# Project Report - “Project Opportunity”

Michael Impey - E10569412 - October 2021

This is my submission to UCD, to demonstrate my leanings for the Specialist Certificate in Data Analytics Essentials.

The associated files are saved to GitHub on the following GitHub URL :

<https://github.com/UCDPA-E10569412/Michael_2021_Specialist_Cert_Data.A/upload>

Table Of Contents

[Project Report - “Project Opportunity” 1](#_Toc31023)

[Abstract 2](#_Toc14668)

[Introduction 2](#_Toc31017)

[Data-set 3](#_Toc4324)

[Implementation Process 4](#_Toc31728)

[Step.0 - Project\_Problem 7](#_Toc15232)

[Step.1 - Data\_Gathering 10](#_Toc29798)

[Step.2 - Basic\_Data\_Cleaning 15](#_Toc13789)

[Step.3 - Baseline\_Model\_Testing 21](#_Toc29746)

[Step.4 - Optimise\_Data\_Cleaning 27](#_Toc7035)

[Step.5 - Tune\_Model\_and Select 44](#_Toc32631)

[Step.6 - HyperTune\_model 46](#_Toc1005)

[Step.7 - Optimise\_Validation\_Data\_Cleaning 52](#_Toc27512)

[Step.8 - Model\_Prediction 55](#_Toc896)

[Results 67](#_Toc1818)

[Insights 68](#_Toc12840)

[References 69](#_Toc1179)

[Appendix 1 - API - Twitter 70](#_Toc29403)

[Appendix 2 - Regex 73](#_Toc5636)

[Appendix 3 - Deep Learning 79](#_Toc18111)

[Appendix 4 - Seven points and page numbers 80](#_Toc31258)

# Abstract

The project attempt to build a supervised machine leaning classifier model that will predict if a loan is likely to be repaid or defaulted on. The data file was downloaded from Kaggle.com using the following link.

<https://www.kaggle.com/ajay1735/hmeq-data>

In order to full fill the requirement of the course report, I have attached the examples and project code snippets that were used. Where the project did not specifically request something required for grading, I have submitted an example and application in appendix. These items include Twitter API and expanded example of Regex I used for my Titanic project.

# Introduction

The Inspiration for this project is to predict clients who default on their loans. The story was the consumer credit department of a bank wants to automate the decision making process for approval of home equity lines of credit. The model was built to use a model. The model will be a supervised classifier model, as it a binary target decision.

The decision is to determine if the loan be likely to be repaid or defaulted. I have wrote the report to bring the reader along the various steps involved. I have created a flow chart which is printed in the appendix. The project starts by reviewing the data-set and examining the project problem. Then the reader will be brought through the various steps of building the data data set and model.

Finally, I will discuss some of the insight gained when reviewing the data and model.

# Data-set

The data file is from [https://www.kaggle.com/ajay1735/hmeq-data](https://www.kaggle.com/ajay1735/hmeq-data.). The data-set (HMEQ) contains loan performance information for 5,960 recent home equity loans. The target is a binary variable indicating whether an applicant eventually repaid, defaulted. The outcome of defaulted occurred in 1,189 cases (20%). For each applicant, 12 input feature variables were recorded.

I chose this data-set as it looked interesting to me as I am currently applying for a mortgage. I was initially interested in the Titanic data-set but this is heavily covered by other data analyst reports and I liked the thought of a similar challenge but in a different context. I was interested to see can a machine learning model actually be used to predict an outcome and could I configure it to do it successfully. I would like to follow up on this method to see can I use machine learning, to predict if a stock price is more probably to go up or down (Classification) as opposed to a target price (Linear Regression).

The legend for the original data is as follows;

* BAD: 1 = client defaulted on loan 0 = loan repaid
* LOAN = Amount of the loan request
* MORTDUE = Amount due on existing mortgage
* VALUE = Value of current property
* REASON = DebtCon = debt consolidation / HomeImp = home improvement
* JOB = Six occupational categories
* YOJ = Years at present job
* DEROG = Number of major derogatory reports
* DELINQ = Number of delinquent credit lines
* CLAGE = Age of oldest trade line in months
* NINQ = Number of recent credit lines
* CLNO = Number of credit lines
* DEBTINC = Debt-to-income ratio

I changed the legend for the original data which is shown above and I added an “ID” column. I modified the original data-frame slightly adding in a “ID” column and splitting into two file so that I could demonstrate a merge function which is asked for in the project report. This detail and more will be covered below in the implementation process.

# Implementation Process

Below is an overview of the project steps taken. I will include some detail on what the step entails. After this initial description, I will detail each step on its own. The outstanding elements required in report but not covered in the project are included in the appendices.

