**Introduction**

In this report, we will outline the steps we took in using Logistic Regression, Support Vector Machine, Decision Tree and K-Nearest Neighbor to predict whether a particular client subscribes and the approaches we took to improve the accuracy of these models. We will then discuss the strengths and weaknesses of each approach and explain how we settled on our final model. We also used the final model to predict subscription in an unlabeled file.

**Summary**

To prepare the data for training on the models, we first had to deal with the unknown values in the dataset. We imputed the missing values, inferring what the values would be based on the other column values. We also performed binning on the *poutcome* column to handle the unknown values. In order to decide on the best model to use, we trained the dataset on several machine learning models to determine which model could describe the data’s function best. We looked at the classification report and confusion matrix for the statistics (precision and recall) to help us determine the goodness of a model.

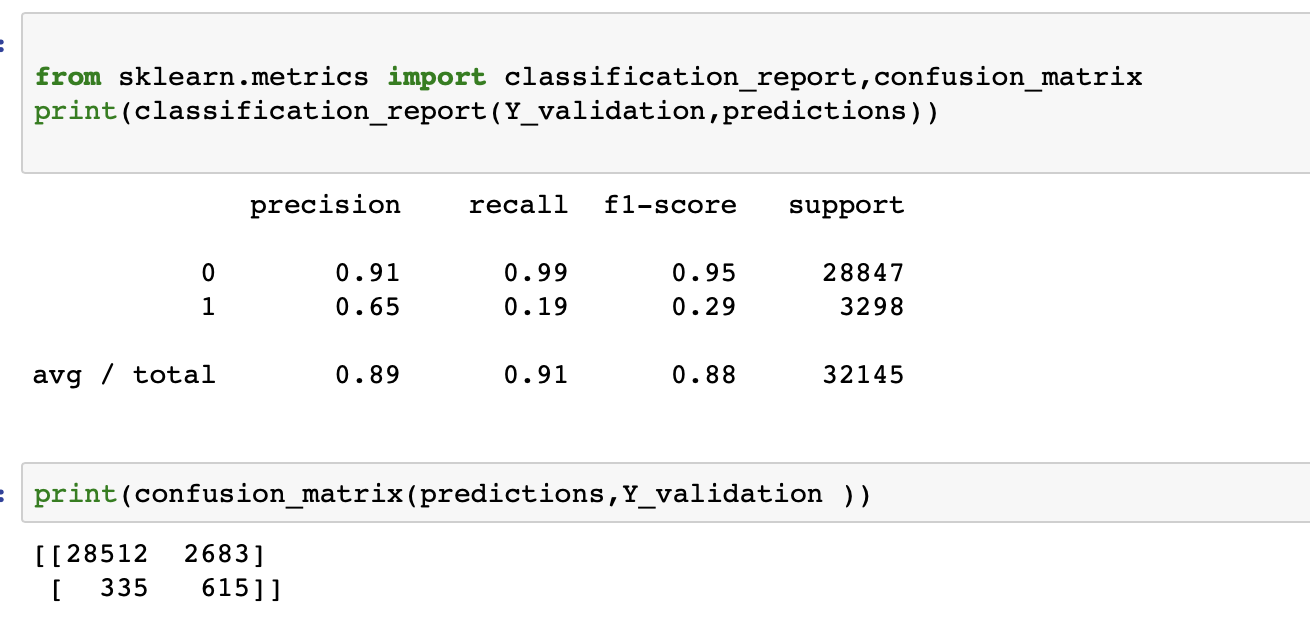
**Experimental Details**

We were provided with 4 files - ‘*data.csv*’, ‘*description.pdf*’, ‘*futures.csv*’, ‘*Project.pdf*’ and we used the libraries - ‘*pandas*’, ‘*numpy*’, ‘*seaborn*’ and ‘*scikit-learn*’ to perform exploratory data analysis and implement these models on Jupyter Notebook.

