Predictive Analytics for a Bank Marketing Campaign

Project Report

Robert Maheux

April 22 2019

Contents

[1. Introduction 3](#_Toc6409505)

[2. Dataset 4](#_Toc6409506)

[2.1 Outliers 5](#_Toc6409507)

[2.2 Imputation of Values 6](#_Toc6409508)

[2.3 Converting Categorical Values to Numerical Values 6](#_Toc6409509)

[2.4 Standardizing the Data 6](#_Toc6409510)

[2.5 Resolving the Imbalanced Distribution of the Outcome Variable 6](#_Toc6409511)

[2.6 Feature Reduction 7](#_Toc6409512)

[3. Building Predictive Models 9](#_Toc6409513)

[3.1 Model Descriptions 9](#_Toc6409514)

[3.2 Metrics Used to Evaluate the Models 10](#_Toc6409515)

[3.2.1 Accuracy Score 10](#_Toc6409516)

[3.2.2 Precision Score 10](#_Toc6409517)

[3.2.3 Recall Score 10](#_Toc6409518)

[3.2.4 F1 Score 10](#_Toc6409519)

[3.2.5 Receiver Operating Characteristics Curve and Area Under the Curve 10](#_Toc6409520)

[3.3 Comparing the Performance of the Models 12](#_Toc6409521)

[4. Model Strengths and Limitations 13](#_Toc6409522)

[5. Conclusion 14](#_Toc6409523)

[Appendix A – Model Results Obtained with Various Training/Testing Sets 15](#_Toc6409524)

[Appendix B – Listing of Python Commands 18](#_Toc6409525)

# 1. Introduction

Banks use telemarketing campaigns to increase the number of subscriptions to longer term investments such as term deposit bank accounts. This strategy works but it’s very time consuming and almost 90% of the phone calls don’t yield results. The bank could save time and money if it was able to focus its marketing efforts on the clients that are most likely to open a term deposit bank account.

This problem was solved by building a machine learning model to predict the outcome of the marketing campaign for each client. This will allow the marketing team to strategically target the clients that will react positively to the campaign. The predictive model will be a valuable tool that will use data that is already at the bank’s disposal.

Four types of machine learning models were built using algorithms that are best suited to classification problems with a binomial output variable (“yes” or no”). The model that proves to be the most accurate will be selected for deployment.

# 2. Dataset

The dataset that was used was obtained from the web site for the UC Irvine Machine Learning Repository**:** <http://archive.ics.uci.edu/ml/datasets/Bank+Marketing>.

The observations in the dataset are related to previous direct marketing campaigns based on phone calls to offer a term deposit bank account. Each observation contains features related to a call made between May 2008 and November 2010. The dataset contains 41,188 observations with 21 features (including the output variable that indicates the client’s response).

The dataset was enriched by adding five social and economic features that were published by the Banco de Portugal and are publicly available at: <https://www.bportugal.pt/estatisticasweb>.

It contains the following features:

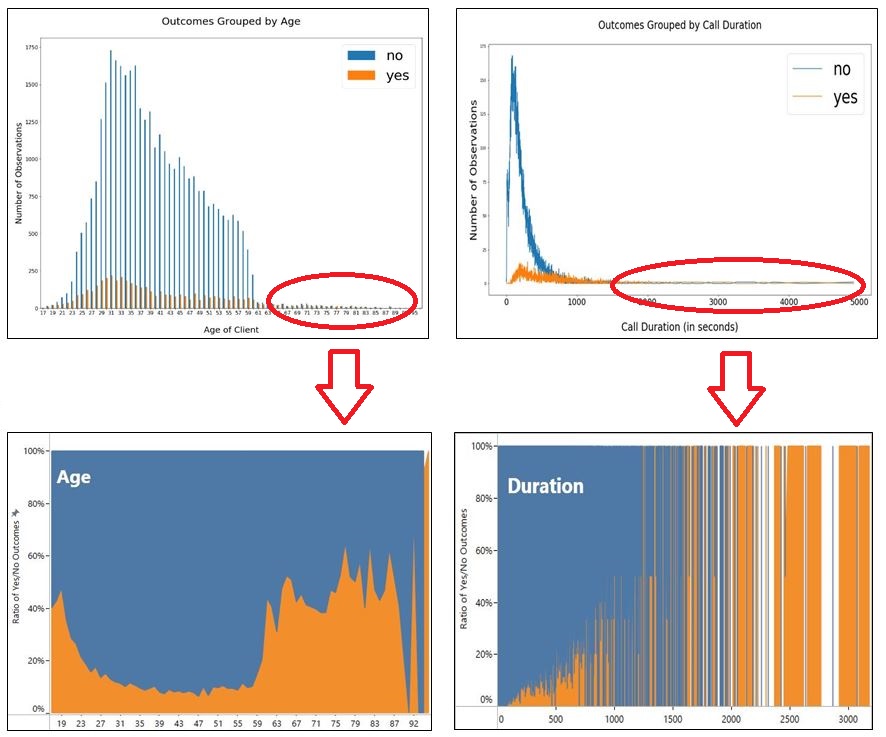
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Feature** | **Description** | **Type** | **Values** |
| 1 | age | Age of client | Numeric | 17 to 98 |
| 2 | job | Type of job | Categorical | admin, blue-collar, technician… |
| 3 | marital | Marital status | Categorical | divorced, married, single … |
| 4 | education | Education level | Categorical | high.school, university.degree… |
| 5 | default | Does the client have credit in default? | Categorical | yes, no, unknown |
| 6 | housing | Does the client have a housing loan | Categorical | yes, no, unknown |
| 7 | loan | Does the client have a personal loan? | Categorical | yes, no, unknown |
| 8 | contact | Contact communication type | Categorical | cellular, telephone |
| 9 | month | Last contact month of the year | Categorical | jan, feb, … dec |
| 10 | day\_of\_week | Last contact day of the week | Categorical | mon, tue, … fri |
| 11 | duration | Last contact duration, measured in seconds | Numeric | 0 to 4918 |
| 12 | campaign | Number of contacts during this campaign for this client | Numeric | 1 to 56 |
| 13 | pdays | Number of days since client was last contacted for a previous campaign | Numeric | 0 to 27, 999 |
| 14 | previous | Number of contacts before this campaign for this client | Numeric | 0 to 7 |
| 15 | poutcome | Outcome of the previous marketing campaign | Categorical | failure, nonexistent, success |
| 16 | emp.var.rate | Employment variation rate - quarterly indicator | Numeric | -3.4 to 1.4 |
| 17 | cons.price.idx | Consumer price index - monthly indicator | Numeric | 92.201 to 94.767 |
| 18 | cons.conf.idx | Consumer confidence index - monthly indicator | Numeric | -50.8 to -26.9 |
| 19 | euribor3m | Euribor 3 month rate - daily indicator | Numeric | 0.634 to 5.045 |
| 20 | nr.employed | Number of employees - quarterly indicator | Numeric | 4963.6 to 5228.1 |
| 21 | outcome | Does client want a term deposit account? | Categorical | yes, no |

The data was very clean – there were no missing or invalid values.

## 2.1 Outliers

Several features contained outliers but they retained. It was considered to be beneficial to retain them since they are legitimate values and contain valuable information.

For instance, the following bar charts display the distribution of the outcomes grouped by *Age* and *Call Duration*. The two distributions are skewed to the right with outliers in the upper regions. However, when charts are generated to illustrate the *proportion* of “yes/no” outcomes, it becomes obvious that the outliers contain a large proportion of “yes” outcomes.



Since the “yes” outcomes represented a mere 11% of the complete dataset, it was important to keep as many of them as possible. Therefore, the outliers were retained.

## 2.2 Imputation of Values

The dataset did not contain any missing values. However, several values had been entered with the indicator ‘*unknown’* or ‘*999’*. The reason for the values being unspecified was unknown so imputing them might have damaged the validity of the data.

There is a famous anecdote concerning a churn project for a large retailer, based on loyalty card data. The best predictor of churn turned out to be whether or not the customer had filled in his/her email address when registering for the card. Sometimes, a lack of information is valuable information itself.

The bank marketing values that had been marked as unknown were not imputed. The fact that they were unspecified was considered to be just another value for the categorical variable.

## 2.3 Converting Categorical Values to Numerical Values

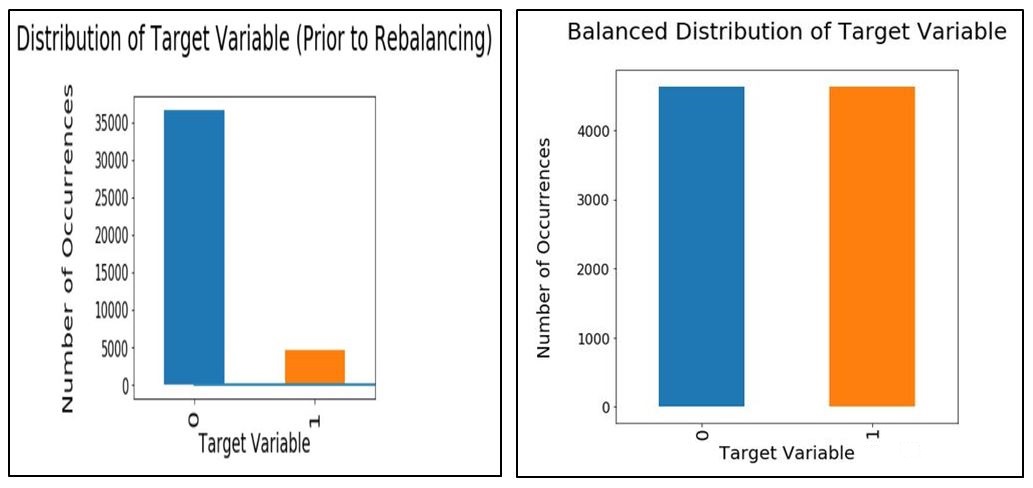
The categorical values needed to be converted into numerical values so that they could be used by machine learning algorithms. The *one-hot* encoding method was used to separate the values into individual columns and convert them into 1 or 0.