You will notice that I like to load the data-fame needed for the step at the start of a program and save the outputs at the end of the step. I did this as it was very cumbersome and time consuming to run the whole project as one whole program and it also allowed me to test the impact of changes on one step at a time before moving onto the next step. It also facilitated going back on earlier steps, isolating and modifying.

I used functions where possible and where they made sense. While this code was for this project I also wanted to make it re-usable and to be the foundation for other project I intend to work on. For example step 3 could be used by any supervised classifier project where the user wants to run data through a number of classifier machine learning models.

I also created a pause() function, so that I could pause the program as it cycled through the various steps. This gave me time to review the program output as some times it can loop through very quick. It is implemented as just a wait for input command to proceed. Simple and effective.

def pause():

'''used to pause program to view output'''

input('===> Press Return to Continue Program ?')

The plan when writing this code was to build a program that builds on the previous step. I have attached a chart in the appendices of how the steps interact in more detain. Submitted with this report is a more comprehensive detail of what each steps contain. The project starts with perhaps the most important step. The problem statement, “what problem are we trying to solve?” then we are essentially building / cleaning data and testing models until we are at a stage where we are happy enough with our model. Then its time to test on validation data and make a prediction. I was constantly gaining insights as I went through the project. From the visual and descriptive outputs.

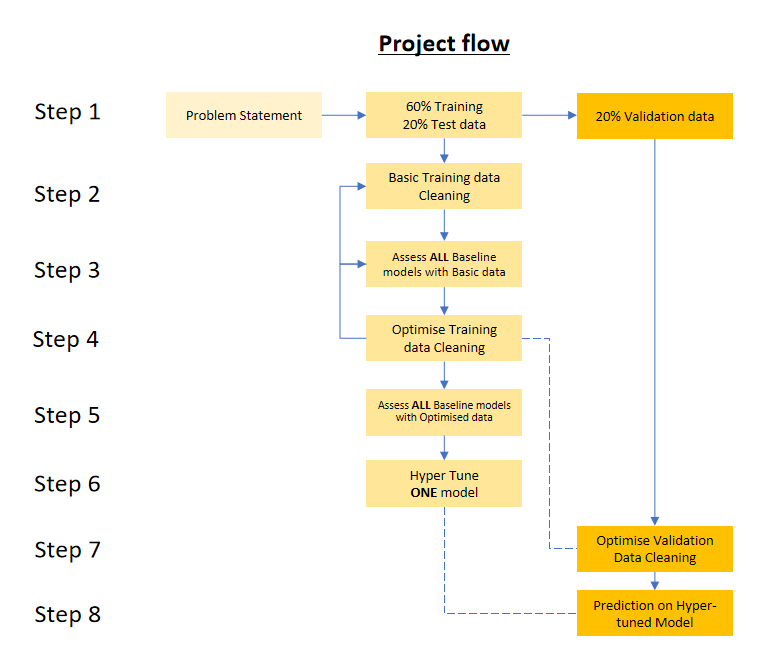
**Project “Steps”.**

|  |  |  |
| --- | --- | --- |
| Step | Title | Overview |
| 0 | Project\_Problem | Preliminary step. What is the problem the project is trying to answer and what data do we need answer it. |
| 1 | Data\_Gathering | Import CSV data, Merge data-frames. I randomly split the data to create a Train,Test Data-set and Validation Data-set |
| 2 | Basic\_Data\_Cleaning | Perform EDA, basic data cleaning so that we can create basic data to run through a number of models to get initial model performance results on the Training data-set |
| 3 | Baseline\_Model\_Testing | Run a number of models on the basic cleaned data |
| 4 | Optimise\_Data\_Cleaning | Optimize data cleaning so that we can run through the model again to get final performance results |
| 5 | Tune\_Model\_and Select | Run a number of models on the Optimised cleaned data. From this trial I selected a model. |
| 6 | HyperTune\_model | On the selected model, Hyper-tune model and assess performance. |
| 7 | Validation\_Data\_Cleaning | Perform basic and Optimised data cleaning on the Validation data set |
| 8 | Model\_Predictions | Perform prediction using the Validation training set on the hyper-tuned model and asses against a number of classifier evaluation metrics |

**Project flow chart:**

This is an overview of the project flow. There is more detail in the file attached called;

**“Project Steps Template.xls”**



## Step.0 - Project\_Problem

This step was very important to the projects success. It involved understanding what problem the project trying to answer. The project is trying to predict the clients that would default on their loans? To do this I had to hypothesize what information would be required and compare to what is available.

If I was stating from scratch, some of the features that I would expect to need would include the following. Those highlighted in yellow below are the items provided in the Kaggle data-frame. I added my hypothesis regrading each feature.