**Results & Discussions**

Logistic Regression

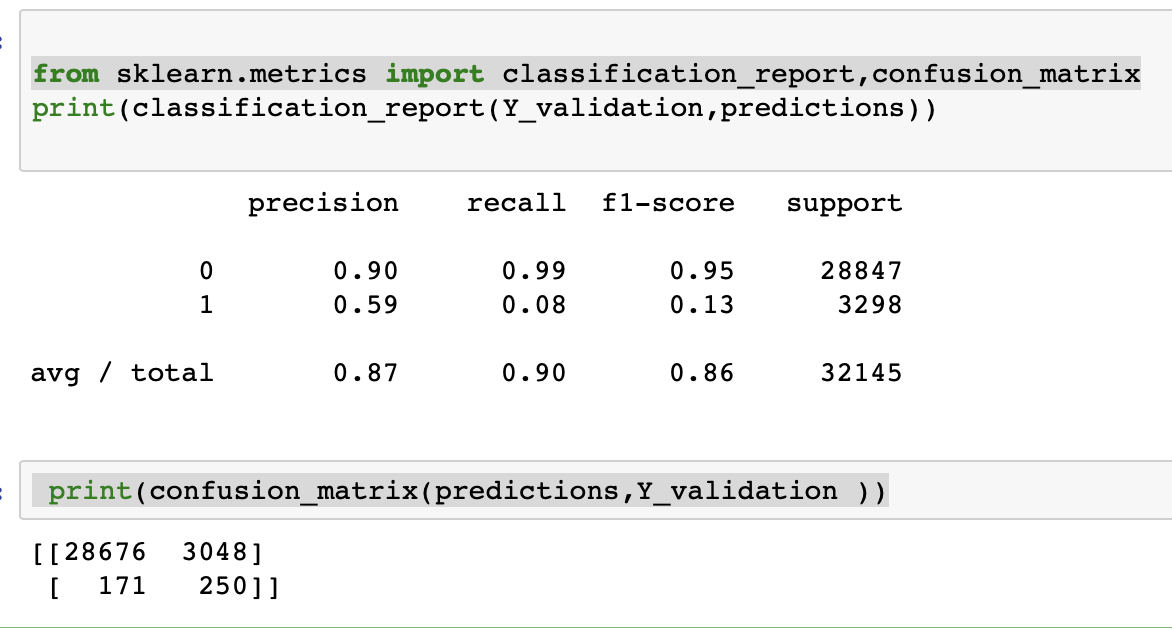
When the data was trained on a logistic regression model, we can see that the precision of the *Yes* value is 0.65 and the recall 0.19. This model has a good precision but the recall of 0.19 means that 80% of the predicted *Yes* values are false positives.



*Fig 1. Logistic Regression Classification Report*

SVM

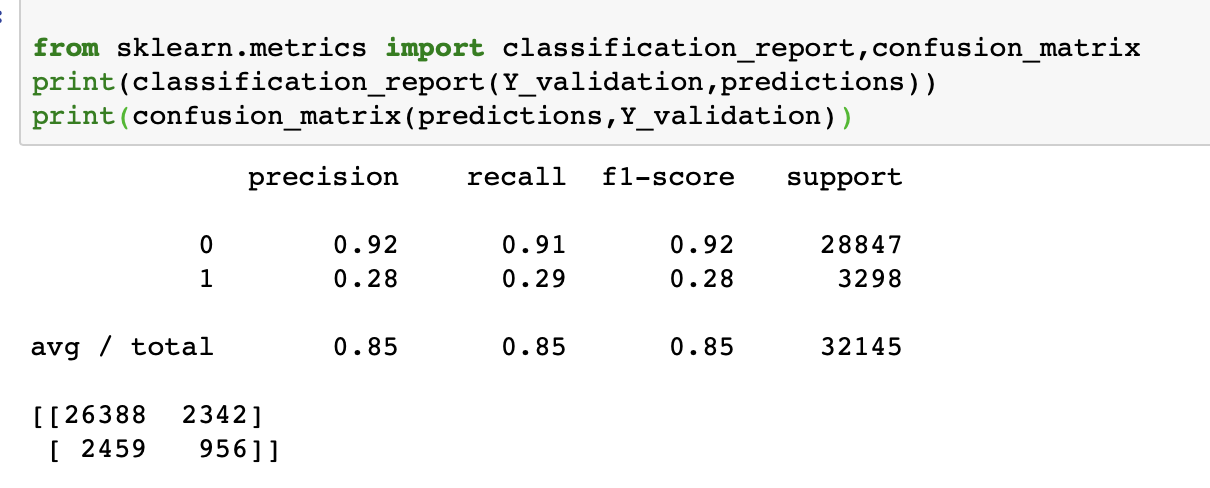
When we used Support Vector Machines to classify the data, we can see that for the *Yes* values, it has a decent precision of 0.59 but a very low recall of 0.08. These statistics suggest that SVM would not be a good model to use for prediction because it doesn’t capture the *Yes* values in the dataset well.

