can be a feature that I want to implement.

Chart, scatter chart

Description automatically generated

However, as the image shows, there’s very little incentive to consider Office\_PIN as a factor that influences the target variable.

**Application\_Receipt\_Date** – As I earlier mentioned with **ID**, I want to check if a selected application/row belonged to an earlier applicant, they might have a higher chance of success. This is something I can check per day, per week, per year. But for the ease of my process, I will consider day-wise early applicants and see if there’s an insight that can be drawn from it. If not, I will further continue with week-wise and year-wise application dates.

Chart, pie chart

Description automatically generated

The insight shows a strong relevance that earlier applicants are generally highly successful.

**Applicant\_City\_PIN** – One of the earlier insights I wanted to check was the difference between **Office\_PIN** and **Applicant\_City\_PIN** to see if the distance between the applicant’s office and homeplace would add value in predicting the target variable better. However, I see missing values within both test and train datasets for this value and **Applicant\_City\_PIN** is a geographical mark that I cannot impute with fake values, and neither can I label-encode them since that would destroy the distance measuring capability. I’d like to explore and see if encoding them would help but for now, I am going to remove this feature because of its hard nature to impute like a user’s name

**Applicant\_Gender** – I will consider this as one of the features. It’s categorical and missing very few rows. My assumption for the missing data is that the survey includes a “Don’t wish to reveal” as a choice in the sex column of the fill-up form. Could be, couldn’t be. But since this value is missing in the test data too, I can either generalize it as Male, since it’s more common. Since there are a lot of other missing values, I can’t cluster them to predict them. So, my choices I am aware of, are either to choose Male or I can consider them as a third category. The missing roles being a third category would also assume Male if they are statistically similar, so third category is a safe way to go.

**Applicant\_BirthDate** – I used the applicant birthdate to create **Age**,but the distribution for the new feature didn’t change when compared when compared between the two target variable values, implying that **Age** is an unnecessary factor as well. Since **Age** is useless and there are missing values for **Applicant\_BirthDate**, I decided to drop the column because it’s not very logical to fill birthdays. Although I could bin these values to see if I can make further progress in the model. This is room for improvement.

**Remainder of Applicant features** – The last three features of the Applicant are categorical and have missing values and so they are label-encoded in the similar way as **Applicant\_Gender**. I have only later realized that a few groups in **Applicant\_Qualification** are similar and have potential to be grouped into a single category. This is another room for improvement.

Initial view of the manager details shows significant missing values in a few columns in both train and test data. The reasoning behind this could that they do not have a manager assigned, or that they have a manager assigned that doesn’t have their details stored. Either way, since a lot of this data is missing, I chose to consider the empty rows as a separate category like I did in **Application\_Gender**.

**Manager\_DOJ and Manager\_DoB** – These two features are very unique to a single manager, as in there’s very little possibility for two managers to be born on the same day as well as joining on the same day. Hence, the combination of both these values is going to be a unique identifier for each manager, that is, a certain unique **DOB\_DOJ** combination can be considered Manager A, and then another combination as Manager B and so on. This, I can further label-encode the combo feature **DOB\_DOJ** and drop these two columns as they again have a huge missing value data that cannot be guessed or filled.

**Manager\_Grade** – I have witnessed that the higher-grade managers have a high success rate among their applicants, so I could either consider a bifurcation, that is, applicants with managers’ grade higher than 5 are one group, and managers with less than or equal to 5 grade as another group, in the future.

After this, I encoded the rest of the categorical variables of the manager details. Coming further to the numerical variables:

**Manager\_Num\_Application and Manager\_Num\_Coded** – I would like to consider these two features after making the manager categories considering that these provide a significance on how many applicants the manager reviews. And the number of applicants he recruited as a deciding factor for his future recruits. The best approach would be to divide to see his selection rate to add further to the data. However, this is room for improvement.

**Manager\_Num\_Products2 and Manager\_Business2** – These values have a hidden feature, that is, business from their Category A advisor. To get these values, I am going to subtract these values from the total values of their similar features **Manager\_Num\_Products** and **Manager\_Business** respectively. I also realized that **Manager\_Business** values, despite being very high cannot be scaled logarithmically because of the negative values in it, so I decided to be safe and just scale them using StandardScaler, along with the other features.

Since I now have my features ready to go through model testing, I have chosen Logistic Regression, Random Forest, GaussianNB and Decision Tree learning models to find the model with the best ROC score. For this, I have used filler code that I previously used.

The working of the filler code is simple.

1. All the four models go through cross\_validation with 5 folds and their ROC scores are recorded.
2. After selecting the models with highest ROC scores, they are tested with various hyperparameters of the model using GridSearchCV.
3. After identifying the best model and best hyperparameters, the scaled train data is used to train the model.
4. The final model is run against the scaled test data to predict the probabilities for each applicant.

This approach should result in an ROC score of approximately 0.87 on the public test dataset.