1. How much do you need? Possibly the higher the loan the higher the risk of not repaying. LOAN = Amount of the loan request
2. How long do you want the loan for? Possibly the longer the duration of the loan the higher the risk of default through Job loss, death.
3. Interest repayments? What is cost of loan? Higher the interest the higher overall cost of the loan, this may contribute to a loan default
4. Do they work? Obviously not working is a significant risk
5. Is employment permanence or temporary employment? In permanent employment you could assume the risk of default is lower
6. How long in current employment? I would imagine the longer in current job the more reliable the job is and means of paying loan and possibly the more reliable the person in the role is YOJ = Years at present job
7. Current job category? I would imagine certain job have higher established pays and security of paying - Temporary worker versus surgeon JOB = Six occupational categories
8. Existing debt - mortgage? I would guess that the higher the current debt the applicant has the more risk they expose the loan provider to. MORTDUE = Amount due on existing mortgage
9. Existing credit card age - line of credit? I would assume a person with an established line of credit for lengthy period is more likely to pay back the loan.

CLAGE = Age of oldest trade line in months

1. How many existing lines of credit? I would imagine having many lines of credit to be negative regarding repaying a loan back. Not sure you can say this but I would imagine the higher the number of lines of credit the higher the indebtedness

NINQ = Number of recent credit lines & CLNO = Number of credit lines

1. How much do they earn? I would imagine this is very important as this would dictate the loan paying power if indebtedness was known
2. Do they have a good credit score? This would be important in indicating are they a good customer or had issues in the past. DEROG = Number of major derogatory reports
3. Savings history? I would imagine if you have a track record for saving you will have a higher potential and discipline for paying back a loan
4. Existing assets? Important for securing loan as collateral VALUE = Value of current property
5. Have they dependents / re-occurring costs? Similar to current debts. Can the income exceed the existing costs to cover loan and interest repayments.
6. Reason for loan? Not sure actual reason impact repayment but perhaps if its for an asset that can be sold in the event of default, this may reduce loan risk.

REASON = DebtCon = debt consolidation / HomeImp = home improvement

These are items I did not think of;

* 1. DELINQ = Number of delinquent credit lines - have they paid back loans in the past, I am not to clear what this means.
  2. CLAGE = Age of oldest trade line in months - are they struggling to pay and need credit or have an existing mortgage.
  3. DEBTINC = Debt-to-income ratio - I would imagine this is important as it should gauge there ability to pay. Lower debt to income being preferred.

So when looking at the data provided for the project I notice two issues. 1) a lot of the above features I would have thought important are not available for modeling. For example the is no income feature, and 2). the quality of the information available was not complete. For example the “”Debt\_to\_Income” is missing in 20% of the instances and there is no data feature relating to income that would allow me back engineer a figure.

For evaluation, there are two approaches I will take. Initially, the models are going to be evaluated using Classification Accuracy which is the ratio of number of correct predictions to the total number of input samples. There is a risk with this approach, if the I have a high number of samples belonging to one class of prediction, the prediction will not be reliable.

The code from Scikit-learn is;

>>> from sklearn.metrics import accuracy\_score

>>> y\_pred = [0, 2, 1, 3]

>>> y\_true = [0, 1, 2, 3]

>>> accuracy\_score(y\_true, y\_pred)

Fractional accuracy = 0.5

>>> accuracy\_score(y\_true, y\_pred, normalize=False)

Number of correct predictions = 2

Confusion Matrix gives you a lot of information about how well your model does. When performing classification predictions, there's four types of outcomes.

The code from Scikit-learn is;

>>> from sklearn.metrics import confusion\_matrix

>>> y\_true = [0, 1, 0, 1]

>>> y\_pred = [0, 1, 0, 0]

>>> confusion\_matrix(y\_true, y\_pred)

confusion\_matrix\_results = confusion\_matrix(y\_test, y\_pred)

print("True negatives - correctly classified as not Target: ", confusion\_matrix\_results[0][0])

print("False negatives - wrongly classified as not Target: ",confusion\_matrix\_results[0][1])

print("False positives - wrongly classified as Target: ", confusion\_matrix\_results[1][0])

print("True positives - correctly classified as Target: " ,confusion\_matrix\_results[1][1]

## Step.1 - Data\_Gathering

This step is where I import the original Kaggle data, reviewed it and prepared it for the following steps.

This step has the following actions;

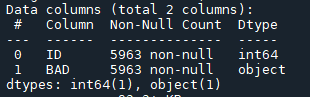
1. Import file and Create data-frame
2. Explore and tidy data-frame
3. Create Train and Test data-frame
4. Save to file

Initially, I saved the csv data as **“loan\_bad\_Orignal.csv”** to my desk top. This was then modified to create two files, **'loan\_bad\_ID\_Target.csv’** and **'loan\_bad\_ID\_Features.csv'** .