**

*Fig 2. Classification Report for SVM*

Decision Tree

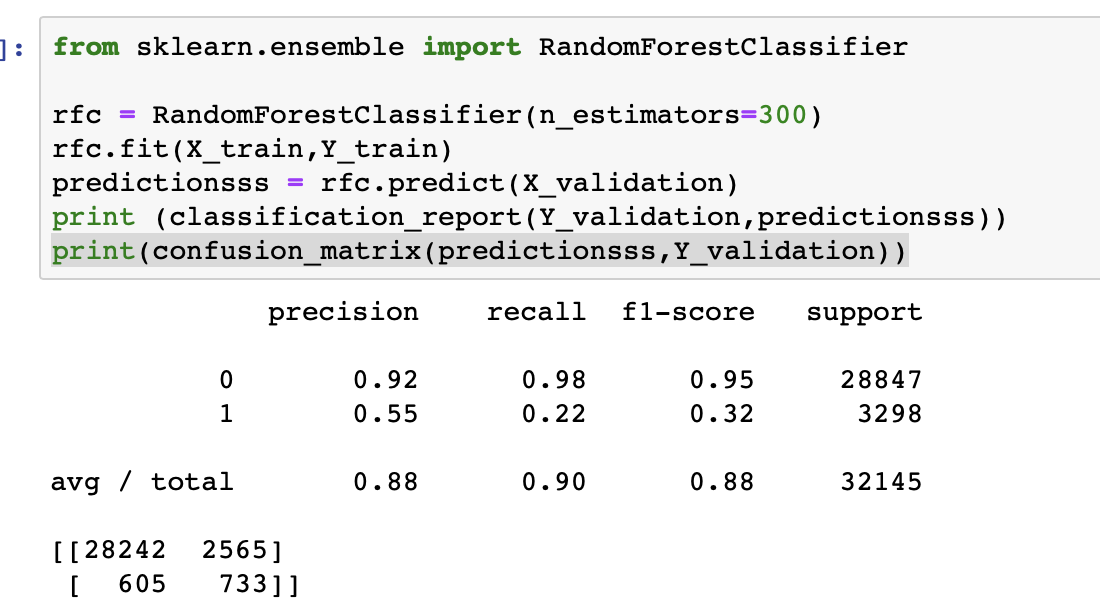
The data was then trained on a decision tree which resulted in a precision of 0.28 and recall of 0.29 for the *Yes* values.



*Fig 3. Classification Report for Decision Tree*

Alternative Approach: Random Forests

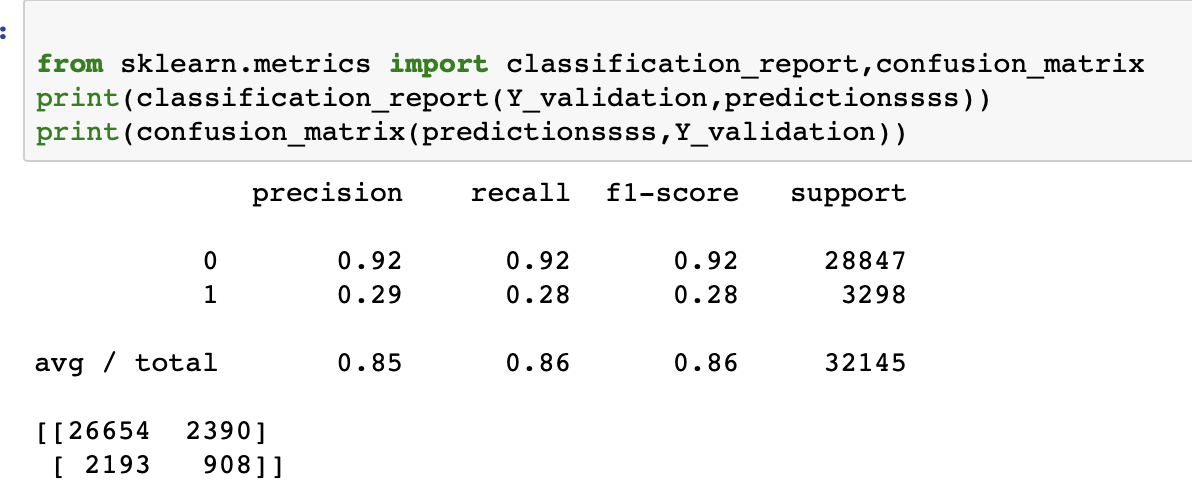
We tried to improve the performance of the decision tree by using random forests. By taking the majority vote from a multitude of decision trees, it reduces the bias from a single tree and gives us a better result. From this model, we got a precision of 0.55 and a recall of 0.22 for *Yes* values.



