CISC-5790 Course Project
César A. Vargas-Pérez & Fernando Martínez
A15822925 & A19737173

Predicting Term Deposit Subscriptions with Bank Marketing Dataset

Introduction

Marketing is a vital part of almost every business, since it is the sum of activities involved in directing the flow of goods and services from producers to consumers[1] and because of this, many companies decide to do what is known as a marketing campaign. According to E.Taver[2], marketing campaigns are often used by companies to help promote products to current and future clients and this also helps companies that lose sales due to major negative press often use marketing campaigns to rehabilitate their image.

Due to the importance of marketing, we decided to use a bank marketing dataset to create a model that can predict if a certain client is likely to open a **term deposit** account or not. For context, in banking a term deposit account is a type of account held at a financial institution where money is locked up for some set period of time, that pays the depositor compensation in the form of interest on the account balance[3].

During this project, we play the roleplaying game of working on a team developing a model to set it into a production environment in the banking market.

Data source details

We extract this dataset from Kaggle under the name of *Bank Marketing Dataset.* As stated in the summary of the dataset by J. Martinez-Bachmann[4], *This is the classic marketing bank dataset uploaded originally in the UCI Machine Learning Repository. The dataset gives information about a marketing campaign of a financial institution in which you will have to analyze in order to find ways to look for future strategies in order to improve future marketing campaigns for the bank.*

Said dataset contains 17 features (1 of which will be our objective variable), and 11,162 observations. More details about each feature and what it means can be found in the appendix table 1.

Exploratory Data Analysis (EDA)

Before starting to create our model, we did an exploratory data analysis to check if there are any patterns, anomalies, and possible outliers.

Our process was the following (in order):

1. Checked for missing values
    - We found out that this dataset does not contain any missing values (*nulls*) at first glance.
2. Checked for the unique values of the categorical features.
    - Something interesting is that this data seems to be pretreated, and possible missing values were transformed into a category called unknown, or from where it was extracted all missing values were imputed with the default value *unknown*.
    - Result of this can be found in the appendix figure 1.
3. Visualized how the objective variable behaves across 4 features that at this stage we considered important to affect if a client opened a term depositing account or not.
    - We plotted a bar chart per feature, with are:
        i. *Job* (fig. 2)
        ii. *Marital Status* (fig. 3)
        iii. *Education* (fig. 4)
    - We also plotted a histogram with *age* (fig. 5) towards looking at its distribution splitted by if the client deposited or not.
    - We noticed that the objective variable is almost equally distributed in most of the cases per every category per feature.
4. Plotted pairwise relationships in a dataset of numerical features against the Y (fig. 6)
    - This showed us the relationship between all the continuous variables in our dataset, observed by our objective variable.
5. Checked the correlation between numerical features (fig. 7)
    - From this we can clearly see that the features *previous*[1] and *pdays*[2] have a positive correlation.

previous[1] : Number of contacts performed before this campaign.
pdays[2]: Number of days that passed by after the client was last contacted from a previous campaign.

6. Finally, we wanted to check if the dataset is balanced and found that it's relatively balanced since 52.6% of the data corresponds to 'no' and 47.4% of the data corresponds to 'yes'. (Actual values can be seen in fig. 8)

Data Modeling

After the EDA process, we had an idea of how our dataset looks, so we proceeded to get our hands dirty and started to work on it.

As a summary what we did was:

1. Split the dataset into two.
2. Scaled continuous features.
3. Imputed values
4. Encoded categorical features.
5. Selected a model based on the results of step 3 and 4.
6. Balanced the data
7. Hyperparameter tuning
8. Feature selection / relevant features by models.

For the model, we decided to use the following algorithms:

- K-nearest neighbors (k-NN)
- Logistic Regression*
- Naive Bayes
- Decision Tree Classifier
- Random Forest
- Ensemble Model applying a hard-voting (mode of all predicted results), using the 5 models previously mentioned.

*In class we learned about linear regression, but since we are working with a binary classifier we decided to use a logistic regression which is similar since both the models are parametric regression i.e. both the models use linear equations for predictions according to S. Mondal[5].

1. Splitting the dataset

We created two new datasets:

- Train dataset
- Test dataset (in the code we named it *valid*)

To achieve this in an easy way we used the famous python package scikit-learn, specifically the `train_test_split()` method from `sklearn.model_selection`. Here we splitted the data in 80/20, 80% for training and 20% for testing.

2. Scaling continuous features

Since our algorithms can be affected by the value on the continuous features (k-NN is very sensitive to outliers), we decided to normalize said features using a Min-Max Scaler method. Using the package `sklearn.preprocessing` we used the function `MinMaxScaler()`, rescaling the dataset such that all feature values are in the range (0-1)[6].

3. Imputing missing values

In order to prevent our model from being affected by the "*unknown*" values that we found we decided to run multiple experiments, creating a copy of the splitted dataset to maintain the baseline dataset filled "*unknown*" values, and experiment with the new copy substituting the "*unknown*" with NA/NaN values and playing around with them using different techniques.

From what we learned in class, we decided to impute these values using three approaches: using the most frequent value (mode) and using two predictive models (k-NN and Decision tree approach).

One thing that we found out during our EDA stage is that the feature `poutcome` has almost 75% of it filled with "*unknown*", so we decided to use this in every imputation case as well as the creation of a helper new feature called `NotContacted` letting further models know if the client was reached or not (based on *pdays*[1]). Another thing that is worth mentioning is that the columns with NA values were (`['job', 'education', 'contact']`), so we applied each approach to only those.

pdays[1]: Number of days that passed by after the client was last contacted from a previous campaign. If a client has the value *-1*, it means the client has not been reached by any agent.

We also created a new dataset based on the result of every imputation process in order to analyze each experiment in a further stage.

3.1 Imputing using mode

According to what we learned in Lecture 6 by Dr. Zhao[7], using mean, median, or mode is one of the most frequently used methods, so we decided to use such a method.

After removing `poutcome`, we were ready to impute the data using the mode.
We selected the columns that had missing entries (`['job', 'education', 'contact']`) and used the method `SimpleImputer()` from the package `sklearn.impute` and applied `Fit_transform` and `transform` for the train and test dataset respectively.

