Unlocking Client Subscriptions: A Data Driven Approach to Understanding Key Relationships

By Connor Stanley

Problem Statement

I believe this data originates from the “Bank Marketing” dataset on the UC Irvine Machine Learning Repository.

Each row represents a marketing phone call made to a client by a Portuguese bank, with the goal of encouraging the client to subscribe to a term deposit.

A term deposit refers to when a customer deposits money with a bank for a set period of time and agrees to not touch the money until the term ends. The customer then gets back the original sum of money, along with interest that has accrued. This benefits banks because they provide a steady, reliable pool of money that can be used to issue loans.

The objective of this project is:

1. To develop and evaluate a predictive model that can help the bank prioritize future outreach efforts.

Variable Understanding

The dataset contains the following columns: age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, poutcome, and the target variable y.

Predictor Variables

- age : Client's age in years. (Numerical)
- job : Client's occupation. (Categorical)
- marital : Client's marital status. (Categorical)
- education : Client's education level. (Categorical)
- default : Whether the client has credit in default. (Categorical: yes or no)
- balance : Client's average yearly balance in euros. (Numerical)
- housing : Indicates if the client has a housing loan. (Categorical: yes or no)
- loan : Indicates whether the client has a personal loan. (Categorical: yes or no)
- campaign : Number of contact attempts made during the current campaign for this client (includes the current contact, and includes unanswered attempts). (Numerical)
- pdays : Number of days since the client was last contacted during a previous campaign, -1 indicates they were not previously contacted. (Numerical)
- previous : Number of contact attempts (successful or not) made in campaigns prior to the current one. (Numerical).
- poutcome : Outcome of the most recent previous marketing campaign. (Categorical)

These columns initially caused some confusion due to the wording, but they all refer to the current contact attempt:

- contact : Communication used to contact the client in the current campaign. The word "last" in the documentation refers to this most recent/current call, not a past one. (Categorical)
- day : Day of the month when the current contact was made. (Numeric)
- month : Month of the year when the current contact was made. (Categorical)
- duration : Duration of the current call in seconds. This refers to the length of the conversation with the client during the contact attempt represented by this row. (Numerical)

Target Variable

- y : Whether the client subscribed to a term deposit. (Categorical: yes or no)

Variable Considerations

1. The duration variable will be removed prior to modeling. Initially, I was under the assumption it referred to the duration of a previous contact with the client, but it actually refers to the current call. Since I assume the later goal of this analysis will be to build a predictive model before making a call, we wouldn't know in advance how long a conversation would last. As such, including duration is not useful at prediction time.
2. Similarly, I am hesitant to include the campaign variable. Although we will know how many previous attempts have been made to contact a client, we won't know if the contact will even answer. Even if our model suggests the client may subscribe, they may not answer. The actual call count will vary based on whether the client answers or not. For that reason, I'll exclude the campaign feature from the model.
3. Another variable that presents a challenge is pdays. The value -1 indicates that the client was not previously contacted. Including these values would 1, indicate that clients were called from a previous campaign -1 days ago (which doesn't make sense), and 2, the -1 value would skew summary statistics and correlations. To address, I've decided to convert pdays into a categorical variable. This way, I can preserve the information while avoiding the problems caused by the -1 entries. While it would usually be beneficial to do this in the feature engineering stage, early binning will simplify the exploratory analysis.
4. Interpreting the day variable using traditional summary statistics (mean, median, standard deviation) proved to be challenging. Therefore, I opted to group it into three bins to facilitate clearer analysis and insights.
5. One thing I wish were available is the year of the contact. However, according to the dataset page on the UC Irvine Machine Learning Repository, the original file (bank-full.csv) appears to be ordered chronologically (likely from oldest to most recent). As a result, it may be more appropriate to avoid random sampling when creating training and testing sets. Instead, using the first 70% of rows for training and the last 30% for testing helps ensure that future data is not used to predict past outcomes. Since we have 45211 rows, we can use a calculator to determine that 70% is 31647.7. Due to the issue of splitting up the rows into .7 and .3, we will just round up to use the first 31648 rows for training and the rest for testing. Furthermore, the exploratory analysis stage will look only at the training set.

Exploratory Analysis

Univariate (Single Variable) Analysis

Numerical Summary Statistics

- age : The minimum age is 19 years old, the first quartile is 33, the median is 40 years old, the mean is 41.11, the third quartile is 49, the maximum is 94, and the standard deviation is 9.60.
- balance : The minimum average yearly account balance is -8,091 euros, the first quartile is 47, the median is 397, the mean is 1,293.29, the third quartile is 1,328, the maximum is 98,417, and the standard deviation is 2,961.23.
- campaign : The minimum number of contact attempts during the campaign is 1, the first quartile is 1, the median is 2, the mean is 3.06, the third quartile is 3, the maximum is 63, and the standard deviation is 3.51.
- previous : The minimum number of contact attempts from a previous campaign is 0, the first quartile is 0, the median is 0, the mean is 0.21, the third quartile is 0, the maximum is 275, and the standard deviation is 1.95.
- duration : The minimum number of seconds talking to client is 0, the first quartile is 98, the median is 171, the mean is 252.40, the third quartile is 305, the maximum is 4,918, and the standard deviation is 262.35.

All variables appear to be positively (right) skewed, as indicated by their means being greater than their medians. This pattern suggests that most values are clustered on the lower end, while a smaller number of high values pull the mean upward. This is a typical reason to consider log-transforming these features when building models, especially if the model performance is impacted by skewness.

There also appears to be a strong possibility of outliers. These could distort model estimates later and on, and if needed, should be dealt with (capping out at a certain value, removal, transformation, etc).

Histograms

- age : The distribution for age is positively skewed, with most clients falling between 25 and 60 years old.
- balance : The distribution for average yearly balance is heavily positively skewed. Most clients appear to have an average yearly balance in the -1,000 to 10,000 euro range, though the exact range is hard to tell from the plot.
- campaign : The distribution for contact attempts is positively skewed, with most clients needing 1-5 attempts.

- previous : The distribution for previous contact attempts is also heavily positively skewed, with approximately a most clients having between 0 and 10 contact attempts from prior campaigns
- duration : The distribution of duration is positively skewed, with most conversations lasting between 0 and 1000 seconds.

As expected, all numeric variables show positive skewness.
All graphs can be viewed here (or go to the H-Graphs section in the Graphs section at the end of this report).

Categorical Proportion Plots

- job - The most frequent category is “blue-collar”, followed by “management”, “technician”, “admin”, “services”, “retired”, “self-employed”, “entrepreneur”, “housemaid”, “unemployed”, “students”, and “unknown” is the least frequent category.
- marital - The most frequent category is “married”, followed by “single”, and “divorced” is the least frequent category.
- education - The most frequent category is “secondary”, followed by “tertiary”, “primary”, and “unknown” is the least frequent category.
- default - The most frequent category is “no” and “yes” is the least frequent category.
- housing - The most frequent category is “yes” and “no” is the least frequent category.
- loan - The most frequent category is “no” and “yes” is the least frequent category.
- contact - The most frequent category is “cellular”, followed by “unknown”, and “telephone” is the least frequent category.
- month - The most frequent category is “may”, followed by “july”, “august”, “june”, “november”, “february”, “january”, “march”, “april”, “october”, and “december” is the least frequent category. “September” also doesn’t show up in the training set.
- poutcome - The most frequent category is “unknown”, followed by “failure”, “other”, and “success” is the least frequent category.
- day_bin - The most frequent category is “21-31 days”, followed by “11-20 days”, and “0-10 days” is the least frequent category.
- pdays_bin - The most frequent category is “not contacted”, followed by “101-200 days”, “201+ days”, and “0-100 days” is the least frequent category.