## 2.4 Standardizing the Data

The features in the dataset had differing scales. To compensate for this, the data was standardized so that each feature has its’ mean = 0 and its’ standard deviation = 1.

## 2.5 Resolving the Imbalanced Distribution of the Outcome Variable

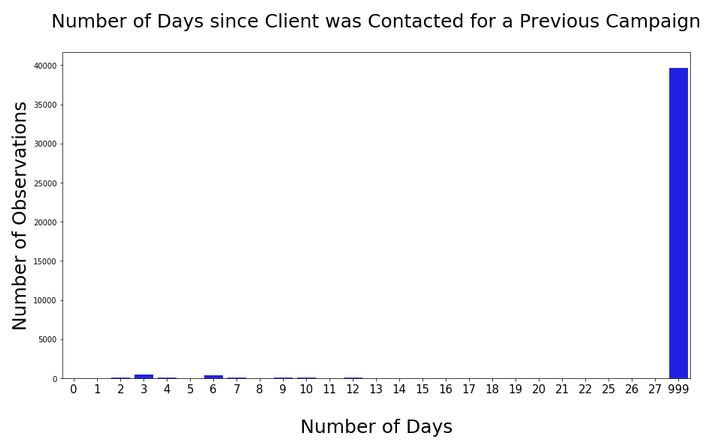
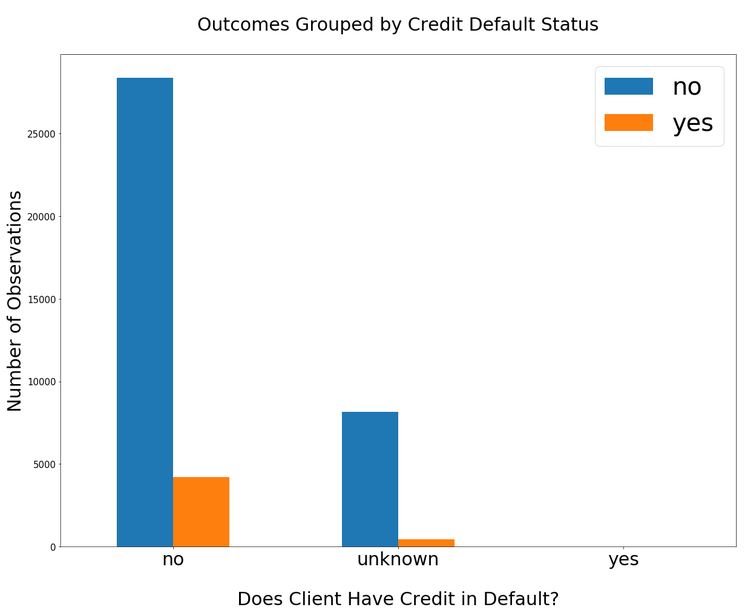
The values for the output variable were very imbalanced, as shown in the first chart below. Only 11% of the observations had an outcome of “yes”. This is a problem because many classification algorithms struggle with imbalanced data. It was resolved by using a technique called *under-sampling* which involves the random removal of “no” records until the number of records for both classes is the same. The second chart shows the resulting dataset containing 9280 observations with an equal number of “yes” and “no” outcomes.



## 2.6 Feature Reduction

When the features were converted to categorical variables, the number of features expanded from 20 to 50. Machine learning algorithms function better with a smaller number of features, so some analysis was performed to identify and remove the least important ones in terms of predicting the outcomes.

During the data exploration phase, the *pdays* and *default* features were judged to have minimal or no impact on the outcome variable. The following charts illustrate that when the outcomes are grouped by the *pdays* or *default* values, there are almost no observations with “yes” outcomes.

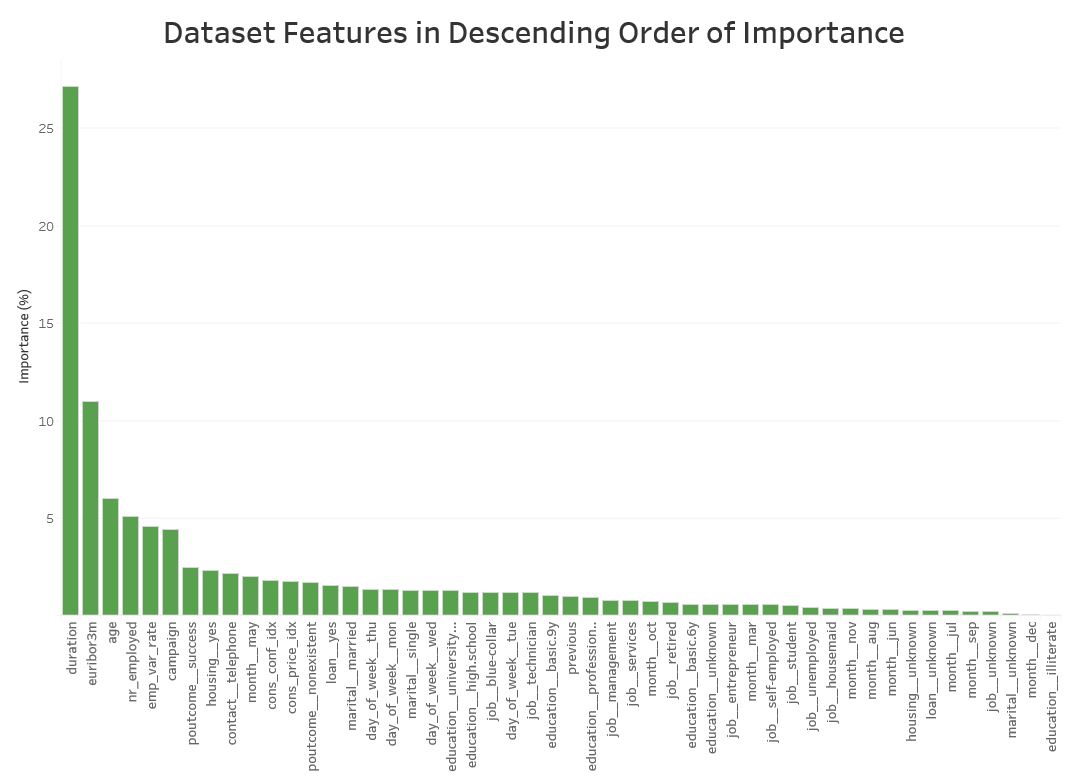
The two features were removed from the dataset prior to the data preparation phase.

In order to further reduce the number of features, the *Primary Component Analysis* (PCA) method was applied to the dataset. The number of features to retain is subjective. In order to preserve as much information as possible, it was decided to retain the features that represent 95% of the total importance score.

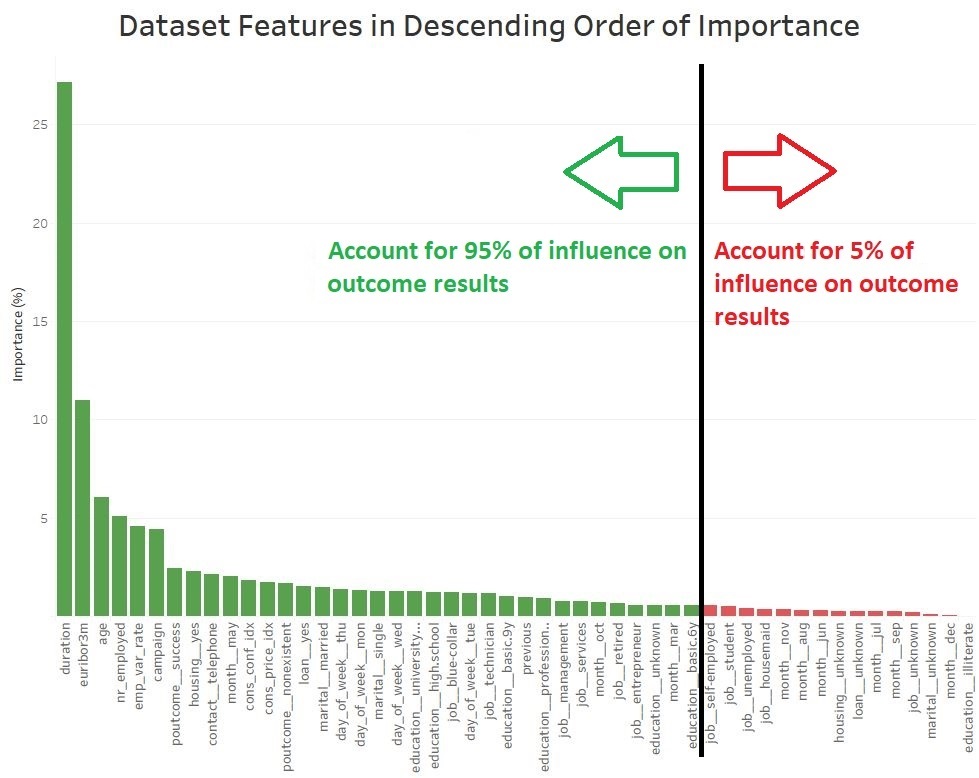
The PCA technique calculated that it would take 37 generic components to explain at least 95% of the variance. This didn’t represent a substantial reduction of features so the benefit of using PCA on this dataset would have been marginal at best. More importantly, the use of the PCA technique would have caused the individual features to be replaced with generic components and this would have eliminated the ability to interpret how the individual features contributed to the outcomes. It was determined that the PCA technique offered little benefit for the bank marketing dataset.