3.2 Imputing using k-NN and Decision Tree

In these steps we basically did the same process for both imputation experiments, the only thing that changed was the algorithm used.

Here, in order to take more advantage of our classifiers we decided to use a technique called One Hot Encoding. According to A. Fawcett[8], *One Hot Encoding is one method of converting data to prepare it for an algorithm and get a better prediction. With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns*. For this we selected all the categorical features and transformed them into binary features.

Finally, we created a new binary feature per imputed column in which we marked as 1 if the value was imputed and 0 if not. We decided to do this as a way to tell the model to "remember" that the value was imputed.

Something to keep in mind for these experiments is that for each imputation process, we temporary imputed the rest of variables with missing values using the mode, and then predict the current feature we were working with.

4. Encoding categorical features.

Previously, we explained what categorical encoding means, especially One Hot Encoding. In this step, we did the same but for the whole dataset. We decided to do such a thing since we will use this temporary dataset to run our experimentation process. It is relevant to mention that after

encoding each value of each categorical feature, we renamed all these new columns following the structure of [`FeatureName_Value`]. After running the several previously mentioned experiments, the winner dataset will be used in our further experiments.

5. Selecting the Best Feature treatment setup based on the imputing process

In this step, we are going to measure how good our model will behave based on the preprocessing of the data that we did. For this, we used a cross validation using 10 folds,and since this is a supervised learning model, we are using the external quality criteria that we learned in previous lecture Precision-Recall, espifically F1, Accuracy, Precision and Recall.

Let's recall that marketing campaigns are often used by companies to help promote products to current and future clients and this also helps companies that lose sales due to major negative press often use marketing campaigns to rehabilitate their image.

Because these decisions set in production could affect future investment led by new operations and marketing activities, we care about both *Precision* and *Recall* since the first one measures the percentage of the clients predicted to make a deposit that were correctly classified, and the second shows the percentage of actual clients that made a deposit that were correctly classified.

Given that the F1-score is the harmonic mean of Precision and Recall, we are going to focus on this metric to decide which model is best. The results were the following:

1. For the original dataset with "*unknown*" values (baseline dataset), we found out that the best algorithm is Random Forest (F1: 0.8513), followed by ensemble (0.8190). (Figure 9)
2. In the model with imputed values using the mode, Random forest also got the best F1-score (0.8424) (Figure 10)
3. With values imputed using k-NN, random forest was the best choice (Figure 11) as well as in the Decision Tree (Figure 12)

After we did that, we decided to compute an average of every approach to see which imputation method obtained F1-score. We found out that the imputation with *"unknown"* (original dataset), on average, is the method that has a higher external quality (Figure 13).

6. Balancing the data

The dataset that we chose is by no means highly imbalanced. However, we decided to run some experiments to check how our models behave after balancing the data using various techniques to prevent the learning algorithms from being more biased towards the majority class. Thus, we want to prevent our model from misclassifying the minority class, which in our case is "yes."

For this, we used the following techniques learned in class:

- Undersampling: select a subset of the majority examples to match the number of minority examples to create a balanced dataset[7]
- Oversampling: Select the minority examples multiple times to create a balanced dataset[7]
- SMOTE: This is a better version of oversampling and means "Synthetic Minority Over-sampling Technique"

To achieve this, we used a python package called `imblearn` (imbalanced-learn) which contains the following functions: `RandomUnderSampler()`, `RandomOverSampler()`, and `SMOTE()`. For SMOTE, since it is based on k-NN, we randomly selected 7 as the k.

To remember what we did in section 5, on average our Imputer with "unknown" is slightly the best method to impute unknown variables(figure 13). With that being said, we used this method with each algorithm and tested them, following a greedy approach.

We obtained similar results of best F1-scores using Random Forest. Apply undersampling, results of can be seen in figure 14, oversampling in figure 15, and SMOTE in figure 16.

After applying the balancing techniques, we measured how good the F1 score is against each one and what we learned in lectures is confirmed, SMOTE was indeed the best technique to handle our dataset, even not being highly imbalanced (results can be seen in figure 17).

Because of this, we applied SMOTE to the dataset that we used in further steps.

7. Hyperparameter tuning

In this stage, we wanted to make our model even better, so we resorted to a technique called hyperparameter tuning. This was subtly covered in the lectures, applying cross-validation, since this is often used in more advanced machine learning models, but it's a good practice to do. According to the AI Platform from Google, *hyperparameters are the variables that govern the training process itself. These are tuned by running your whole training job, looking at the aggregate accuracy, and adjusting. In both cases you are modifying the composition of your model in an effort to find the best combination to handle your problem* [9].

During this experimentation step, we decided to use a different way to make our K-Folds in cross-validation and decided to implement Stratified K-Folds, to preserve the class distribution balanced for our objective variable at each fold. All these experiments were developed using a grid search algorithm for hyperparameter optimization.

After running this step (more details in the notebook attached in the report), we obtained the best parameters to use at every algorithm (see figure 18).

8. Feature selection / relevant features by models.

In this final step before running our models on our test data, we evaluate which features are the best to describe our target variable, or use which specific features our models perform the best. It is good to remember that we used one hot encoding and that process added more features to the model, but those new features might be relevant or not. Here we reviewed this.

For k-NN and Naives Bayes, we used the `SequentialFeatureSelector()` method `from sklearn.feature_selection` and configured forward direction, which is the wrapper method using "Sequential Forward Search". This starts with no features greedily including the most relevant feature and stops when selected the desired number of features[7].

In figure 19 we can see the top 25 features selected for k-NN and for Naives Bayes in figure 20. In both , the features are not ordered by importance since the process in which we did it doesn't have estimators, but we got the best performance using the extracted features.

For Logistic Regression, Decision Tree, Random Forest we used a filter method approach, using `RFECV()` function `from sklearn.feature_selection.` This will rank features independently of the algorithm and select the top ranked features using a recursive feature elimination with cross-validation to select the number of features[11].