Target variable

- y - The most frequent category is “no” and “yes” is the least frequent category.

All graphs can be viewed here (or go to the Proportion Bar Charts section in the Graph section at the end of this report).

Bivariate (Double Variable) Analysis

Scatterplots

- age vs balance : A clear outlier appears around age 60 with a balance near 100,000 euros. Aside from this, there doesn't seem to be any obvious linear or non-linear relationship between age and balance.
- age vs campaign : There does seem to be some outliers with approximately 60 contact attempts in the current campaign. There doesn't seem to be any linear or non-linear relationships.
- age vs previous : Similar to the first plot, there is a distinct outlier corresponding to a client contacted over 250 times in previous campaigns. Otherwise, no notable linear or non-linear relationship is evident.
- age vs duration : Some potential outliers exist with call durations around 5,000 seconds. There does not appear to be any strong linear or non-linear relationship..
- balance vs campaign : The outlier with a balance near 100,000 euros remains obvious. There seems to be a non-linear relationship, possibly inversely proportional, though it is not the strongest in the world.
- balance vs previous : The outliers for previous contacts (~250) and high balance (~100,000 euros) stand far apart from the other points. A slight non-linear curve, similar to the balance vs campaign plot, may be present.
- balance vs duration : The extreme points in duration (~5,000 seconds) and balance (~100,000 euros) stand out clearly again. There is no obvious non-linear relationship between balance and duration.
- campaign vs previous: The outlier with previous contacts around 250 is prominent. No moderate or strong linear or non-linear relationship is visible.
- campaign vs duration : Outliers for duration (~5,000 seconds) and campaign attempts (~60) are present. The same slight non-linear relationship noted earlier also appears here.
- previous vs duration: The previous contacts outlier (~250) is evident. There appears to be a slight non-linear relationship similar to that observed in the age vs campaign scatterplot.

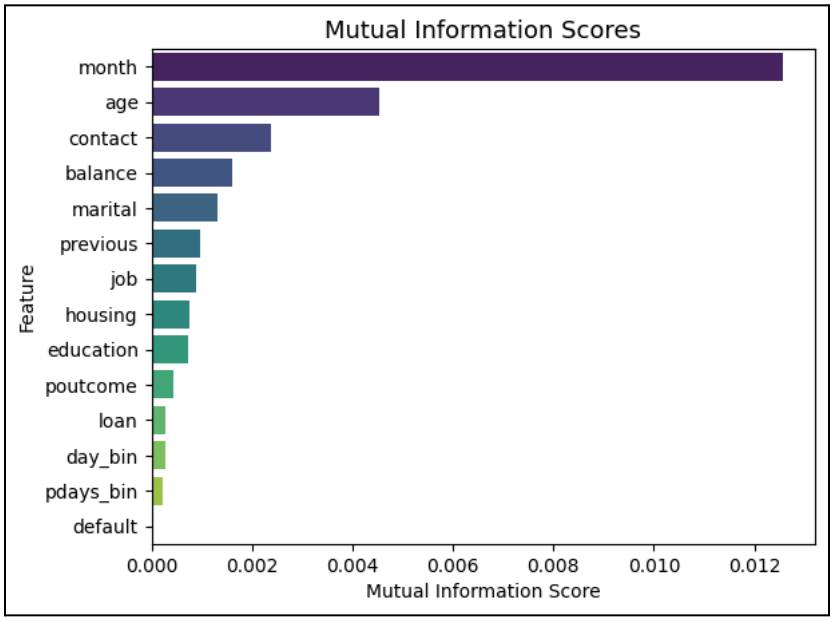
All graphs can be viewed here (or go to the Scatterplot section in the Graph Section at the end of this report).

Feature Selection and Feature Engineering

Feature Selection (Small Outlier Removal Also)

Prior to calculating mutual information scores for feature selection, we first address two standout data points identified in the scatterplots: a client with a balance of approximately 95,000 euros and another with around 270 previous contact attempts. While these values may be technically possible, their distance from the rest of the data suggests they are outliers. To reduce their potential influence on the analysis, we remove them by filtering the dataset to include only rows where balance < 80,000 and previous < 250.

With these adjustments made, we will proceed to using the mutual_info_classifier from scikit-learn to determine which features hold the most information regarding our target features.



The higher the mutual information score, the more predictive information a feature has about the target variable. The most important feature by far is “month”, followed by “age”, “contact”, “balance”, “marital”, “previous”, “job”, “housing”, “education”, “poutcome”, “loan”, “day_bin”, “pdays_bin”, “default”.

In order to determine what features should be used further, we will select all features that have a mutual information score greater than the median mutual information score. Those features are “month”, “age”, “contact”, “balance”, “marital”, “previous”, and “job”.

Outliers and Feature Engineering

Before creating any interactions for the Random Forest model, it is important to revisit potential outliers. Using the IQR method, identifying those values outside the range of Q1 + 1.5 * IQR and Q3 + 1.5 * IQR, we calculated the proportion of outliers for each numerical variable:

- Approximately 10.87% of the “balance” values are classified as outliers.
- Approximately 6.5% of the “previous” values are outliers.
- Approximately 0.09% of the “age” values are outliers.

Since the “age” feature has such a low proportion of outliers, it is unlikely to significantly affect the model. However, the “balance” and “previous” features each contain over 6% outliers, which poses more of a challenge. Removing these rows entirely could result in a substantial loss of training data. As an alternative, we can create binary indicator variables to flag whether each observation is an outlier. This preserves the data while still allowing the model to capture the potential influence of extreme values.

Now that a strategy for handling outliers is in place, the next step is to look at creating some interaction terms.

In terms of the numerical features, we can first create three interaction terms:

1. Between age and balance
2. Between balance and previous
3. Between previous and age

While these may not yield drastic improvements, they could allow the model to capture any interaction between the two.

Why Random Forest?

Random Forest was chosen because it can capture non-linear relationships; which appears useful here given the slight non-linearity observed between balance and previous. While logistic regression relies on a linear combination of the predictor variables (transformed into a probability), Random Forest is composed of many decision trees, each of which uses rules to split the data in the most effective way.

Although a single decision tree is useful for interpretability, it can be sensitive to small changes in the data. This is where Random Forest comes in: it trains multiple decision trees on random subsets of both the data and the features. Each tree makes an individual prediction, and these are aggregated, typically through majority vote in classification, to form the final output.

While it’s often said that Random Forest is made up of many “weak learners,” that phrase can be a bit misleading. A more helpful analogy is the U.S. Senate: no single senator makes decisions alone. Each one contributes their perspective based on their unique experience. Similarly, each decision tree in a Random Forest brings a different “view” of the data. By combining these diverse perspectives, the model can arrive at a more balanced and robust prediction.

Scoring Metrics:

The next step is to determine the most appropriate scoring metric. Given the significant imbalance between the number of clients who did not subscribe and those who did, traditional accuracy is not a reliable metric. While a model with 95% accuracy may sound impressive, it

would likely achieve this by overwhelmingly predicting the majority class (clients who did not subscribe). As a result, many clients who would subscribe may be overlooked (which defeats the purpose of the model's help identifying promising leads).

One alternative is precision, which measures the percentage of clients predicted to subscribe who actually did subscribe. A high precision model would be valuable in practice, especially if the goal is to reduce wasted calls to clients unlikely to convert.