*Fig 4. Classification Report for Random Forests*

K-Nearest Neighbors

The last model we trained the data on was K-Nearest Neighbors which gave a precision of 0.29 and a recall of 0.28 for Yes values. The recall is pretty high among the models that we trained the dataset on but the low precision means that the majority of the Yes values predicted by the model are false positives.



*Fig 5. Classification Report for K-Nearest Neighbors*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Summary | **Precision** | | **Recall** | |
| *no* | *yes* | *no* | *yes* |
| Logistic Regression | 0.91 | **0.65** | 0.99 | 0.19 |
| SVM | 0.90 | 0.59 | 0.99 | 0.08 |
| Decision Tree | 0.92 | 0.28 | 0.91 | **0.29** |
| Decision Tree (Random Forest) | 0.92 | 0.55 | 0.98 | 0.22 |
| K-Nearest Neighbors | 0.92 | 0.29 | 0.92 | 0.28 |

*Table 1. Summary of results*

From the table summarising the results that we had gotten from training the models, we can see that the precision and recall of *No* values in the output is pretty good, with all scores > 0.90. On the other hand, we can see that the precision and recall values of *Yes* values fluctuates greatly between models.

A reason why the precision and recall of *No* values are so high is because the data is skewed in terms of the number of *Yes* and *No* in the dataset. We have 89.8% of the data being *No* which means that if the output that the model returns mostly consists of *No*, the Precision (# of True *No* / # of Total Predicted *No*) and Recall (# of True *No* / # of Total Actual *No*) would both have really high scores. On the contrary, a smaller number of wrong predictions or missed predictions could vary the Precision and Support of the *Yes* result more greatly.

To determine the ability of the model to correctly predict *Yes* values given a new dataset, we can look at the Precision and Recall of *Yes* values of the different models. First, we look at the Precision column. Precision, in this case, refers to the total number of correctly predicted *Yes* values over the total number of values which had been predicted to be *Yes*. This metric can also be interpreted as how frequently the model gets the correct answer when it predicts *Yes*. It is important that this number is high because it shows that the model is not blindly predicting *Yes*.

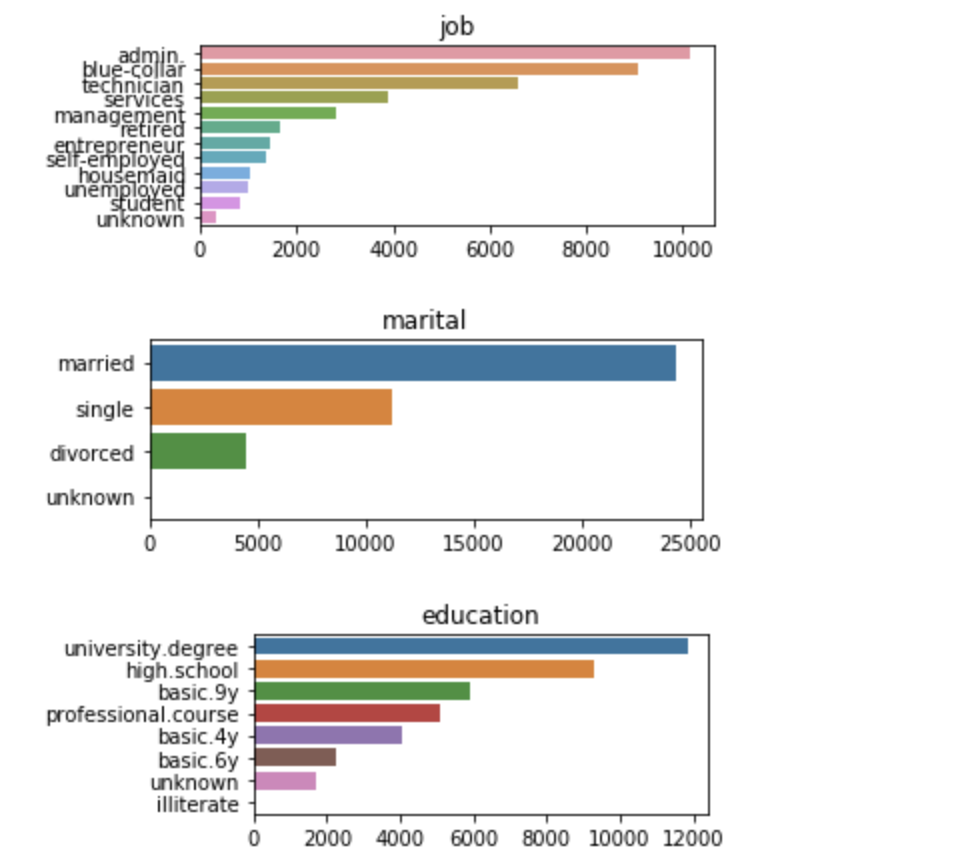
Secondly, the Recall column, which in this case refers to the number of correctly predicted Yes values over the total number of values which are actually *Yes*. In other words, how good the model is at capturing the *Yes* values given a dataset. A recall value of 0.20 means that in a dataset with 100 *Yes* values as the output, 20 of those *Yes* will have been correctly predicted by our model. We view this metric to be an integral part of the accuracy of the model and we want the value to be high as possible for our final model.

