Machine Learning Model Project

Standard Bank Marketing Data Set

By

Kudakwashe Rodrick Makamure

Submitted on 29/10/2019

Contents

1. Introduction 1
2. Business Understanding and Data Understanding 2
   2.1. Description of the features 3
   2.2. Understanding the features 4
3. Data Preparation 7
   3.1. Replacing unknown values 7
   3.2. Handling Missing Values 8
   3.3. Encoding Categorical Features 9
   3.4. Thresholding 11
   3.5. Descriptive Statistics 12
   3.6. Handling Outliers 13
   3.7. Co-Variance 14
   3.8. Normalization 15
   3.9. Train-Test Split 15
4. Modelling 16
   4.1. Decision Tree Classifier 16
   4.2. Logistic Regression 17
5. Evaluation of Classification Model 18

1. Introduction

The Standard Bank Group serves a number of retail customers across the African continent. In order to service customers better, Standard Bank needs to ensure that any marketing campaigns target only potentially interested customers relating to that campaign. This ensures that Standard Bank does not bombard customers with unnecessary information whilst maximising the use of personnel who engage with the customers.

The purpose of the project is to build a machine learning model that will make the Standard Bank's campaigns more intelligent based on historical marketing data. The project will make use of a publicly available dataset used in previous international research relating to telemarketing in banking and an additional csv file with more data and features.

The objectives of the project are to; (1) develop a machine learning model that uses the following features: age, job, marital, education, default, housing, loan, contact, month and day of week, to predict the success of a campaign and (2) debiase the machine leaning model based on sensitive columns namely; age and marital columns.

To achieve these objectives, the project will follow the CRISP-DM life-cycle methodology. The iterative procedures included in the life-cycle are: business understanding, data understanding, data preparation, modelling, evaluation and deployment. The procedures are illustrated in Figure 1.

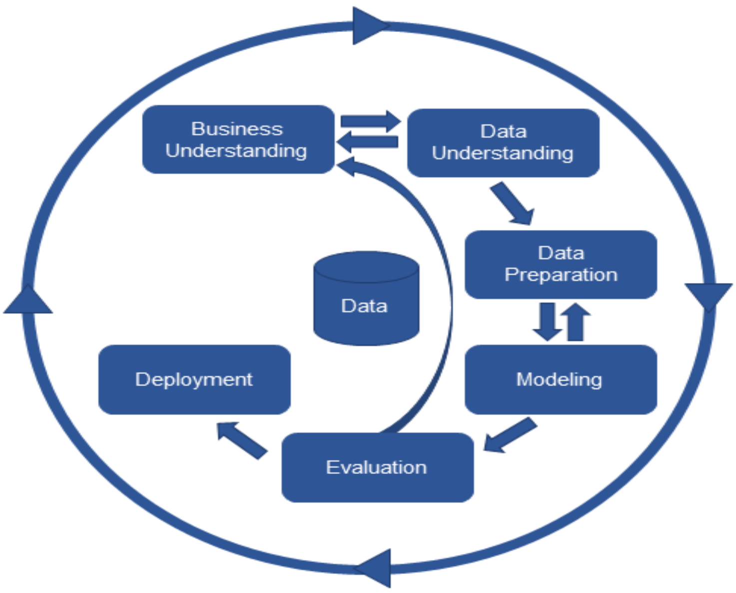


Figure 1: CRISP-DM Life-Cycle

The model will be implemented using Jupyter Notebook which is an open-source web application that allows for the creation and sharing of documents that contain live code, equations, visualizations and narrative text. The model will be coded using Python programming language, which is used for developing complex scientific and numeric applications. Further, python libraries cover data cleaning, data manipulation and visualization, and modelling, which are key for implementing the proposed machine learning model.

The following sections will describe the process followed in building the machine learning model for Standard Bank, the data preparation steps conducted, the classification models chosen and the results achieved from evaluations.

2. Business Understanding and Data Understanding

The dataset is related direct marketing campaign of Standard Bank. Marketing campaign involves phone calls to the customers to make the customers accept to make a term deposit with the bank. Therefore, after being in contact with the customer, the call is binary classified to ‘no’ being the client did not make a deposit and ‘yes’ being the client on call accepted to make a deposit.

The purpose of the machine learning model is to predict if the customer on call would accept to make a term deposit or not based on the information of the customers.

The dataset has 4119 rows of data, with 20 features and one column of class information. However, for the purpose of this project, only 11 features are to be used name; 'age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', and 'y'. The features used are depicted in Figure 2 and the total number of rows and columns in Figure 3.