Another option is recall, which measures the percentage of actual subscribers who were correctly identified. Optimizing recall ensures that fewer true subscribers are missed, which is a useful objective if the goal is to capture as many potential customers as possible.

However, there's a natural trade-off between precision and recall: improving one often leads to a decline in the other. To balance both, we can use the F1-score, which is the harmonic mean of precision and recall. This provides a more holistic view of the model's ability to correctly identify subscribers without sacrificing too much on either side.

Feature Transformation and Model Building

While my previous experience involved taking care of any preprocessing in a separate pipeline prior to model training, in the real world, it can be helpful to have a pipeline that takes care of preprocessing and then model training. While some of the preprocessing done already could be integrated into a pipeline, I decided to take care of it earlier on.

Prior to any real advanced techniques, we should first establish a baseline model to determine where we are at currently.

First Baseline Pipeline

In our pipeline, we will need two main parts:

1. Preprocessing our data
2. Training the Random Forest model

In the preprocessing, we will be transforming the numerical features we have, along with turning the categorical variables we have into a format that the Random Forest model will understand.

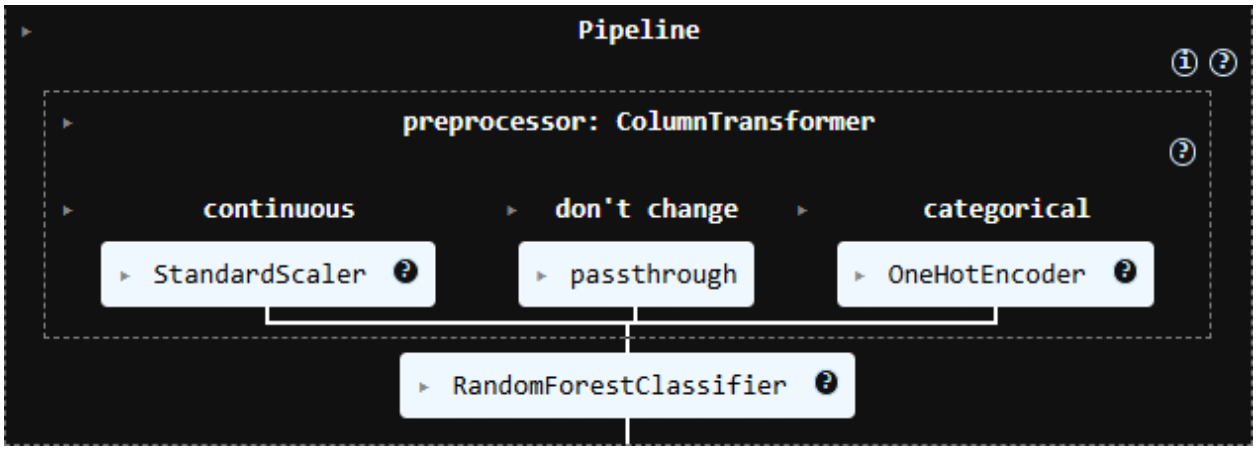
For the categorical features, we will use one-hot encoding, which converts each category in a feature into a separate binary column. If a row belongs to a specific category, that column will be marked as 1, and the rest will be 0 for that feature. Since the "september" category isn't present in the training data, we will set the handle_unknown parameter of the OneHotEncoder to "ignore", ensuring that any previously unseen categories (such as "september") are all encoded as zeroes.

While it's common to drop the first category of each feature to avoid multicollinearity in some instances, doing so here would cause the dropped category to be indistinguishable from September, which is already encoded as all zeros. To avoid this ambiguity, we will retain all one-hot encoded columns.

Finally, the numerical features will be standard scaled (transformed so each feature has a mean of 0 and standard deviation of 1) to ensure that differences in scale, especially in regard to the “balance” feature, do not influence the model. It also should be noted that Random Forest is more robust to differences in scale compared to Logistic Regress or Support Vector Machines, but it shouldn’t hurt to scale the features regardless.

Although we initially split the dataset into training and testing sets, we can use cross-validation on the training data to better evaluate model performance and stability. However, because our data has a time component, we avoid standard cross-validation to prevent data leakage (using future information to predict the past). Instead, we use TimeSeriesSplit from scikit-learn, which ensures that each training fold precedes its corresponding test fold chronologically.

Additionally, in order to limit how much code to write (or copy/paste), I created a function that would take in the predictor features (X), target variable (y), pipeline (estimator) and number of splits to use for TimeSeriesSplit (splits).



The above image is a visual representation of the first pipeline. The continuous, don’t change, and categorical parts all occur first and deal with standard scaling the numerical variables (continuous), one-hot encoding all the categorical variables (categorical), and features that are already in a suitable numerical format and don’t require scaling or encoding (don’t change). After that is done, we then send that data to the Random Forest Classifier.

However, prior to training the pipeline, we need to split up the data. Once the data is split into a training and testing fold, we feed the training fold into the Pipeline. Using the fitted Pipeline, we can then predict whether the customer will subscribe to a term deposit in the testing set. The baseline model achieved a mean F1-score of 0.6% with a standard deviation of 0.6%. Both metrics are concerning: the F1-score is extremely low, and the standard deviation being equal to

the mean, suggests high variability. Since approximately 95% of the training data consists of clients who did not subscribe, the low mean f1_score is understandable.

To improve the f1_score, one immediate approach is to adjust the “class_weight” parameter. In other words, this aims to penalize the model when it predicts "no" for a client who actually did subscribe.

After duplicating the original pipeline (and assigning it a new name), we can set the class_weight parameter to ‘balanced’ and then pass the new pipeline to the custom function, along with the required data. After this, the mean f1_score was 0.9% with a standard deviation of 0.8%. While the mean f1_score did increase, it remains very low, and the model’s variability is concerning. Luckily, there is still more we can try.

Tuned Pipeline with SMOTE

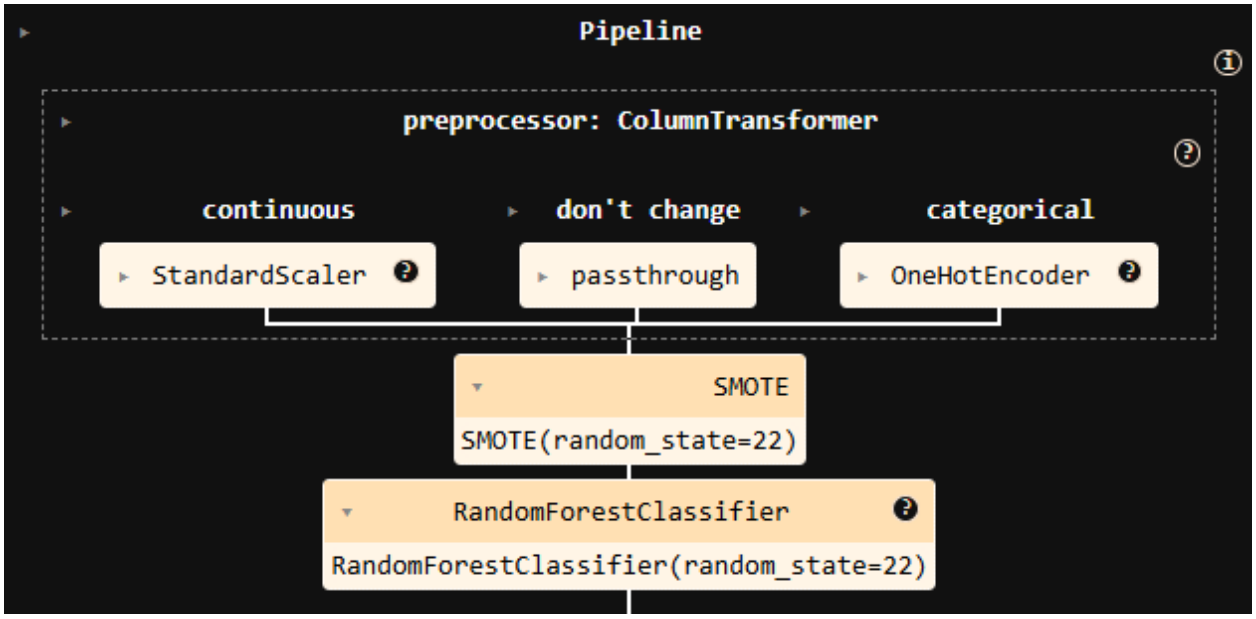
One potential solution to address the class imbalance is to use SMOTE (Synthetic Minority Over-Sampling Technique). SMOTE works by generating synthetic examples of the minority class based on existing observation, which can help the model learn more effectively about this underrepresented group.