**Data Pre-Processing**

Unknown Values in Data

In our initial exploration of the data, we found that many of the values in columns are unknown. One of the ways to deal with missing data is to discard the rows that have unknown values in them. However, this would lead to a reduction in the size of the usable dataset and given the relatively small dataset (~40k rows), we would like to avoid having to do this.

Another way we can deal with the unknown values is to cleverly infer the value of the unknown value based on other variables. This is a way of performing an imputation where we make use of other known variables to infer the value of the missing variable. This method does not guarantee that all missing values will be addressed but this way, most of them will have a reasonable value which can be used in the prediction.

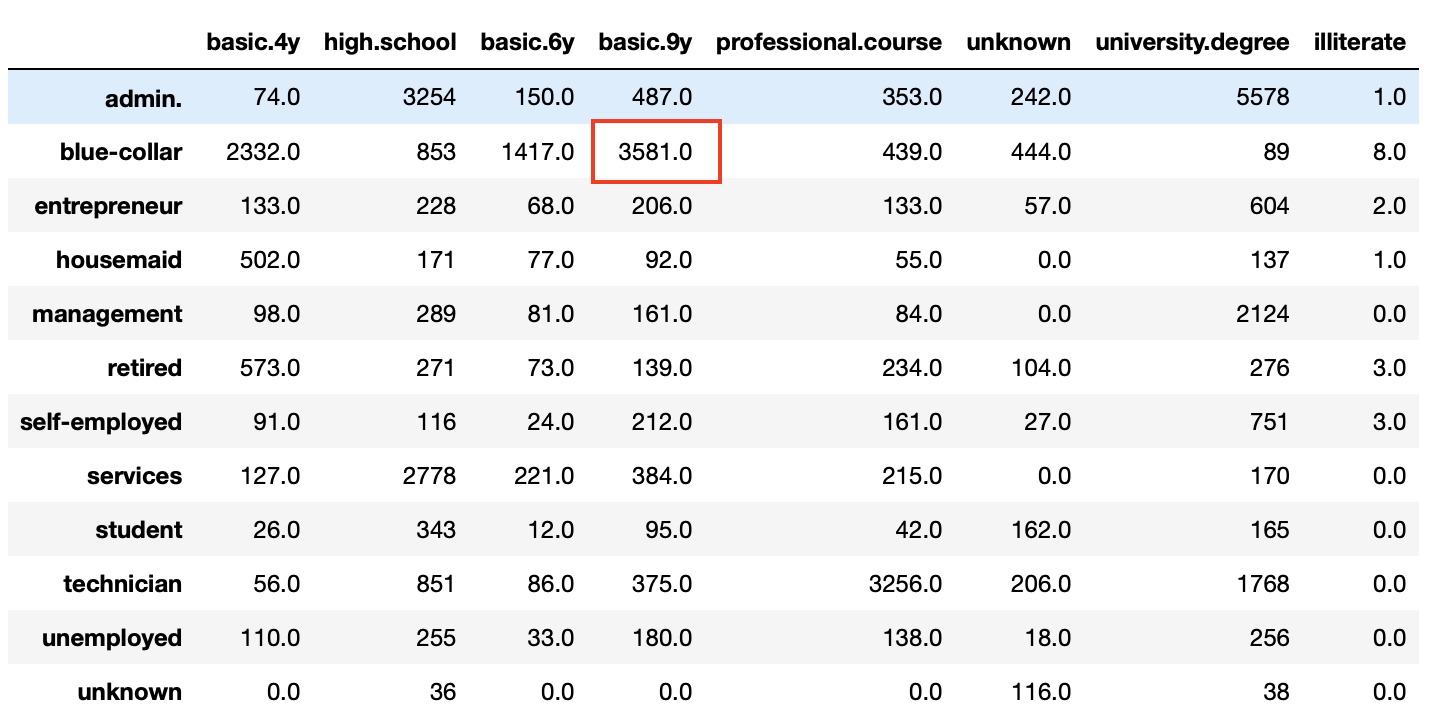


*Fig 6. Exploration of Values in Job, Marital and Education*

The variables with unknown/missing values are: *‘education’*, *‘job’*, *‘housing’*, *‘loan’*, *‘default’* and *‘marital’*. Out of these variables, the number of unknowns for the first four are significant. The unknown values in ‘default’ variable are understandable because the client might not be willing to disclose this information to the banking representative, thus we can leave it as it is. Also, the number of unknowns for *‘marital’* is very low, so we can just remove those rows.

Imputations

Hence, to get a sense of the relationships between classes in each variable, we created a column to count the number of unknowns given a class in a variable (eg. *‘blue-collar’* in *‘Occupation’*) and if the number of unknowns is significantly greater than the others, we would posit that a relationship exists between those two classes (eg. *‘blue-collar’* and ‘*basic.9y*’ as seen below).



*Fig 7. Cross Tabulation of Occupation and Education (Number of Unknowns)*

The first two variables that we imputed for were *‘job’* and *‘education’*. The underlying assumption is that the *‘job’* of a client is affected by his/her ‘education’. An example of the imputation we did is filling in unknown occupations for clients with education levels of ‘*basic.4y*’, ‘*basic.6y*’ or ‘*basic.9y*’ with ‘*blue-collar*’, which makes sense because people without ‘*university.degree*’ tend to become blue-collar workers. Since the unknown values that we are filling in are a small subset of all the data, we do not need to worry about correlation between the two variables and we can continue to use the variable to predict *y*.