In [3]:
```
#importing bank data
data = pd.read_csv('bank-additional.csv', sep =';')
data = data[['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week','y']
data
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	y
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	no
1	39	services	single	high.school	no	no	no	telephone	may	fri	no
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	no

Figure 2: Original DataFrame

...	...	...	...	...	...	...	...	...	...	...	...
4089	25	admin.	single	university.degree	no	yes	yes	cellular	oct	fri	yes
4090	43	blue-collar	married	basic.4y	unknown	yes	yes	telephone	may	tue	no
4091	38	management	married	high.school	unknown	no	no	telephone	may	thu	no
4092	30	blue-collar	single	high.school	no	no	no	telephone	jul	wed	no
4093	56	retired	married	basic.4y	unknown	no	no	cellular	jul	tue	no
4094	62	blue-collar	married	basic.4y	no	yes	no	cellular	nov	mon	no
4095	36	admin.	single	university.degree	no	no	yes	cellular	aug	fri	no
4096	33	services	married	high.school	no	no	no	telephone	may	mon	no
4097	41	blue-collar	divorced	basic.9y	no	no	no	cellular	aug	tue	no
4098	34	housemaid	single	university.degree	no	yes	no	cellular	aug	thu	no
4099	58	admin.	divorced	high.school	no	no	no	cellular	aug	tue	no
4100	41	admin.	divorced	high.school	no	no	no	cellular	apr	fri	no
4101	35	entrepreneur	single	university.degree	no	yes	no	cellular	jul	mon	no
4102	31	blue-collar	single	basic.9y	unknown	no	yes	telephone	jun	fri	no
4103	43	services	married	high.school	no	no	no	telephone	may	mon	no
4104	42	technician	divorced	professional.course	no	yes	no	cellular	aug	mon	no
4105	47	housemaid	married	basic.4y	unknown	yes	no	telephone	jul	tue	no
4106	45	entrepreneur	divorced	basic.9y	no	yes	no	cellular	may	tue	no
4107	36	admin.	married	university.degree	unknown	yes	no	cellular	aug	wed	no
4108	32	admin.	married	university.degree	no	yes	no	telephone	may	thu	no
4109	63	retired	married	high.school	no	no	no	cellular	oct	wed	no
4110	53	housemaid	divorced	basic.6y	unknown	unknown	unknown	telephone	may	fri	no
4111	30	technician	married	university.degree	no	no	yes	cellular	jun	fri	no
4112	31	technician	single	professional.course	no	yes	no	cellular	nov	thu	no
4113	31	admin.	single	university.degree	no	yes	no	cellular	nov	thu	no
4114	30	admin.	married	basic.6y	no	yes	yes	cellular	jul	thu	no
4115	39	admin.	married	high.school	no	yes	no	telephone	jul	fri	no
4116	27	student	single	high.school	no	no	no	cellular	may	mon	no
4117	58	admin.	married	high.school	no	no	no	cellular	aug	fri	no