In theory, this should improve the model’s ability to detect minority class instances. One downside of SMOTE is that it can lead to overfitting, especially if the model is trained on synthetically balanced data but evaluated on imbalanced data. Because of this, the goal will be to generate just enough synthetic samples to give the model more information.

Furthermore, there are different parameters that are part of a Random Forest that can be adjusted, that could possibly lead to a better performance overall.

How does this fit into a Pipeline?

First, we add one step to our original pipeline (SMOTE).



As we can see, this pipeline is almost identical to the previous one. The primary difference is that after preprocessing, the data is fed through SMOTE, and then to a Random Forest model.

We will also create a new function that will accept the same parameters as the previous function. This one will include the powerful BayesSearchCV function.

First, we will use the TimeSeriesSplit function to generate a training set and a testing set. In this case, 5 folds were used, meaning the entire process will be repeated five times. After the training set is established, we can then create the setup for BayesSearchCV, for which we will need to specify several key elements.

- The pipeline featured above serves as the estimator
- Defined search space for each hyperparameter that we want to tune
- How many times we want to change hyperparameters per run (iterations)
- What scoring we want to maximize
- Custom cross validation strategy
- Fixed random_state to ensure reproducibility

We will be tuning five different hyperparameters:

1. sampling_strategy for SMOTE : The higher this value is, the more samples are generated for the minority class
2. n_estimators for Random Forest : Number of trees
3. max_depth for Random Forest : Maximum depth of each tree
4. min_samples_split : Minimum samples required to split an inner node
5. class_weight

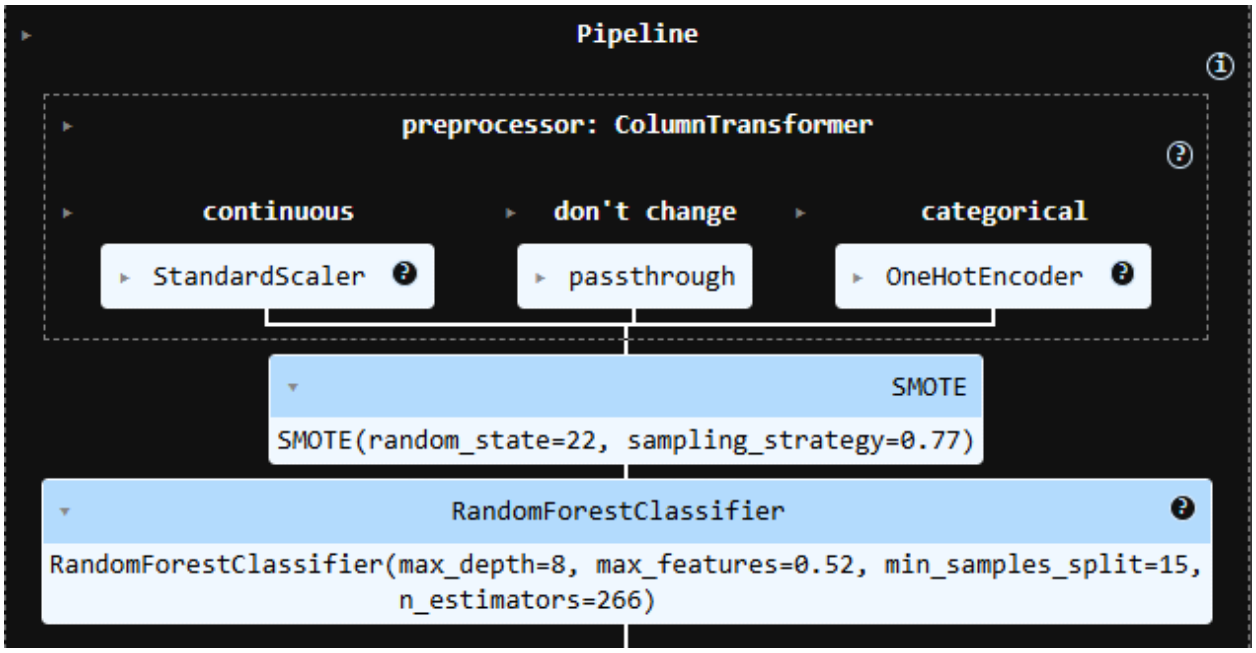
Then, we can run BayesSearchCV. Initially, parameters are randomly sampled from the search space provided. Then, instead of randomly sampling a new set of parameters (unlike RandomizedSearchCV), it builds a probabilistic model to decide which hyperparameter combinations are most likely to improve performance. It iteratively chooses different combinations based on prior results, theoretically leading to a faster and more efficient solution that maximizes the scoring function compared to other methods.

Given that 4 inner folds are used and 30 iterations, 120 models will be trained and evaluated per outer fold. After evaluating these, it selects the set of hyperparameters that yielded the best average f1_score across the inner folds. This tuned pipeline is then used to predict on the outer folds test set, and the resulting f1_score is recorded. This completes the process for one outer fold. Since we have 5 outer folds, this entire process repeats four additional times. At the end, we will have 5 f1_scores, along with the set of hyperparameters that lead to those scores.

In the end, out of the five scores, the mean was .123, or 12.3% with a standard deviation of 4.7%.

