Coursework Task 3

Technical Report

Mihai Stavila

SID 8977658

**Coventry GitHub Repository URL**:

<https://github.coventry.ac.uk/stavilam/8977658-MS-s1>

Contents

[Introduction 4](#_Toc87726957)

[Exploratory Data Analysis 4](#_Toc87726958)

[Data Processing and Further Analysis 11](#_Toc87726959)

[Non-numerical values and One-Hot Encoding of Categorical Variables 11](#_Toc87726960)

[Dataset Outliers 11](#_Toc87726961)

[Data Correlation and Multicollinearity 11](#_Toc87726962)

[Feature Scaling 15](#_Toc87726963)

[Train/Set Data Splitting 15](#_Toc87726964)

[Initial Model Performance Evaluation 16](#_Toc87726965)

[Feature Selection 16](#_Toc87726966)

[Recursive Feature Elimination 16](#_Toc87726967)

[Backward Elimination 17](#_Toc87726968)

[Data Sampling 17](#_Toc87726969)

[Model Tuning 18](#_Toc87726970)

[Logistic Regression – Evaluation after Hyperparameter Optimization 18](#_Toc87726971)

[Random Forest Classification – Evaluation after Hyperparameter Optimization 19](#_Toc87726972)

[Model Evaluation 20](#_Toc87726973)

[Conclusion 22](#_Toc87726974)

Table of figures

[Figure 1 - Unequal Class Distribution 4](#_Toc88724122)

[Figure 2 - Target Variable/Mean Features 5](#_Toc88724123)

[Figure 3 - Client Age Distribution 5](#_Toc88724124)

[Figure 4 - Contact Duration 5](#_Toc88724125)

[Figure 5 - Days After Contact from the Previous Campaign 6](#_Toc88724126)

[Figure 6 - Subscription Frequency per Job Type 6](#_Toc88724127)

[Figure 7 - Client Education Level 7](#_Toc88724128)

[Figure 8 - Client Marital Status 7](#_Toc88724129)

[Figure 9 - Contact Month 7](#_Toc88724130)

[Figure 10 - Contact Day of the Month 8](#_Toc88724131)

[Figure 11 - Previous Campaign Outcome 8](#_Toc88724132)

[Figure 12 - Previous Contacts and Subscription Outcome 9](#_Toc88724133)

[Figure 13 - Subscribed Clients per Number of Contacts 9](#_Toc88724134)

[Figure 14 - Client Personal Loan 9](#_Toc88724135)

[Figure 15 - Client Housing Loan 10](#_Toc88724136)

[Figure 16 - Contact Type/Subscription Ratio 10](#_Toc88724137)

[Figure 17 - Default Credit Subscription Ratio 10](#_Toc88724138)

[Figure 18 - Data Outliers 12](#_Toc88724139)

[Figure 19 - Correlation Matrix 1 13](#_Toc88724140)

[Figure 20 - Correlation Matrix 2 14](#_Toc88724141)

[Figure 21 - Predictor/Target Correlation 15](#_Toc88724142)

[Figure 22 - Initial Model Performance Classification Report 16](#_Toc88724143)

[Figure 23 - Initial Model CV Mean Performance 16](#_Toc88724144)

[Figure 24 - RFE Classification Report 16](#_Toc88724145)

[Figure 25 - RFE CV Mean Performance 17](#_Toc88724146)

[Figure 26 - Backward Elimination CV Mean Performance 17](#_Toc88724147)

[Figure 27 - SMOTE CV Mean Performance 18](#_Toc88724148)

[Figure 28 - Logistic Regression Tuning Performance 19](#_Toc88724149)

[Figure 29 - Random Forest Tuning Performance 19](#_Toc88724150)

[Figure 30 - ROC Curve 20](#_Toc88724151)

[Figure 31 - Final Logical Regression CV Mean Performance 21](#_Toc88724152)

[Figure 32 - Final Random Forest CV Mean Performance 21](#_Toc88724153)

|  |
| --- |
| **Academic Report** |

# Introduction

The Bank Marketing Data Set publicly available for research at UCI Machine Learning Repository (Moro et al., 2014) was chosen as the dataset for the classification problem. The data is related to direct marketing campaigns, based on phone calls, of a Portuguese banking institution. The classification goal was to predict whether the client will subscribe a term deposit or not.

Previous approaches to this problem have been the identification of patterns, relationships and rule extraction through data mining and use of the C5.0 supervised learning algorithm, resulting in a 93.37% correct prediction rate (Sing'oei and Wang, 2013). IG and Chi-square feature selection methods have been used to find the highest ranked ten features (Parlar and Acaravci, 2017), which were *duration, poutcome, month, pdays, contact, previous, age, job, housing* and *balance*. Data-processing with WEKA and SMOTE sampling have yielded an AUCPR of 0.935 (Logistic Regression) and 0.989 (Random Forest) for the minority class (Verma, 2019).

# Exploratory Data Analysis

EDA involved studying the dataset’s structure, observations, attributes and their relationships to determine attribute significance, identify flaws in the dataset or data collection processes and enable fine-tuning.

The dataset contains 4521 observations with 17 features (10 categorical, 7 numerical), and no missing values or duplications. The data is highly unbalanced, with a class distribution ratio of 89:11 - 521 (≈ 11.52%) subscribed and 4000 (≈ 88.47%) not subscribed (Figure 1).

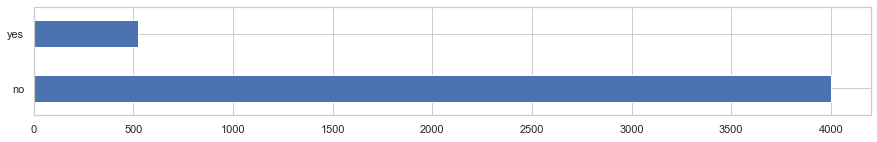


Figure - Unequal Class Distribution

Clients who subscribed a term deposit were slightly older on average (Figure 2); most clients are in the age range of 30-40 (Figure 3).

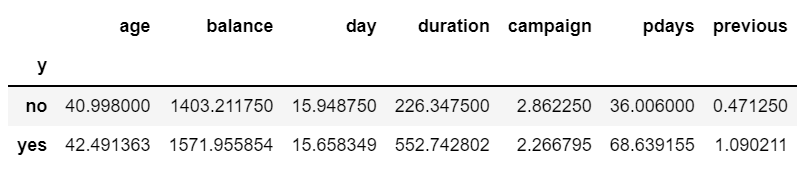


Figure - Target Variable/Mean Features

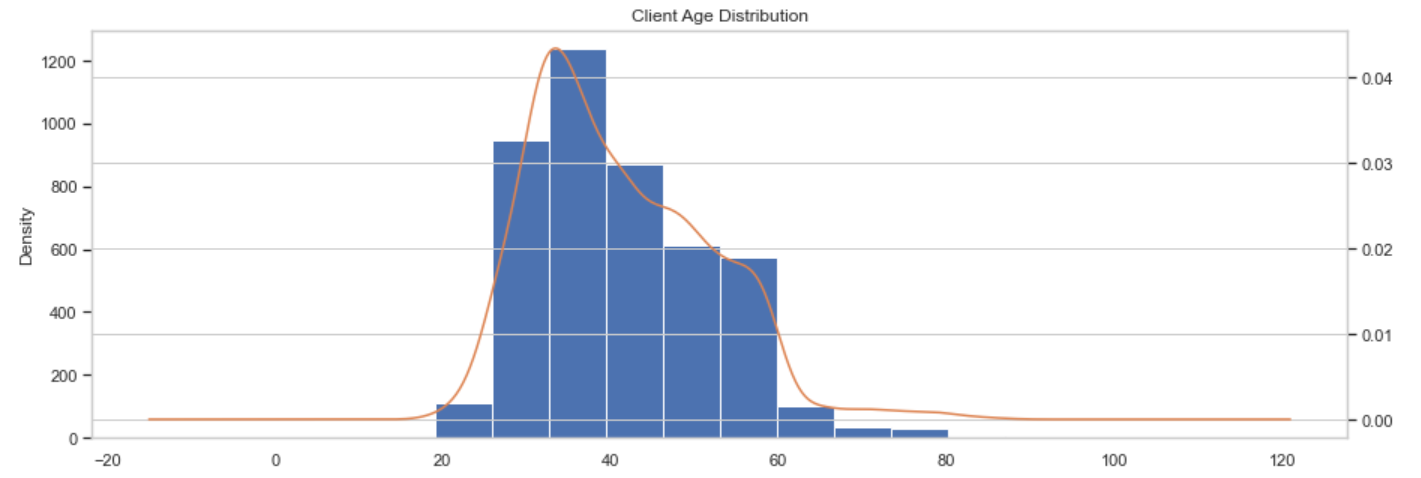


Figure - Client Age Distribution

Subscribed clients have a higher balance, though this may be influenced by outliers, as *balance* is continuous and can vary greatly. (Figure 2)

Contact duration strongly correlates with the outcome - the longer the duration, the higher chance of subscription. (Figure 4)

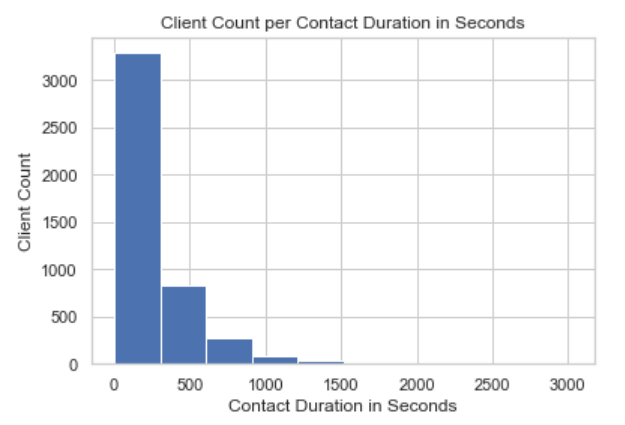
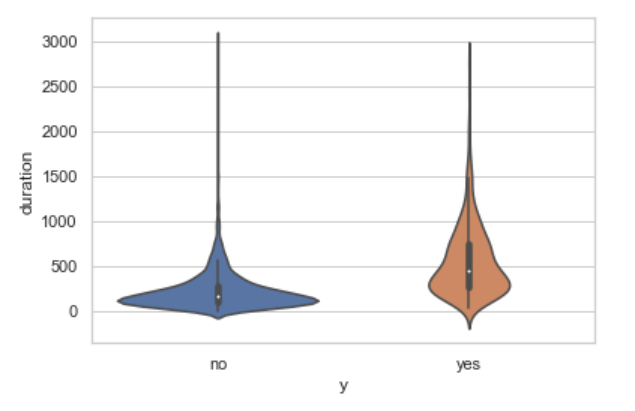


Figure - Contact Duration

Surprisingly, the more days passed after the client was last contacted from a previous campaign, the higher chance of subscription. (Figure 5)

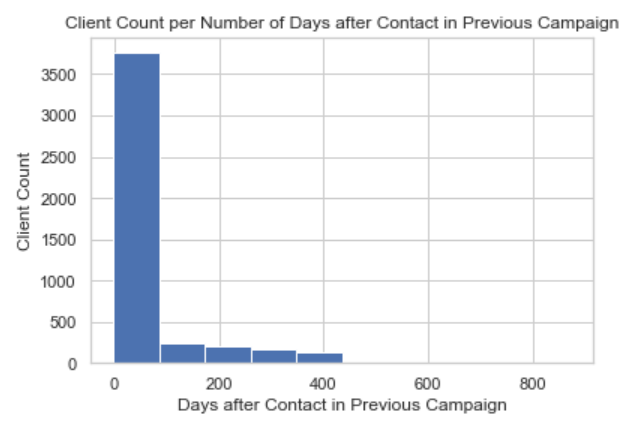
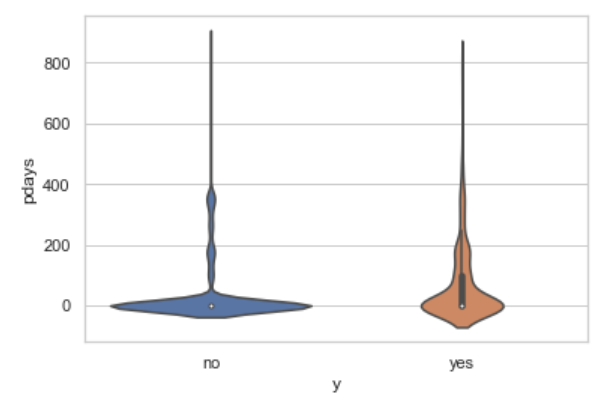


Figure - Days After Contact from the Previous Campaign

Subscription frequency depends greatly on the job type (esp. student and retired), making it a strong outcome predictor (Figure 6). Clients mostly belong to the management, blue-collar and technician, with the least being “unknown”.