An alternative technique using an *Extra Trees Classifier* decision tree was attempted. This technique could be used to compute the importance of each feature without replacing the individual features with generic components. It allowed the least important features to be discarded while retaining the ability to interpret how the remaining features could impact the outcomes.

The following bar chart displays the importance levels for each feature, as calculated by the Extra Trees Classifier. The most important feature received a score of 27.1% while the least important feature received an almost irrelevant score of 0.02%. The chart illustrates that the first six features are much more important than the others, but a lot of valuable information would be lost if the remaining 44 features were discarded.



The Extra Trees Classifier calculated that 35 features would need to be retained in order to preserve 95% of the total importance score. By discarding the 15 least important features, the dataset was simplified without losing valuable feature-related information required to interpret the results.



# 3. Building Predictive Models

## 3.1 Model Descriptions

The Python programming language was used to build four types of machine learning models using algorithms best suited to classification problems with a binomial outcome (“yes” or no”).

The mathematical workings behind each model are beyond the scope of this document but here’s a rough idea of how each algorithm works:

Logistic Regression

This algorithm accepts the features of an observation and predicts the probability that the output belongs to a certain class. It plots the training observations using a *sigmoid* which is a mathematical function with a characteristic "S"-shaped curve. This allows it to determine a boundary between two different classes of outcomes. A new observation is plotted and its outcome is predicted depending on which side of the boundary it was mapped.

Random Decision Tree

A decision tree breaks down the training dataset into smaller and smaller subsets and builds a set of *if-then-else* decision rules in the form of a tree structure. The final result is a tree with several decision nodes leading down to leaf nodes. Leaf nodes represent a classification or outcome.

Decision trees are prone to overfitting but this can be corrected by using a collection of multiple decision trees called a *Random Decision Forest*. This classification method operates by constructing a multitude of decision trees with different predictions and combining the results of those individual trees to give the final outcome.

K-Nearest Neighbours

This algorithm compares a new observation with all the observations in the training set and finds the K observations that most closely resemble it. It then takes the majority of the outcomes of these “nearest neighbours” to be the predicted outcome for the new observation. This model will be built using K = 25 nearest neighbours.

Support Vector Classification

This algorithm creates a representation of the training observations as points in space. They are mapped and are divided into two categories by a clear gap that is as wide as possible. New observations are then mapped into that same space and predicted to belong to one of the categories based on which side of the gap they fall.

## 3.2 Metrics Used to Evaluate the Models

Each model was evaluated using the same set of 6 metrics suitable for classification algorithms.

### 3.2.1 Accuracy Score

The Accuracy score is the simplest and most commonly used performance metric. It is simply the ratio of correct predictions divided by the total number of predictions.

### 3.2.2 Precision Score

The Precision **score** tells us the proportion of clients that were correctly predicted to open a term account compared to *the total number that were predicted to do so*.

A low Precision score means that the marketing team would waste a lot of time contacting clients that aren’t likely to open a bank account. Given the bank’s business objective, the Precision score is an important measure to use.

### 3.2.3 Recall Score

The Recall **score** tells us the proportion of clients that were correctly predicted to open a term account compared to *the total number that would actually do so*.

A low Recall score means that the marketing team wouldn’t be informed about several clients that are likely to open a bank account. The model would erroneously direct the marketing team away from new business opportunities. Given the bank’s business objective, the Recall score is an important measure to use.

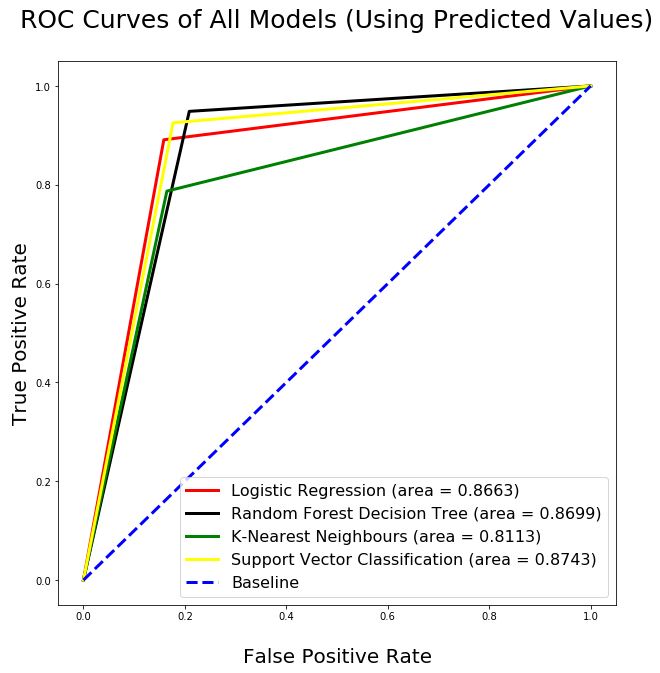
### 3.2.4 F1 Score

The F1 Score is the harmonic average of the precision and recall scores. Given the bank’s business objective, the Precision and Recall scores are equally important so the F1 score will provide a weighted harmonic mean of both values.

### 3.2.5 Receiver Operating Characteristics Curve and Area Under the Curve

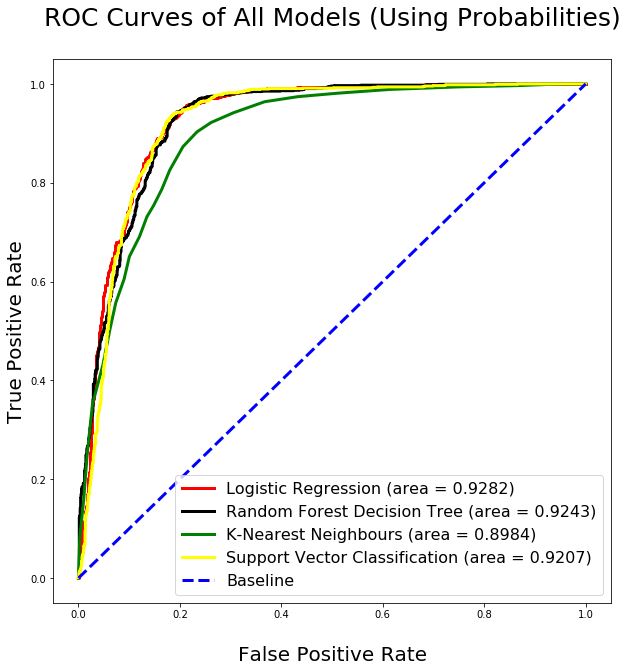
TheReceiver Operating Characteristics(ROC) curve provides a visual comparison of classification models.

The curves are plotted with the true positive rates on the Y axis and the false positive rates on the X axis. The top left corner of the plot is the *ideal* point - a false positive rate of zero, and a true positive rate of one. This means that a larger area under the curve (AUC) is usually better. The AUC summarizes the curve information in one number and can be used as a measure of the accuracy of the model.



The number of points on the graph reflects the number of unique values in the input. The plotted lines have a triangular shape because the input vectors have only 2 unique values for the predictions. (“yes” or “no”).

When performing classifications, it is sometimes desirable to predict not only the outcome values but also the associated probabilities of predicted values (for each observation). These probabilities provide confidence on the predictions and can be plotted in the same way, as shown below. The area under the ROC curve that was created using the probabilities is a valuable metric to evaluate the performance of a model.



## 3.3 Comparing the Performance of the Models

It’s difficult to compare models when several metrics are used because different metrics point to different models as being the “best one”. The following table is a compilation of the various scores obtained by each model. Identifying the one with the highest number of “best” and “second-best” scores can provide an overall ranking of each model. The highest scores for each column are shaded in green. The second highest scores are shaded in yellow.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 86.64 | 85.01 | 89.10 | 87.01 | .8663 | .9282 |
| **Random Decision Forest** | 87.03 | 82.10 | 94.85 | 88.01 | .8699 | .9243 |
| **K-Nearest Neighbours** | 81.12 | 82.84 | 78.71 | 80.72 | .8113 | .8984 |
| **Support Vector** | 87.46 | 84.09 | 92.53 | 88.11 | .8743 | .9207 |

The overall scores in the table indicate that the Support Vector model is a strong performer, closely followed by the Random Forest model. The worst performer is the model using the K-Nearest Neighbours algorithm.