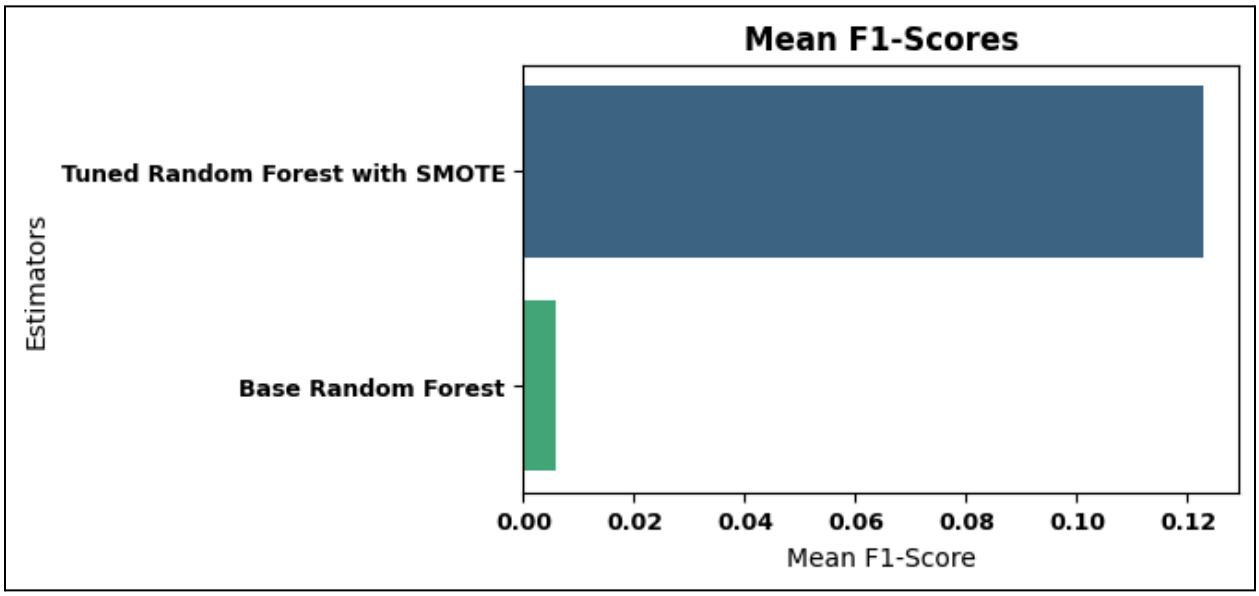
Final Pipeline

Since the five f1_scores we got all came with their own different hyperparameters, we need to come up with a way to grab the best ones. For simplicity purposes, we can simply average all 5 parameters values (since all parameters were numerical ones). Once a pipeline with these parameters is fitted, we can then compare the f1_score truly.



As we can see, the sampling strategy for SMOTE is at .77, and the parameters for random forest are (max_depth-8, max_features=0.52, min_samples_split=15, and n_estimators=266).

When this pipeline, along with our training data, was sent to the initial custom function (excluding the BayesSearchCV step), our mean F1-score was 12.3% with a standard deviation of 4.9%



Evidently, the score has significantly increased from its initial baseline. Specifically, we achieved a 1950% increase.

Conclusion and Potential Next Steps

Finally, we can fit the final pipeline using all of our training data and then test it on the previously unseen data. On the testing data, our f1_score was 28.7%. Under these circumstances, a random guess (like flipping a coin) could theoretically yield better F1-scores.However, a coin cannot predict the probability that a client will subscribe, which our pipeline is capable of providing.

Using this probability, we can then identify, for example, the top 10% of clients most likely to subscribe and prioritize outreach to them. Following this logic, if we use our pipeline to select the top 10% of clients most likely to subscribe, we would identify 16.36% of all actual subscribers. If we didn't have enough callers for that week, and just had enough to call the 1% of clients most likely to subscribe. In this case, we will have gotten 1.77% of all clients who would have subscribed.

A balance is clearly needed between these two objectives: to increase loan offerings to our customers (assume this is the goal), we must secure more clients who agree to term deposits. However, increasing weekly caller requirements or asking current staff to work additional hours would necessitate additional compensation.

Potential Next Steps:

The first step should be to incorporate all initial preprocessing, outlier checks, and interaction terms directly into our pipeline. In a real-world scenario, consolidating preprocessing and the estimator within a single pipeline is considered best practice, ideally leading to easier debugging by centralizing all components.

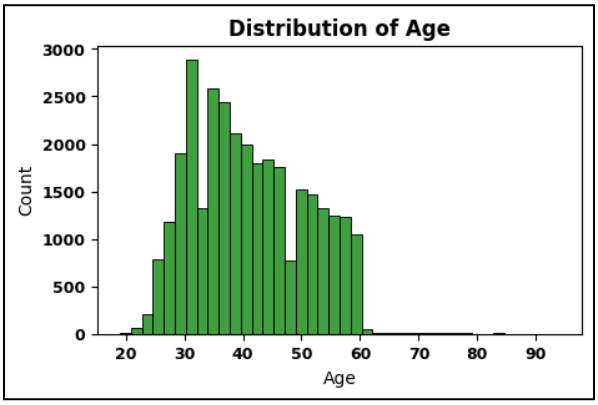
Next, while the basic SMOTE function was used, other versions (e.g., BorderlineSMOTE, ADASYN, SMOTEN) could be explored, along with their respective parameters for tuning. Similar to machine learning models, SMOTE offers simpler and more complex versions that warrant further experimentation.

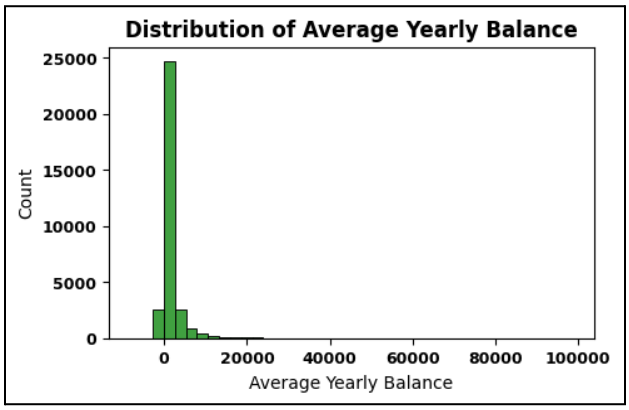
Although more challenging in this scenario, acquiring additional data in the future could also be beneficial (since this is a public dataset and is from a bank in Portugal). In general, while overfitting and variability will always be present in predictive models to some extent, having sufficient data should enable a model to learn more generalized patterns that can be successfully applied to testing data.

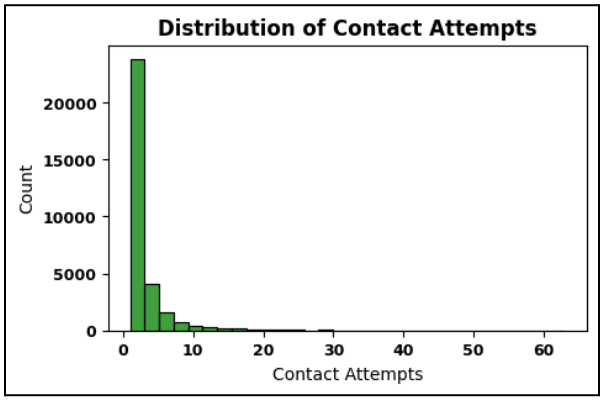
Additionally, exploring gradient boosting models, such as XGBoost, HistGradientBoosting, or LightGBM, is another avenue. While Random Forest models excel at training on diverse data subsets and features, Gradient Boosting focuses on iteratively learning from past errors to improve performance in subsequent steps.