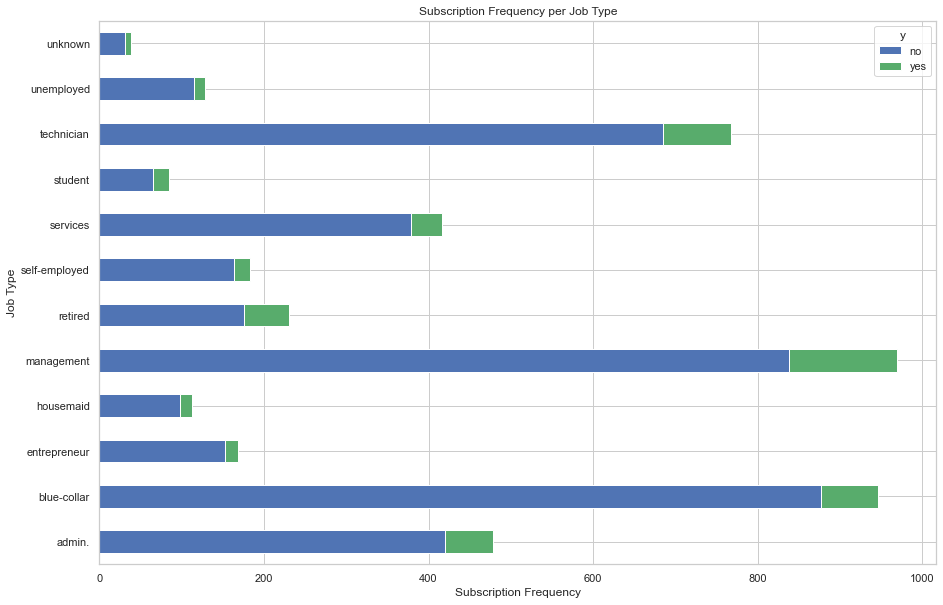


Figure - Subscription Frequency per Job Type

Subscription frequency increases with education level, with tertiary having the largest subscribed ratio (Figure 7); education is not quantitative but categorical due to the lack of equal intervals.

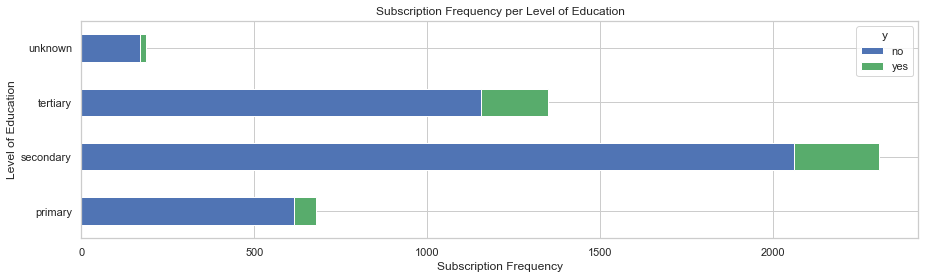


Figure - Client Education Level

Subscription frequency is similar between all marital statuses; most clients are married. (Figure 8)

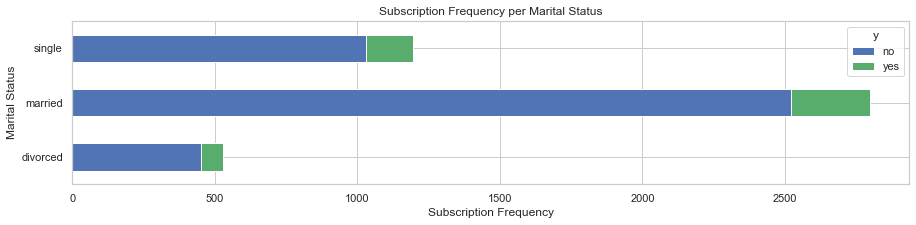


Figure - Client Marital Status

Most clients were contacted in May, with fewest in December and September. March and October have a higher ratio of subscriptions. (Figure 9).

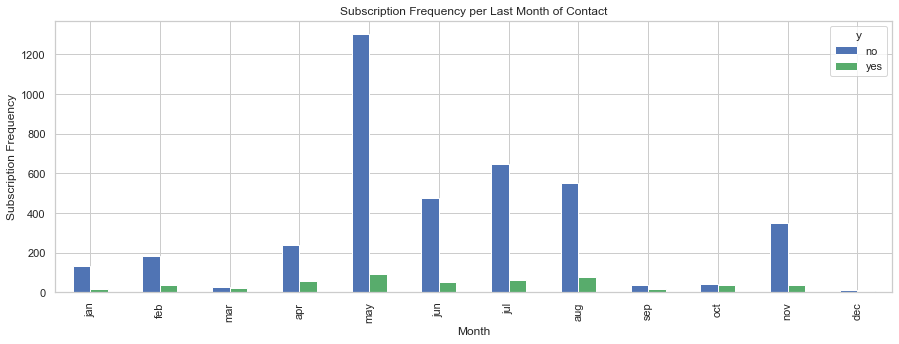


Figure - Contact Month

Slightly more clients were last contacted during the middle of the month, with most contacts being on the 20th (Figure 10), and most subscriptions on the 1st.

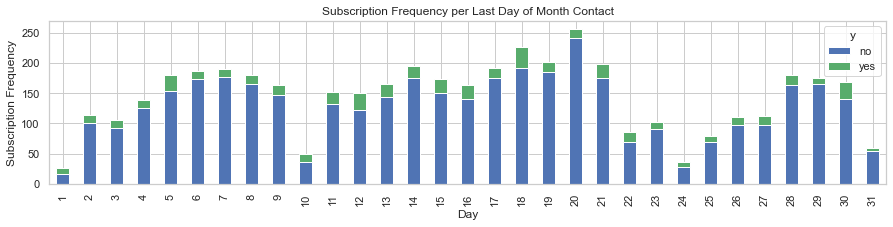


Figure - Contact Day of the Month

Previous campaign outcome has an especially strong correlation to the target variable: If the client has subscribed a term deposit before, there is a high chance they will do so again. (Figure 11)

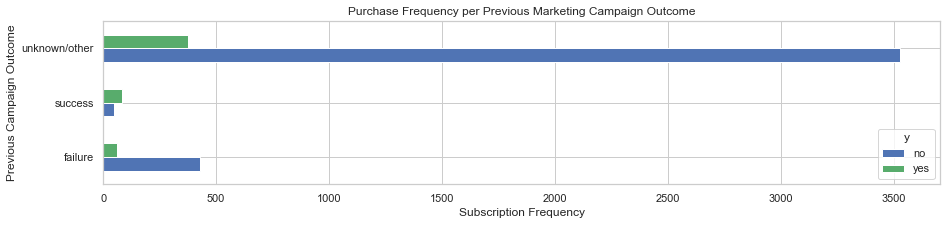


Figure - Previous Campaign Outcome

Clients that were previously contacted have a higher chance of subscribing, making *previous* a strong predictor. However, most clients were not contacted prior to the current campaign. (Figure 12).

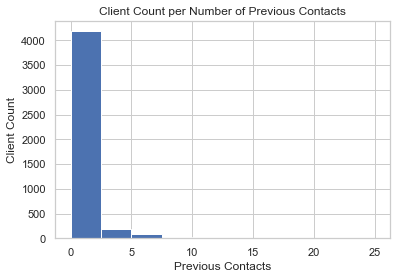
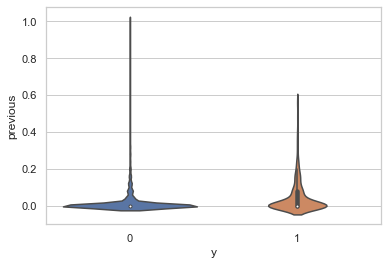


Figure - Previous Contacts and Subscription Outcome

Contacts performed during the current campaign had an unexpectedly negative impact, decreasing the probability of subscription as the number increased. (Figure 13)

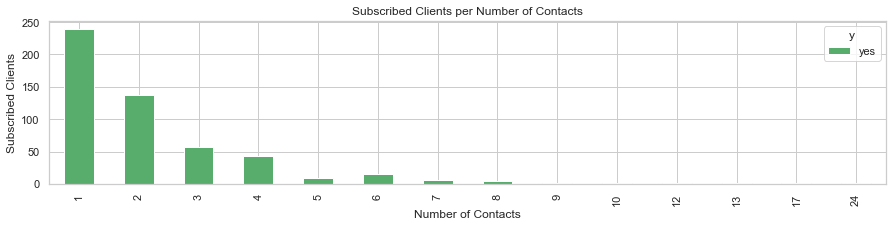


Figure - Subscribed Clients per Number of Contacts

Clients without personal or housing loans are more likely to subscribe (Figure 11 and Figure 14).

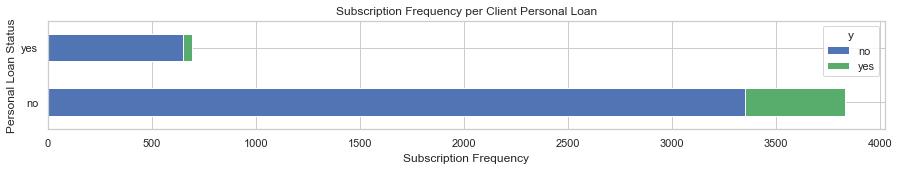


Figure - Client Personal Loan

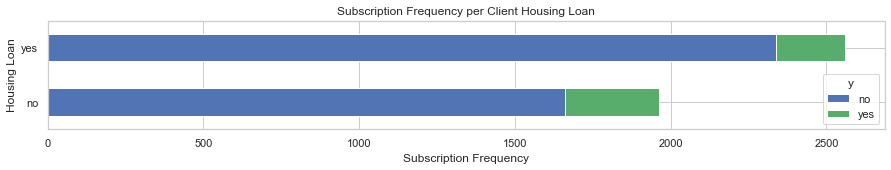
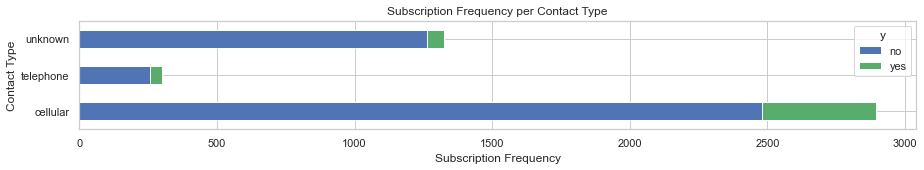


Figure - Client Housing Loan

Most contacts were cellular; subscription ratio is very similar between cellular and telephone, but significantly lower for “unknown” (Figure 16).



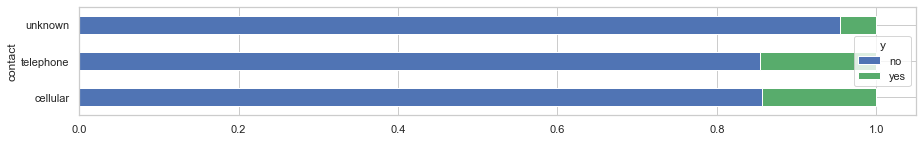


Figure 16 - Contact Type/Subscription Ratio

Few clients have credit in default, though it makes little difference, making *default* a weak predictor (Figure 17).



Figure - Default Credit Subscription Ratio

# Data Processing and Further Analysis

The data should be processed and transformed into a useable (and more informative) form for modelling.

## Non-numerical values and One-Hot Encoding of Categorical Variables

All categorical variables were converted to binary. Those with multiple categories (e.g., *job*, *month*) were One-Hot Encoded into Boolean dummy variables without quantifiable relationships. The least significant resultant of each variable was dropped to reduce multicollinearity (“dummy variable trap”).

## Dataset Outliers

The dataset contains many samples that deviate significantly from the data's mainstream (Figure 18), and although transformations that lessen the impact of data skewness or outliers might result in considerable performance gains (Kuhn and Johnson, 2013) by decreasing data variability, this should be done appropriately, such as when the data is incorrect or not part of the study.

Excluding extreme values purely because of their extremeness might skew the results by erasing information about the research area's intrinsic variability (Frost, 2020).

Thus, outliers were not removed from the dataset as they were considered important for model building due to their numbers (by definition, outliers are rare) and accuracy: after further analysis, they are natural and there seem to be no measurement errors.

## Data Correlation and Multicollinearity

Highly correlated predictors are linearly reliant on other features and contribute less to output prediction while increasing computational costs.

The Variance Influence Factor and correlation matrices were used to mitigate multi-collinearity by identifying predictors with high correlation (>0.7) to one another (Figure 19) or with a VIF above the cut-off point of 5.0. Those features were removed to improve the statistical significance of independent variables (Figure 20).

When choosing which feature to remove, the target/predictor correlation plot (Figure 21) was used to determine significance. Features with high VIF scores were removed iteratively starting from the highest value, with model rebuilding and re-checking in-between, to avoid any unnecessary removal.