When the main dataset is split into training and testing datasets, the observations are divided randomly. By performing a new random split and re-executing the models on the new training and testing sets, different results are obtained. **Appendix A** lists the results for the same models built on 10 different versions of the training and testing sets. The Support Vector model was the best one in 70% of the cases while the Random Forest model was the best one in the remaining 30%. The K-Nearest Neighbours model was consistently the worst performer.

An alternative comparison method would be to focus on the metrics that are the most relevant for the problem being addressed. For the Bank Marketing problem, it’s extremely important for the model to have high Precision and Recall scores. A low Precision score would result in the marketing team wasting time contacting clients that are *false positives*. A low Recall score would mean that the marketing team would not be directed towards many clients that are *true positives*.

Since the F1 score is a harmonic average of the Precision and Recall scores, we can focus on that single metric to compare the performance of the models. The F1 score indicates that the Support Vector and Random Forest models are almost equal, with a slight edge in favour of the Support Vector model. Again, the K-Nearest Neighbours model was consistently the worst performer.

There are some techniques that can be used to see if one of the models can be improved so that it consistently outperforms the others, such as:

* Tuning the model parameters (amount of tree pruning, number of nearest neighbours used…)
* Removing some of the independent variables being used
* Using boosting algorithms

# 4. Model Strengths and Limitations

The F1 score for the Support Vector model is 88.11%. (The average F1 score of all the variations of training/testing sets is 88.56%). This is an impressive score. It’s a very good score without being *too good*, which would have indicated overfitting.

The predictive model will have some limitations. It can only be used for telemarketing campaigns related to term deposit bank accounts. In addition, it won’t determine the causes for a client’s willingness to open a new account. The factors that influence the client’s decisions will need to be investigated in order to improve the rate of subscriptions.

Many factors could cause the clients’ behaviour to change over time. The performance of the model will need to be reassessed on a regular basis.

After every marketing campaign, new data will become available. This data must be added to the original dataset to maintain the relevancy of the model. The dataset will need to be cleaned and prepared using the same steps outlined in this project. The model will need to be trained and tested using the latest version of the dataset.

# 5. Conclusion

This report outlined the steps to build, test and compare four different types of classification models. The Bank Marketing dataset was used to make predictions.

The Support Vector and Random Forest models performed almost equally, with a slight edge in favour of the Support Vector model.

In addition, this analysis has provided some valuable insights concerning features that impact the outcome of the marketing calls:

* The calls with the highest duration yield the highest ratio of “yes” outcomes. The reason for this will need to be investigated. It’s possible that some marketing resources are more adept at keeping clients on the call and this gives them more time to convince them to open a term bank account. However, it’s also possible that interested clients asked a lot of questions and chose to open an account immediately which extended the call duration. It will be beneficial to interview the marketing team for their insights concerning the call durations.
* The clients that are younger than 25 or older than 60 have the highest ratio of “yes” outcomes. This corresponds to the ratios for the job types: The clients that are students or retired have the highest ratio of “yes” outcomes.
* The best time to run this type of campaign is when the rates for the Euribor3M, NR Employed and Employee Variation are low.
* Clients who were contacted for previous marketing campaigns are most likely to respond positively to the new campaign. The ratio of “yes” outcomes increases with the number of previous contacts.
* Clients who responded positively to a previous campaign are much more likely to respond positively to new campaigns than not.
* The ratio of “yes” outcomes is much higher during the months of March, September, October and December. The ratio is the lowest during the months of May, June, July, August and November.
* The clients who answered while using a cell phone are more than twice as likely to respond positively. This may simply be related to the fact that younger people and students have a higher ratio of “yes” outcomes. This demographic group is more likely to use a cell phone rather than a land line as their main contact phone.

The new predictive model as well as the insights on the data features will allow the marketing team to predict which clients will most likely respond positively to the campaign. This will allow the bank to increase the efficiency of the marketing resources and thereby reduce costs.

# Appendix A – Model Results Obtained with Various Training/Testing Sets

When the main dataset is split into training and testing datasets, the observations are divided randomly. By performing a new random split and re-executing the models on the new training and testing sets, different results are obtained.

This appendix lists the results obtained when the models were built on 10 versions of the training and testing sets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.28 | 86.86 | 87.98 | 87.42 | .8728 | .9397 |
| **Random Decision Forest** | 87.84 | 83.88 | 93.82 | 88.57 | .8782 | .9332 |
| **K-Nearest Neighbours** | 82.50 | 84.85 | 79.31 | 81.99 | .8251 | .9087 |
| **Support Vector** | 88.19 | 85.22 | 92.53 | 88.72 | .8817 | .9344 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.72 | 88.06 | 87.38 | 87.72 | .8772 | .9473 |
| **Random Decision Forest** | 89.31 | 85.08 | 95.45 | 89.97 | .8928 | .9400 |
| **K-Nearest Neighbours** | 83.79 | 86.29 | 80.52 | 83.30 | .8381 | .9165 |
| **Support Vector** | 88.71 | 86.68 | 91.59 | 89.07 | .8869 | .9437 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.76 | 87.17 | 88.67 | 87.91 | .8775 | .9403 |
| **Random Decision Forest** | 88.53 | 83.92 | 95.45 | 89.32 | .8850 | .9359 |
| **K-Nearest Neighbours** | 82.54 | 84.67 | 79.66 | 82.09 | .8256 | .9095 |
| **Support Vector** | 87.72 | 85.03 | 91.67 | 88.23 | .8770 | .9294 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.41 | 86.59 | 88.67 | 87.62 | .8741 | .9398 |
| **Random Decision Forest** | 87.03 | 84.18 | 91.33 | 87.61 | .8701 | .9334 |
| **K-Nearest Neighbours** | 82.76 | 85.38 | 79.23 | 82.19 | .8277 | .9075 |
| **Support Vector** | 87.50 | 84.53 | 91.93 | 88.08 | .8748 | .9319 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 86.34 | 85.27 | 87.98 | 86.61 | .8633 | .9332 |
| **Random Decision Forest** | 87.07 | 82.69 | 93.91 | 87.94 | .8704 | .9281 |
| **K-Nearest Neighbours** | 81.98 | 83.32 | 80.17 | 81.71 | .8199 | .8984 |
| **Support Vector** | 87.16 | 83.74 | 92.36 | 87.84 | .8713 | .9230 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.89 | 87.52 | 88.50 | 88.01 | .8789 | .9412 |
| **Random Decision Forest** | 87.46 | 84.63 | 91.67 | 88.01 | .8744 | .9322 |
| **K-Nearest Neighbours** | 83.84 | 86.04 | 80.94 | 83.41 | .8385 | .9149 |
| **Support Vector** | 88.45 | 85.34 | 92.96 | 88.99 | .8843 | .9349 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 86.81 | 85.70 | 88.50 | 87.08 | .8680 | .9358 |
| **Random Decision Forest** | 87.59 | 82.36 | 95.79 | 88.57 | .8755 | .9263 |
| **K-Nearest Neighbours** | 82.03 | 84.06 | 79.23 | 81.57 | .8204 | .9069 |
| **Support Vector** | 87.93 | 84.54 | 92.96 | 88.55 | .8791 | .9337 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.93 | 87.40 | 88.76 | 88.07 | .8793 | .9459 |
| **Random Decision Forest** | 86.51 | 83.92 | 90.47 | 87.07 | .8649 | .9274 |
| **K-Nearest Neighbours** | 83.15 | 85.44 | 80.09 | 82.68 | .8316 | .9124 |
| **Support Vector** | 88.41 | 85.33 | 92.88 | 88.94 | .8839 | .9412 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.63 | 87.14 | 88.41 | 87.77 | .8763 | .9433 |
| **Random Decision Forest** | 88.15 (tie) | 83.11 | 95.88 | 89.04 | .8811 | .9363 |
| **K-Nearest Neighbours** | 82.72 | 85.05 | 79.57 | 82.22 | .8273 | .9110 |
| **Support Vector** | 88.15 (tie) | 85.54 | 91.93 | 88.62 | .8813 | .9398 |
|  | **Accuracy Score** | **Precision Score** | **Recall Score** | **F1**  **Score** | **AUC**  **Using Predicted Values** | **AUC**  **Using Probabilities** |
| **Logistic Regression** | 87.72 | 87.16 | 88.58 | 87.87 | .8771 | .9459 |
| **Random Decision Forest** | 88.36 | 84.24 | 94.51 | 89.08 | .8834 | .9372 |
| **K-Nearest Neighbours** | 82.97 | 85.39 | 79.74 | 82.47 | .8299 | .9115 |
| **Support Vector** | 88.62 | 85.84 | 92.62 | 89.10 | .8860 | .9395 |

# Appendix B – Listing of Python Commands

**Load the Libraries and Import the data**

#----- Import the libraries and functions that will be used

import pandas as pd

from pandas import Series, DataFrame

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

from sklearn.ensemble import ExtraTreesClassifier

%matplotlib inline

#----- Import the data into a data frame

bank\_file = pd.read\_csv("C:/Users/Robert/Desktop/Banking\_with\_5\_Extra.csv")

full\_df = DataFrame(bank\_file)

#----- Look at the shape of the data

print(full\_df.shape)

#----- Peek at the first few lines of the data frame to ensure that all columns are loaded

full\_df.head(10)

**Look for NULL Values**

#----- Check to see if any columns contain null values.

full\_df.isnull().sum()

**Look for Invalid Data**

#----- Explore the data types of the features to ensure that they are correct. If not, it could indicate that there are

#----- some invalid values (such as text characters in a feature that should be numerical.)

full\_df.dtypes

#----- Look at the minimum values for the numerical features.