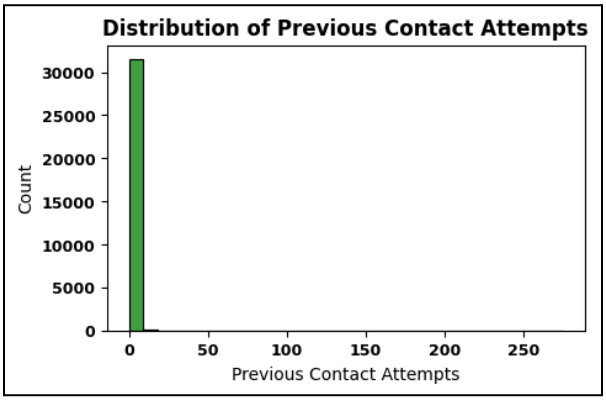
Graphs Section

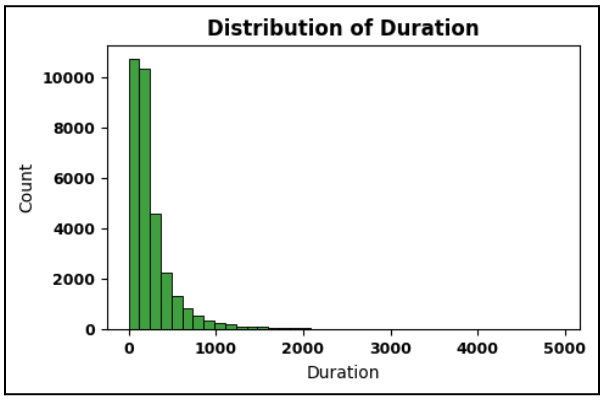
H-Graphs



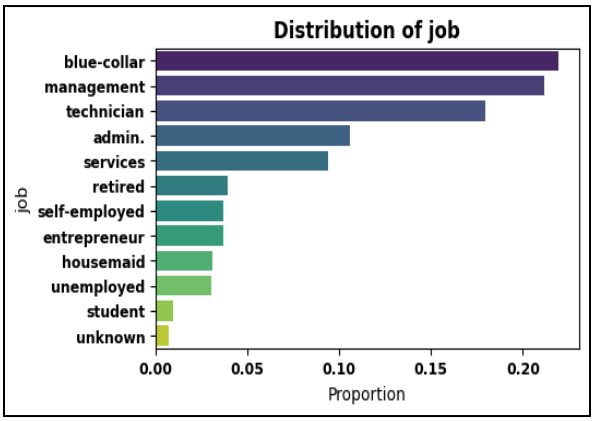


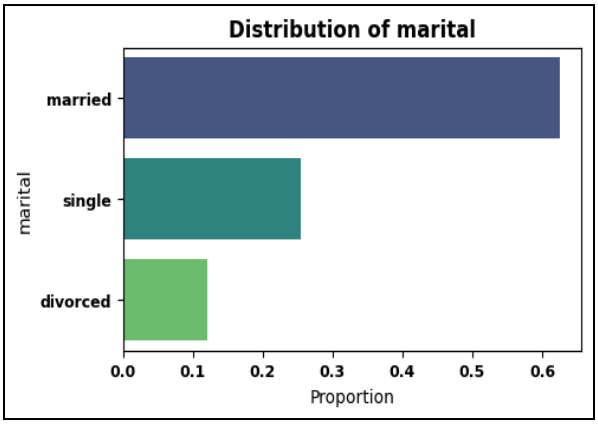


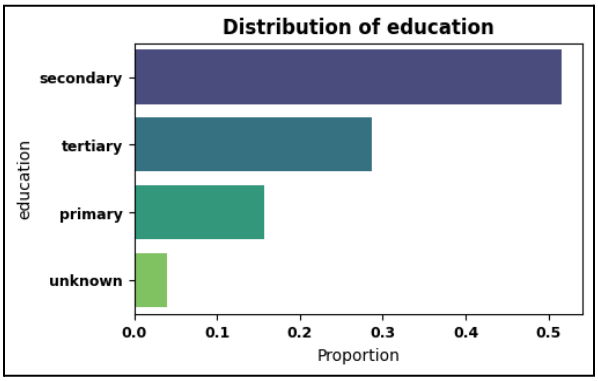


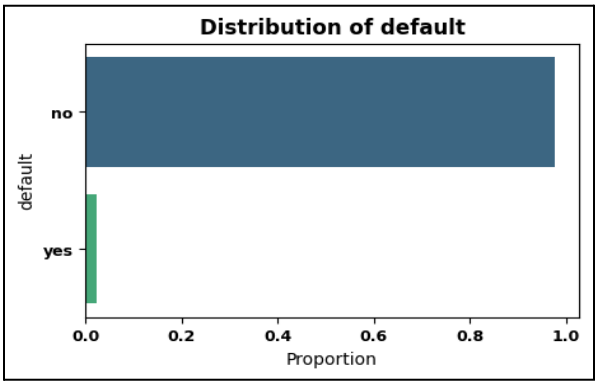


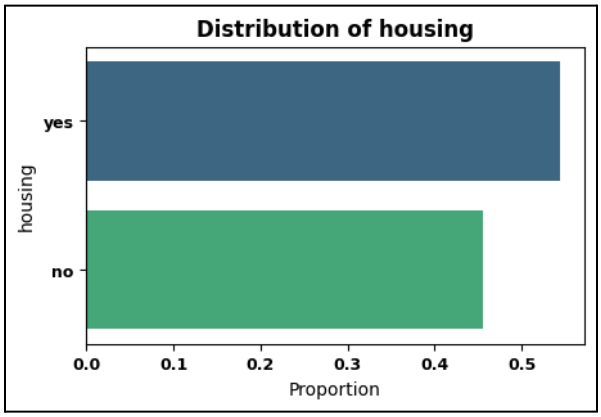
Proportion Bar Charts

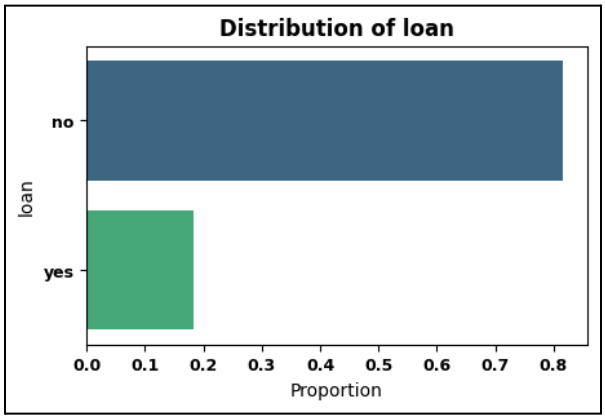


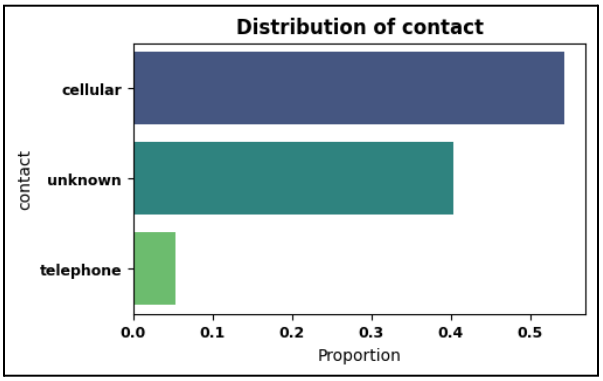


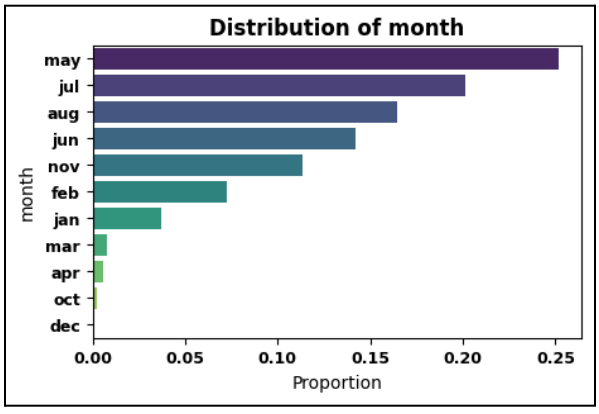


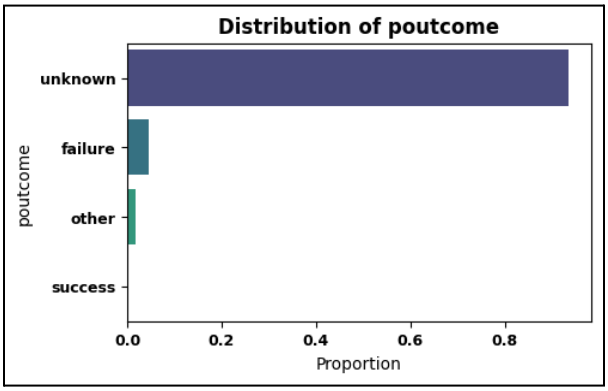


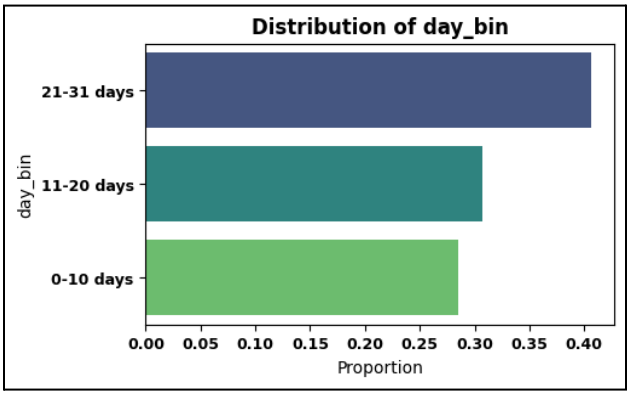