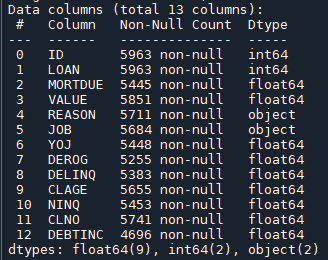
I then imported and merge both to make a project data-frame. They were merged on the ‘ID’ column. I created functions to complete the task, as I plan to use this program as a template for future projects. This will be a theme through out the project.

I modified the original data set so that I could demonstrate MERGE function of two data-frames using the ‘ID’. The following is the head from both of these data-frames I used to complete the merge;

**“loan\_bad\_ID\_Target.csv”** consisted of the following (5963 Rows, 2 Columns).



**“loan\_bad\_ID\_Features.csv”** consisted of the following (5963 Rows, 13 Columns).



I used the following code to complete this. You can see I created functions so that the code is reusable. The filename is set and sent to the function. This is a little overkill but for me, it helped with repeatability and standardizing my code.

Code:

#===============================================================================

#Functions

#===============================================================================

def import\_file(filename):

'''Import data - import and set up data frames'''

file = pd.read\_csv(filename)

print("\n"+str(filename)+" in imported file:\n", file.info())

return file

def create\_project\_file(A, B, Merge\_on):

'''create project file from imported files'''

file = pd.merge(A, B, on=merge\_on)

print("\nMerged file info():\n", file.info())

return file

#===============================================================================

#Start of program - import data

#===============================================================================

#import file for project

filename1 = 'loan\_bad\_ID\_Target.csv'

df\_a = import\_file(filename1)

#import file for project

filename2 = 'loan\_bad\_ID\_Features.csv'

df\_b = import\_file(filename2)

#create project file and merge on 'ID'

merge\_on = "ID"

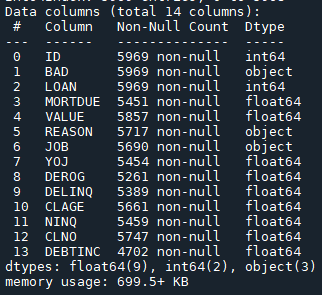
df\_merge = create\_project\_file(df\_a, df\_b, merge\_on)

#make a copy in case we ruin orignal dataframe

data = df\_merge.copy()

Merging the both of them produced the following data-frame info;

(5963 Rows, 14 Columns)



I then modified the legend so that the column titles where clear to me. I did not find the original column titles that descriptive and so I made these changes.

#rename columns to more understanable titles

data.rename(columns={'BAD': 'BAD\_LOAN',

'LOAN': 'AMOUNT\_REQUESTED',

'MORTDUE':'EXIST\_MORTG\_DEBT',

'VALUE': 'EXIST\_PROPERTY\_VALUE',

'REASON':'LOAN\_REASON',

'YOJ': 'EMPLOYED\_YEARS',

'DEROG': 'DEROG\_REPORTS',

'DELINQ': 'DELINQ\_CR\_LINES',

'CLAGE': 'CR\_LINES\_AGE(MTS)',

'NINQ': 'NO\_OF\_RECENT\_CR\_LINES',

'CLNO': 'NO\_OF\_CR\_LINES',

'DEBTINC': 'DEBT\_TO\_INCOME'}, inplace=True)

I then split this data-frame into (80%) Train, Test and (20%) Validation data sets. Row selection was random. The 20% will be used later in the project to make the predictions on a clean validation data-frame to avoid over-fitting.

The most important thing you can do, to properly evaluate your model, is not train the model on the entire data set. The train/test split I used was 70% for training and 30% for testing. I will also be using K-Fold Cross validation where appropriate. I usually used KFold CV when I was trying to determine the model performance metrics.

I evaluated my model as I was building it so I could find that best parameters. I did not use the validation data-set for evaluation as I might of ended up selecting parameters that perform best on the test data but these maybe not parameters that generalize best. So to evaluate the model I create a third subset of the data known as the validation set. From the web it would appear that a typical train/test/validation split would be to use 60% of the data for training, 20% of the data for testing, and 20% of the data for validation.