From this, we observe that:

- Logistic regression, we obtained 42 ranked features (figure 21). For this model we found that the top most relevant features are: *duration, campaign, poutcome_success, month_mar, month_dec, month_jan, balance, month_oct, contact_uknown, previous,...*
- Decision Tree, 45 ranked features (figure 22). For this model we found that the top most relevant features are: *duration, poutcome_success, contact_unknown, housing, month_aug, month_jul, loan_no, day, pdays,…*
- Random Forest, 50 features (figure 23). For this model we found that the top most relevant features are: *duration, balance, age, day, campaign, poutcome_success, pdays, contact_uknown, previous, housing, month, education,…*

A relevant insight we get from this is that for these 3 algorithms, the feature *duration*[1] is the most important by far, and also is present at the k-NN's list and the Naive Bayes' list . The observed information denotes that this attribute highly affects if a customer will deposit or not, in summary, followed by *month*[2], *poutcome_success*[3], *campaign*[4], *housing*[5], *previous*[6], and so on.

Duration[1] : last contact duration, if the customer was contacted previously. *Month*[2]: last contact month of the year. *Poutcome*[3]: outcome of the previous marketing campaign. *Ccampaign*[4] : number of contacts performed during this campaign and for this client. *Housing*[5] : has a housing loan?
Previous[6] : number of contacts performed before this campaign and for this client.

9. Metrics on Test Data

In the banking industry, marketing campaign results lead to new marketing and operational activities, which could translate into the need for future investments to retain reached customers. Because of that, we care about both *Precision* and *Recall* since the first one measures the percentage of the clients predicted to make a deposit that were correctly classified, and the second shows the percentage of actual clients that made a deposit that were correctly classified.

With that being said, we tested out all of the possible scenarios we prepared to work with completely unseen data, in our case Test Data (`valid_data`). For each of the machine learning algorithms we cited at the beginning of the Data Modeling Section, we tested the following experiments:

- Base models (Default hyperparameters).
- Models with best hyperparameters after executing a Grid Search for hyperparameter optimization (as seen at Section 7).
- Models with best hyperparameters after executing a Grid Search for hyperparameter optimization and using exclusively the most relevant columns for each model.

After all the process was done, we decided to evaluate each potential model obtained in previous steps (see figure 24). We see that the Random Forest algorithm in all of its versions performed the best overall to predict our test data, considering all selected metrics. The Random Forest without hyperparameter tuning and no specific feature filtering performed slightly better than the other Random Forest variants we tested out at each of the metrics; however, we consider it is not a representative difference. On the other, we can see a significant improvement in other models such as k-NN, where the baseline model got an F1-Score of approximately 0.70, but after implementing hyperparameter tuning and using the relevant features exclusively for the model, we got an F1-Score of roughly 0.8520, getting truly close to our best model (Random Forest).

Insights and possible further steps

As we developed a greedy approach treating our dataset as branches, preprocessing, testing them, and then merging the best experiments for further experiments; an interesting exercise would be to use a recursive approach, trying all imputation methods, with all feature selection methods, combined with all hyperparameter tuning techniques. The problem arises when considering our computational resources available since this exercise requires intensive computation.

Additionally, we could have tried other categorical encoding techniques, such as Label Encoding (ranking feature from 1 to N, better known as Ordinal Encoder), Helmert Encoding, Leave One Out Encoding, Probability Ratio Encoding, and so on; to see how it affects the performance of our modeling experiments.

References

[1] Kotler, Philip , Grayson, Kent A. and Hibbard, Jonathan D.. "marketing". Encyclopedia Britannica, 20 Dec. 2021, https://www.britannica.com/topic/marketing. Accessed 8 May 2022.
[2] E. Tarver, "Create a great marketing campaign to attract customers," *Investopedia*, 27-Jul-2021. [Online]. Available: https://www.investopedia.com/terms/m/marketing-campaign.asp. [Accessed: 08-May-2022].
[3] J. Chen, "Term deposit definition," *Investopedia*, 20-Mar-2022. [Online]. Available: https://www.investopedia.com/terms/t/termdeposit.asp. [Accessed: 08-May-2022].
[4] J. Martinez-Bachmann, "Bank Marketing Dataset," Kaggle, 12-Nov-2017. [Online]. Available: https://www.kaggle.com/datasets/janiobachmann/bank-marketing-dataset. [Accessed: 07-May-2022].
[5] S. Mondal, "Linear vs logistic regression: Linear and logistic regression," *Analytics Vidhya*, 02-Dec-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/. [Accessed: 08-May-2022].
[6] Scikit Learn, Ed., "Compare the effect of different scalers on data with outliers," scikit-learn, 2007. [Online]. Available: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html. [Accessed: 09-May-2022].
[7] Y. Zhao, "Techniques," in *CISC5790 Data Mining*, 28-Feb-2022.
[8] A. Fawcett, "Data Science in 5 minutes: What is one hot encoding?," *Educative*, 11-Feb-2021. [Online]. Available: https://www.educative.io/blog/one-hot-encoding. [Accessed: 08-May-2022].
[9] Google, "Overview of hyperparameter tuning | AI platform training | Google Cloud," *Google*, 6AD. [Online]. Available: https://cloud.google.com/ai-platform/training/docs/hyperparameter-tuning-overview. [Accessed: 08-May-2022].
[10] S. Yildirim, "How to train_test_split: KFold vs StratifiedKfold," *TowardsDataScience*, 21-May-2020. [Online]. Available: https://towardsdatascience.com/how-to-train-test-split-kfold-vs-stratifiedkfold-281767b93869 [Accessed: 08-May-2022].
[11] Sklearn, "Sklearn.feature_selection.RFECV," *scikit*, 2007. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html. [Accessed: 09-May-2022].