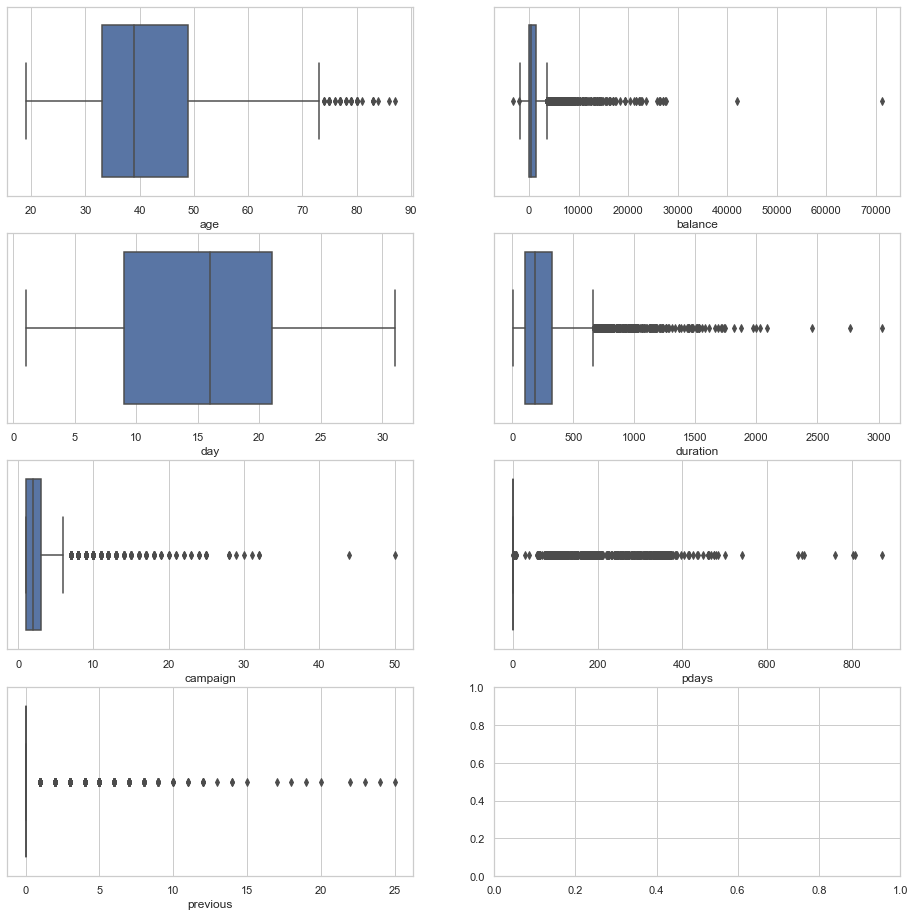


Figure - Data Outliers

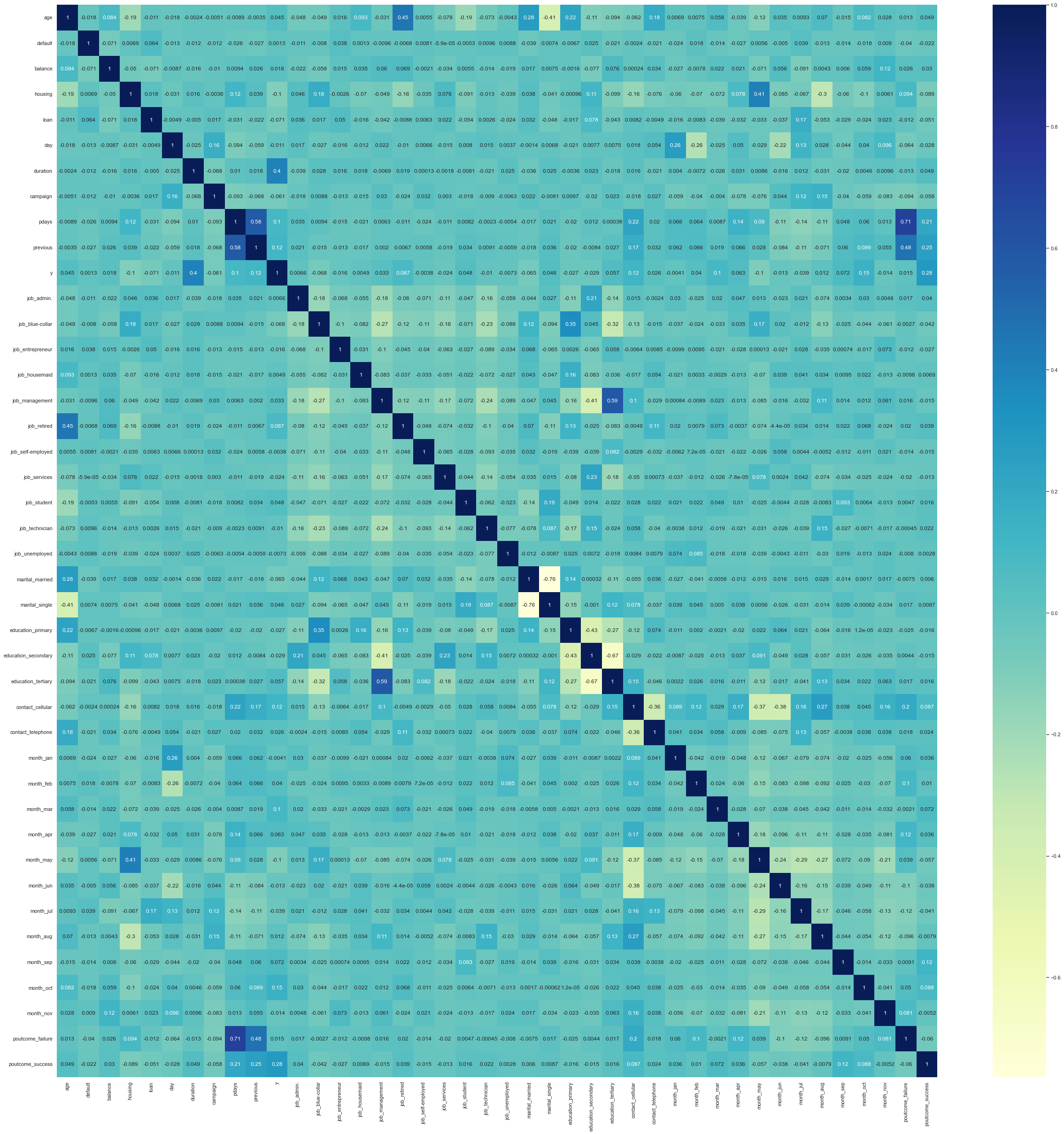


Figure - Correlation Matrix 1

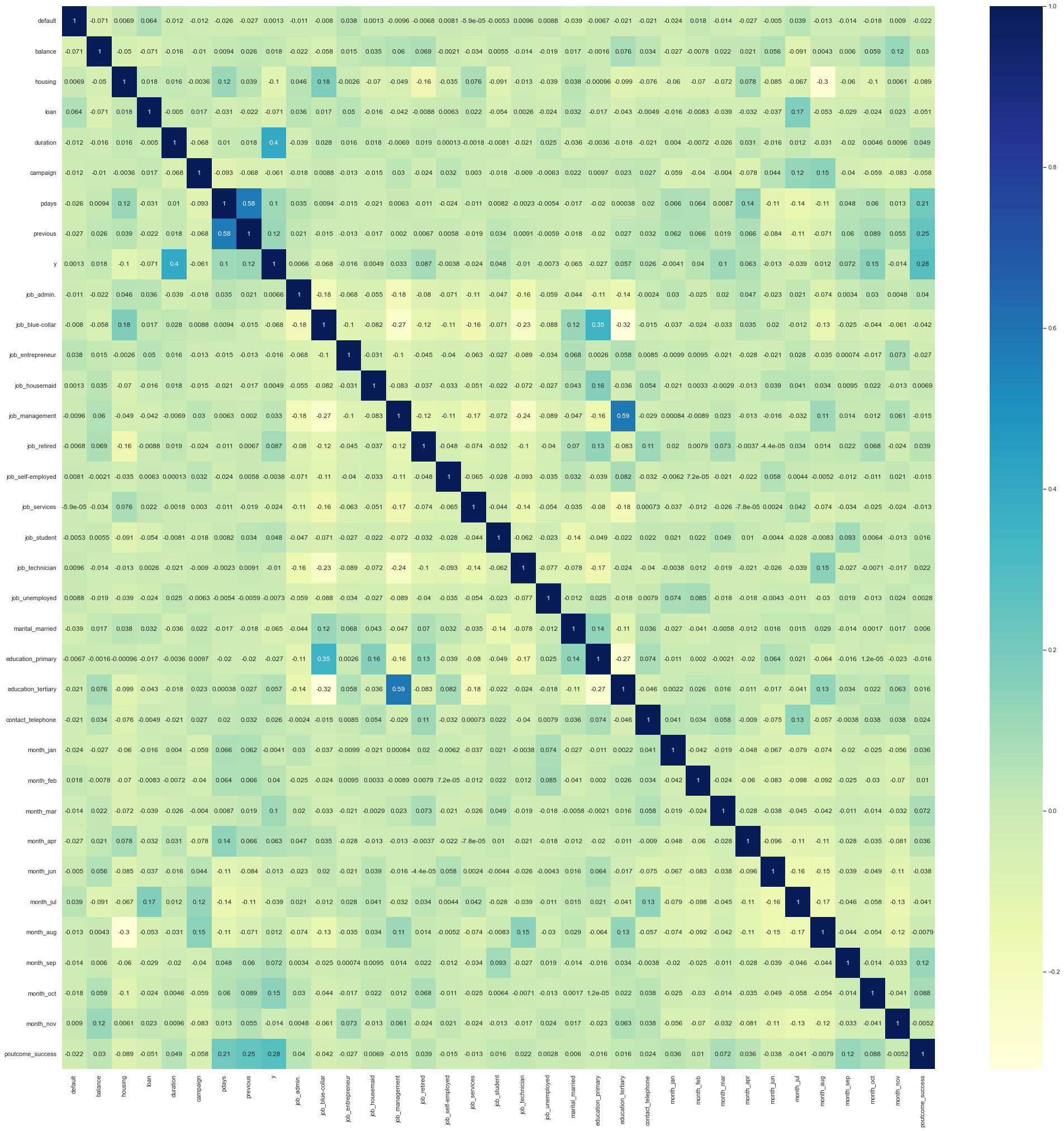


Figure - Correlation Matrix 2

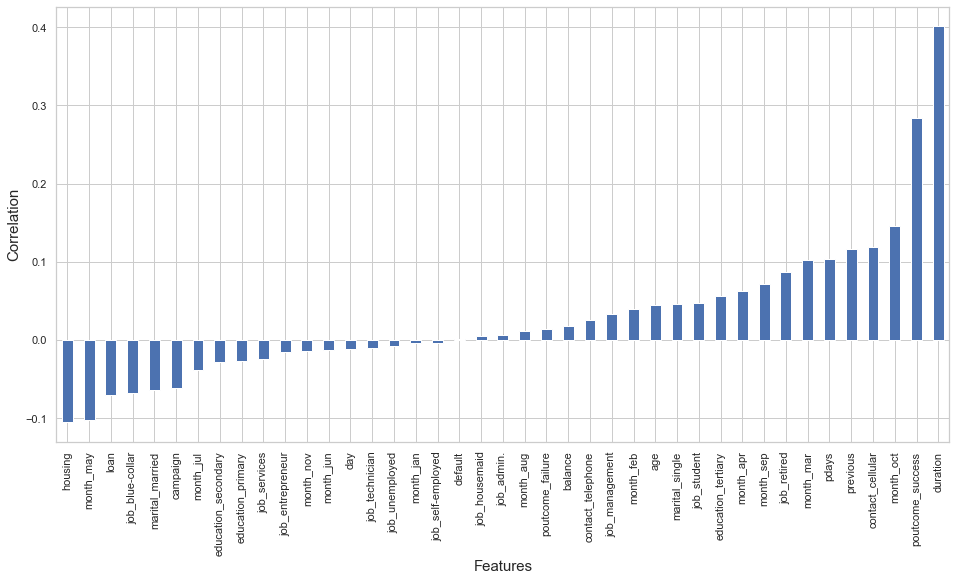


Figure - Predictor/Target Correlation

## Feature Scaling

The data was normalized to ensure features are given equal importance, using MinMaxScaler to preserve the shape of the dataset.

Although scaling is not necessary for the Random Forest Classifier, Logistic Regression uses stochastic gradient descent, and small value ranges will help increase the speed as the data will descend faster. The speed of RFE and other algorithms was also improved.

## Train/Set Data Splitting

The data was split into a training and testing set, following the Pareto Principle (80-20 rule) - approximately 80% of outcomes result from 20% of causes. The train set will be used to build and validate the model, while the test data will give an impartial assessment of the model on previously unseen data. Only the train set will be sampled, to avoid biased models and unduly optimistic estimations (Santos et al., 2018).