#----- This will highlight anomalies such as values that illogically contain negative numbers (eg: duration = -45).

print(full\_df.min())

#----- Look at the maximum values for the numerical features.

#----- This will highlight anomalies such as values that are exorbitantly high (eg: age = 205).

print(full\_df.max())

#----- List the distinct values for each categorical feature.

#----- This will highlight invalid values.

print("\nValues for JOB feature:\n-----------------------")

print(full\_df.job.value\_counts())

print("\nValues for MARITAL feature:\n---------------------------")

print(full\_df.marital.value\_counts())

print("\nValues for EDUCATION feature:\n-----------------------------")

print(full\_df.education.value\_counts())

print("\nValues for DEFAULT feature:\n---------------------------")

print(full\_df.default.value\_counts())

print("\nValues for HOUSING feature:\n---------------------------")

print(full\_df.housing.value\_counts())

print("\nValues for LOAN feature:\n------------------------")

print(full\_df.loan.value\_counts())

print("\nValues for CONTACT feature:\n---------------------------")

print(full\_df.contact.value\_counts())

print("\nValues for MONTH feature:\n-------------------------")

print(full\_df.month.value\_counts())

print("\nValues for DAY\_OF\_WEEK feature:\n-------------------------------")

print(full\_df.day\_of\_week.value\_counts())

print("\nValues for POUTCOME feature:\n----------------------------")

print(full\_df.poutcome.value\_counts())

print("\nValues for TARGET feature:\n--------------------------")

print(full\_df.target.value\_counts())

**Explore the Values for the TARGET Variable**

#----- Create a histogram to show the distribution of the TARGET values

plt.figure(figsize=(15,8))

ax = sns.countplot(x="target", data=full\_df)

plt.title('\nDistribution of the Outcome Values\n',size=25)

plt.xlabel('Outcome',size=25)

plt.xticks(size=25)

plt.ylabel('Number of Observations',size=25)

#----- Display the counts for each outcome value

print(full\_df.target.value\_counts())

#----- Display the proportional amount of each outcome value

print(full\_df.groupby(['target']).size() / len(full\_df) \* 100)

**Calculate Interquartile Range (IQR) to Determine Outliers**

#----- Calculate the IQR value for each column

Q1 = full\_df.quantile(0.25)

Q3 = full\_df.quantile(0.75)

IQR = Q3 - Q1

print(IQR)

#----- Calculate the lower limit to determine lower outliers.

#----- These values will be used in later steps when the numerical features are explored.

print(Q1 - 1.5 \* IQR)

#----- Calculate the upper limit to determine upper outliers.

#----- These values will be used in later steps when the numerical features are explored.

print(Q3 + 1.5 \* IQR)

**Explore the Numerical Variables**

**Explore the Values for the AGE Feature**

#----- Create a histogram to show the distribution of the AGE values

plt.figure(figsize=(15,8))

ax = sns.countplot(x="age", data=full\_df, color = 'Blue')

plt.title('\nDistribution of Age Values\n',size=30)

plt.xlabel('Age of Client',size=25)

plt.xticks(rotation=70)

plt.ylabel('Number of Observations',size=25)

# For readability, only display every 5th tick on the X-axis.

for index, label in enumerate(ax.xaxis.get\_ticklabels()):

if index % 5 != 0:

label.set\_visible(False)

#----- Use a Boxplot to display outliers in the AGE column

sns.boxplot(x=full\_df['age'])

#----- Count the number of outliers for the AGE attribute (ie: age < 10 or age > 69)

print(full\_df.age[full\_df.age < 10].count())

print(full\_df.age[full\_df.age > 69].count())

#----- What proportion of the AGE values are outliers?

print(469 / 41188 \* 100)

#----- How many of the AGE outliers have a target value of 'yes'?

full\_df.loc[full\_df['age'] > 69, 'target'].value\_counts()

#----- Create a histogram showing relationship between AGE and target variable

temp\_df = pd.crosstab(full\_df['age'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Age\n', fontsize='30')

ax.set\_xlabel('\nAge of Client', fontsize='25')

ax.set\_ylabel('Number of Observations', fontsize='25')

ax.legend(fontsize='40')

plt.xticks(rotation=0)

# For readability, only display every 2nd tick on the X-axis.

for index, label in enumerate(ax.xaxis.get\_ticklabels()):

if index % 2 != 0:

label.set\_visible(False)

**Explore the Values for the DURATION Feature**

#----- Create a histogram to show the distribution of the DURATION values

plt.figure(figsize=(15,8))

ax = sns.countplot(x="duration", data=full\_df, color = 'Blue')

plt.title('\nDistribution of Call Durations\n',size=25)

plt.xlabel('Call Duration (in Seconds)',size=25)

plt.ylabel('Number of Observations',size=25)

plt.xticks(rotation=70)

plt.xticks(size=15)

# For readability, only display every 100th tick on the X-axix.

for index, label in enumerate(ax.xaxis.get\_ticklabels()):

if index % 100 != 0:

label.set\_visible(False)

#----- Use a Boxplot to display outliers in the DURATION column

sns.boxplot(x=full\_df['duration'])

#----- Count the number of outliers for the DURATION attribute (ie: duration <-223.5 or age > 644.5)

print(full\_df. duration [full\_df.duration < -223.5].count())

print(full\_df. duration [full\_df.duration > 644.5].count())

#----- What proportion of the DURATION values are outliers?

print(2963 / 41188 \* 100)

#----- How many of the DURATION outliers have a target value of 'yes'?

full\_df.loc[full\_df['duration'] > 644.5, 'target'].value\_counts()

#----- Create a histogram showing relationship between DURATION and target variable

temp\_df = pd.crosstab(full\_df['duration'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(fontsize=9,figsize=[19,10])

ax.set\_title('\nOutcomes Grouped by Call Duration\n', fontsize='25')

ax.set\_xlabel('\nCall Duration (in seconds)', fontsize='25')

ax.set\_ylabel('Number of Observations', fontsize='25')

ax.legend(fontsize='40')

plt.xticks(size=20)

**Explore the Values for the CAMPAIGN Feature**

#----- Create a histogram to show the distribution of the CAMPAIGN values

plt.figure(figsize=(15,8))

ax = sns.countplot(x="campaign", data=full\_df, color = 'Blue')

plt.title('\nDistribution of Contacts Made During Campaign\n',size=25)

plt.xlabel('\nNumber of Contacts Made During Campaign (per Client)',size=25)

plt.ylabel('Number of Observations',size=25)

#----- Use a Boxplot to display outliers in the CAMPAIGN column

sns.boxplot(x=full\_df['campaign'])

#----- Count the number of outliers for the CAMPAIGN attribute (ie: campaign < -2 or age > 6)

print(full\_df.campaign [full\_df.campaign < -2].count())

print(full\_df.campaign [full\_df.campaign > 6].count())

#----- What proportion of the CAMPAIGN values are outliers?

print(2406 / 41188 \* 100)

#----- How many of the CAMPAIGN outliers have a target value of 'yes'?

full\_df.loc[full\_df['campaign'] > 6, 'target'].value\_counts()

#----- Create a histogram showing relationship between CAMPAIGN and target variable

temp\_df = pd.crosstab(full\_df['campaign'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Number of Contacts Made During Campaign\n', fontsize='25')

ax.set\_xlabel('\nNumber of Contacts Made During Campaign', fontsize='25')

ax.set\_ylabel('Number of Observations', fontsize='25')

ax.legend(fontsize='40')

plt.xticks(rotation=0)

**Explore the Values for the PDAYS Feature**

#----- Create a histogram to show the distribution of the PDAYS values

plt.figure(figsize=(15,8))

ax = sns.countplot(x="pdays", data=full\_df, color = 'Blue')

plt.title('\nNumber of Days since Client was Contacted for a Previous Campaign\n',size=25)

plt.xlabel('\nNumber of Days',size=25)

plt.ylabel('Number of Observations',size=25)

plt.xticks(size=15)

#----- Use a Boxplot to display outliers in the PDAYS column

sns.boxplot(x=full\_df['pdays'])

#----- Count the number of outliers for the PDAYS attribute (ie: pdays != 999)

print(full\_df.pdays [full\_df.pdays != 999].count())

print(full\_df.pdays [full\_df.pdays == 999].count())

#----- What proportion of the PDAYS values have values that are not '999'?

print(1515 / 41188 \* 100)

#----- How many of the PDAYS outliers have a target value of 'yes'?

full\_df.loc[full\_df['pdays'] != 999, 'target'].value\_counts()