I shuffled the data before making these splits so that each split has an accurate representation of the orignal data-set. (msk = np.random.rand(len(data)) < 0.8)

<https://www.jeremyjordan.me/evaluating-a-machine-learning-model/>

The following code shows how that was achieved and saved. I had an issue saving and reloading the files. The data-frame was loading with a new index, to resolved this I use ‘index=False ‘ when saving;

Code:

*#*==========================================================

#Split dataFrame to Train and test and save

#==========================================================

#create a test and traing dataframe that has not been cleaned

#use the random function to select random rows and assign to a mask

msk = np.random.rand(len(data)) < 0.8

#save the test dataframe - not in mask

test = data[~msk]

filename1 = 'S1\_test\_Loan\_Basic\_Data\_Cleaning.csv'

test.to\_csv(filename1, index=False)

print("\n>>Saved test data.shape: ", test.shape);print(test.info())

#save the train dataframe in mask

train = data[msk]

filename2 = 'S1\_train\_Loan\_Basic\_Data\_Cleaning.csv'

train.to\_csv(filename2, index=False)

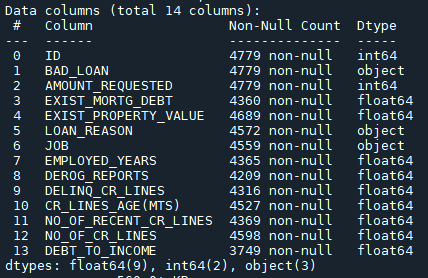
print("\n>>Saved train data.shape: ", train.shape);print(train.info())

#Load df from file - used to see how the saved file loads back as I had an issue with index column

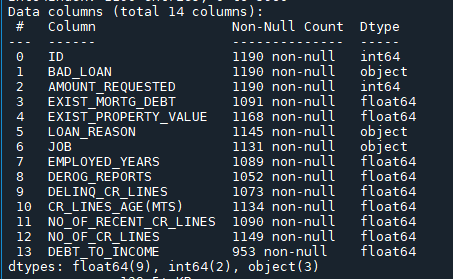
df = pd.read\_csv(filename2)

print("\n<<Loaded dataframe.shape: ", df.shape);print(df.info())#(4826, 14)

**“S1\_Train\_Test\_Loan\_Basic\_Data\_Cleaning.csv”** contains;



**“S1\_Validation\_Loan\_Basic\_Data\_Cleaning.csv”** contains;



Step.2 - Basic\_Data\_Cleaning

This step used the data merged from the previous step. The overall aim of this step is to perform basic Exploratory Data Analysis (EDA). I performed basic data cleaning with minimal cleaning so that I can use this data quickly in the next step and get performance results for a number of classifier Machine Learning models.

This step has the following actions;

1. Import data and Perform basic Exploratory Data Analysis (EDA)
2. Perform Basic data cleaning
3. Save data-frame to use in Optimize Data Clean
4. Impute data for baseline models and Save data-frame for next step

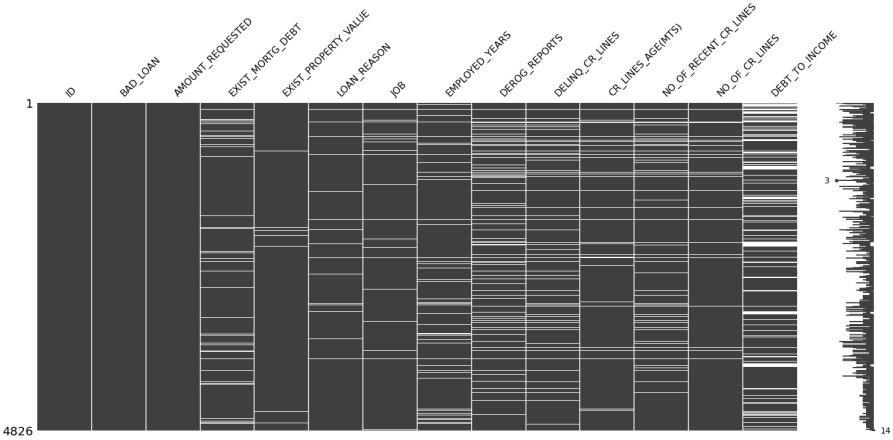
Action 1 - Perform basic Exploratory Data Analysis

In this action I conducted these three checks;

1. Column dtype check: What were the dtype for the individual columns? I need to know where they correct for the information contained as this would also affect the machine learning models as for example some cannot handle categorical data. I also printed out the numeric and non-numeric column titles.
2. EDA Descriptive: This action consisted of displaying the shape of the data-frame, info on the data-frame and I completed a for loop where I iterate through the data-frame columns and printed out a description of the column content. I used describe(include=all) so I could get details on counts and max min etc. The (include=all) setting is very useful as it give a fuller description of the column contents.
3. EDA Visual: This function displayed a chart of the missing data and a chart of any possible correlations. The missingness chart is an excellent way to see how much missing data we have and where? The heat map is an excellent first step in examining any possible correlations in the data.