## Initial Model Performance Evaluation

The Logistic Regression and Random Forest models were fitted and their performance evaluated using classification reports, confusion matrices and cross-validation mean scores, to determine if further processing is required (Figure 22 and Figure 23). Due to unequal class distribution, the models’ accuracy was falsely high (Accuracy paradox), with a low recall for the minority class. The data would therefore need to be further processed, including train set balancing.

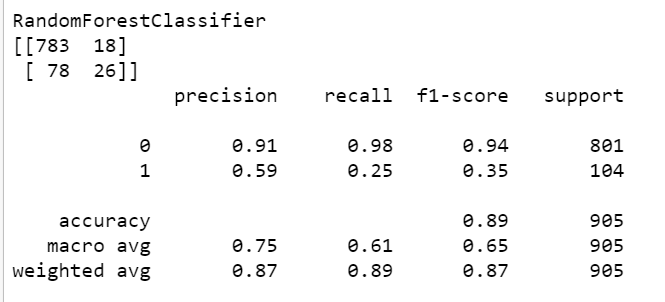
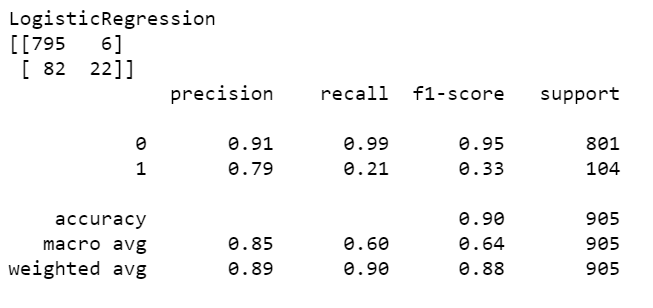


Figure - Initial Model Performance Classification Report

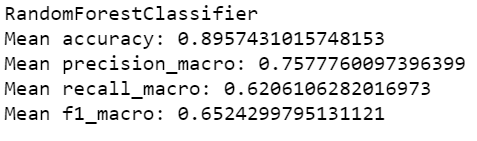
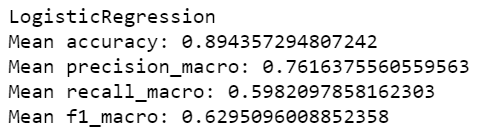


Figure - Initial Model CV Mean Performance

## Feature Selection

Recursive Feature Elimination was used to select features with most significance to the target, by building a model repeatedly and pruning the least important features until all have been exhausted.

RFE increased the recall of both models, especially Random Forest, but came at the cost of precision (Figure 24 and Figure 25) – a good trade-off, as it is far more crucial to identify clients who will subscribe to a term deposit than it is to find customers who may or may not subscribe.

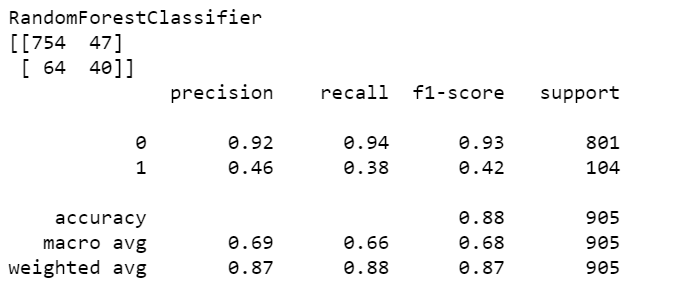
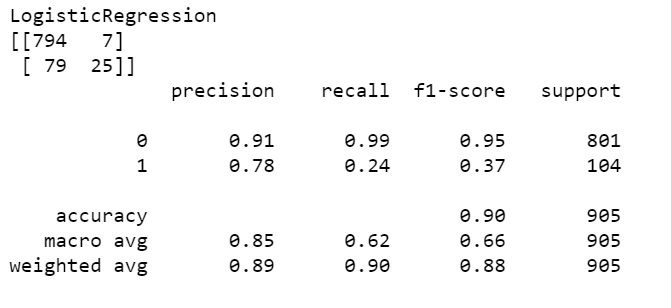


Figure - RFE Classification Report

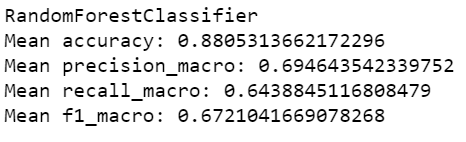
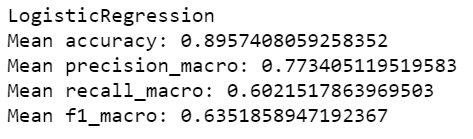


Figure - RFE CV Mean Performance

Backward Elimination was applied for further feature selection, with the Significance Level set at 0.05, a standard that is frequently used to distinguish between statistically significant and non-significant p-values. This does not, however, mean that there is a scientific basis to regard results on opposite sides of a threshold as qualitatively distinct. (Sterne and Davey, 2001; Amrhein et al., 2017). Since p-values should be used to help with decision making, not make the decision, model performance will be compared to decide whether the variables above the threshold should be left in.

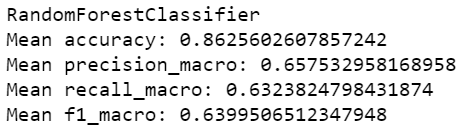
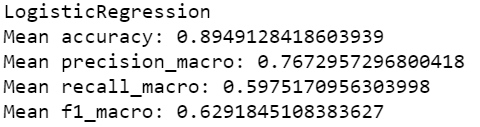


Figure - Backward Elimination CV Mean Performance

Removing features with a p-value above threshold provided no substantial gains, and on the contrary, slightly decreased performance, especially for Random Forest. Since the models perform better with the values included, the data frames will be restored. Ideally, each feature with a high p-value should be individually tested to see if it benefits the model or not.

## Data Sampling

The data was balanced using SMOTE (Synthetic Minority Oversampling Technique) (Chawla et al., 2002). Synthetic samples were generated from existing minority cases (no-subscription), by combining the features of neighbors to generate new examples, without changing the majority cases. This was applied only to the training set to avoid information leakage from the test set into model training. During Cross-Validation, proper over-sampling within each fold was performed with the use of pipelines.

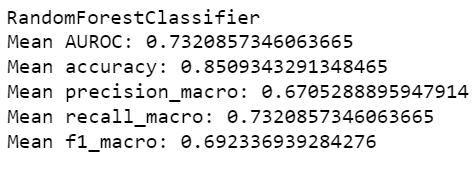
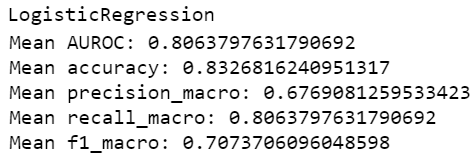


Figure - SMOTE CV Mean Performance

Cross-validation results show that balancing the data has considerably increased the models’ performance, greatly increasing recall at the cost of precision, resulting in a positive f1-score gain, and bringing accuracy down to a more expected level. (Figure 27)

# Model Tuning

Grid Search performs an exhaustive search, looping through the predefined hyperparameters to select the optimal combination. Fitting and evaluating each combination can take a very long time, as GS suffers from the curse of dimensionality, and the more combinations, the less feasible grid-search becomes, especially with the slow predictions of Random Forest.

Therefore, Cross-validated Randomized Search was used instead; choosing between a number of random combinations, it manages to achieve comparable accuracy at only a fraction of the time, and it has been shown empirically and theoretically that it poses higher parameter-optimization efficiency than grid search (Bergstra and Bengio, 2012).

The models were trained and tested using different data subsets via K-fold cross-validation to detect overfitting.

## Logistic Regression – Evaluation after Hyperparameter Optimization

Logistic regression is a supervised learning classification algorithm that uses a logistic sigmoid function to predict the probability of a target variable binary in nature (e.g., subscribed/not-subscribed).

The confusion matrix (Figure 28) shows that the number of true negatives is much larger than true positives. This is due to data class imbalance, as there are not as many TP predictions to be made by comparison to TN, since only ≈11% clients subscribed. The model correctly predicted ≈84% of non-subscribers and ≈73% subscribers.

A small difference between the performance of the train and test sets can be seen (more apparent for recall), which shows that the model is not severely overfitted, although it may perform slightly differently when generalizing new data.

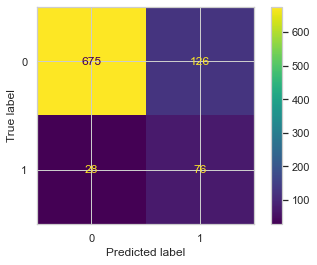
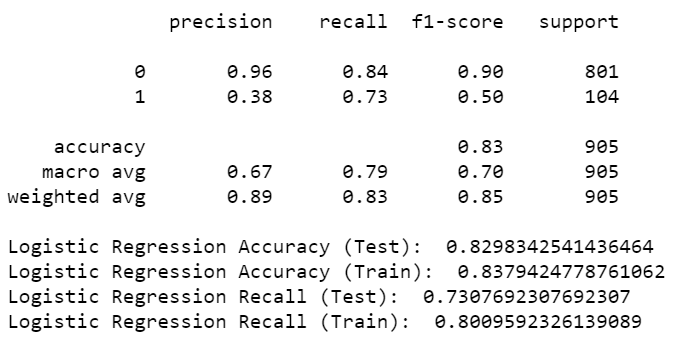


Figure - Logistic Regression Tuning Performance

## Random Forest Classification – Evaluation after Hyperparameter Optimization

The Random Forest Classifier is a decision tree-based classification algorithm that uses randomness in generating individual trees to promote uncorrelated forests, which then uses the forest’s predictive powers to make accurate decisions (Niklas, 2021).

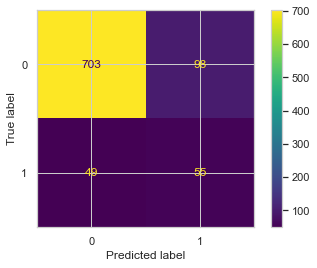
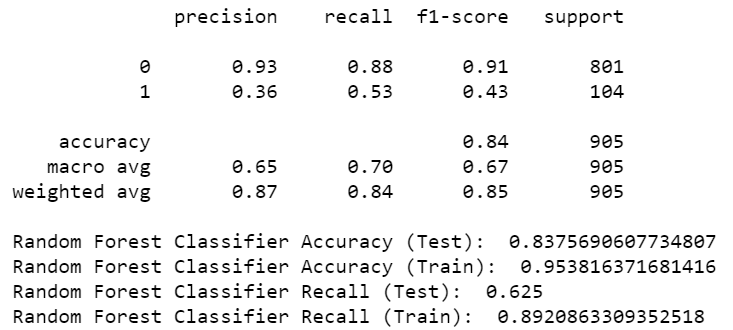


Figure - Random Forest Tuning Performance

Model tuning increased the performance of the Random Forest Classification, especially when predicting subscribed clients, and f1-score improvements can be seen across both classes (Figure 29).

Although Random Forest becomes inherently resistant (but not immune) to overfitting as the number of trees increases, there is a notable difference between the performance of test and train sets. While not as severe from an accuracy perspective, a degree of overfitting is certainly indicated by the recall, arguably the more significant metric.

# Model Evaluation

Cross-validation was once again used to estimate the stability and performance of both the Logistic Regression and Random Forest, using Stratified K-Fold to preserve the dataset’s class distribution.

The ROC (receiver operating characteristic) curve plot illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied ("Receiver operating characteristic", 2021, para. 1) and was used to compare the two models by evaluating their performance at all classification thresholds.