**Explore the Values for the PREVIOUS Feature**

#----- Create a histogram to show the distribution of the PREVIOUS values

plt.figure(figsize=(15,12))

ax = sns.countplot(x="pdays", data=full\_df, color = 'Blue')

plt.title('\nNumber of Times Client was Contacted for Previous Campaigns\n',size=25)

plt.xlabel('Number of Times Client was Contacted',size=25)

plt.ylabel('Number of Observations',size=25)

plt.xticks(size=15)

#----- Use a Boxplot to display outliers in the PREVIOUS column

sns.boxplot(x=full\_df['previous'])

#----- Count the number of outliers for the PREVIOUS attribute (ie: previous != 0)

print(full\_df.campaign [full\_df.previous != 0].count())

print(full\_df.campaign [full\_df.previous == 0].count())

#----- What proportion of the PREVIOUS values have values that are not '0'?

print(5625 / 41188 \* 100)

#----- How many of the PREVIOUS outliers have a target value of 'yes'?

full\_df.loc[full\_df['previous'] != 0, 'target'].value\_counts()

#----- Create a histogram showing relationship between PREVIOUS and target variable

temp\_df = pd.crosstab(full\_df['previous'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Number of Times Client was Contacted for Previous Campaigns\n',

fontsize='25')

ax.set\_xlabel('Number of Contacts Made for Previous Campaigns', fontsize='25')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=15)

plt.xticks(rotation=0)

#----- Create a histogram showing relationship between EMP\_VAR\_RATE and target variable

temp\_df = pd.crosstab(full\_df['emp\_var\_rate'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Employment Variation Rate\n', fontsize='30')

ax.set\_xlabel('\nEmployment Variation Rate', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- Create a histogram showing relationship between CONS\_PRICE\_IDX and target variable

temp\_df = pd.crosstab(full\_df['cons\_price\_idx'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Consumer Price Index\n', fontsize='30')

ax.set\_xlabel('\nConsumer Price Index', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=20)

#----- Create a histogram showing relationship between CONS\_CONF\_IDX and target variable

temp\_df = pd.crosstab(full\_df['cons\_conf\_idx'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Consumer Confidence Index\n', fontsize='30')

ax.set\_xlabel('\nConsumer Confidence Index', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=20)

#----- Create a histogram showing relationship between EURIBOR3M and target variable

temp\_df = pd.crosstab(full\_df['euribor3m'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Euribor 3-Month Rate\n', fontsize='30')

ax.set\_xlabel('\nEuribor 3-Month Rate', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=20)

# For readability, only display every 100th tick on the X-axix.

for index, label in enumerate(ax.xaxis.get\_ticklabels()):

if index % 8 != 0:

label.set\_visible(False)

#----- Create a histogram showing relationship between NR\_EMPLOYED and target variable

temp\_df = pd.crosstab(full\_df['nr\_employed'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Quarterly Employment Indicator\n', fontsize='30')

ax.set\_xlabel('\nQuarterly Employment Indicator', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=20)

**Explore the Categorical Features**

#----- Create a histogram showing relationship between JOB and target variable

temp\_df = pd.crosstab(full\_df['job'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Job Type\n', fontsize='30')

ax.set\_xlabel('Job Type', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=20)

# ----- What is the proportion of UNKNOWN value in the JOB feature?

print(full\_df.groupby(['job']).size() / len(full\_df) \* 100)

#----- Create a histogram showing relationship between MARITAL and target variable

temp\_df = pd.crosstab(full\_df['marital'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Marital Status\n', fontsize='30')

ax.set\_xlabel('\nMarital Status', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- What is the proportion of UNKNOWN value in the MARITAL feature?

print(full\_df.groupby(['marital']).size() / len(full\_df) \* 100)

#----- Create a histogram showing relationship between EDUCATION and target variable

temp\_df = pd.crosstab(full\_df['education'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Education Level\n', fontsize='30')

ax.set\_xlabel('\nEducation Level', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=70)

#----- What is the proportion of UNKNOWN value in the EDUCATION feature?

print(full\_df.groupby(['education']).size() / len(full\_df) \* 100)

#----- Create a histogram showing relationship between DEFAULT and target variable

temp\_df = pd.crosstab(full\_df['default'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Credit Default Status\n', fontsize='30')

ax.set\_xlabel('\nDoes Client Have Credit in Default?', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- What is the proportion of UNKNOWN value in the DEFAULT feature?

print(full\_df.groupby(['default']).size() / len(full\_df) \* 100)

#----- Create a histogram showing relationship between HOUSING and target variable

temp\_df = pd.crosstab(full\_df['housing'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Housing Loan Status\n', fontsize='30')

ax.set\_xlabel('\nDoes Client Have a Housing Loan?', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- What is the proportion of UNKNOWN value in the HOUSING feature?

print(full\_df.groupby(['housing']).size() / len(full\_df) \* 100)

#----- Create a histogram showing relationship between LOAN and target variable

temp\_df = pd.crosstab(full\_df['loan'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Personal Loan Status\n', fontsize='30')

ax.set\_xlabel('\nDoes Client Have a Personal Loan?', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- What is the proportion of UNKNOWN value in the LOAN feature?

print(full\_df.groupby(['loan']).size() / len(full\_df) \* 100)

#----- Create a histogram showing relationship between CONTACT and target variable

temp\_df = pd.crosstab(full\_df['contact'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Contact Communication Type\n', fontsize='30')

ax.set\_xlabel('\nContact Communication Type', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- Create a histogram showing relationship between MONTH and target variable

temp\_df = pd.crosstab(full\_df['month'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Contact Month\n', fontsize='30')

ax.set\_xlabel('\nContact Month', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- Create a histogram showing relationship between DAY\_OF\_WEEK and target variable

temp\_df = pd.crosstab(full\_df['day\_of\_week'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Contact Day of Week\n', fontsize='30')

ax.set\_xlabel('\nContact Day of Week', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- Create a histogram showing relationship between POUTCOME and target variable

temp\_df = pd.crosstab(full\_df['poutcome'].sort\_values(), full\_df['target'])

ax = temp\_df.plot(kind='bar',fontsize=15,figsize=[20,15])

ax.set\_title('\nOutcomes Grouped by Outcome of the Previous Campaign\n', fontsize='30')

ax.set\_xlabel('\nOutcome of the Previous Marketing Campaign', fontsize='30')

ax.set\_ylabel('Number of Observations', fontsize='30')

ax.legend(fontsize='40')

plt.xticks(size=30)

plt.xticks(rotation=0)

#----- What is the proportion of NONEXISTANT value in the POUTCOME feature?

print(full\_df.groupby(['poutcome']).size() / len(full\_df) \* 100)

**Explore Correlation between the Features**

#----- Generate a correlation matrix to get a sense of the correlation between the numerical features.

rs = np.random.RandomState(0)

corr = full\_df.corr()

corr.style.background\_gradient()

**Drop Some Features from the Dataset**

#----- Drop some features from the dataset because they were deemed to have no value.

del full\_df['pdays']

del full\_df['default']

**Transform Categorical Features into Numerical Dummy Variables**

#----- Create a new dataframe called 'dummies\_df' and convert

#----- all categorical variables to numerical variables.

#-----

#----- Add a prefix to each new column for ease of identification.

#----- Drop first column of each variable in order to minimize the dimensionality of the dataset.

#-----

dummies\_df = pd.get\_dummies(full\_df, prefix='job\_', columns=['job'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='marital\_', columns=['marital'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='education\_', columns=['education'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='housing\_', columns=['housing'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='loan\_', columns=['loan'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='contact\_', columns=['contact'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='month\_', columns=['month'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='day\_of\_week\_',columns=['day\_of\_week'],drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, prefix='poutcome\_', columns=['poutcome'], drop\_first=True)

dummies\_df = pd.get\_dummies(dummies\_df, columns=['target'], drop\_first=True)

#----- The name of the "target" variable has been changed to "target\_yes".

#----- Restore the name to "target" (just for the sake of cleanliness).

dummies\_df.rename(columns={'target\_yes':'target'}, inplace=True)

#----- Display the structure of the 'dummies\_df' dataframe.

dummies\_df.dtypes

#----- Peek at the first few lines of the data frame to confirm that the column values are as expected.

dummies\_df.sample(5)

**Standardize the Features**

#----- Standardize all of the features in the dummies\_df dataframe.

#----- The result will be placed in a numpy array.

scaler = StandardScaler()

scaled\_array = scaler.fit\_transform(dummies\_df)

#----- Convert the numpy array to a dataframe.