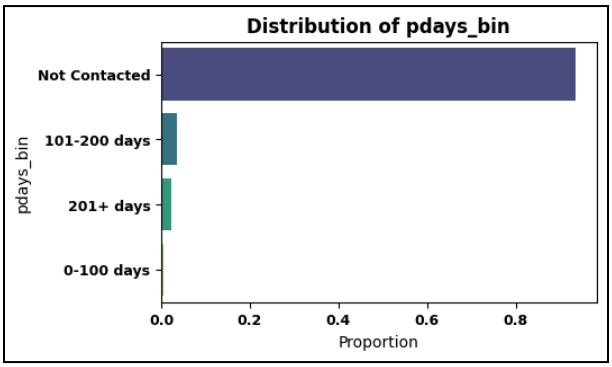


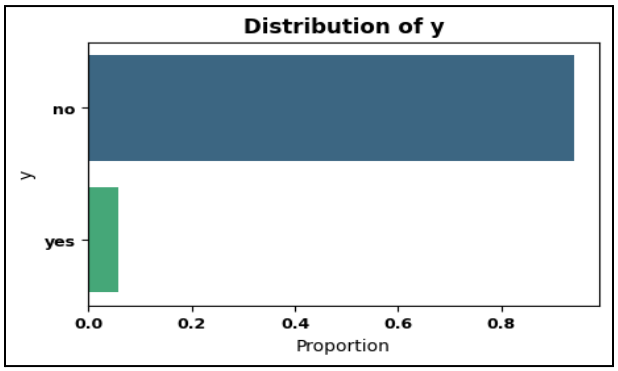




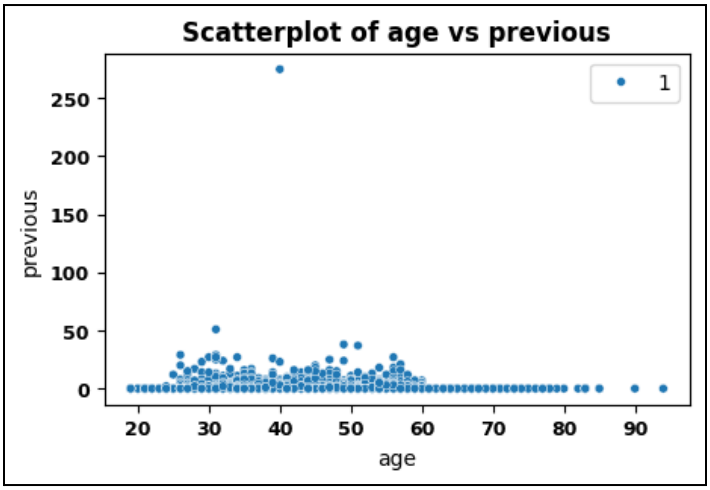

Distribution of poutcome
unknown
failure
other
success
poutcome
0.0
0.2
0.4
0.6
0.8
Proportion

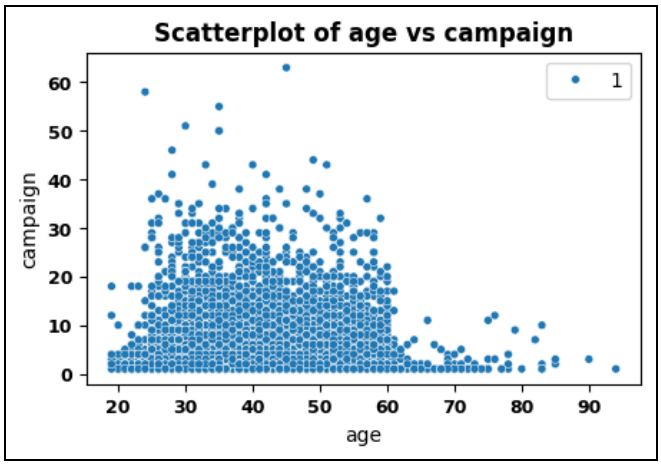

Distribution of day_bin
21-31 days
11-20 days
0-10 days
day_bin
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Proportion

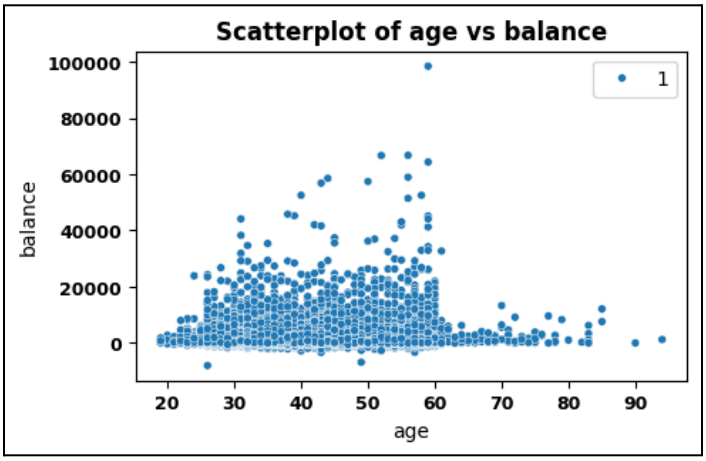

Distribution of pdays_bin
Not Contacted
101-200 days
201+ days
0-100 days
pdays_bin
0.0
0.2
0.4
0.6
0.8
Proportion