4118	34	management	single	high.school	no	yes	no	cellular	nov	wed	no

4119 rows × 11 columns

Figure 3: Rows and Columns of dataset

2.1. Description of the features

To gain a better understanding of the dataset, the features have to be explored. The various features of the dataset had different data types, attributes and descriptions.

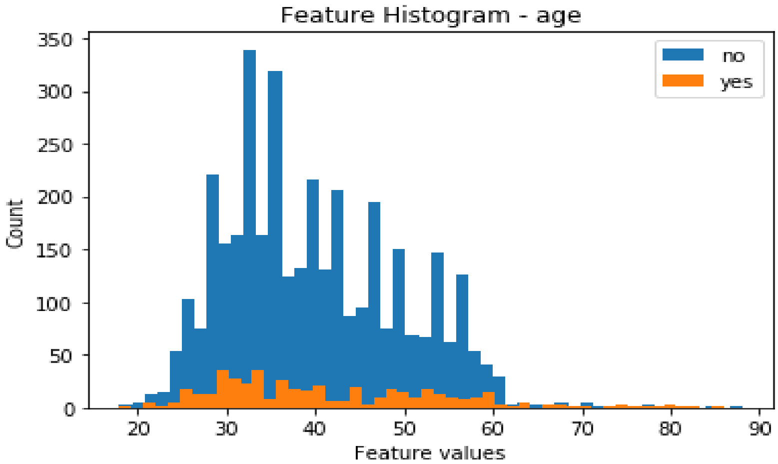
Table 2-1: Description of Dataset

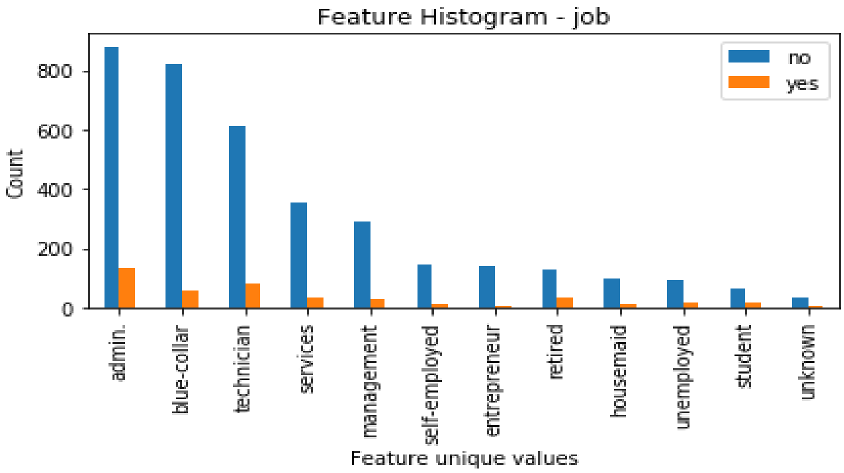
No.	Feature Name	Feature Type	Feature Description
1	Age	Numeric, Bank Client Data	Age of the client
2	Job	Categorical, Bank Client Data	Type of job the client is
3	Marital	Categorical, Bank Client Data	Marital Status of the client
4	Education	Categorical, Bank Client Data	Education level of the client
5	Default	Categorical, Bank Client Data	Has Credit in Default or not
6	Housing	Categorical, Bank Client Data	If the client has a Housing Loan or not
7	Loan	Categorical, Bank Client Data	If the client has a Personal Loan or not
8	Contact	Categorical, Related to Last Contact	Mode of contact last time
9	Month	Categorical, Related to Last Contact	Month when last contact was made
10	Day_of_Week	Categorical, Related to Last Contact	Day of week last contact was made
11	Duration	Numeric, Related to Last Contact	Total time of last contact
12	Campaign	Numeric, Other Attributes	Number of contacts performed during this campaign with a particular client
13	Pdays	Numeric, Other Attributes	Number of days passed by after the client was contacted from a previous campaign
14	Previous	Numeric, Other Attributes	Number of contacts performed before this campaign for a particular client
15	Poutcome	Categorical, Other Attributes	Outcome of the precious marketing campaign
16	Emp.var.rate	Numeric, Socio Economic Attributes	Employment Variation Rate – quarterly indicator
17	Cons.price.idx	Numeric, Socio Economic Attributes	Consumer Price Index– monthly indicator
18	Cons.conf.idex	Numeric, Socio Economic Attributes	Consumer Confidence Index – monthly indicator
19	Euribor3m	Numeric, Socio Economic Attributes	Euribor 2-month rate – daily indicator

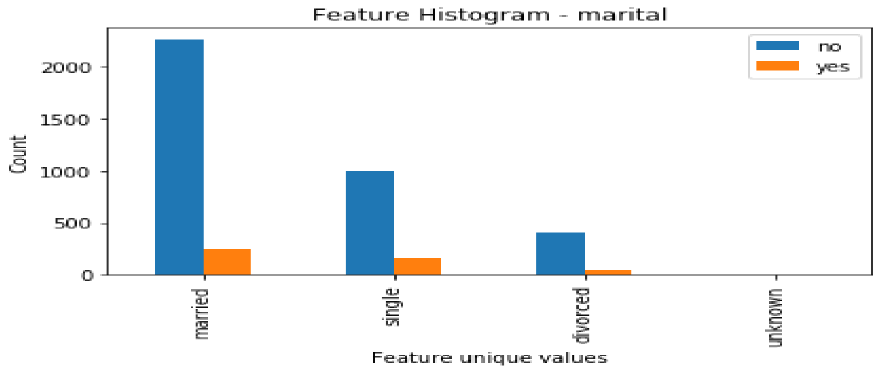
20	Nr.employed	Numeric, Socio Economic Attributes	Number of employees – quarterly indicator

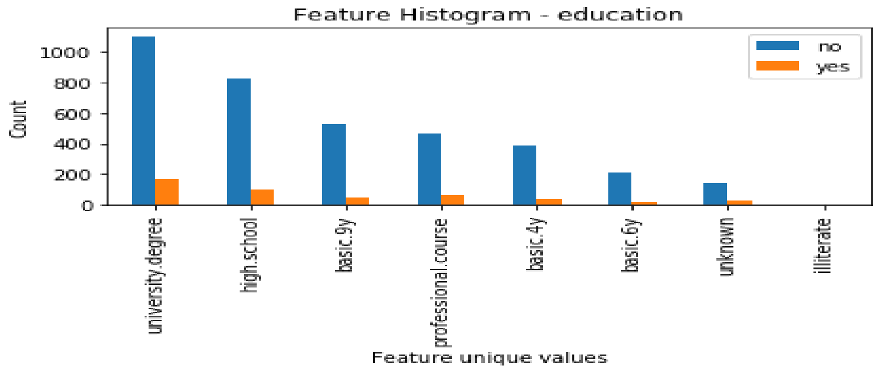
2.2. Understanding the features

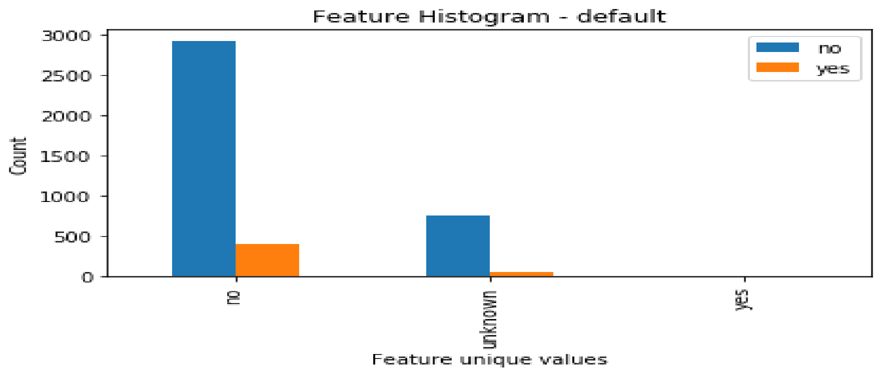
To proceed with the project, features were plotted on histograms in relation to the target class in terms of distribution to gain an understanding of the features.