#----- The index and columns keyword arguments are used to keep the original indices and column names.

scaled\_df = pd.DataFrame(scaled\_array, index=dummies\_df.index, columns=dummies\_df.columns)

#----- Peek at the data to confirm that the scaling process was successful.

scaled\_df.sample(5)

**Handle the Unbalanced Data Using Random Under-Sampling**

#----- The target variable was scaled in the previous steps. Convert its' values back to '0' and '1' because

#----- working with 0/1 is more intuitive than working with -0.35/2.80 and the columns will need to contain

#----- integers for the predictive models.

scaled\_df['target'][(scaled\_df['target'] < 0)] = 0

scaled\_df['target'][(scaled\_df['target'] > 0)] = 1

scaled\_df['target'] = scaled\_df['target'].apply(int)

#----- Generate a bar chart showing the current distribution of the target variable.

scaled\_df.target.value\_counts().plot(kind='bar');

#plt.plot(scaled\_df)

plt.title('Distribution of Target Variable (Prior to Rebalancing)\n',size=25)

plt.ylabel('Number of Occurrences\n',size=20)

plt.xlabel('Target Variable\n',size=20)

plt.xticks(fontsize=20)

plt.yticks(fontsize=15)

plt.rcParams['figure.figsize'] = [7, 7]

#----- Create a new dataframe to hold the rows for each target value ('0' and '1')

target\_0\_df = scaled\_df[scaled\_df['target'] == 0]

target\_1\_df = scaled\_df[scaled\_df['target'] == 1]

#----- Use the undersampling technique to reduce the number of '0' rows.

#-----

#----- Undersampling means that we will reduce the number of rows with target = '0' to n rows,

#----- where n is the number of rows with a target of '1'.

#----- In this dataset, there are 4640 rows with a target of '1'. Therefore, the number of rows

#----- with a target of '0' will be reduced to a random sample of 4640 rows.

count\_target\_1 = target\_1\_df.target.count() # Count the number of '1' rows.

target\_0\_df\_under = target\_0\_df.sample(count\_target\_1) # Reduce the number of '0' rows.

#----- Concatenate the rows with an outcome of '1' with the reduced number of rows with an outcome of '0'.

#----- Place the results in a dataframe called 'balanced\_df'.

balanced\_df = pd.concat([target\_0\_df\_under, target\_1\_df], axis=0)

#----- Display the shape of the dataframe that has a balanced target column.

print(balanced\_df.shape)

#----- Generate a bar chart showing the new distribution of the target variable.

#----- The distribution is now balanced.

balanced\_df.target.value\_counts().plot(kind='bar');

plt.plot(balanced\_df)

plt.title('Balanced Distribution of Target Variable\n',size=25)

plt.ylabel('Number of Occurrences\n',size=20)

plt.xlabel('Target Variable\n',size=20)

plt.xticks(fontsize=20)

plt.yticks(fontsize=15)

plt.rcParams['figure.figsize'] = [7, 7]

**Separate the Dataset Features into X and Y Datasets**

#----- Create a dataframe called 'X' to hold only the model’s independent variables.

X = balanced\_df.copy()

del X['target'] # Remove the target variable.

#----- Create a dataframe called 'Y' to hold only the model’s target variable.

Y = balanced\_df['target']

**Perform Primary Component Analysis on the Dependent Variables**

#----- Perform the Principal Component Analysis on the balanced dataset of dependent variables.

pca = PCA(n\_components = 50)

sklearn\_pca\_X = pca.fit\_transform(X)

#----- The goal is find out how many how many components would be enough to explain almost all of the

#----- data variance. First, the covariance matrix must be generated.

covariance\_matrix = np.cov(X.T)

#----- Display the covariance matrix to confirm that it was generated successfully.

print(covariance\_matrix)

#----- Use the covariance matrix to calculate the Eigen Vectors and Eigen Values.

eig\_vals, eig\_vecs = np.linalg.eig(covariance\_matrix)

#----- Display the Eigen Values to confirm that they were generated successfully.

print(eig\_vals)

#----- Display the Eigen values (in descending order) as well as their percentages of variance and cumulative percentages.

print("\n Eigen value\t\tPercentage of Variance\t\tCumulative Percentage\n")

counter = 1

cumulative\_pct = 0.0

for i in list(np.sort(eig\_vals)[::-1]):

cumulative\_pct = cumulative\_pct + (i/sum(eig\_vals)\*100)

print("%d\t%.3f\t\t\t%.3f\t\t\t\t%.3f" %

(counter,i,(i/sum(eig\_vals)\*100),cumulative\_pct))

counter = counter + 1

#----- Plot the variance ratios for each component. This will show us where the variance contributions become minimal.

explained\_variance = pca.explained\_variance\_ratio\_

plt.plot(explained\_variance,color='Blue', linewidth=3)

plt.rcParams['figure.figsize'] = [9, 9]

plt.title('Variance Ratios\n',size=30)

plt.ylabel('Variance Contributions\n',size=25)

plt.xlabel('\nNumber of Components',size=25)

plt.xticks(fontsize=14)

plt.yticks(fontsize=14)

**Estimate the Importance of Features using Extra Trees Classifier Decision Tree**

#----- Build a forest and compute the importance of each feature.

forest = ExtraTreesClassifier(n\_estimators=250, random\_state=0)

forest.fit(X,Y)

#----- Store the importance values and display them to confirm that they have been generated.

importances = forest.feature\_importances\_

print(importances)

#----- Sort the array elements by the importance values, in descending order.

indices = np.argsort(importances)[::-1]

#----- Display the features and their importance values, in descending order of importance.

print("\n Importance Cumulative")

print(" Importance")

print("----------------------------------------------------------------------------------------")

cumulative\_pct = 0

for i in range(0,X.shape[1]):

cumulative\_pct += importances[indices[i]]

print("%d\tFeature %d: %30s\t\t%0.2f\t\t%0.2f" % (i+1, indices[i], X.columns.values[indices[i]],

importances[indices[i]]\*100, cumulative\_pct\*100))

#----- Plot the feature importances, in descending order of importance.

plt.figure(figsize=(15,10))

plt.title("\nImportance Levels for Each Feature\n", size=25)

plt.ylabel('Importance Level\n',size=25)

plt.xlabel('\nFeature Number',size=25)

plt.bar(range(X.shape[1]), importances[indices], color="green", align="center")

plt.xticks(range(X.shape[1]), indices, size=10)

plt.xlim([-1, X.shape[1]])

plt.show()

**Remove the Least Important Features**

#----- Remove the features with the lowest importances from the dataframe that holds the independent variables.

del X['job\_\_self-employed']

del X['job\_\_student']

del X['job\_\_unemployed']

del X['job\_\_housemaid']

del X['month\_\_nov']

del X['month\_\_aug']

del X['month\_\_jun']

del X['housing\_\_unknown']

del X['loan\_\_unknown']

del X['month\_\_jul']

del X['month\_\_sep']

del X['job\_\_unknown']

del X['marital\_\_unknown']

del X['month\_\_dec']

del X['education\_\_illiterate']

#----- Display the remaining columns to see the list of features

#----- that will be used from this point onwards.

X.dtypes

**Split the Dataset into Training and Testing Sets**

#----- Split the data into training/testing sets using a 75/25 ratio.

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=1)

#----- Look at the shape of the X training data set and peek at the data.

print(x\_train.shape)

print(x\_train.head())

#----- Look at the shape of the X testing data set and peek at the data.

print(x\_test.shape)

print(x\_test.head())

#----- Look at the shape of the Y training data set and peek at the data.

print(y\_train.shape)

print(y\_train.head())

#----- Look at the shape of the Y testing data set and peek at the data.

print(y\_test.shape)

print(y\_test.head())

L**ogistic Regression**

#----- Build the logistic regression model and train it using the training data sets

LR\_model = LogisticRegression()

LR\_model.fit (x\_train, y\_train)

#----- Make predictions using the testing data set

LR\_predictions = LR\_model.predict(x\_test)

print("\nPredicted values are:\n",LR\_predictions)

**Random Forest (Decision Tree)**

#----- Build the random forest model and train it using the training data sets

DT\_model = RandomForestClassifier(criterion='entropy', min\_samples\_split=200, max\_features='auto',

max\_leaf\_nodes=None, verbose=0)

DT\_model =DT\_model.fit(x\_train, y\_train)

#----- Make predictions using the testing data set

DT\_predictions = DT\_model.predict(x\_test)

print("\nPredicted values are:\n",DT\_predictions)

**K-Nearest Neighbours**

#----- Build the K-Nearest Neighbours model and train it using the training data sets

KNN\_model = KNeighborsClassifier(25)

KNN\_model.fit (x\_train, y\_train)

#----- Make predictions using the testing data set

KNN\_predictions = KNN\_model.predict(x\_test)

print("\nPredicted values are:\n",KNN\_predictions)

**Support Vector Classification**

#----- Build the Support Vector model and train it using the training data sets

SV\_model = SVC(probability=True)

SV\_model.fit (x\_train, y\_train)

#----- Make predictions using the testing data set

SV\_predictions = SV\_model.predict(x\_test)

print("\nPredicted values are:\n",SV\_predictions)

**Generate the Metrics for the Models**

#----- Generate the accuracy scores for the models

LR\_accuracy = accuracy\_score(y\_test, LR\_predictions)

DT\_accuracy = accuracy\_score(y\_test, DT\_predictions)

KNN\_accuracy = accuracy\_score(y\_test, KNN\_predictions)

SV\_accuracy = accuracy\_score(y\_test, SV\_predictions)

print('\nLR accuracy score: \t %0.2f' % (LR\_accuracy \* 100))

print('DT accuracy score:\t %0.2f' % (DT\_accuracy \* 100))

print('KNN accuracy score:\t %0.2f' % (KNN\_accuracy \* 100))

print('SV accuracy score:\t %0.2f' % (SV\_accuracy \* 100))