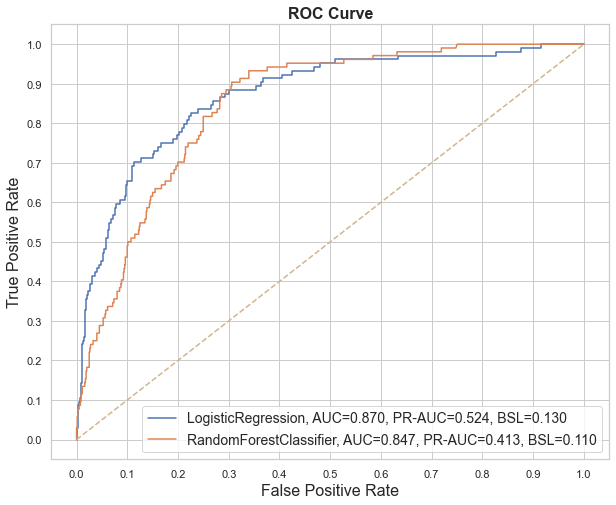


Figure - ROC Curve

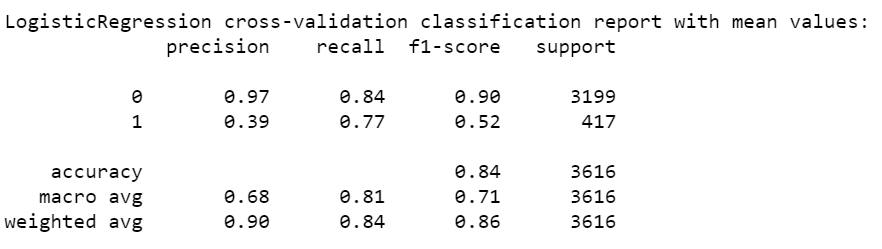


Figure - Final Logical Regression CV Mean Performance

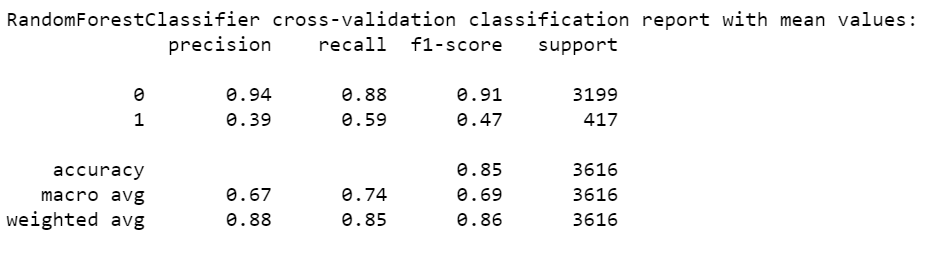


Figure - Final Random Forest CV Mean Performance

Both models display good Area Under the ROC Curve (AUC) scores (Figure 30), with ≈ 87% (LR) and ≈ 85% (RF) change to distinguish between clients who subscribed and those who did not. However, since the AUC score can be unreliable when evaluating a low sample size of minority instances (Fernández et al., 2018), the Precision-Recall Curve (PR-AUC) score will be used as well in estimating the model, as it is recommended for highly skewed domains where ROC curves may provide an excessively optimistic view of the performance (Branco et al., 2015).

Unlike the ROC curve, which covers both classes, the PR curve focuses on the subscribed clients (minority class). Since ≈ 11% of clients subscribed, a random estimator would be expected to have a PR-AUC score of ≈ 0.11 (Saito and Rehmsmeier, 2015). With ≈ 0.52 for Logistic Regression and ≈ 0.41 for Random Forest Classification, the PR-AUC score shows that both models display a substantial performance increase.

The Brier score is the average squared difference between the expected probability and actual outcome. With 0 being the best possible score, the BSL score of both models - LR ≈ 0.13 BSL, RF ≈ 0.11 BSL - infers a high degree of accuracy, especially for Random Forest.

# Conclusion

Data processing techniques such as feature normalization with MinMaxScaler, feature selection with RFE and Backward Elimination, data sampling with SMOTE and handling multicollinearity using VIF and correlation matrices, have been applied.

The performance evaluation of the two models using classification reports, confusion matrices, Stratified K-fold Cross-Validation and the ROC Curve, shows that while there are some small differences, both models perform similarly well in predicting whether the client will subscribe a term deposit, with Logistic Regression displaying a slightly better performance thanks to the higher AUC, PR-AUC and recalling of positive predictions.

Other works using data mining with Logistic Regression on a 70/30 split have shown a 90.42% accuracy, 65.53% sensitivity (recall) and 92.16% specificity (Elsalamony, 2014). An experiment carried out using data mining with WEKA resulted in AUC scores of 90.9% for LR, and 92.7% for RF (Asare-Frempong and Jayabalan, 2017).

[**word count**: 2180 - including in-text citations, and excluding figure citations and cross-references]

|  |
| --- |
| **Bibliography** |

Blagus, R., & Lusa, L. (2010). Class prediction for high-dimensional class-imbalanced data. *BMC bioinformatics*, *11*(1), 1-17.

S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P., 2002. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, pp.321-357.

Dunford, R., Su, Q. and Tamang, E., 2014. The pareto principle.

Santos, M.S., Soares, J.P., Abreu, P.H., Araujo, H. and Santos, J., 2018. Cross-validation for imbalanced datasets: avoiding overoptimistic and overfitting approaches [research frontier]. ieee ComputatioNal iNtelligeNCe magaziNe, 13(4), pp.59-76.

Kuhn, M. and Johnson, K., 2013. Applied predictive modeling (Vol. 26, p. 13). New York: Springer.

Frost, J., 2020. Guidelines for Removing and Handling Outliers in Data.

Bergstra, J. and Bengio, Y., 2012. Random search for hyper-parameter optimization. Journal of machine learning research, 13(2).

Sterne, J.A. and Smith, G.D., 2001. Sifting the evidence—what's wrong with significance tests?. Physical therapy, 81(8), pp.1464-1469.

Goodman, S., 2008, July. A dirty dozen: twelve p-value misconceptions. In Seminars in hematology (Vol. 45, No. 3, pp. 135-140). WB Saunders.

Amrhein, V., Korner-Nievergelt, F. and Roth, T., 2017. The earth is flat (p> 0.05): Significance thresholds and the crisis of unreplicable research. PeerJ 5: e3544.

Niklas, D., 2021, A Complete Guide to the Random Forest Algorithm. Available at: https://builtin.com/data-science/random-forest-algorithm (Accessed: 10 Nov 2021).

'Receiver operating characteristic', 2021, Wikipedia. Available at: https://en.wikipedia.org/wiki/Receiver\_operating\_characteristic (Accessed: 10 Nov 2021)

Saito, T. and Rehmsmeier, M., 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PloS one, 10(3), p.e0118432.

Fernández, A., García, S., Galar, M., Prati, R.C., Krawczyk, B. and Herrera, F., 2018. Learning from imbalanced data sets (Vol. 10, pp. 978-3). Berlin: Springer.

Branco, P., Torgo, L. and Ribeiro, R., 2015. A survey of predictive modelling under imbalanced distributions. arXiv preprint arXiv:1505.01658.

Sing'oei, L., & Wang, J. (2013). Data mining framework for direct marketing: A case study of bank marketing. International Journal of Computer Science Issues (IJCSI), 10(2 Part 2), 198.

Parlar, T., & Acaravci, S. K. (2017). Using data mining techniques for detecting the important features of the bank direct marketing data. International journal of economics and financial issues, 7(2), 692-696.

Verma, A. (2019). Evaluation of classification algorithms with solutions to class imbalance problem on bank marketing dataset using WEKA. IJACSA) International Journal of Advanced Computer Science and Applications.

Elsalamony, H. A. (2014). Bank direct marketing analysis of data mining techniques. International Journal of Computer Applications, 85(7), 12-22.

Asare-Frempong, J., & Jayabalan, M. (2017, September). Predicting customer response to bank direct telemarketing campaign. In *2017 International Conference on Engineering Technology and Technopreneurship (ICE2T)* (pp. 1-4). IEEE.

|  |
| --- |
| **Appendix A** |

< A suggested checklist for you, for full details please refer to the coursework brief >

1. The following naming convention is used for the Coventry GitHub Repository and Coventry OneDrive

StudentID-Initials-s1

For example, for a student Alan Turing whose student ID was 1234567, it should be

1234567-AT-s1

Failing to follow the naming convention may delay the release of marks and feedback for your coursework.

1. **Coventry** GitHub Repository URL **or** **Coventry** OneDrive URL: added to the top of this report
   1. Coventry GitHub Repository includes:

* URL of the selected dataset(s) included in README
* The original selected dataset(s)
* Source-code (.ipynb)
* Demonstration video (.mp4)
  1. Coventry OneDrive folder includes:
* URL of the selected dataset(s) included in a separated text file
* The original selected dataset(s)
* Source-code (.ipynb)
* Demonstration video (.mp4)

1. Source-code added **as text** in Appendix B (below)
2. Submission in the form of a **Word** document. *\*\*Other format is not accepted.*

|  |
| --- |
| **Appendix B** |

1. Coventry University
2. School of Computing, Engineering **and** Mathematics (CEM)
3. Faculty of Engineering, Environment **and** Computing (EEC)
5. ***# 6006CEM – Machine Learning***
6. 2021/2022
8. *# Mihai Stavila*
9. SID 8977658
11. *# Introduction*
12. "The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to assess if the product (bank term deposit) would be ('yes') or not ('no') subscribed."
14. This dataset **is** public available **for** research; the details are described **in** [Moro et al., 2014].
16. The classification goal **is** to predict whether the client will subscribe (yes/no) a term deposit (variable y).
18. *# Setup & Basic Configuration*
20. **import numpy**
21. **import** pandas
22. **import** psutil
23. **from** sklearn **import** preprocessing, metrics
24. **from** sklearn.linear\_model **import** LogisticRegression
25. **from sklearn.model\_selection import train\_test\_split, StratifiedKFold, \**
26. cross\_val\_score, RandomizedSearchCV
27. **from** sklearn.linear\_model **import** LogisticRegression
28. **from** sklearn.feature\_selection **import** RFE
29. **from** sklearn.ensemble **import** RandomForestClassifier
30. **from imblearn.over\_sampling import SMOTE**
31. **from** imblearn.pipeline **import** Pipeline **as** imbpipeline
32. **from** statsmodels **import** api **as** sm\_api
33. **from** statsmodels.stats.outliers\_influence **import** variance\_inflation\_factor
34. **from** matplotlib **import** pyplot
35. **import seaborn as sns**
37. *# make plot outputs appear and be stored within the notebook*
38. %matplotlib inline
40. ***# initial plot settings***
41. color\_settings = {'yes':'#58ac6c', 'no':'#5074b4'}
42. pyplot.rc("font", size=14)
43. sns.set(style="whitegrid", color\_codes=True)
45. **pandas.options.mode.chained\_assignment = None *# default='warn'***
47. *# Loading the data*
48. Load the dataset **and** display the columns, **and** first/last 5 rows
50. **data = pandas.read\_csv('bank.csv', sep=';', header=0)**
51. data.columns
53. data
55. ***# Data Analysis***
56. Check the dataframe **and** relationships between the different attributes **and** uncover important predictors **for** the result variable by visualizing the data.
58. *# Check for null values per column.*
59. *# Since there appear to be none in this dataset, there is no need to use pandas.DataFrame.dropna*
60. **data.isnull().sum()**
62. data.isna().any()
64. *# check if there are any duplicate rows, and if there are, analyse and decide whether it's an error or real data*
65. **print(data.shape)**
66. **print**(data.drop\_duplicates().shape)
68. *# Display further information about the dataset, such as data types and NaN values*
69. data.info()
71. data.info() shows that there are 10 categorical features **and** 7 numerical, **and** there are no null values **in** the dataset.
73. """
74. "Descriptive statistics include those that summarize the central tendency,
75. **dispersion and shape of a dataset’s distribution, excluding NaN values."**
76. """
77. data.describe(include='all')
79. count\_subscribed = len(data[data.y =='yes'])
80. **count\_not\_subscribed = len(data[data.y =='no'])**
81. percentage\_subscribed = count\_subscribed / (count\_not\_subscribed + count\_subscribed)
82. percentage\_not\_subscribed = count\_not\_subscribed / (count\_not\_subscribed + count\_subscribed)
83. **print**(f'Subscribed: {count\_subscribed} (% {percentage\_subscribed \* 100})')
84. **print**(f'Not subscribed: {count\_not\_subscribed} (% {percentage\_not\_subscribed \* 100})')
86. data.y.value\_counts(normalize=False).plot(kind='barh', figsize=(15,2))
87. pyplot.show()
89. data.groupby(['y']).mean()
91. pyplot.figure(figsize=(15,5))
92. ax = data.age.plot.hist()
93. data.age.plot.kde(ax=ax, secondary\_y=True)
94. pyplot.title('Client Age Distribution')
95. **pyplot.xlabel('Age')**
96. pyplot.show()
98. sns.pairplot(data=data, hue='y', vars=['age'], height=5.5)
99. pyplot.show()
101. *## Job Type*
102. job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
104. data.groupby('job').mean()
106. table = pandas.crosstab(data.job, data.y).plot(kind='barh', figsize=(15,10), stacked=True, color=color\_settings)
107. pyplot.title('Subscription Frequency per Job Type')
108. pyplot.xlabel('Subscription Frequency')
109. pyplot.ylabel('Job Type')
110. **pyplot.show()**
112. *## Education Level*
113. education (categorical: 'primary','secondary','tertiary','unknown')
115. **Note: education is a categorical (and not quantitative) variable due to inconsistent distinctions across categories and the lack of equal intervals.**
117. data.groupby('education').mean()
119. table=pandas.crosstab(data.education, data.y).plot(kind='barh', stacked=True, figsize=(15,4), color=color\_settings, ylabel='test')
120. **pyplot.title('Subscription Frequency per Level of Education')**
121. pyplot.xlabel('Subscription Frequency')
122. pyplot.ylabel('Level of Education')
123. pyplot.show()
125. ***## Marital Status***
126. marital : marital status (categorical: 'divorced','married','single'; note: 'divorced' means divorced **or** widowed)
128. data.groupby('marital').mean()
130. **table=pandas.crosstab(data.marital, data.y).plot(kind='barh', stacked=True, figsize=(15,3), color=color\_settings)**
131. pyplot.title('Subscription Frequency per Marital Status')
132. pyplot.xlabel('Subscription Frequency')
133. pyplot.ylabel('Marital Status')
134. pyplot.show()
136. *## Campaign Contacts*
137. number of contacts performed during this campaign **and** **for** this client (numeric, includes last contact)
139. table=pandas.crosstab(data.campaign, data.y).plot(kind='bar', stacked=True, figsize=(15,3), color=color\_settings)
140. **pyplot.title('Subscription Frequency per Number of Contacts')**
141. pyplot.xlabel('Number of Contacts')
142. pyplot.ylabel('Clients')
143. pyplot.show()
145. **table=pandas.crosstab(data.campaign, data.y[data['y'] == 'yes']).plot(kind='bar', stacked=True, figsize=(15,3), color=color\_settings)**
146. pyplot.title('Subscribed Clients per Number of Contacts')
147. pyplot.ylabel('Subscribed Clients')
148. pyplot.xlabel('Number of Contacts')
149. pyplot.show()
151. *## Contact Month*
152. month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
154. month = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']
155. **data['month'] = pandas.Categorical(data['month'], categories=month, ordered=True)**
156. data.sort\_values(by='month', inplace=True)
157. pandas.crosstab(data.month, data.y).plot(kind='bar', figsize=(15,5), color=color\_settings)
158. pyplot.title('Subscription Frequency per Last Month of Contact')
159. pyplot.xlabel('Month')
160. **pyplot.ylabel('Subscription Frequency')**
161. pyplot.show()
163. *## Contact Day of Month*
164. day : on which day of the month was the client contacted
166. table=pandas.crosstab(data.day, data.y).plot(kind='bar', stacked=True, figsize=(15,3), color=color\_settings)
167. pyplot.title('Subscription Frequency per Last Day of Month Contact')
168. pyplot.xlabel('Day')
169. pyplot.ylabel('Subscription Frequency')
170. **pyplot.show()**
172. *## Previous Marketing Campaign Outcome*
173. poutcome: outcome of the previous marketing campaign (categorical: 'failure','success','unknown','other')
175. **data['poutcome'] = numpy.where(data['poutcome'] == 'unknown', 'unknown/other', data['poutcome'])**
176. data['poutcome'] = numpy.where(data['poutcome'] == 'other', 'unknown/other', data['poutcome'])
177. pandas.crosstab(data.poutcome, data.y).plot(kind='barh', figsize=(15,3), color=color\_settings)
178. pyplot.title('Purchase Frequency per Previous Marketing Campaign Outcome')
179. pyplot.xlabel('Subscription Frequency')
180. **pyplot.ylabel('Previous Campaign Outcome')**
181. pyplot.show()
183. *## Previous Contacts*
184. previous: number of contacts performed before this campaign **and** **for** this client (numeric)
186. data.previous.hist()
187. pyplot.title('Client Count per Number of Previous Contacts')
188. pyplot.xlabel('Previous Contacts')
189. pyplot.ylabel('Client Count')
190. **pyplot.show()**
192. *## Days after Previous Contact*
194. pdays: number of days that passed by after the client was last contacted **from** a previous campaign (numeric; -1 means client was **not** previously contacted).
196. data.pdays.hist()
197. pyplot.title('Client Count per Number of Days after Contact in Previous Campaign')
198. pyplot.xlabel('Days after Contact in Previous Campaign')
199. pyplot.ylabel('Client Count')
200. **pyplot.show()**
202. sns.violinplot(x='y', y='pdays', data=data)
203. pyplot.show()
205. ***## Client Personal Loan***
206. loan: has personal loan? (categorical: 'no','yes','unknown')
208. table=pandas.crosstab(data.loan, data.y).plot(kind='barh', stacked=True, figsize=(15,2), color=color\_settings, ylabel='test')
209. pyplot.title('Subscription Frequency per Client Personal Loan')
210. **pyplot.xlabel('Subscription Frequency')**
211. pyplot.ylabel('Personal Loan Status')
212. pyplot.show()
214. *## Client Housing Loan*
216. housing: has housing loan? (categorical: 'no','yes','unknown')
218. table=pandas.crosstab(data.housing, data.y).plot(kind='barh', stacked=True, figsize=(15,2), color=color\_settings, ylabel='test')
219. pyplot.title('Subscription Frequency per Client Housing Loan')
220. **pyplot.xlabel('Subscription Frequency')**
221. pyplot.ylabel('Housing Loan')
222. pyplot.show()
224. *## Contact Type*
225. **contact: contact communication type (categorical: 'cellular','telephone')**