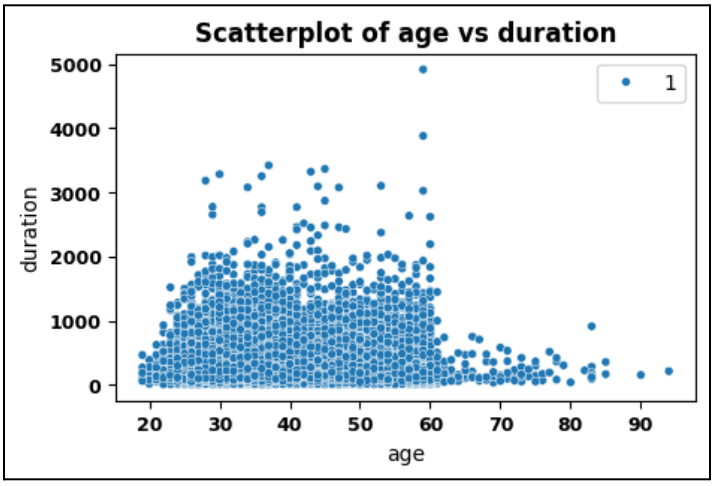

Distribution of y
no
yes
y
0.0
0.2
0.4
0.6
0.8
Proportion

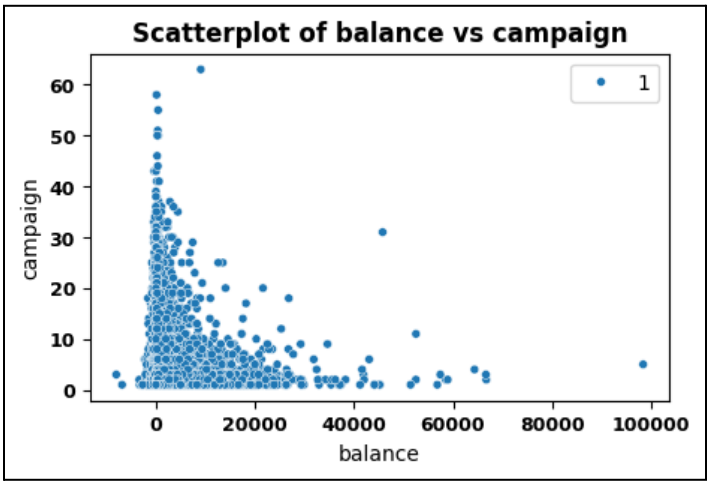
Scatterplots

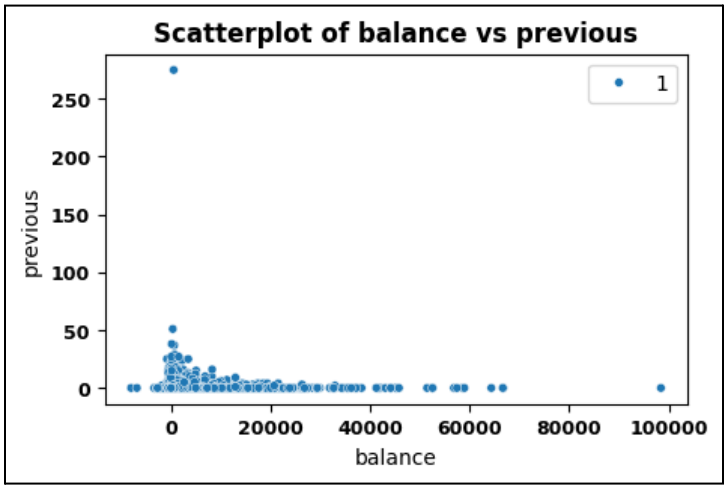


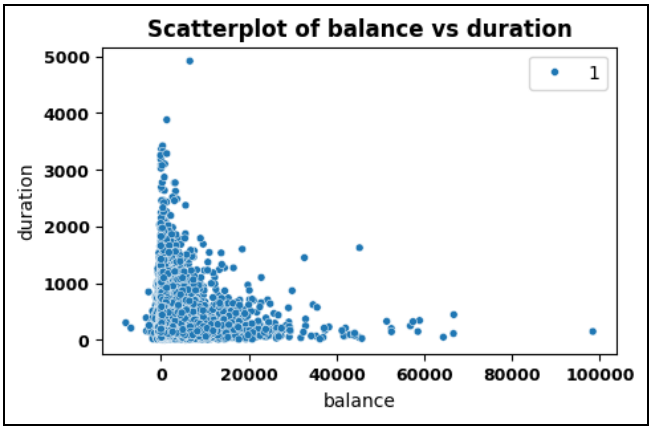


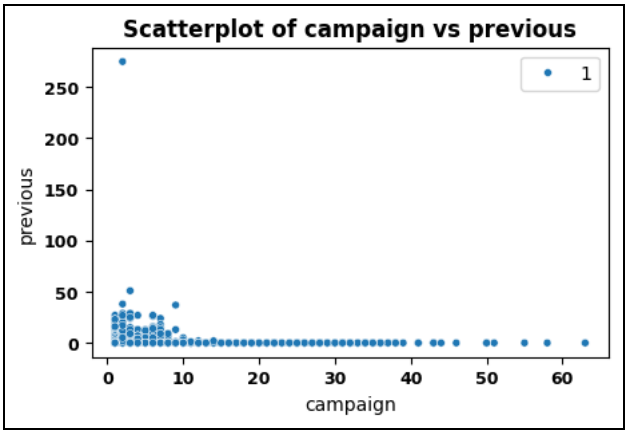


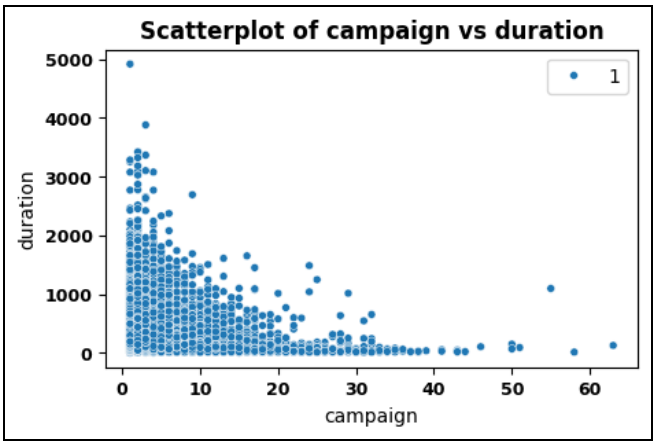


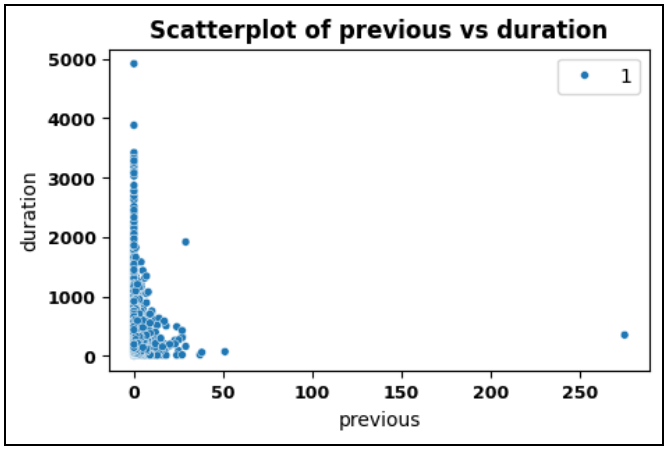





Scatterplot of balance vs duration
duration
balance
1
5000
4000
3000
2000
1000
0
0
20000
40000
60000
80000
100000


Scatterplot of campaign vs previous
previous
campaign
1
250
200
150
100
50
0
0
10
20
30
40
50
60


Scatterplot of campaign vs duration
duration
campaign
1
5000
4000
3000
2000
1000
0
0
10
20
30
40
50
60


Scatterplot of previous vs duration
duration
previous
1
5000
4000
3000
2000
1000
0
0
50
100
150
200
250