#----- Generate the precision scores for the models

LR\_precision = precision\_score(y\_test, LR\_predictions)

DT\_precision = precision\_score(y\_test, DT\_predictions)

KNN\_precision = precision\_score(y\_test, KNN\_predictions)

SV\_precision = precision\_score(y\_test, SV\_predictions)

print('\nLR precision score: \t %0.2f' % (LR\_precision \* 100))

print('DT precision score:\t %0.2f' % (DT\_precision \* 100))

print('KNN precision score:\t %0.2f' % (KNN\_precision \* 100))

print('SV precision score:\t %0.2f' % (SV\_precision \* 100))

#----- Generate the recall scores for the models

LR\_recall = recall\_score(y\_test, LR\_predictions)

DT\_recall = recall\_score(y\_test, DT\_predictions)

KNN\_recall = recall\_score(y\_test, KNN\_predictions)

SV\_recall = recall\_score(y\_test, SV\_predictions)

print('\nLR recall score: \t %0.2f' % (LR\_recall \* 100))

print('DT recall score:\t %0.2f' % (DT\_recall \* 100))

print('KNN recall score:\t %0.2f' % (KNN\_recall \* 100))

print('SV recall score:\t %0.2f' % (SV\_recall \* 100))

#----- Generate the F1 scores for the models

LR\_f1 = f1\_score(y\_test, LR\_predictions)

DT\_f1 = f1\_score(y\_test, DT\_predictions)

KNN\_f1 = f1\_score(y\_test, KNN\_predictions)

SV\_f1 = f1\_score(y\_test, SV\_predictions)

print('\nLR F1 score: \t %0.2f' % (LR\_f1 \* 100))

print('DT F1 score:\t %0.2f' % (DT\_f1 \* 100))

print('KNN F1 score:\t %0.2f' % (KNN\_f1 \* 100))

print('SV F1 score:\t %0.2f' % (SV\_f1 \* 100))

#----- Generate the confusion matrices for the models using the predicted and actual results

LR\_conf\_matrix = confusion\_matrix(y\_test, LR\_predictions)

DT\_conf\_matrix = confusion\_matrix(y\_test, DT\_predictions)

KNN\_conf\_matrix = confusion\_matrix(y\_test, KNN\_predictions)

SV\_conf\_matrix = confusion\_matrix(y\_test, SV\_predictions)

print('\nLogistic Regression model:\n',LR\_conf\_matrix)

print('\nDecision Tree model:\n',DT\_conf\_matrix)

print('\nK-Nearest Neighbours model:\n',KNN\_conf\_matrix)

print('\nSupport Vector model:\n',SV\_conf\_matrix)

#----- Generate a heat map of the confusion matrix for the Logistic Regression model

LR\_heatmap = sns.heatmap(LR\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for Logistic Regression Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Generate a heat map of the SNS confusion matrix for the Decision Tree model

DT\_heatmap = sns.heatmap(DT\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for Decision Tree Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Generate a heat map of the confusion matrix for the K-Nearest Neighbours model

KNN\_heatmap = sns.heatmap(KNN\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for K-Nearest Neighbours Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Generate a heat map of the confusion matrix for the Support Vector model

SV\_heatmap = sns.heatmap(SV\_conf\_matrix, cmap="Blues", annot=True, fmt="g")

plt.title('\nConfusion Matrix for Support Vector Model\n',size=20)

plt.xlabel('\nTrue Values',size=20)

plt.ylabel('Predicted Values',size=20)

#----- Compare the Receiver Operating Characteristic (ROC) Curves of All Models

#----- Calculate the true and false positive rates for the models

LR\_fpr, LR\_tpr, LR\_threshholds = roc\_curve(y\_test, LR\_predictions,drop\_intermediate=False)

DT\_fpr, DT\_tpr, DT\_threshholds = roc\_curve(y\_test, DT\_predictions, drop\_intermediate=False)

KNN\_fpr, KNN\_tpr, KNN\_threshholds = roc\_curve(y\_test, KNN\_predictions)

SV\_fpr, SV\_tpr, SV\_threshholds = metrics.roc\_curve(y\_test, SV\_predictions, drop\_intermediate=False)

#----- Calculate the area under the curve for the models

LR\_roc\_auc = metrics.auc(LR\_fpr, LR\_tpr)

DT\_roc\_auc = metrics.auc(DT\_fpr, DT\_tpr)

KNN\_roc\_auc = metrics.auc(KNN\_fpr, KNN\_tpr)

SV\_roc\_auc = metrics.auc(SV\_fpr, SV\_tpr)

#----- Plot the ROC curves for all models

plt.figure(figsize=(10,10))

#----- Add the ROC Curve for the Logistic Regression to the Graph

LR\_line, = plt.plot(LR\_fpr, LR\_tpr, color='Red', lw=3, label='Logistic Regression (area = %0.4f)' % LR\_roc\_auc)

#----- Add the ROC Curve for the Random Decision Forest to the Graph

DT\_line, = plt.plot(DT\_fpr, DT\_tpr, color='Black', lw=3, label='Random Decision Forest (area = %0.4f)' % DT\_roc\_auc)

#----- Add the ROC Curve for the K-Nearest Neighbours to the Graph

KNN\_line, = plt.plot(KNN\_fpr, KNN\_tpr, color='Green', lw=3, label='K-Nearest Neighbours (area = %0.4f)' % KNN\_roc\_auc)

#----- Add the ROC Curve for the Support Vector Classification model to the Graph

SV\_line, = plt.plot(SV\_fpr,SV\_tpr, color='Yellow',lw=3,label='Support Vector Classification (area = %0.4f)' % SV\_roc\_auc)

#----- Add the baseline

baseline, = plt.plot([0, 1], [0, 1], color='Blue', lw=3, label='Baseline', linestyle='--')

plt.xlabel('\nFalse Positive Rate',size=20)

plt.ylabel('True Positive Rate', size=20)

plt.title('\nROC Curves of All Models (Using Predicted Values)\n',size=25)

plt.legend(loc="lower right",handles=[LR\_line, DT\_line, KNN\_line, SV\_line, baseline], prop={'size': 16})

plt.show()

#----- Plot the ROC curves for all models using the probabilities

plt.figure(figsize=(10,10))

#----- Add the ROC Curve for the Logistic Regression to the Graph

LR\_y\_pred\_proba = LR\_model.predict\_proba(x\_test)[:,1]

LR\_fpr, LR\_tpr, LR\_threshholds = metrics.roc\_curve(y\_test, LR\_y\_pred\_proba)

LR\_roc\_auc = metrics.roc\_auc\_score(y\_test, LR\_y\_pred\_proba)

LR\_line, = plt.plot(LR\_fpr, LR\_tpr, color='Red', lw=3, label='Logistic Regression (area = %0.4f)' % LR\_roc\_auc)

#----- Add the ROC Curve for the Random Decision Forest to the Graph

DT\_y\_pred\_proba = DT\_model.predict\_proba(x\_test)[:,1]

DT\_fpr, DT\_tpr, DT\_threshholds = metrics.roc\_curve(y\_test, DT\_y\_pred\_proba)

DT\_roc\_auc = metrics.roc\_auc\_score(y\_test, DT\_y\_pred\_proba)

DT\_line, = plt.plot(DT\_fpr,DT\_tpr,color='Black', lw=3, label='Random Decision Forest (area = %0.4f)' % DT\_roc\_auc)

#----- Add the ROC Curve for the K-Nearest Neighbours to the Graph

KNN\_y\_pred\_proba = KNN\_model.predict\_proba(x\_test)[:,1]

KNN\_fpr, KNN\_tpr, KNN\_threshholds = metrics.roc\_curve(y\_test, KNN\_y\_pred\_proba)

KNN\_roc\_auc = metrics.roc\_auc\_score(y\_test, KNN\_y\_pred\_proba)

KNN\_line, = plt.plot(KNN\_fpr,KNN\_tpr,color='Green', lw=3, label='K-Nearest Neighbours (area = %0.4f)' % KNN\_roc\_auc)

#----- Add the ROC Curve for the Support Vector Classification model to the Graph

SV\_y\_pred\_proba = SV\_model.predict\_proba(x\_test)[:,1]

SV\_fpr, SV\_tpr, SV\_threshholds = metrics.roc\_curve(y\_test, SV\_y\_pred\_proba)

SV\_roc\_auc = metrics.roc\_auc\_score(y\_test, SV\_y\_pred\_proba)

SV\_line, = plt.plot(SV\_fpr,SV\_tpr,color='Yellow',lw=3,label='Support Vector Classification (area = %0.4f)' % SV\_roc\_auc)

#----- Add the baseline

baseline, = plt.plot([0, 1], [0, 1], color='Blue', lw=3, label='Baseline', linestyle='--')

plt.xlabel('\nFalse Positive Rate',size=20)

plt.ylabel('True Positive Rate', size=20)

plt.title('\nROC Curves of All Models (Using Probabilities)\n',size=25)

plt.legend(loc="lower right",handles=[LR\_line, DT\_line, KNN\_line, SV\_line, baseline], prop={'size': 16})

plt.show()