**Missingness chart:**

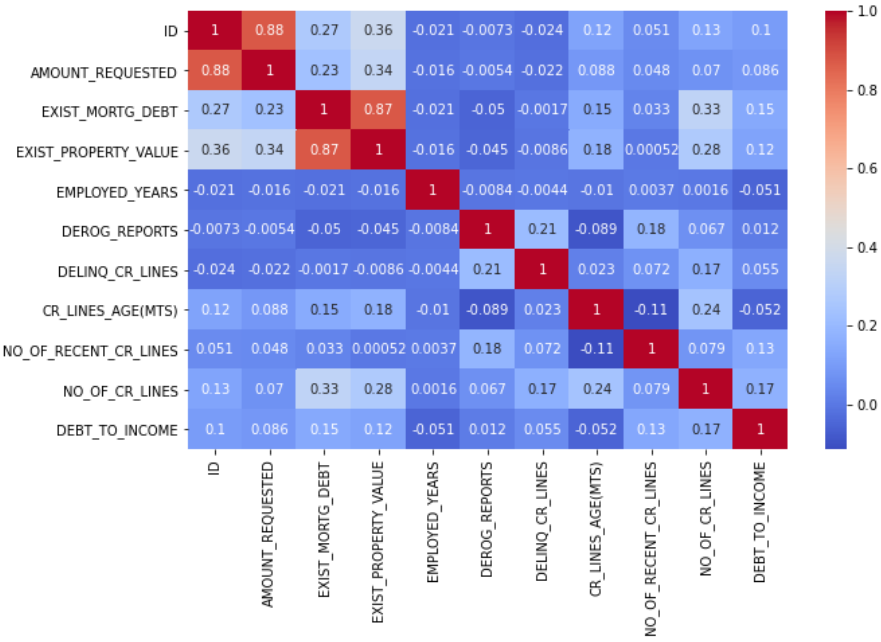
The missingness chart is excellent for quickly assessing which columns are really affected by missing data. It appears there are 11 columns missing data. We can see that “Debt\_to\_Income” appears to be the worst affected column. With “Derog\_Reports” appearing to be the next worst affected. If you look carefully it appears some rows are missing a number of columns of data.



**Heat map:**

The heat map is an excellent way of visually inspecting correlations or relationships between the features. Taking the the extremes we can see that (excluding “ID”)

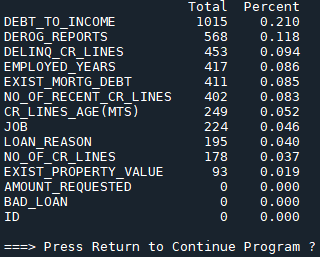
“Amount\_requested” to “Existing\_Mortg\_Debt” and to “Existingp\_roperty\_value” would appear to have some correlation which is not unsurprising as perhaps the existing debt is for a loan for the existing property. “No\_Of\_CR\_Lines” appear correlated to “CR\_Line\_age” which is not surprising as perhaps its suggests that perhaps the reason they have more is that they are with the bank longer? In general, I only work off the extreme correlations as they are more reliable.



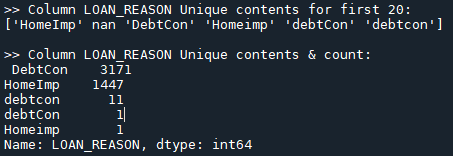
Action 2 - Perform Basic data cleaning

In this action I conducted the following checks;

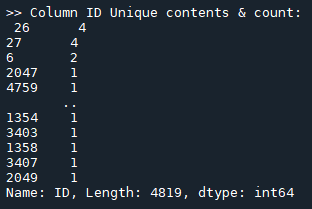
1. Descriptive review of the missing data from the data-frame. I ran a function to table all the columns and their content that was missing. The detail was combined and recorded in my notes for correction. The table looked like this;



1. Describe the unique content in the columns. I used a function to iterate through the data-frame columns looking at the unique data in each column. This was especially important as I can see very quickly possible text errors for example miss-spelt “DebtCon” or “debtCon” and missing values “nan” as shown below;



and duplicated rows for example 4 counts of ‘ID’ = 26, 27 and 2 counts of ‘ID’ = 6.



1. Using the information gathered so far I was able to drop rows and standardize the content of a number of columns very quickly. The purpose was to do the most obvious basic data clean to allow me run and score the models. The following code was used to complete this;

#==========================================================

#baseline - basic cleaning of data - defining consistent feature contents

#==========================================================

##is there missing data?

print(draw\_missing\_data\_table(df));pause()

##What is the unique data?

dataframe\_unique\_check(df)

##Drop all duplicate rows based

print("\nNumber of rows before drop\_duplicated: ",len(df))

df.drop\_duplicates(subset=None, keep='first', inplace=True, ignore\_index=False)

print("\nNumber of rows After drop\_duplicated: ", len(df));pause()

##Lets re-examine the offending unique data of the dataframe after the duplicate row drops and the drop ID column?