Appendix

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        11162 non-null  int64
 1   job        11162 non-null  object
 2   marital    11162 non-null  object
 3   education  11162 non-null  object
 4   default    11162 non-null  object
 5   balance    11162 non-null  int64
 6   housing    11162 non-null  object
 7   loan       11162 non-null  object
 8   contact    11162 non-null  object
 9   day        11162 non-null  int64
 10  month      11162 non-null  object
 11  duration   11162 non-null  int64
 12  campaign   11162 non-null  int64
 13  pdays      11162 non-null  int64
 14  previous   11162 non-null  int64
 15  poutcome   11162 non-null  object
 16  deposit    11162 non-null  object
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
```

Table 1 - Description of the dataset

```
Feature -> 'job'
management       2566
blue-collar      1944
technician       1823
admin.           1334
services          923
retired           778
self-employed     405
student           360
unemployed        357
entrepreneur      328
housemaid         274
unknown            70
Name: job, dtype: int64


Feature -> 'marital'
married     6351
single      3518
divorced    1293
Name: marital, dtype: int64


Feature -> 'education'
secondary    5476
tertiary     3689
primary      1500
unknown       497
Name: education, dtype: int64
```

```
Feature -> 'default'
no     10994
yes      168
Name: default, dtype: int64


Feature -> 'housing'
no     5881
yes    5281
Name: housing, dtype: int64


Feature -> 'loan'
no     9702
yes    1460
Name: loan, dtype: int64


Feature -> 'contact'
cellular     8042
unknown      2346
telephone     774
Name: contact, dtype: int64
```

```
Feature -> 'month'
may    2824
aug    1519
jul    1514
jun    1222
nov     943
apr     923
feb     776
oct     392
jan     344
sep     319
mar     276
dec     110
Name: month, dtype: int64


Feature -> 'poutcome'
unknown    8326
failure    1228
success    1071
other       537
Name: poutcome, dtype: int64