228. table=pandas.crosstab(data.contact, data.y).plot(kind='barh', stacked=True, figsize=(15,2), color=color\_settings, ylabel='test')
229. pyplot.title('Subscription Frequency per Contact Type')
230. **pyplot.xlabel('Subscription Frequency')**
231. pyplot.ylabel('Contact Type')
232. pyplot.show()
234. table=pandas.crosstab(data.contact, data.y)
235. **table.div(table.sum(1).astype(float), axis=0).plot(kind='barh', stacked=True, figsize=(15,2), color=color\_settings, ylabel='test')**
236. pyplot.show()
238. *## Default (Credit)*
239. default: has credit **in** default? (categorical: 'no','yes','unknown')
241. table=pandas.crosstab(data.default, data.y).plot(kind='barh', stacked=True, figsize=(15,2), color=color\_settings, ylabel='test')
242. pyplot.title('Subscription Frequency per Client Credit in Default')
243. pyplot.xlabel('Subscription Frequency')
244. pyplot.ylabel('Credit in Default')
245. **pyplot.show()**
247. table=pandas.crosstab(data.default, data.y)
248. table.div(table.sum(1).astype(float), axis=0).plot(kind='barh', stacked=True, figsize=(15,2), color=color\_settings, ylabel='test')
249. pyplot.show()
251. *## Contact Duration*
253. duration: last contact duration, **in** seconds (numeric).
255. **Note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.**
257. data.duration.hist()
258. pyplot.title('Client Count per Contact Duration in Seconds')
259. pyplot.xlabel('Contact Duration in Seconds')
260. **pyplot.ylabel('Client Count')**
261. pyplot.show()
263. sns.violinplot(x='y', y='duration', data=data)
264. pyplot.show()
266. *# Data Processing*
267. Transform the data to make it more relevant **and** useable **with** sklearn.
269. data.head(10)
271. *## Converting all yes/no columns to binary*
273. *# get columns that store a value of yes/no*
274. yes\_no\_columns = ['y', 'default', 'loan', 'housing']
276. *# replace yes/no with 0/1 and make sure the feature type is changed to integer*
277. **for** column **in** yes\_no\_columns:
278. data[column] = numpy.where(data[column] == 'yes', 1, data[column])
279. data[column] = numpy.where(data[column] == 'no', 0, data[column])
280. **data[column] = data[column].astype(str).astype(int)**
282. data.head()
284. *## One-Hot Encoding for the Categorical Variables*
285. **Convert categorical values (e.g. job, month, marital etc.) into Boolean dummy/indicator variables with no quantifiable relationships between them, to prepare the data for modeling.**
287. *# Return subset of all the object and categorical dtype columns in the dataframe*
288. categorical\_subset = data.select\_dtypes(include=['object', 'category'])
289. categorical\_subset
291. categorical\_columns = categorical\_subset.columns.values.ravel()
293. **for** column **in** categorical\_columns:
294. *# convert categorical variables into dummy indicator variables*
295. **dummy\_vars = pandas.get\_dummies(data[column], prefix=column)**
296. temp = data.join(dummy\_vars)
297. data = temp
299. *# drop one of the results for each categorical var to mitigate dummy variable trap (multicollinearity)*
300. ***# manually instead of drop\_first=True, as columns of less obvious significance can be removed instead***
301. result\_cols\_to\_drop = ['job\_unknown', 'marital\_divorced', 'education\_unknown', 'contact\_unknown', 'month\_dec', 'poutcome\_unknown/other']
302. data = data.drop(result\_cols\_to\_drop, axis=1)
304. *# remove categorical variables from the data*
305. **data = data.drop(categorical\_columns, axis=1)**
307. **print**(data.columns.values)
308. **print**(data.info())
310. ***## Data Outliers***
311. Get attributes **with** continuous values **and** plot them to identify potential outliers.
313. *# Get a dictionary of max column values (e.g., {'age': 90, 'balance': 110000, ...})*
314. columns\_max\_val = data.max().to\_dict()
315. ***# Take only the columns with max values above 1 (continuous variables only)***
316. continuous\_columns = [key **for** key, val **in** columns\_max\_val.items() **if** val > 1]
317. continuous\_columns
319. continuous\_subset = data[continuous\_columns]
320. **continuous\_subset.describe(percentiles=[0.25,0.5,0.75,0.90,0.95,0.99])**
322. *# calculate how many plots are needed*
323. cont\_count = len(continuous\_columns)
324. fig, axes = pyplot.subplots(int(numpy.ceil(cont\_count/2)), 2, figsize=(16,16))
326. *# programmatically display the plots of outliers*
327. r, c = 0, 0
328. **for** column **in** continuous\_columns:
329. sns.boxplot(x=data[column], ax=axes[r, c])
330. **if c == 1:**
331. r += 1
332. c = 0
333. **else**:
334. c += 1
336. *# check for data points located outside the whiskers of the box plot*
337. pyplot.show()
339. *## Data Correlation*
340. **Correlation coefficients of less than 0.3 are regarded weak, 0.3-0.7 are considered moderate and those above 0.7 are considered high. A color-encoded matrix of the dataframe's correlation of columns will be plotted and used to detect strong correlations between predictor features.**
342. ### Correlation between target and predictors
344. # Apply a function alongside the axis of the dataframe to get correlation to target variable for each predictor column/feature
345. **target\_correlation = data.drop("y", axis=1).apply(lambda x: x.corr(data.y))**
346. target\_correlation = target\_correlation.sort\_values(ascending=True)
347. target\_correlation.plot(kind='bar', figsize=(16,8))
349. pyplot.xlabel("Features", fontsize=15)
350. **pyplot.ylabel("Correlation", fontsize=15)**
351. pyplot.show()
353. The target/predictor correlation plot shows that contact duration has a great impact on client subscription. During analysis, it was shown that the longer the call duration, the higher the chance of the client subscribing.
355. **Previous campaign outcome also highly affects the output: if a client has subscribed before, he is more likely to do so again.**
357. Cellular contact is shown to have a high correlation due to the majority of contacts being of this type.
359. Other high-correlation predictors include housing, month\_may & month\_oct, previous contact count and days that passed since client was contacted from a previous campaign.


363. ### Predictor Multicollinearity
364. Plot a color-encoded correlation matrix of all variables, then compute the variance inflation factor for each column, and begin the iterative process of removing multicollinear features, starting with the highest-correlated predictors with the largest VIF values.
366. pyplot.figure(figsize=(data.shape[1], data.shape[1]))
367. sns.heatmap(data.corr(), annot=True, cmap="YlGnBu")
368. pyplot.show()
370. **Using the correlation matrix and VIF (Variance Inflation Factor) of the data, features with high correlation to other predictors have been identified.**
372. def show\_high\_correlation():
373. """ Display high correlations between features/columns """
374. high\_corr = {}
375. **columns\_correlations = data.corr().to\_dict()**
376. for column, correlations in columns\_correlations.items():
377. for col, corr in correlations.items():
378. if (corr > 0.7 or corr < -0.7) and column != col:
379. high\_corr[column] = {}
380. **high\_corr[column][col] = corr**
381. return high\_corr
383. show\_high\_correlation()
385. **# Manually drop features of less importance to the outcome using the target/predictor correlation bar graph**
386. data = data.drop(['poutcome\_failure', 'marital\_single'], axis=1)
388. show\_high\_correlation()
390. **# One recommendation is that if Variance Inflation Factor is greater than 5, then the explanatory variable given by exog\_idx is**
391. # highly collinear with the other explanatory variables, and the parameter estimates will have
392. # large standard errors because of this.
393. # Therefore, the cut-off point will be 5.0
394. vif\_cutoff = 5.0
395. **def high\_vif\_variables():**
396. """ Compute the VIF for all variables and return those with a VIF value greater than the cutoff point. """
397. high\_corr\_col = {}
398. for column in data.columns:
399. i = data.columns.get\_loc(column)
400. **vif = variance\_inflation\_factor(data, i)**
401. if vif > vif\_cutoff:
402. high\_corr\_col[column] = vif
403. return high\_corr\_col
405. **high\_vif\_variables()**

408. # Remove high VIF variables starting with the largest, until there are no more variables with a VIF higher than the cutoff point.
409. # When dropping variables with a high VIF, the VIF value of other variables will be consequently lowered.
410. **# Dropping the variable with the highest VIF on each iteration ensures that other variables are not dropped needlessly.**
412. dropped\_cols = {}
413. while high\_vif\_variables():
414. high\_vif = high\_vif\_variables()
415. **print(f'Variables with high VIF:\n{high\_vif}')**
416. highest\_vif\_col = max(high\_vif, key=high\_vif.get)
417. highest\_vif\_val = high\_vif[highest\_vif\_col]
418. data = data.drop([highest\_vif\_col], axis=1)
419. print(f'Dropped "{highest\_vif\_col}" **with** VIF {highest\_vif\_val}', end='\n\n')
420. **dropped\_cols[highest\_vif\_col] = highest\_vif\_val**
422. print(f'Dropped all variables **with** a VIF greater than {vif\_cutoff} **from** the dataset.')
424. print(f'List of dropped variables **and** their respective VIF value at time of removal:\n{dropped\_cols}')