check\_columns = df[['ID', 'BAD\_LOAN','LOAN\_REASON','JOB']]

dataframe\_unique\_check(check\_columns)

# BAD\_LOAN: Expected TWO unique values but got FIVE. I will change to column data to Repaid, Defaulted.

# Then I will change to 1:Default and 0:Repaid as its affecting categorising

df['BAD\_LOAN'].replace(['paid','Repaid'],0,inplace=True)

df['BAD\_LOAN'].replace([ 'default', 'Dfault','Default'],1,inplace=True)

# LOAN\_REASON: Expected TWO unique values but got SIX. will change to DebtCon, HomeImp

df['LOAN\_REASON'].replace(['homeImp','Homeimp'],'HomeImp',inplace=True)

df['LOAN\_REASON'].replace([ 'debtCon' , 'debtcon'],'DebtCon',inplace=True)

#Going to use this oppourtunity to impute a value other than leave empty

df['LOAN\_REASON'].fillna('Other', inplace = True)

# Expected SIX unique values but got SEVEN. Will investigate empty features and call Other

# replacing na values in 'JOB' with 'Other'

df['JOB'].fillna('Other', inplace = True)

##Lets rexamine after replace function has been used

check\_columns = df[['ID', 'BAD\_LOAN','LOAN\_REASON','JOB']]

dataframe\_unique\_check(check\_columns)

Action 3 - Save data-frame to use when Optimize Data Clean

Now I saved the data-frame that would later be used when I wanted to optimize the data. I saved this data-frame before imputing as this was going to be my import data-frame into the optimize data clean step and I did not want the next impute action to affect this. In the optimize data step I intend to take a closer look at the following imputations and optimize them. To state once again, my aim here is to create a clean data-frame so I can assess the baseline models performance scores. These scores are then compared and becomes my baseline score. This data-frame was saved as **'S2\_Loan\_Basic\_Data\_Cleaning.csv.**

Action 4 - Impute data for baseline models and Save data-frame for next step

For this action I created a simple function that would cycle through the data-frame columns and where there was a missing values, insert a multiple of 10 by the max value of the column. I chose 10 times the max value as I wanted any impute to negatively affect the models performance or make no difference. In the Optimize data step I would correct this.The columns where now all numeric so I did not need to worry about non-numeric errors from the compiler. The loop was a simple for loop that iterated through the data-frame columns, I checked for missing values before and after the change was completed.

The output of the imputation can be seen below;

column is BAD\_LOAN

Filled column[BAD\_LOAN] with 10

column is AMOUNT\_REQUESTED

Filled column[AMOUNT\_REQUESTED] with 892000

column is EXIST\_MORTG\_DEBT

Filled column[EXIST\_MORTG\_DEBT] with 3995500.0

column is EXIST\_PROPERTY\_VALUE

Filled column[EXIST\_PROPERTY\_VALUE] with 8559090.0

column is EMPLOYED\_YEARS

Filled column[EMPLOYED\_YEARS] with 99990.0

column is DEROG\_REPORTS

Filled column[DEROG\_REPORTS] with 100.0

column is DELINQ\_CR\_LINES

Filled column[DELINQ\_CR\_LINES] with 150.0

column is CR\_LINES\_AGE(MTS)

Filled column[CR\_LINES\_AGE(MTS)] with 11682.33561

column is NO\_OF\_RECENT\_CR\_LINES

Filled column[NO\_OF\_RECENT\_CR\_LINES] with 170.0

column is NO\_OF\_CR\_LINES

Filled column[NO\_OF\_CR\_LINES] with 710.0

column is DEBT\_TO\_INCOME

Filled column[DEBT\_TO\_INCOME] with 2033.121487

The code for this is shown below;

#==========================================================

#baseline - basic cleaning of data - Impute data for baseline model testing

#==========================================================

##Correct the missing data - first review

print(draw\_missing\_data\_table(df));pause()

##Impute columns missing data - lets use a basic impute of the max value (by 10) in the column

for col in df.columns:

print("column is "+str(col))

n = df[col].max()

n = n \* 10

df[col].fillna(n, inplace=True)

print("Filled column["+str(col)+"] with "+str(n))

##Correct the missing data - review changes

print(draw\_missing\_data\_table(df));pause()

This data-frame which is now ‘Basically Cleaned’ was then saved to **'S2\_Loan\_Basic\_Data\_for\_Baseline\_Models.csv'** for use in the next step where I would apply this data and assess the models performance.

## Step.3 - Baseline\_Model\_Testing

This step used the data-frame created in the last step. The purpose of this step is to run the data-frame through a number of Supervised Classifier Machine Learning Models and assess there performance on basic cleaned data.