Feature -> 'deposit'
no     5873
yes    5289
Name: deposit, dtype: int64
```

Figure 1. Unique entries per categorical features.

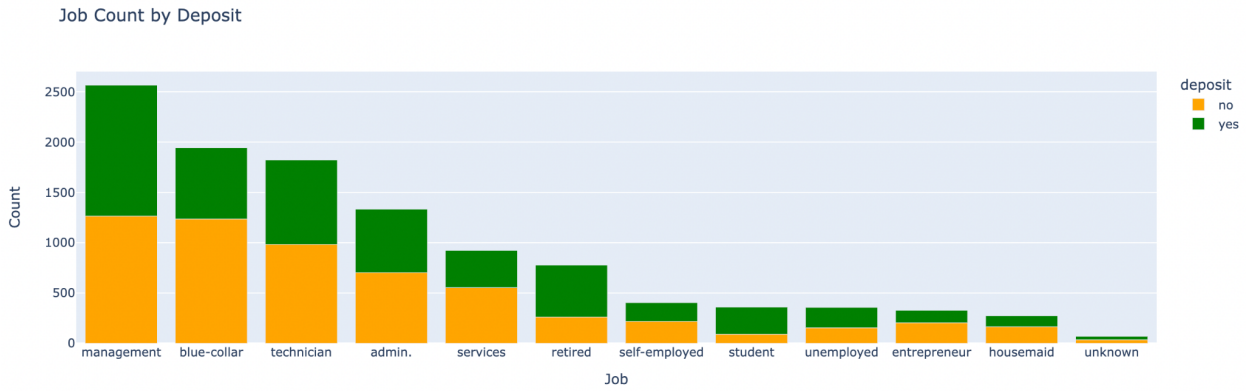

Figure 2

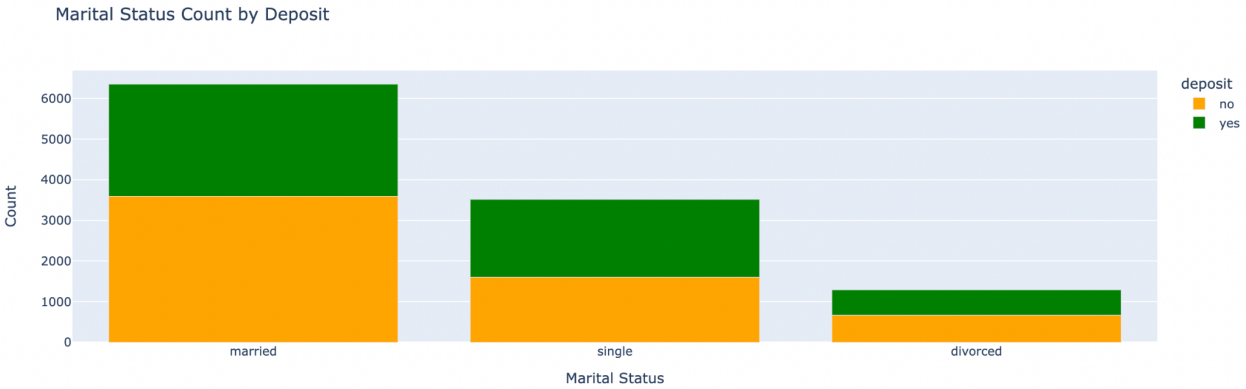

Figure 3

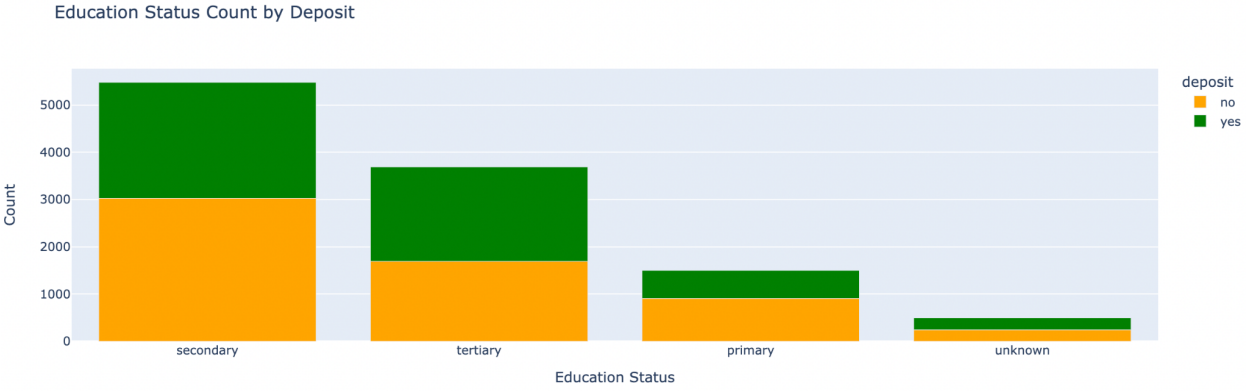