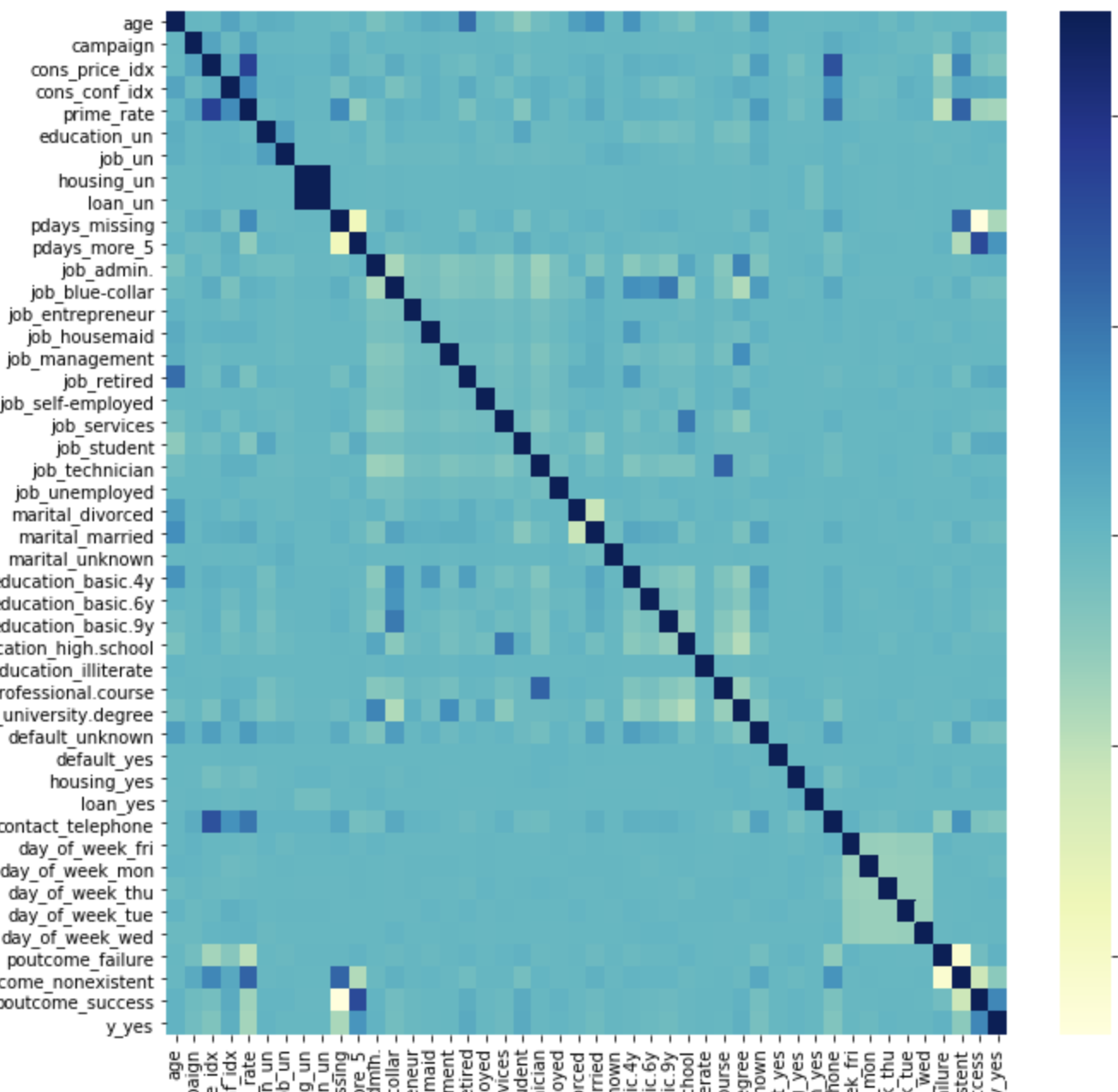
We also imputed the unknown values of *‘house’* & *‘occupation* and *‘loan’* & ‘*occupation’*. The underlying assumption is that the distribution of people who have and do not have loans that belong to a certain class remains the same. For example, we take rows with known loan values and we see that 60% of the clients with *occupation* = *‘admin’* have taken loans. To fill in the unknown values, we assume that in general, 60% of the clients who have *occupation* = *‘admin’* have taken loans and we randomly assign ‘yes’ and ‘no’ values to the unknown values such that the 60% proportion is maintained. In the same way, by maintaining the distributions we filled in the unknown values in *‘house’* as well.

Binning

We observed that the majority of the values of pdays is unknown. Most of these unknown values occur when the *‘poutcome’* is null. This means that the majority of unknown values in pdays are unknown because the customer was never contacted before. To deal with the unknown values in this variable, we replaced the numerical continuous variable pdays with a categorical variable with the following categories: *p\_days\_missing*, *pdays\_less\_5*, *pdays\_bet\_5\_15*, and *pdays\_greater\_15*.

**Feature Selection**

In order to determine which variables should we included in the model, we determined the correlation between each variable to the final output *y*. In order to do so, we plotted a normalised distribution of the classes within each feature and from there, we could see the difference between the positive and negative correlations. A positive value implies that a client with that class as an attribute are more likely to subscribe and negative values implies otherwise. The formula we used to calculate the correlation value is as follows,



*Fig 8. Correlation between variables (Blue is Higher)*

From the above heat map we can see that result has good correlation with  **‘*prime\_rate*’, *‘poutcome\_success’* and *‘pdays\_bet\_5\_15’***. We should pay more attention to these for building models.

**Conclusions**

With all the factors considered, we decided on the Logistic Regression model. It had the highest Precision out of all the models and its recall was only slightly less than the other models. We believe that this model performed well because it was trained on enough data that a reasonable threshold was discovered. Depending on which side of the threshold each row of data falls on, the output would be a *Yes* or a *No* and it seems that this kind of classification describes the problem space pretty well.

**Recommendations**

Our recommendation to improve on this project would be to obtain domain knowledge about this problem. Having this knowledge about the problem would allow us to narrow down the variables needed in the model since we may be able to infer some sort of relationship between variables by knowing more about the problem. Reducing the number of variables in the model would help with preventing overfitting and also reduce dependence between variables.

Domain knowledge would also allow us to craft better features that deal with/represent the problem more specifically. By combining two variables into one, the data may become more easily separated along this axis.

**Appendix**

For more information about the code used in generating these models and graphs, please refer to the iPython Notebooks attached in the zip file.