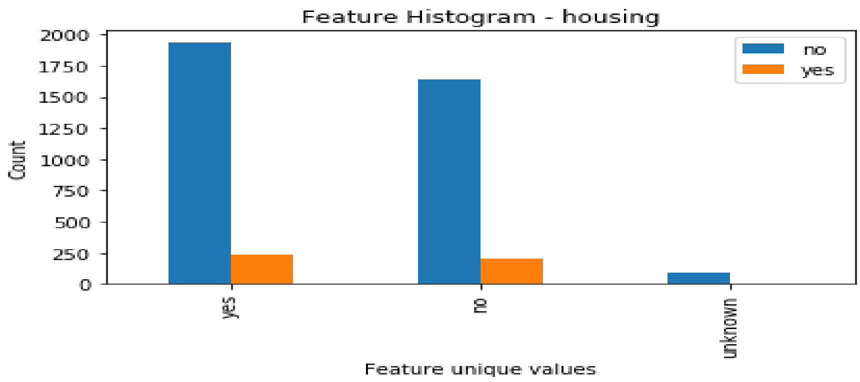


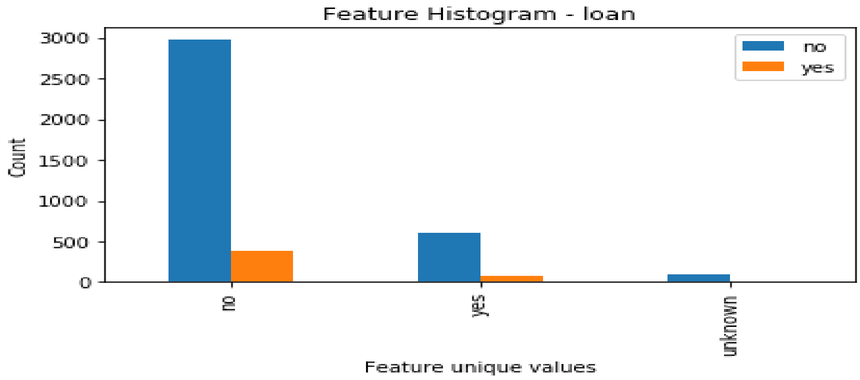


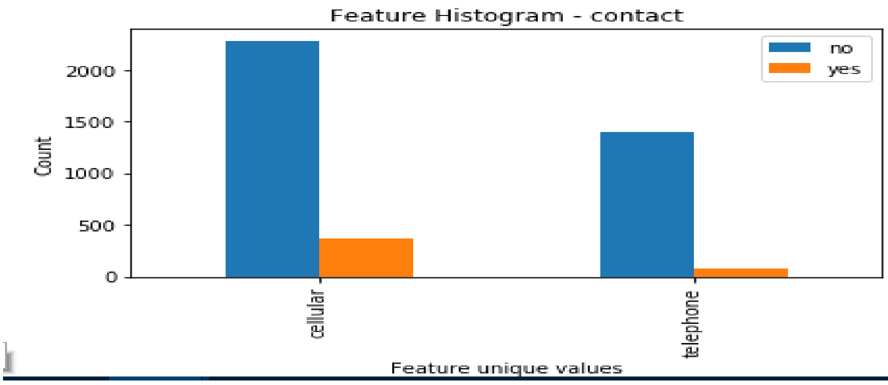

Feature Histogram - marital
Count
0
500
1000
1500
2000
no
yes
married
single
divorced
unknown
Feature unique values

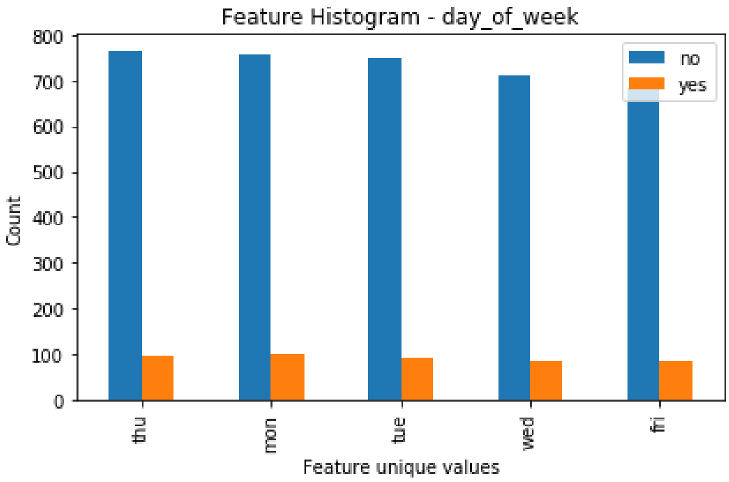

Feature Histogram - education
Count
0
200
400
600
800
1000
no
yes
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
unknown
illiterate
Feature unique values


Feature Histogram - default
Count
0
500
1000
1500
2000
2500
3000
no
yes
no
unknown
yes
Feature unique values


Feature Histogram - housing
Count
2000
1750
1500
1250
1000
750
500
250
0
no
yes
yes
no
unknown
Feature unique values


Feature Histogram - loan
Count
3000
2500
2000
1500
1000
500
0
no
yes
no
yes
unknown
Feature unique values


Feature Histogram - contact
Count
2000
1500
1000
500
0
no
yes
cellular
telephone
Feature unique values


Feature Histogram - day_of_week
Count
800
700
600
500
400
300
200
100
0
no
yes
thu
mon
tue
wed
fri
Feature unique values

Age is the only numerical feature and the rest are categorical features. This feature has outliers, which will skew results if not handled.

Some key observations from the Categorical Dataset are: categorical data cannot be used directly into a classifier. Categorical features need to be converted to meaningful numerical values for them to be used in a classifier. The second observation is the presence of unknown values in the dataset which will need to be handled.

3. Data Preparation

Data preparation is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions. Some datasets have values that are missing, invalid, or otherwise difficult for an algorithm to process. If data is missing, the algorithm cannot use it. If data is invalid, the algorithm produces less accurate or even misleading outcomes. Good data preparation produces clean and well-curated data which leads to more practical, accurate model outcomes. For exploring the data set, third party Python libraries are used to help us process the data so that it can be effectively used with scikit-learn's powerful algorithms.

Thus, for the purposes of the project the data had to be pre-processed. Several data pre-processing steps were done before classification and these are discussed the following sections.

3.1. Replacing unknown values

The unknown values were handled using the replace function provided by the pandas library. Replace function allows for values of the DataFrame to be replaced with other values dynamically as illustrated in Figure 5.