Figure 4

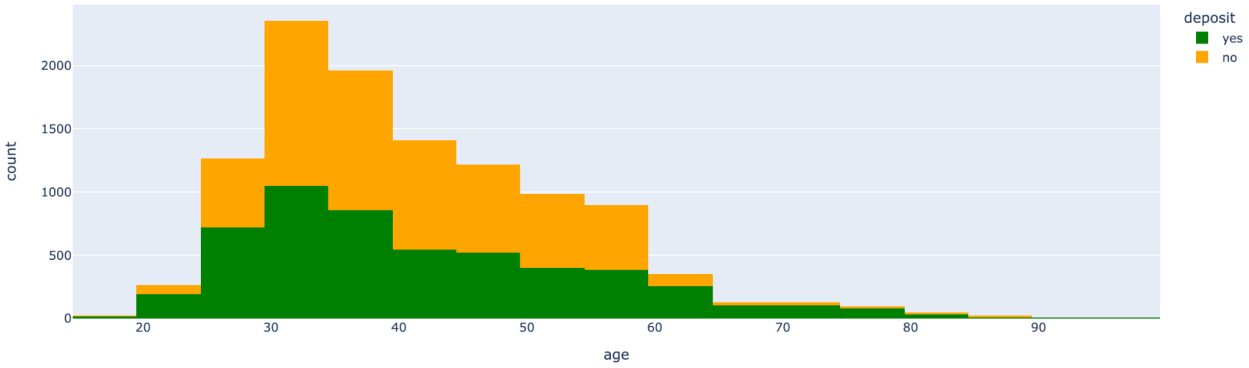

Figure 5

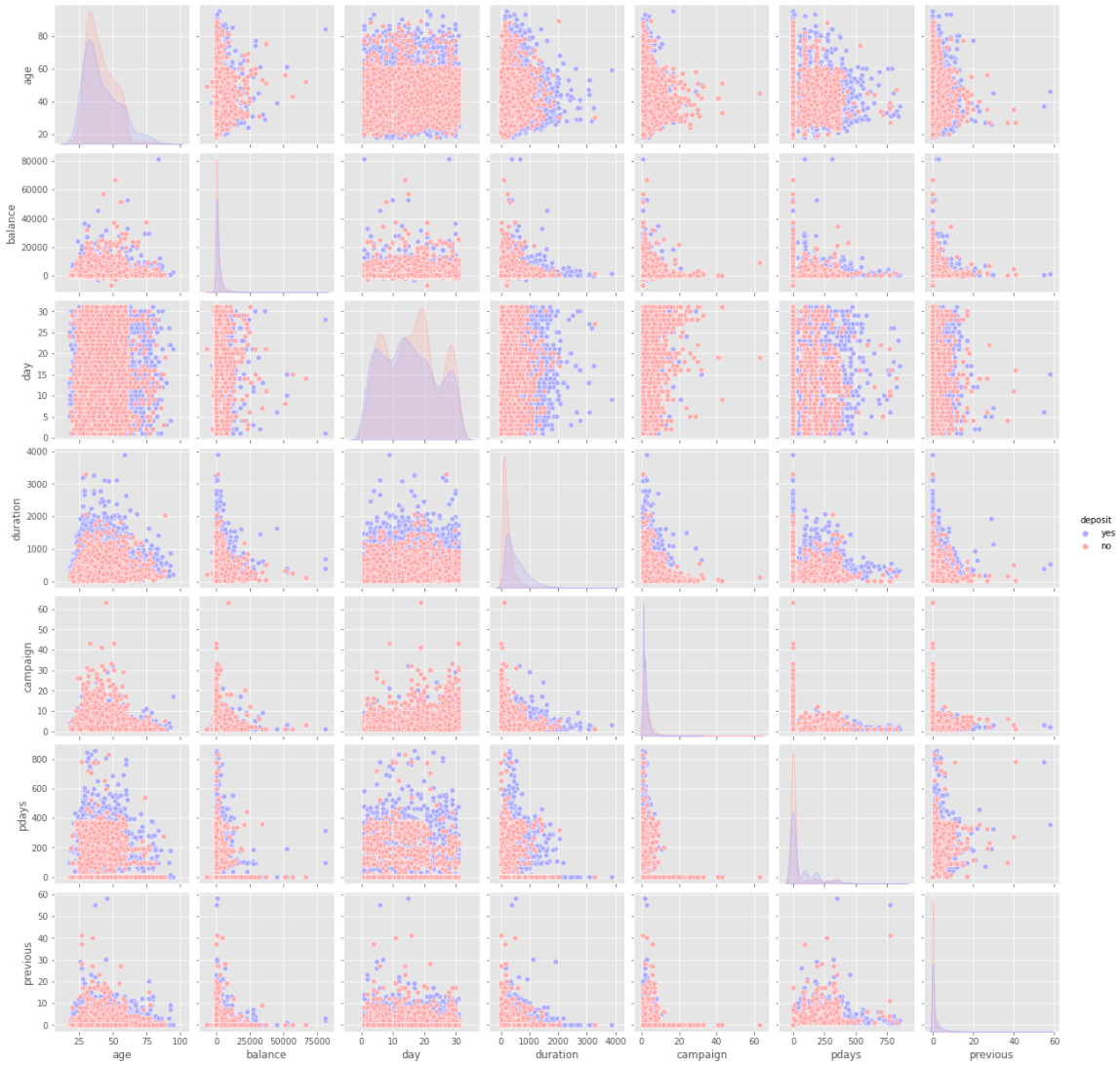

Figure 6 - Plot of the distribution of the Y variable.

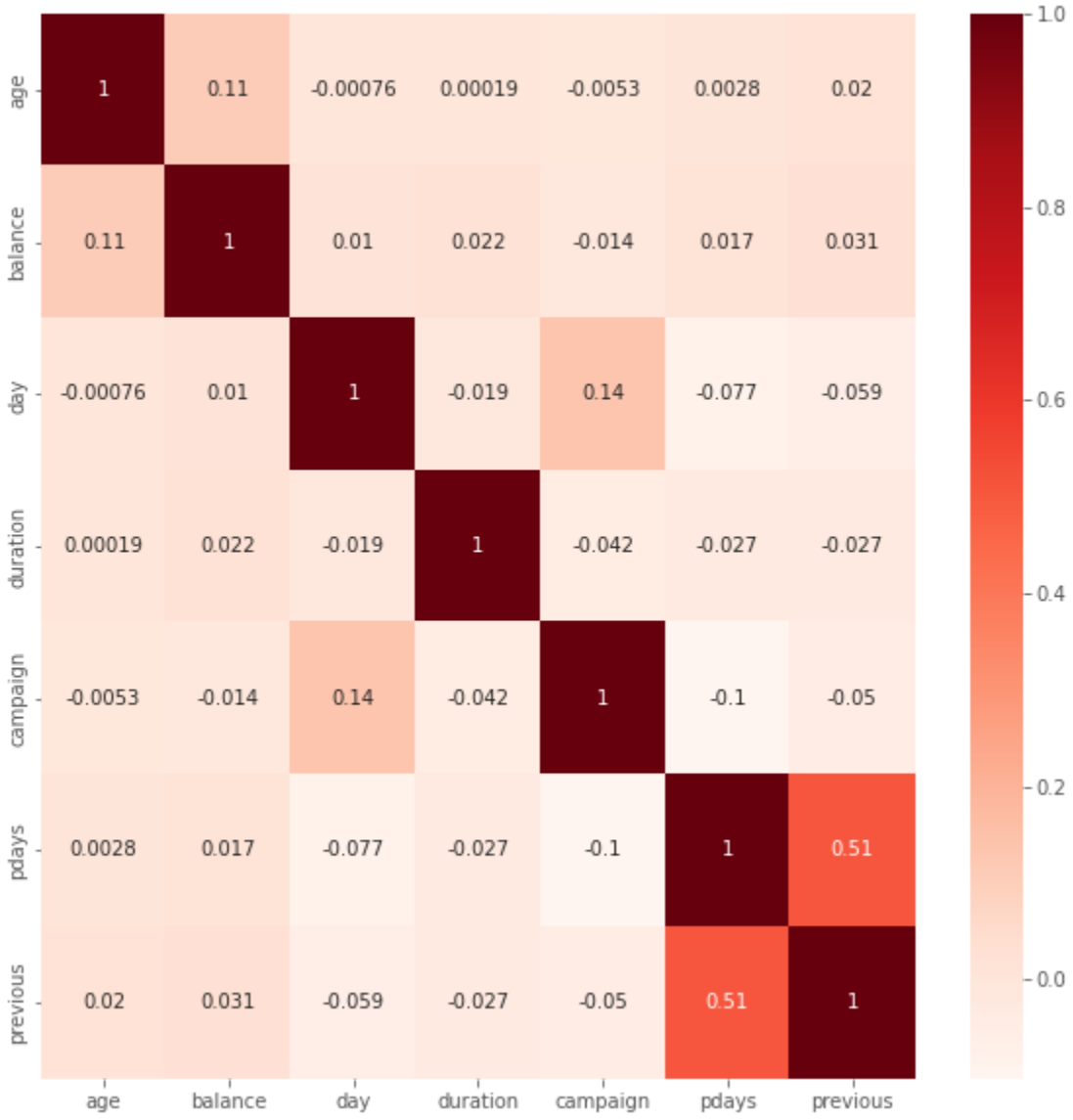

Figure 7 - Correlation between numeric features