427. ### Re-checking the correlation matrix
428. After removing the high-correlation/high-VIF variables, the overall dataset predictor collinearity has been visibly reduced, and therefore the statistical significance of independent variables has been improved.
430. **pyplot.figure(figsize=(data.shape[1], data.shape[1]))**
431. sns.heatmap(data.corr(), annot=True, cmap="YlGnBu")
432. pyplot.show()
434. ## Feature Scaling
435. **Scale all values in the [0, 1] range and preserve the dataset shape with MinMaxScaler.**
437. data.head()
439. # Only scale columns with values larger than one.
440. **columns\_to\_scale = [col for col in data.columns if numpy.issubdtype(data[col].max(), numpy.integer)**
441. and data[col].max() > 1]
443. mm\_scaler = preprocessing.MinMaxScaler()
445. **# Scale the training and test data in the same way**
446. data[columns\_to\_scale] = mm\_scaler.fit\_transform(data[columns\_to\_scale])
448. data.head()
450. **## Data Splitting**
451. Split the data into train set (80%) for model building and validation and test set (20%) for a final unbiased evaluation.
453. ### Separating the predictor features and target variable
455. **X = data.loc[:, data.columns != 'y']**
456. y = data.loc[:, data.columns == 'y']
458. X
460. **y**
462. ### Train Test Split
463. Split the data into train and test subsets.
465. **X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=24, stratify=y)**
466. print(f'X\_train: {len(X\_train)}, X\_test: {len(X\_test)}')
468. ## Initial Model Performance
469. Display the confusion matrix and classification report to help analyze the models' performance (Logistic Regression & Random Forest Classification).
471. **def** display\_model\_performance():
472. """
473. Fit each classifier to the current training data and display the model's performance with default parameters
474. using its confusion matrix and classification report.
475. **"""**
476. model\_logit = LogisticRegression(max\_iter=10000)
477. model\_logit = model\_logit.fit(X\_train, y\_train.values.ravel())
478. **print**(model\_logit.\_\_class\_\_.\_\_name\_\_)
479. **print**(metrics.confusion\_matrix(y\_test, model\_logit.predict(X\_test)))
480. **print(metrics.classification\_report(y\_test, model\_logit.predict(X\_test)), end='\n\n')**
482. rforest\_model = RandomForestClassifier()
483. rforest\_model = rforest\_model.fit(X\_train, y\_train.values.ravel())
484. **print**(rforest\_model.\_\_class\_\_.\_\_name\_\_)
485. **print(metrics.confusion\_matrix(y\_test, rforest\_model.predict(X\_test)))**
486. **print**(metrics.classification\_report(y\_test, rforest\_model.predict(X\_test)))
488. display\_model\_performance()
490. ***### Cross-Validation Mean Score***
492. *# Preserve dataset class distribution with StratifiedKFold*
493. *# Fixed random state ensures the same shuffle and therefore same split each time the code is run*
494. *# Shuffle to get meaningful cross-validation result as the data ordering is not arbitrary*
495. **stratified\_kfold = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=11)**
497. **def** display\_model\_performance\_mean():
498. **for** estimator **in** [LogisticRegression(max\_iter=1000), RandomForestClassifier()]:
499. **print**(estimator.\_\_class\_\_.\_\_name\_\_)
500. **for metric in ['accuracy', 'precision\_macro', 'recall\_macro', 'f1\_macro']:**
501. score = cross\_val\_score(estimator=estimator, X=X\_train, y=y\_train.values.ravel(), cv=stratified\_kfold, scoring=metric)
502. **print**(f'Mean {metric}: {numpy.mean(score)}')
503. **print**('**\n**')
505. **display\_model\_performance\_mean()**
507. *## Further Processing*
509. *### Feature Selection - Recursive Feature Elimination*
510. **Use the RFE algorithm to select the most relevant attributes to the target variable and remove those with low significance by iteratively building a model and pruning the least important features each time, until all the dataset's features have been checked. On the trimmed set, this approach is continued recursively until the required number of features to choose is attained, hence the name "Recursive Feature Elimination".**
512. Note: According to the SMOTE paper, feature selection should be performed before over-sampling.
514. # initialize and fit the train set to RFE with Logit estimator
515. **selector = RFE(estimator=LogisticRegression(max\_iter=10000), step=1)**
516. selector = selector.fit(X\_train, y\_train.values.ravel())
518. # get the list of features supported by the RFE algorithm
519. selected\_features = [X\_train.columns[i] for i in range(len(X\_train.columns)) if selector.support\_.tolist()[i] is True]
521. # get the RFe results (feature support status and ranking)
522. rfe\_results = list(zip(X\_train.columns, selector.support\_, selector.ranking\_))
524. X\_train = X\_train[selected\_features]
525. **X\_test = X\_test[selected\_features]**
527. print(f'Selected features: {selected\_features}')
528. rfe\_results

531. display\_model\_performance()
533. display\_model\_performance\_mean()
535. **### Feature Selection - Backward Elimination**
537. Backward elimination will be used as part of the feature selection process. The significance level (alpha, or threshold) will be set at 0.05. Any variables with a p-value of less than 0.05 will be considered statistically significant. As a result, any variables with a p-value greater than 0.05 will be considered not significant and removed from the dataset.
539. The model will be fitted and the predictor with the highest p-value will be programmatically deleted in each iteration until the highest p-value is less than the significance level (0.05). Thus, only the most essential features will be chosen for the model, and those that are not significant enough will be filtered.
541. print(sm\_api.Logit(y\_train, X\_train).fit().summary())
543. def high\_pvalues():
544. """
545. **Fit model with the current train data and return the column name**
546. and p-value for all columns with a p-value above the threshold.
547. """
548. # disp set at 0 to silence statsmodels.fit()
549. fit = sm\_api.Logit(y\_train, X\_train).fit(disp=0)
550. **high\_pvalues = {}**
552. # find all columns whose P value is higher than the cut-off point and return them
553. for column\_index in range(0, len(X\_train.columns)):
554. if fit.pvalues[column\_index] > 0.05:
555. **high\_pvalues[X\_train.columns[column\_index]] = fit.pvalues[column\_index]**
556. return high\_pvalues
558. # Make train & test copies before alteration
559. X\_train\_bkp = X\_train
560. **X\_test\_bkp = X\_test**
562. dropped\_cols\_pvalue = {}
563. # while there are features above the threshold, remove the highest one then repeat
564. # this way there is no need to do it manually
565. **while high\_pvalues():**
566. high\_pvalues\_ = high\_pvalues()
567. print(f'Variables **with** P-value greater than 0.05:\n{high\_pvalues\_}')
568. highest\_pvalue\_col = max(high\_pvalues\_, key=high\_pvalues\_.get)
569. highest\_pvalue = high\_pvalues\_[highest\_pvalue\_col]
570. **X\_train = X\_train.drop([highest\_pvalue\_col], axis=1)**
571. X\_test = X\_test.drop([highest\_pvalue\_col], axis=1)
572. print(f'Dropped "{highest\_pvalue\_col}" **with** p-value {highest\_pvalue}', end='\n\n')
573. dropped\_cols\_pvalue[highest\_pvalue\_col] = highest\_pvalue
575. **print(f'Dropped all variables with p-value greater than 0.05 from the train/test sets.')**
577. print(f'List of dropped variables **and** their respective p-value at time of removal:\n{dropped\_cols\_pvalue}')

580. **print(sm\_api.Logit(y\_train, X\_train).fit().summary())**
582. print(f'Variables **with** p-value above 0.05 threshold: {high\_pvalues()}')
584. display\_model\_performance()
586. display\_model\_performance\_mean()
588. Since the models seem to perform better with the high p-value features, the dataframes will be restored.
590. **X\_train = X\_train\_bkp**
591. X\_test = X\_test\_bkp
593. ### Data Balancing with SMOTE Over-Sampling
594. The Synthetic Minority Oversampling Technique algorithm will be used to balance the dataset by generating new instances (synthetic samples) from existing minority cases (no-subscription), by combining the features of neighbours to generate new examples. The number or majority cases will not be changed.
596. Since the synthetic examples created via oversampling techniques are not real observations, and thus not suitable for testing purposes, only the training dataset will be used for oversampling, so there is no information leakage from the test set into model training, and no synthetic observations will be created from test set information.
598. For cross-validation, proper SMOTE over-sampling within each fold will be performed with the use of pipelines in order to avoid an inaccurate cross-validation score.
600. **# Get model performance metrics with unbalanced data**
601. display\_model\_performance()
603. pipeline\_logit = imbpipeline(steps = [['smote', SMOTE(sampling\_strategy='minority', random\_state=11)],
604. ['classifier', LogisticRegression(random\_state=11, max\_iter=10000)]
605. **])**
607. # Attempt to speed up RandomForestClassifier with parallelization
608. # Max used CPU cores set to 6 (https://stackoverflow.com/a/50996992)
609. # Note: If this is causing problems on your machine please set n\_jobs below to None
610. **cpu\_cores\_to\_use = min(6, psutil.cpu\_count(logical=False))**
612. pipeline\_rforest = imbpipeline(steps = [['smote', SMOTE(sampling\_strategy='minority', random\_state=11)],
613. ['classifier', RandomForestClassifier(random\_state=11, n\_jobs=cpu\_cores\_to\_use)]
614. ])
616. # make custom scorer to print the classification report for each fold
617. def classification\_report\_and\_auroc(estimator):
618. """ Display classification reports and AUROC scores via custom scorer."""
619. def \_scorer(y\_true, y\_pred):
620. **print(metrics.classification\_report(y\_true, y\_pred))**
621. return metrics.roc\_auc\_score(y\_true, y\_pred)
623. auroc\_score = cross\_val\_score(estimator=estimator, X=X\_train, y=y\_train.values.ravel(), cv=stratified\_kfold,
624. scoring=metrics.make\_scorer(\_scorer))
625. **return f'Mean AUROC: {numpy.mean(auroc\_score)}'**
627. # display the mean performance of the models with upsampled train data
628. def display\_model\_os\_performance(estimator):
629. print(estimator[1].\_\_class\_\_.\_\_name\_\_)
630. **print(classification\_report\_and\_auroc(estimator))**
631. for metric in ['accuracy', 'precision\_macro', 'recall\_macro', 'f1\_macro']:
632. score = cross\_val\_score(estimator=estimator, X=X\_train, y=y\_train.values.ravel(), cv=stratified\_kfold, scoring=metric)
633. print(f'Mean {metric}: {numpy.mean(score)}')
634. print('\n')
636. display\_model\_os\_performance(pipeline\_logit)
638. display\_model\_os\_performance(pipeline\_rforest)
640. **# IMPORTANT: In case of threading errors, this is likely due to a known issue of the loky library:**
641. # https://github.com/pycaret/pycaret/issues/38#issuecomment-676963390
642. # https://github.com/scikit-learn/scikit-learn/issues/13354
643. # pipeline\_rforest.set\_params(classifier\_\_n\_jobs=None)
644. # To fix (if the above exception handling didn't work), remove n\_jobs **or** set it to None **in** the pipeline\_rforest RandomForestClassifier.
646. *# Logistic Regression Model Implementation*
647. Logistic regression **is** a supervised learning classification algorithm that uses a logistic sigmoid function **in** order to predict the probability of a target variable that **is** binary **in** nature (e.g., subscribed/not-subscribed).
649. *## Hyperparameter Optimization (Model Tuning)*
651. *# While Logistic Regression does not have any hyperparameters whose tuning is of critical importance,*
652. *# differences in performance can be seen in some cases.*
654. *# Group parameters separately to avoid fit errors and silence warnings due to incompatible parameters.*
655. **p\_max\_iter = [100] + numpy.arange(1000, 10000, 1000).tolist()**
656. p\_C = [100, 10, 1.0, 0.1, 0.01]
657. parameters\_logit = [
658. {
659. *# Maximum number of iterations taken for the solvers to converge.*
660. **'classifier\_\_max\_iter' : p\_max\_iter,**
661. *# Algorithm to use in the optimization problem.*
662. 'classifier\_\_solver' : ['liblinear', 'saga'],
663. *# Regularization methods to be used*
664. 'classifier\_\_penalty' : ['l1', 'l2'],
665. ***# Inverse of regularization strength - smaller values specify stronger regularization.***
666. 'classifier\_\_C' : p\_C
667. },
668. {
669. 'classifier\_\_max\_iter' : p\_max\_iter,
670. **'classifier\_\_solver' : ['lbfgs', 'liblinear', 'saga'],**
671. 'classifier\_\_penalty' : ['l2'],
672. 'classifier\_\_C' : p\_C
673. },
674. {
675. **'classifier\_\_max\_iter' : p\_max\_iter,**
676. 'classifier\_\_solver' : ['sag', 'newton-cg'],
677. 'classifier\_\_penalty' : ['l2'],
678. 'classifier\_\_C' : p\_C
679. },
680. **{**
681. 'classifier\_\_max\_iter' : p\_max\_iter,
682. 'classifier\_\_solver' : ['saga'],
683. 'classifier\_\_penalty' : ['elasticnet'],
684. 'classifier\_\_l1\_ratio' : numpy.arange(0.0, 1.0, 0.1).tolist(),
685. **'classifier\_\_C' : p\_C**
686. }
687. ]
689. *### Accuracy*
691. *# Search for optimal combination of hyperparameters*
692. *# Scoring - https://scikit-learn.org/stable/modules/model\_evaluation.html#scoring-parameter*
693. *# cv - cross-validation splitting strategy: number of fols in (Stratified)KFold*
694. iter\_num = 10
695. **classifier\_logit = RandomizedSearchCV(estimator=pipeline\_logit, param\_distributions=parameters\_logit,**
696. scoring='accuracy', cv = stratified\_kfold, n\_iter=iter\_num)
698. *# For GridSearch, though it takes a considerable amount of time:*
699. *# from sklearn.model\_selection import GridSearchCV*
700. ***# classifier\_logit = GridSearchCV(estimator=pipeline\_logit, param\_distributions=parameters\_logit, scoring='accuracy', cv = stratified\_kfold, n\_iter=iter\_num)***
702. *# Train model*
703. model\_logit = classifier\_logit.fit(X\_train, y\_train.values.ravel())
705. **print(f'Logistic Regression optimal hyperparameters for accuracy: {model\_logit.best\_params\_}')**
707. *### Recall*
709. classifier\_logit\_recall = RandomizedSearchCV(estimator=pipeline\_logit, param\_distributions=parameters\_logit,
710. **scoring='recall', cv = stratified\_kfold, n\_iter=iter\_num)**
711. model\_logit\_recall = classifier\_logit\_recall.fit(X\_train, y\_train.values.ravel())
713. **print**(f'Logistic Regression optimal hyperparameters for recall: {model\_logit\_recall.best\_params\_}')
715. ***## Evaluation***