```
In [13]: data.head()
```

Out[13]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	y
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	no
1	39	services	single	high.school	no	no	no	telephone	may	fri	no
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	no
3	38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri	no
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	no

Figure 4: Unknown Values

```
In [14]: data["housing"].unique()
Out[14]: array(['yes', 'no', 'unknown'], dtype=object)

In [15]: data["housing"].replace(
         {
              'unknown':np.nan

         }, inplace = True)
```

Figure 5: Replacing Unknown Values

The unique() function was used to return unique values as a NumPy array. If the values returned contained 'unknown', the replace function was then used to replace the 'unknown' value in that column with 'NaN'.

```
In [29]: data.head()
Out[29]:
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	y
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	no
1	39	services	single	high.school	no	no	no	telephone	may	fri	no
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	no
3	38	services	married	basic.9y	no	NaN	NaN	telephone	jun	fri	no
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	no

Figure 6: Updated Data with NaN

3.2. Handling Missing Values

As shown in the histogram of features, there are many unknown values. These unknown values should be filled with some other feature values present in the training data. As discussed in section 3.1, the unknown values were replaced with NaNs to show that the absence of values in those cells. This was done to be able to use the fillna and interpolate functions provided by the Pandas library. Fillna() manages and let the user replace NaN values with some value of their own. As shown in Figure 7, the isnull().sum() revealed the totals of null values in the features.

```
In [32]: data.isnull().sum()

Out[32]: age               0
         job              39
         marital          11
         education       167
         default         803
         housing         105
         loan            105
         contact           0
         month             0
         day_of_week       0
         y                 0
         dtype: int64
```

Figure 7: Totals of null values in features

Using fillna with the method ffill the values that have NaN were replaced in the dataframe.

```
In [39]: #using fillna to replace the values in the dataframe that have NaN values, to make the model more efficient
         data = data.fillna(method="ffill", limit = 2)
         data
```

Figure 8: Replacing NaN values

Another way for filling NaN values is the function interpolate(). It uses various interpolation technique to fill the missing values rather than hard-coding the value. This will interpolate the values by getting a better guess for missing values by replacing with an intermediate value as shown in Figure 9

```
In [40]: #using the interpolate method. This will interpolate the values by getting a better guess for missing values by replaci
         data = data.interpolate()
         data.interpolate()
```

Figure 9: Interpolate Function

3.3. Encoding Categorical Features

Label encoding and one hot encoding were performed on the categorical features. Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Scikit-learn provides LabelEncoder library for encoding labels with a value between 0 and one less than the number of discrete classes. However, label encoding has limitations. Label encoding convert the data in machine readable form, but it assigns a unique number (starting from 0) to each class of data. This may lead to the generation of priority issue in training of data sets. A label with high value may be considered to have high priority than a label having lower value.

One hot encoding refers to splitting the column which contains numerical categorical data to many columns depending on the number of categories present in that column. Each column contains “0” or “1” corresponding to which column it has been placed.

Therefore, label encoding was just done for the target column and one hot encoding for the rest of the independent categorical features. Figure 10 and Figure 11 illustrate the dataset features before encoding and after label and one hot encoding was conducted.

Encoding categorical data

In [41]:
```
data.head()
```

Out[41]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	y
0	30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri	no
1	39	services	single	high.school	no	no	no	telephone	may	fri	no
2	25	services	married	high.school	no	yes	no	telephone	jun	wed	no
3	38	services	married	basic.9y	no	yes	no	telephone	jun	fri	no
4	47	admin.	married	university.degree	no	yes	no	cellular	nov	mon	no

Figure 10: Before Encoding

```
In [49]: le = preprocessing.LabelEncoder()
         data['y'] = le.fit_transform(data['y'])
         #one hot encoding on all the dataset

         df_encoded = pd.get_dummies(data=data, columns=oj_columns)
         df_encoded.head(15)
```

Out[49]:

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired
0	30	0	0	1	0	0	0	0
1	39	0	0	0	0	0	0	0
2	25	0	0	0	0	0	0	0
3	38	0	0	0	0	0	0	0
4	47	0	1	0	0	0	0	0
5	32	0	0	0	0	0	0	0
6	32	0	1	0	0	0	0	0
7	41	0	0	0	1	0	0	0
8	31	0	0	0	0	0	0	0
9	35	0	0	1	0	0	0	0
10	25	0	0	0	0	0	0	0
11	36	0	0	0	0	0	0	0
12	36	0	1	0	0	0	0	0
13	47	0	0	1	0	0	0	0
14	29	0	1	0	0	0	0	0

15 rows × 46 columns

Figure 11: After One Hot Encoding

As illustrated in Figure 11, unique feature values would be found, and they would make the new columns. The rows would be filled with binary values of 0 or 1 based on if that feature value is present or not for that row.

3.4. Thresholding

Thresholding values in columns by counts. A recurring problem with some of the columns such as education in the dataset are rare categories, for example illiterate. One hot encoding will create a new feature for every rare category and this will create many redundant parameters. I will deal with this by setting a count threshold. Categories that are counted more times than the threshold will be left as is, and the others will be labelled as 'rare'. The count threshold will be a hyper parameter for this model as it is hard to predict what will be the best value for it.

```
In [47]: instances = data.shape[0]
         threshold = instances*0.005
         print ('The minimum count threshold is: ' +str(threshold))
```

```
The minimum count threshold is: 20.595
```

```
In [48]: #applying the minimum threshold to all the categoricals values
         oj_columns = list(data.select_dtypes(include=['object']).columns)
         oj_columns.remove('y') #as this is the target column, it should not be one hot encoded
         data = data.apply(lambda x: x.mask(x.map(x.value_counts())<threshold, 'Rare')if x.name in oj_columns else x)
```