```
no      0.52616
yes     0.47384
Name: deposit, dtype: float64
```

Figure 8 - Y variable distribution.

	F1	Accuracy	Precision	Recall
Decision Tree: Imputed with "unknown"	0.773440	0.787100	0.775858	0.771735
Ensemble (Voting): Imputed with "unknown"	0.819004	0.832792	0.835704	0.803237
KNN: Imputed with "unknown"	0.698800	0.736252	0.756935	0.649497
Linear Regressor: Imputed with "unknown"	0.808444	0.825288	0.836523	0.782618
Naive Bayes: Imputed with "unknown"	0.643700	0.716429	0.789339	0.543808
Random Forest: Imputed with "unknown"	0.851258	0.854967	0.823960	0.880846

Figure 9 - Results

	F1	Accuracy	Precision	Recall
Decision Tree: Most Frequent	0.761255	0.775786	0.764355	0.758939
Ensemble (Voting): Most Frequent	0.800575	0.818457	0.830341	0.773493
KNN: Most Frequent	0.678978	0.719565	0.737578	0.629770
Linear Regressor: Most Frequent	0.793840	0.814088	0.831460	0.759832
Naive Bayes: Most Frequent	0.633771	0.714974	0.803452	0.523609
Random Forest: Most Frequent	0.842141	0.846230	0.816125	0.870197

Figure 10

	F1	Accuracy	Precision	Recall
Decision Tree: KNN Imputer	0.774207	0.787996	0.777173	0.771694
Ensemble (Voting): KNN Imputer	0.815218	0.829767	0.834532	0.797159
KNN: KNN Imputer	0.687717	0.726396	0.744246	0.639743
Linear Regressor: KNN Imputer	0.807612	0.824617	0.835827	0.781622
Naive Bayes: KNN Imputer	0.639412	0.715309	0.793030	0.535944
Random Forest: KNN Imputer	0.848982	0.852615	0.821129	0.879115

Figure 11

	F1	Accuracy	Precision	Recall
Decision Tree: Imputed with "unknown": DTC Imputer	0.772827	0.786764	0.776383	0.769809
Ensemble (Voting): Imputed with "unknown": DTC Imputer	0.818402	0.832904	0.838647	0.799415
KNN: Imputed with "unknown": DTC Imputer	0.689664	0.727853	0.745656	0.642109
Linear Regressor: Imputed with "unknown": DTC Imputer	0.808728	0.825737	0.837674	0.782139
Naive Bayes: Imputed with "unknown": DTC Imputer	0.639770	0.715869	0.794878	0.535713
Random Forest: Imputed with "unknown": DTC Imputer	0.850859	0.854518	0.823509	0.880620

Figure 12

	Imputed with "unkown"	Imputed with Most Frequent	KNN Imputer	DTC Imputer
F1	0.765774	0.751760	0.762191	0.763375
Accuracy	0.792138	0.781517	0.789450	0.790607
Precision	0.803053	0.797218	0.800990	0.802791
Recall	0.738624	0.719307	0.734213	0.734967

Figure 13

	F1	Accuracy	Precision	Recall
Decision Tree	0.780050	0.791465	0.775836	0.784886
Ensemble (Voting)	0.812453	0.827865	0.834792	0.791626
KNN	0.708958	0.738156	0.744794	0.676902
Linear Regressor	0.814966	0.827753	0.825739	0.804720
Naive Bayes	0.646255	0.717661	0.789275	0.547570
Random Forest	0.850874	0.852615	0.813728	0.892026

Figure 14 - Undersampling

	F1	Accuracy	Precision	Recall
Decision Tree	0.772003	0.787434	0.780218	0.764311
Ensemble (Voting)	0.810891	0.827640	0.840223	0.783769
KNN	0.705123	0.735356	0.742655	0.671744
Linear Regressor	0.818038	0.830328	0.827281	0.809225
Naive Bayes	0.648033	0.718556	0.789359	0.550014
Random Forest	0.848854	0.851943	0.818085	0.882515

Figure 15 - Oversampling

	F1	Accuracy	Precision	Recall
Decision Tree	0.777913	0.790796	0.778508	0.777521
Ensemble (Voting)	0.815231	0.831000	0.840970	0.791502
KNN	0.705974	0.731996	0.731019	0.683092
Linear Regressor	0.817395	0.829880	0.827458	0.807834
Naive Bayes	0.652638	0.720684	0.788639	0.557099
Random Forest	0.852412	0.855415	0.821878	0.885632

Figure 16 - SMOTE

	Base Model	Undersampling	Oversampling	SMOTE
F1	0.762191	0.768926	0.767157	0.770261
Accuracy	0.789450	0.792586	0.791876	0.793295
Precision	0.800990	0.797361	0.799637	0.798079
Recall	0.734213	0.749622	0.743596	0.750447

Figure 17 - Final Result