718. *# Accuracy*
719. prediction = model\_logit.predict(X\_test)
720. **prediction\_train = model\_logit.predict(X\_train)**
722. *# Recall*
723. prediction\_recall = model\_logit\_recall.predict(X\_test)
724. prediction\_recall\_train = model\_logit\_recall.predict(X\_train)
726. **print**(metrics.classification\_report(y\_test, prediction))
727. **print**("Logistic Regression Accuracy (Test): ", metrics.accuracy\_score(y\_test, prediction))
728. **print**("Logistic Regression Accuracy (Train): ", metrics.accuracy\_score(y\_train, prediction\_train))
730. **print("Logistic Regression Recall (Test): ", metrics.recall\_score(y\_test, prediction\_recall))**
731. **print**("Logistic Regression Recall (Train): ", metrics.recall\_score(y\_train, prediction\_recall\_train))
733. **print**(metrics.confusion\_matrix(y\_test, prediction))
734. metrics.ConfusionMatrixDisplay.from\_predictions(y\_test, prediction)
735. **pyplot.show()**
737. *# Random Forest Model Implementation*
739. The Random Forest Classifier **is** a decision tree-based classification algorithm that uses randomness **in** generating individual tree to promote uncorrelated forests, which then uses the forest’s predictive powers to make accurate decisions. (Niklas, 2021)
741. *## Hyperparameter Optimization (Model Tuning)*
743. parameters\_rforest = [
744. {
745. ***# The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure***
746. *# or until all leaves contain less than min\_samples\_split samples.*
747. *# Note: Max depth may cause overfitting (higher value) or underfitting (lower value).*
748. 'classifier\_\_max\_depth': numpy.arange(10, 100, 10).tolist() + [None],
750. ***# The minimum number of samples required to split an internal node.***
751. 'classifier\_\_min\_samples\_split': [2],
753. *# The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered*
754. *# if it leaves at least min\_samples\_leaf training samples in each of the left and right branches.*
755. ***# This may have the effect of smoothing the model, especially in regression.***
756. 'classifier\_\_min\_samples\_leaf': [1, 2],
758. *# The number of trees in the forest.*
759. 'classifier\_\_n\_estimators': [100] + numpy.arange(200, 2000, 200).tolist(),
761. *# The number of features to consider when looking for the best split*
762. *# auto: max\_features=sqrt(n\_features)*
763. 'classifier\_\_max\_features': ['auto', 'log2'],
765. ***# Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.***
766. 'classifier\_\_bootstrap': [True, False],
768. *# The function to measure the quality of a split.*
769. *# Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.*
770. **'classifier\_\_criterion': ['gini', 'entropy']**
771. }
772. ]
774. Note: Depending on the number of trees (n\_estimators), hyperparameter model tuning can take a long time due to the model's slow predictions times.
776. ### Accuracy
778. classifier\_rforest = RandomizedSearchCV(estimator=pipeline\_rforest, param\_distributions=parameters\_rforest,
779. scoring='accuracy', cv = stratified\_kfold, n\_iter=iter\_num)
780. **model\_rforest = classifier\_rforest.fit(X\_train, y\_train.values.ravel())**
782. print(f'Random Forest optimal hyperparameters **for** accuracy: {model\_rforest.best\_params\_}')
784. ### Recall
786. classifier\_rforest\_recall = RandomizedSearchCV(estimator=pipeline\_rforest, param\_distributions=parameters\_rforest,
787. scoring='recall', cv = stratified\_kfold, n\_iter=iter\_num)
788. model\_rforest\_recall = classifier\_rforest\_recall.fit(X\_train, y\_train.values.ravel())
790. **print(f'Random Forest optimal hyperparameters for recall: {model\_rforest\_recall.best\_params\_}')**
792. ## Evaluation
794. # Accuracy
795. **prediction = model\_rforest.predict(X\_test)**
796. prediction\_train = model\_rforest.predict(X\_train)
798. # Recall
799. prediction\_recall = model\_rforest\_recall.predict(X\_test)
800. **prediction\_recall\_train = model\_rforest\_recall.predict(X\_train)**
802. print(metrics.classification\_report(y\_test, prediction))
803. print("Random Forest Classifier Accuracy (Test): ", metrics.accuracy\_score(y\_test, prediction))
804. print("Random Forest Classifier Accuracy (Train): ", metrics.accuracy\_score(y\_train, prediction\_train))
806. print("Random Forest Classifier Recall (Test): ", metrics.recall\_score(y\_test, prediction\_recall))
807. print("Random Forest Classifier Recall (Train): ", metrics.recall\_score(y\_train, prediction\_recall\_train))
809. print(metrics.confusion\_matrix(y\_test, prediction))
810. **metrics.ConfusionMatrixDisplay.from\_predictions(y\_test, prediction)**
811. pyplot.show()
813. # Models Evaluation
815. **## Cross-Validation with Stratified K-Fold**
816. Cross-validation will be used once more to estimate the model stability and performance of both the Logistic Regression and Random Forest.
818. def get\_model\_name(model):
819. # get the imblearn pipeline used as estimator for RandomizedSearchCV
820. **pipeline\_ = model.best\_estimator\_**
821. # get the classifier from the pipeline
822. classifier\_ = pipeline\_.get\_params()['classifier']
823. # return the classifier's **class** name
824. **return** classifier\_.\_\_class\_\_.\_\_name\_\_
826. stratified\_kfold = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=11)
828. *# custom scorer to save each fold's classification report to calculate the mean for all of them*
829. **def** get\_model\_mean\_report(model):
830. **def \_clf\_report\_scorer(y\_true, y\_pred):**
831. true\_c.extend(y\_true)
832. pred\_c.extend(y\_pred)
833. **return** metrics.accuracy\_score(y\_true, y\_pred)
835. **print(get\_model\_name(model))**
836. true\_c, pred\_c = [], []
837. cross\_val\_score(estimator=model, X=X\_train, y=y\_train.values.ravel(), cv=stratified\_kfold,
838. scoring=metrics.make\_scorer(\_clf\_report\_scorer))
839. **print**(f'**\n**{get\_model\_name(model)} cross-validation classification report with mean values: ')
840. **print(metrics.classification\_report(true\_c, pred\_c))**
842. get\_model\_mean\_report(model\_logit)
844. get\_model\_mean\_report(model\_rforest)
846. *## ROC Curve*
847. The ROC (receiver operating characteristic) curve plot illustrates the diagnostic ability of a binary classifier system **as** its discrimination threshold **is** varied ("Receiver operating characteristic", 2021, para. 1), **and** will be used to compare the two models by evaluating their performance at all classification thresholds.
849. roc\_table = pandas.DataFrame()
851. *# get ROC-related metrics, including brier score, for both models*
852. **for** model **in** [model\_logit, model\_rforest]:
853. prediction\_probability\_estimates = model.predict\_proba(X\_test)
854. auroc\_score = metrics.roc\_auc\_score(y\_test, prediction\_probability\_estimates[:, 1])
855. **false\_positive\_rate, true\_positive\_rate, thresholds = metrics.roc\_curve(y\_test, prediction\_probability\_estimates[:, 1])**
856. roc\_data = {
857. 'model': get\_model\_name(model),
858. 'false\_positive\_rate': false\_positive\_rate,
859. 'true\_positive\_rate': true\_positive\_rate,
860. **'auroc\_score': auroc\_score,**
861. 'pr\_auc\_score': metrics.average\_precision\_score(y\_test.values.ravel(), prediction\_probability\_estimates[:, 1]),
862. 'brier\_score': metrics.brier\_score\_loss(y\_test.values.ravel(), prediction\_probability\_estimates[:, 1])
863. }
864. roc\_table = roc\_table.append(roc\_data, ignore\_index=True)
866. roc\_table.set\_index('model', inplace=True)
868. roc\_table
870. **pyplot.figure(figsize=(10, 8))**
872. *# plot the ROC curve and display the scores in the label next to the model name*
873. **for** model **in** roc\_table.index:
874. pyplot.plot(roc\_table.loc[model]['false\_positive\_rate'],
875. **roc\_table.loc[model]['true\_positive\_rate'],**
876. label="{}, AUC={:.3f}, PR-AUC={:.3f}, BSL={:.3f}".format(
877. model, roc\_table.loc[model]['auroc\_score'], roc\_table.loc[model]['pr\_auc\_score'], roc\_table.loc[model]['brier\_score']))

880. ***# Adjust size and legend position so it's not covering anything and the scores are visible***
881. pyplot.title('ROC Curve', fontweight='bold', fontsize=16)
882. pyplot.legend(loc='lower right', prop={'size':14})
883. pyplot.xlabel("False Positive Rate", fontsize=16)
884. pyplot.ylabel("True Positive Rate", fontsize=16)
886. pyplot.plot([0,1], [0,1], color='tan', linestyle='--')
887. pyplot.xticks(numpy.arange(0.0, 1.1, step=0.1))
888. pyplot.yticks(numpy.arange(0.0, 1.1, step=0.1))
890. **pyplot.show()**
892. *# External Resources Used*
894. \* http://archive.ics.uci.edu/ml/datasets/Bank+Marketing
895. **\* https://towardsdatascience.com/how-to-build-a-machine-learning-model-439ab8fb3fb1**
896. \* https://towardsdatascience.com/how-to-build-your-first-machine-learning-model-in-python-e70fd1907cdd
897. \* https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28
898. \* https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
899. \* https://builtin.com/data-science/random-forest-algorithm
900. **\* https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148**
901. \* https://towardsdatascience.com/3-easy-ways-to-crosstab-in-pandas-4123383bfbf2
902. \* https://towardsdatascience.com/how-to-effortlessly-handle-class-imbalance-with-python-and-smote-9b715ca8e5a7
903. \* https://machinelearningmastery.com/what-is-imbalanced-classification/
904. \* https://machinelearningmastery.com/how-to-score-probability-predictions-in-python/
905. **\* https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28**
906. \* https://en.wikipedia.org/wiki/Accuracy\_paradox
907. \* https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d
908. \* https://www.analyticsvidhya.com/blog/2020/10/how-to-choose-evaluation-metrics-for-classification-model/
909. \* https://arxiv.org/pdf/1106.1813.pdf (SMOTE)
910. **\* https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/smote**
911. \* https://www.simplypsychology.org/p-value.html
912. \* https://statisticsbyjim.com/
913. \* https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/
914. \* https://www.imranabdullah.com/2019-06-01/Drawing-multiple-ROC-Curves-in-a-single-plot
915. **\* https://stackoverflow.com/questions/50993867/increasing-n-jobs-has-no-effect-on-gridsearchcv**