3.5. Descriptive Statistics

Descriptive statistics were generated to summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values. Since the data was numeric due to the encoding, the result's index included count, mean, std, min, max as well as lower, 50 and upper percentiles.

```
In [50]: df_encoded.describe()
Out[50]:
```

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	...	mo
count	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	...	411
mean	40.113620	0.109493	0.246905	0.216557	0.036174	0.027191	0.079145	0.040544	0.038844	0.096140	...	
std	10.313362	0.312294	0.431263	0.411948	0.186745	0.162660	0.269998	0.197255	0.193247	0.294819	...	
min	18.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	38.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
75%	47.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
max	88.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	

8 rows × 46 columns

Figure 12: Descriptive Statistics

Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe. The correlation of a variable with itself is 1.

```
In [51]: #correlations
         df_encoded.corr()
Out[51]:
```

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_servi
age	1.000000	0.060374	-0.103055	-0.038487	0.040608	0.092973	0.066536	0.411821	0.010700	-0.047
y	0.060374	1.000000	0.040832	-0.065438	-0.034620	-0.001258	-0.016400	0.077716	-0.018183	-0.022
job_admin.	-0.103055	0.040832	1.000000	-0.301039	-0.110927	-0.095728	-0.167864	-0.117703	-0.115108	-0.186
job_blue-collar	-0.038487	-0.065438	-0.301039	1.000000	-0.101855	-0.087899	-0.154135	-0.108077	-0.105694	-0.171
job_entrepreneur	0.040608	-0.034620	-0.110927	-0.101855	1.000000	-0.032389	-0.056796	-0.039824	-0.038946	-0.063
job_housemaid	0.092973	-0.001258	-0.095728	-0.087899	-0.032389	1.000000	-0.049014	-0.034368	-0.033610	-0.054
job_management	0.066536	-0.016400	-0.167864	-0.154135	-0.056796	-0.049014	1.000000	-0.060265	-0.058937	-0.095
job_retired	0.411821	0.077716	-0.117703	-0.108077	-0.039824	-0.034368	-0.060265	1.000000	-0.041325	-0.067
job_self-employed	0.010700	-0.018183	-0.115108	-0.105694	-0.038946	-0.033610	-0.058937	-0.041325	1.000000	-0.065
job_services	-0.047360	-0.022047	-0.186741	-0.171468	-0.063183	-0.054526	-0.095613	-0.067043	-0.065564	1.000

Figure 13: Correlations

3.6. Handling Outliers

Outliers can be a result of a mistake during data collection or it can be just an indication of variance in the data. Outliers may skew the results, and thus have to be handled to make the model more efficient. The age feature had several outliers as illustrated in the histograms in section 2.2.

In [53]:
```
low = 0.01
high = 0.99
df_encoded.quantile([low, high])
```

Out[53]:

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	...	month_mar	month_may	mo
0.01	24.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
0.99	68.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1.0	1.0	

2 rows × 46 columns

In [54]:
```
quantiles = df_encoded.quantile([low, high])
```

In [55]:
```
quantiles.age
```

Out[55]:
```
0.01    24.0
0.99    68.0
Name: age, dtype: float64
```

In [56]:
```
df_encoded.age = df_encoded.age.apply(lambda v: v if quantiles.age[low]< v <quantiles.age[high] else np.nan)
```

In [57]:
```
df_encoded.describe()
```

Out[57]:

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	...	mo
count	3976.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	4119.000000	...	411
mean	40.132797	0.109493	0.246905	0.216557	0.036174	0.027191	0.079145	0.040544	0.038844	0.096140	...	
std	9.397951	0.312294	0.431263	0.411948	0.186745	0.162660	0.269998	0.197255	0.193247	0.294819	...	
min	25.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	38.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
75%	47.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
max	67.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	

8 rows × 46 columns

Figure 14: Removing Outliers

Removing the outliers, the lower and upper quantiles were first set to 0.01 for low and 0.99 for high. The ages below 24 which was the lower quantile and above 68 which was the upper quantile were removed. This was done using the apply function and lambda as illustrated in Figure 14.

The ages were then shown in a box and whisker plot, so as to visual see the outliers and the new minimum and maximum age values. This is illustrated in the Figure 15.

```
In [61]: df_encoded.age.unique()
```

```
Out[61]: array([30., 39., 25., 38., 47., 32., 41., 31., 35., 36., 29., 27., 44.,
       46., 45., 50., 55., 40., 28., 34., 33., 51., 48., 56., 58., 60.,
       37., 52., 42., 49., 54., 59., 57., 43., 53., 26., 61., 67., 64.,
       63., 66., 62., 65.])
```

```
In [62]: plt.boxplot(df_encoded.age)
```

```
Out[62]: {'whiskers': [<matplotlib.lines.Line2D at 0x2055eb2c668>,
  <matplotlib.lines.Line2D at 0x2055eb2c9b0>],
 'caps': [<matplotlib.lines.Line2D at 0x2055eb2ccf8>,
  <matplotlib.lines.Line2D at 0x2055eb29080>],
 'boxes': [<matplotlib.lines.Line2D at 0x2055eb2c518>],
 'medians': [<matplotlib.lines.Line2D at 0x2055eb293c8>],
 'fliers': [<matplotlib.lines.Line2D at 0x2055eb29710>],
 'means': []}
```