```
Parameters KNN:
n_neighbors='9',

Parameters NB:
var_smoothing='1e-05',

Parameters LR:
penalty='none',

Parameters DTC:
criterion='entropy',
max_depth='10',
min_samples_leaf='2',
min_samples_split='5',
splitter='random',

Parameters RF:
criterion='gini',
n_estimators='800',
```

Figure 18 - Hyperparameter Tuning Results

	Relevant Features
0	age
1	balance
2	duration
3	previous
4	job_housemaid
5	job_retired
6	job_student
7	job_unemployed
8	job_unknown
9	marital_single
10	education_unknown
11	default_no
12	default_yes
13	housing_no
14	housing_yes
15	contact_unknown
16	month_apr
17	month_dec
18	month_feb
19	month_jun
20	month_mar
21	month_oct
22	month_sep
23	poutcome_other
24	poutcome_success

Figure 19 - k-NN Features

	Relevant Features
0	age
1	day
2	duration
3	job_admin.
4	job_management
5	job_technician
6	job_unknown
7	marital_divorced
8	marital_single
9	education_secondary
10	education_unknown
11	default_no
12	default_yes
13	housing_no
14	contact_telephone
15	contact_unknown
16	month_apr
17	month_aug
18	month_feb
19	month_jan
20	month_jul
21	month_jun
22	month_nov
23	poutcome_failure
24	poutcome_unknown

Figure 20 - Naives Bayes Features

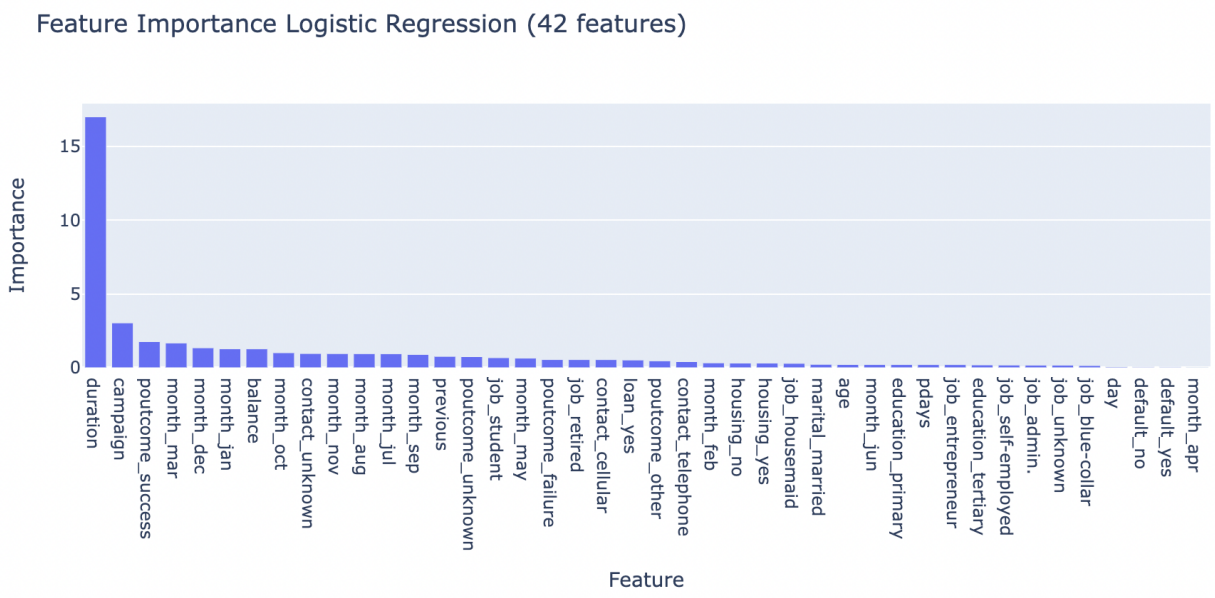

Figure 21

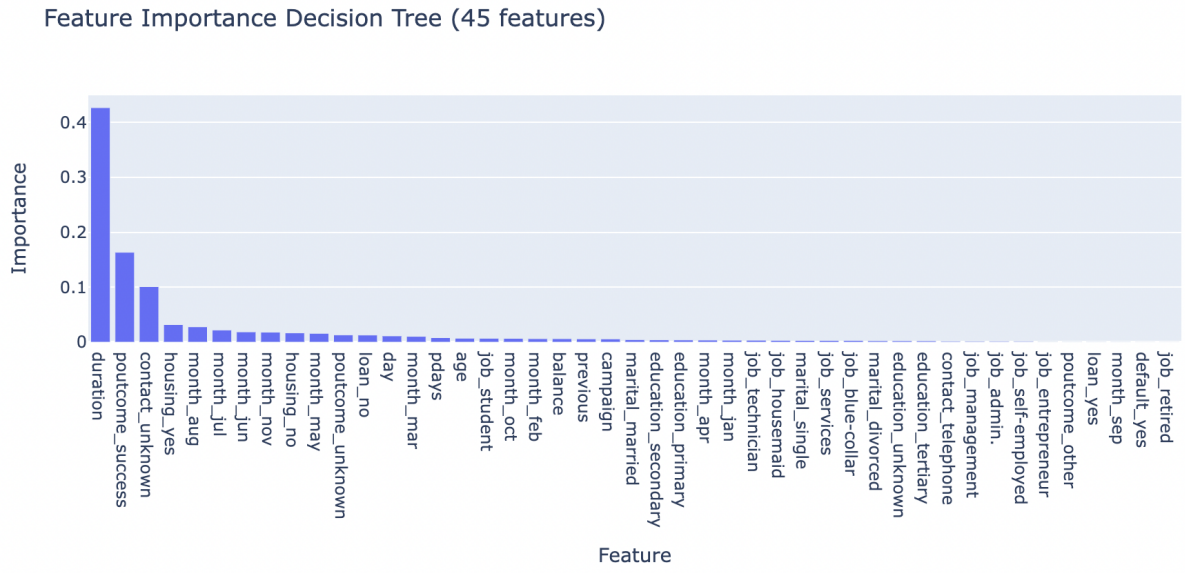

Figure 22

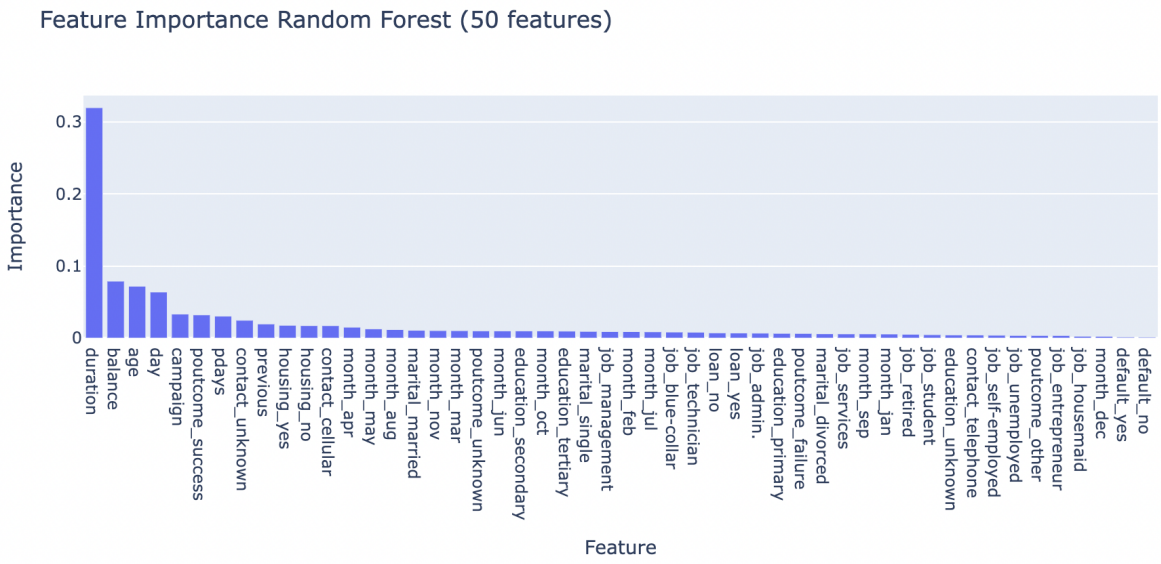
Feature Importance Random Forest (50 features)
Importance
0.3
0.2
0.1
0
duration
balance
age
day
campaign
poutcome_success
pdays
contact_unknown
previous
housing_no
housing_yes
contact_cellular
month_apr
month_may
month_aug
marital_married
month_nov
month_mar
poutcome_unknown
month_jun
education_secondary
month_oct
education_tertiary
marital_single
job_management
month_feb
month_jul
job_blue-collar
job_technician
loan_no
loan_yes
job_admin.
education_primary
poutcome_failure
marital_divorced
job_services
month_sep
month_jan
job_retired
job_student
education_unknown
contact_telephone
job_self-employed
job_unemployed
poutcome_other
job_entrepreneur
job_housemaid
month_dec
default_yes
default_no
Feature

Figure 23

	F1	Accuracy	Precision	Recall
Random Forest	0.855739	0.855352	0.816709	0.898687
Random Forest_HyperParameter_Features_Selected	0.853571	0.853112	0.814310	0.896811
Random Forest_HyperParameter	0.852166	0.851769	0.813299	0.894934
KNN_HyperParameter_Features_Selected	0.852022	0.855799	0.835135	0.869606
Ensemble (Voting)_HyperParameter	0.831084	0.837438	0.824561	0.837711
Ensemble (Voting)	0.827032	0.836095	0.833333	0.820826
Ensemble (Voting)_HyperParameter_Features_Selected	0.824684	0.832064	0.821994	0.827392
Linear Regressor_HyperParameter_Features_Selected	0.820804	0.830273	0.827455	0.814259
Decision Tree_HyperParameter	0.820767	0.822212	0.791123	0.852720
Linear Regressor_HyperParameter	0.819563	0.829825	0.829808	0.809568
Linear Regressor	0.819563	0.829825	0.829808	0.809568
Decision Tree_HyperParameter_Features_Selected	0.817633	0.824004	0.808999	0.826454
Decision Tree	0.784314	0.793103	0.780669	0.787992
Naive Bayes_HyperParameter_Features_Selected	0.776055	0.767129	0.717357	0.845216
KNN	0.703922	0.729512	0.737166	0.673546
KNN_HyperParameter	0.703187	0.733094	0.749469	0.662289
Naive Bayes	0.642417	0.713838	0.796117	0.538462
Naive Bayes_HyperParameter	0.642417	0.713838	0.796117	0.538462

Figure 24 - Final Test