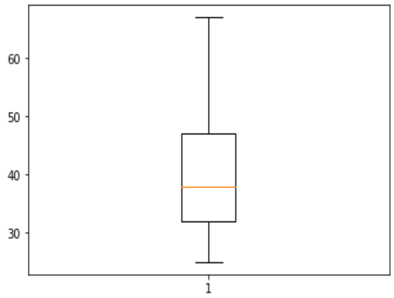


Figure 15: Box and Whisker Plot

3.7. Co-Variance

Covariance provides the measure of strength of correlation between two variable or more set of variables. Pandas dataframe.cov() is used to compute pairwise covariance of columns. This function was utilised to measure the co-variance of the target feature with the rest of the features as shown in Figure 16.

```
In [63]: df_encoded.cov()
```

Out[63]:

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_servic
age	88.321480	0.050471	-0.421433	-0.078524	0.088674	0.125230	0.185402	0.522265	0.025380	-0.0882
y	0.050471	0.097528	0.005499	-0.008419	-0.002019	-0.000064	-0.001383	0.004787	-0.001097	-0.0020
job_admin.	-0.421433	0.005499	0.185988	-0.053482	-0.008934	-0.006715	-0.019546	-0.010013	-0.009593	-0.0237
job_blue-collar	-0.078524	-0.008419	-0.053482	0.169702	-0.007836	-0.005890	-0.017144	-0.008782	-0.008414	-0.0208
job_entrepreneur	0.088674	-0.002019	-0.008934	-0.007836	0.034874	-0.000984	-0.002864	-0.001467	-0.001405	-0.0034
job_housemaid	0.125230	-0.000064	-0.006715	-0.005890	-0.000984	0.026458	-0.002153	-0.001103	-0.001056	-0.0026
job_management	0.185402	-0.001383	-0.019546	-0.017144	-0.002864	-0.002153	0.072899	-0.003210	-0.003075	-0.0076
job_retired	0.522265	0.004787	-0.010013	-0.008782	-0.001467	-0.001103	-0.003210	0.038909	-0.001575	-0.0038
job_self-employed	0.025380	-0.001097	-0.009593	-0.008414	-0.001405	-0.001056	-0.003075	-0.001575	0.037345	-0.0037
job_services	-0.088200	-0.002030	-0.023743	-0.020825	-0.003479	-0.002615	-0.007611	-0.003899	-0.003735	0.0869
job_student	-0.150835	0.002380	-0.005036	-0.004417	-0.000738	-0.000555	-0.001614	-0.000827	-0.000792	-0.0019
job_technician	-0.195412	0.000708	-0.042210	-0.037022	-0.006184	-0.004648	-0.013530	-0.006931	-0.006641	-0.0164
job_unemployed	-0.012547	0.001636	-0.006715	-0.005890	-0.000984	-0.000740	-0.002153	-0.001103	-0.001056	-0.0026
marital_divorced	0.447354	-0.001470	0.000580	-0.007532	0.001407	0.001413	0.001589	0.003117	-0.000098	0.0011
marital_married	1.199361	-0.005433	-0.024761	0.023399	0.004134	0.002335	0.008730	0.005107	-0.000897	0.0015
marital_single	-1.646716	0.006903	0.024181	-0.015867	-0.005541	-0.003748	-0.010319	-0.008225	0.000995	-0.0027
education_Rare	0.000470	-0.000027	-0.000060	-0.000053	-0.000009	-0.000007	-0.000019	0.000233	-0.000009	-0.0000
education_basic.4y	0.618319	-0.002338	-0.023653	0.032248	0.000550	0.009998	-0.005204	0.010045	-0.001189	-0.0057

Figure 16: Co-Variances

3.8. Normalization

Normalization of the dataset is scaling the features between 0 and 1. This was tried only on the continuous features of the dataset. I used sklearn.preprocessing.MinMaxScaler for this process. This function calculates the min and max from the dataset features and scales them between 0 and 1. The figure below illustrates how normalization was done for the age feature.

In [66]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [67]:
```
scaler = MinMaxScaler()
```

In [68]:
```
#normalising all ages
columns = ["age"]
df1[columns]=scaler.fit_transform(df1[columns])
df1.head()
```

Out[68]:

	age	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	...	month_mar	month_may	mo
0	0.119048	0	0	1	0	0	0	0	0	0	...	0	1	
1	0.333333	0	0	0	0	0	0	0	0	1	...	0	1	
2	0.000000	0	0	0	0	0	0	0	0	1	...	0	0	
3	0.309524	0	0	0	0	0	0	0	0	1	...	0	0	
4	0.523810	0	1	0	0	0	0	0	0	0	...	0	0	

5 rows × 46 columns

Figure 17: Normalization using MinMaxScaler

3.9. Train-Test Split

The training set contains a known output and the model learns on this data in order to be generalized to other data later on. The test dataset (or subset) in order to test our model's prediction on this subset. This is illustrated in Figure 18.

#Spliting dependent and independent features

In [69]:
```
#split the dataset to X and Y for model fit
x_orig = df1.fillna(df.median(axis=0))
y_orig = x_orig['y'] #sets the target column
x_orig = x_orig.drop('y', axis=1)#this removes the target column from the training set
```

Train-test split

Training data set can be used specifically for our model building.

The above cells divide data into feature set and target set. The "x_orig " set consists of predictor variables. It consists of data from columns in the dataset df1. The "y_orig" set consists of the outcome variable. It consists of data in the 'y' column.

The cell below will split data into training and test set. X_train, y_train are training data and X_test, y_test belongs to the test dataset. The parameter test_size is given value 0.3; it means test sets will be 30% of whole dataset and training dataset's size will be 70% of the entire dataset. Random_state variable is a pseudo-random number generator state used for random sampling.

In [72]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(np.nan_to_num(x_orig), y_orig, test_size=0.3, random_state=42)
```

Figure 18: Train/Test Split

4. Modelling

The problem posed was a binary classification problem. Thus, the classifiers used to implement the machine learning model were; decision tree classifier and the linear regression classifier.

4.1. Decision Tree Classifier

Decision Tree can be used as a classifier model.

Decision Tree Training

Fit Decision tree algorithm on training data, predicting labels for validation dataset and printing the accuracy of the model using various parameters.

DecisionTreeClassifier()- This is the classifier function for DecisionTree. It is the main function for implementing the algorithms.
Optimizing Decision Tree Performance - criterion parameter set to gini which allows us to use the different-different attribute selection measure. Max_depth is the maximum depth of the tree. In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. Maximum depth of the tree can be used as a control variable for pre-pruning.

In [73]:
```
# Train basic Classifer

clf = DecisionTreeClassifier(criterion = "gini", random_state=0, max_depth=3, min_samples_leaf=5)
clf.fit(X_train, y_train)
```

Out[73]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [74]:
```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100,
 max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)
```

...

Predictions

In [75]:
```
y_pred = clf.predict(X_test)
y_pred
```

Out[75]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [76]:
```
y_pred_en = clf_entropy.predict(X_test)
y_pred_en
```

Out[76]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Figure 19: Decision Tree Classifier training and predicting

Calculating Accuracy Score

```
Classification Accuracy metric will be used. The function accuracy_score() will be used to print accuracy of Decision
Tree algorithm. By accuracy, we mean the ratio of the correctly predicted data points to all the predicted data
points. Accuracy as a metric helps to understand the effectiveness of our algorithm. It takes 4 parameters.
```

```
In [77]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,y_pred)*100

Out[77]: 89.64401294498381

In [78]: accuracy_score(y_test,y_pred_en)*100

Out[78]: 89.80582524271846
```

Figure 20: Accuracy of Decision Tree Algorithm

From the results illustrated in Figure 20, the decision tree model achieved an accuracy score of 89.644%.

4.2. Logistic Regression

Logistic Regression is a supervised machine learning algorithm used in binary classification. I used sklearn.linear_model.LogisticRegression for this Logistic Regression Implementation process and called the fit function with the features and the labels as arguments. Figure 21 illustrates the logistic regression implemented.

Logistic Regression

```
In [81]: from sklearn.linear_model import LogisticRegression
         logreg = LogisticRegression()
         logreg.fit(X_train, y_train)

         C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will b
         e changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
           FutureWarning)

Out[81]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='warn',
                   tol=0.0001, verbose=0, warm_start=False)

In [82]: #make class predictions for the testing set
         y_pred_class = logreg.predict(X_test)

         Classfication accuracy: percetange of correct predictions

In [86]: from sklearn import metrics
         metrics.accuracy_score(y_test, y_pred_class)*100

Out[86]: 89.23948220064725
```

Figure 21: Logistic Regression Algorithm Training, Testing and Accuracy Metric

From the results in Figure 21, the logistic regression model achieved an accuracy score of 89.24%.

```
In [89]: confusion_matrix(y_test, y_pred)

Out[89]: array([[1088,   17],
                [ 111,   20]], dtype=int64)
```

Figure 22: Confusion Matrix

5. Evaluation of Classification Model

```
# Evaluation of the classfication model
Confusion Matrix is a table that describes the perfromance of a classification model.
Compute confusion matrix to evaluate the accuracy of a classification.
```

```
In [79]: tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
         print('tn:',tn, 'fp:',fp, 'fn:',fn, 'tp:',tp)

         tn: 1088 fp: 17 fn: 111 tp: 20

In [85]: tn, fp, fn, tp = confusion_matrix(y_test, y_pred_en).ravel()
         print('tn:',tn, 'fp:',fp, 'fn:',fn, 'tp:',tp)

         tn: 1087 fp: 18 fn: 108 tp: 23

In [89]: confusion_matrix(y_test, y_pred)

Out[89]: array([[1088,   17],
                [ 111,   20]], dtype=int64)

In [88]: from sklearn.metrics import classification_report
         print (classification_report(y_test, y_pred))

                       precision    recall  f1-score   support

                    0       0.91      0.98      0.94      1105
                    1       0.54      0.15      0.24       131

            micro avg       0.90      0.90      0.90      1236
            macro avg       0.72      0.57      0.59      1236
         weighted avg       0.87      0.90      0.87      1236
```

From the confusion matrix it can be concluded that:

- True positive: 20 (predicted a positive result and it was positive)
- True negative: 1088 (predicted a negative result and it was negative)
- False positive: 17 (predicted a positive result and it was negative)
- False negative: 111 (predicted a negative result and it was positive).

The classification_report function builds a text report showing the main classification metrics.

```
In [88]: from sklearn.metrics import classification_report
         print (classification_report(y_test, y_pred))

                      precision    recall  f1-score   support

                   0       0.91      0.98      0.94      1105
                   1       0.54      0.15      0.24       131

           micro avg       0.90      0.90      0.90      1236
           macro avg       0.72      0.57      0.59      1236
        weighted avg       0.87      0.90      0.87      1236


In [ ]:
```

Figure 23: Classification Report

The classification report shown in Figure 23 show the sklearn metrics scores for the model. Precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples. F-1 measure can be interpreted as a weighted harmonic mean of the precision and recall. F-1 measure reaches its best value at 1 and its worst score at 0. Macro avg calculates the mean of the binary metrics, giving equal weight to each class. Weighted average accounts for class imbalance by computing the average of binary metrics in which each class's score is weighted by its presence in the true data sample. Micro gives each sample-class pair an equal contribution to the overall metric.