This step has the following actions;

1. Import data and transform the categorical variables.
2. Create a array of Features values ‘X’ and Target array of values called ‘y’.
3. Normalize the data-frame?
4. Perform K-Fold cross validation on a number of models

Action 1 - Transform the categorical variables

After importing, I transformed the categorical features to numeric variables. This is the first of the pre-processing activities. To do this I passed the data-frame to a function I created. This returned a new data-frame of numeric values. One concern I have with this is when I look at feature importance, you can see that the “JOB” features are lower, as their total has been split across a number of columns. My fear is that the impact of the “JOB” feature will have less importance on the models performance. Qualitatively this was not the case as I ran the model with out the “JOB” feature and there was minimal impact.

The data-frame Column for “JOB” was;

|  |
| --- |
| JOB |
| Mgr |
| Other |
| Office |
| Self |
| ProfExe |

I use the following code to achieve Categorical Variables and passed it the data-frame to transform;

def transform\_categorical\_variables(dataframe):

''' Transform categorical variables into dummy variables - - known as one-hot encoding of the data.

This process takes categorical variables, such as days of the week

and converts it to a numerical representation without an arbitrary ordering.'''

dataframe = pd.get\_dummies(dataframe, drop\_first=True) # To avoid dummy trap

return dataframe

Columns transform are now;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| JOB\_Office | JOB\_Other | JOB\_ProfExe | JOB\_Sales | JOB\_Self |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |

*Side note : Dummy Variable Trap: When the number of dummy variables created is equal to the number of values the categorical value can take on. This leads to multi-collinearity, which causes incorrect calculations of regression coefficients and p-values. <https://www.statology.org/dummy-variable-trap/>*

Action 2 - Create Target and Feature variables

In this part I created the Target and Feature arrays. There are many different options for completing this. The most effective I found allows me to name a Target column and send it as an argument to the function. I like this approach in my code as it allows me easily re-use my code but also it does not matter where the Target column is the data-frame column headers. Also, It returns the X and y arrays. This is important as the Column header needs to be stripped as the machine learning model cannot accept the string title, they need numerical values.

def create\_X\_y\_datasets(df,target\_column\_name):

'''create features and target datasets'''

X = df[df.loc[:, df.columns != target\_column\_name].columns]

y = df[target\_column\_name]

return X, y

# Create datasets for model

target\_column\_name = 'BAD\_LOAN'

X, y = create\_X\_y\_datasets(df, target\_column\_name)

Action 3 - Normalize the data-frame

In this part I would normalize the data. This consisted of setting the max and min of the column to 0 and 1 respectively. I ran the data with and without Normalizing as I read some models do not need the data normalized.

def scale\_data\_normalisation(X):

'''pre-processing - Normalisation'''

scaler = MinMaxScaler()

scaled = scaler.fit\_transform(X)

return scaled

#rescale X between 0 - 1

X = scale\_data\_normalisation(X)

Action 4 - Perform K-Fold Cross Validation on a number of models

This is the work horse of this step. I built it from an idea I found on towardsdatascience.com. The idea was to send the X and y data arrays and perform KFold cross validation against a number of classifier models. The function tests all the models for a specific KFold value, score them using cross\_val\_score and display there accuracy on the data. It would also check there accuracy against the know outputs.

<https://towardsdatascience.com/cross-validation-and-hyperparameter-tuning-how-to-optimise-your-machine-learning-model-13f005af9d7d>

This was then loaded to a data-frame I created and saved to file **"ML3\_Loans\_Models\_Results\_on\_Basic\_Data.csv"**. The idea for this steps was to get baseline data on the various models, for their standard model parameters. In Step 5 of my process I would be repeating this test and need baseline results. I am looking for an improvement in the models performance as it will be tested on Optimised\_data\_cleaning.

def Classifier\_models\_test(df\_model\_values, a, b):

'''Test data on a number of different classifier algorithims, using KFold CV and save performance data'''

# get the list of models to consider

models = get\_models()

# define test conditions

Kfold\_number = range(a,b,1)

for CV\_val in Kfold\_number:

#https://www.askpython.com/python/examples/k-fold-cross-validation

kf = KFold(n\_splits=CV\_val, shuffle=True, random\_state=42)

# evaluate each model

for model in models:

print("\nKfold\_number = ", CV\_val)

#Implementing cross validation and get y\_p

#https://www.bitdegree.org/learn/train-test-split

import sklearn.model\_selection as model\_selection

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, train\_size=0.7, random\_state=CV\_val)

#Fit model and predict on training data

model.fit(X\_train,y\_train)

y\_pred = model.predict(X\_test)

##Test 1 - Accruacy\_score

#Implement accuracy\_score() function

model\_accuracy\_score= accuracy\_score(y\_test, y\_pred)

print("\nAccuracy\_score() is: ", round(model\_accuracy\_score, 3))

##Test 2 - Confusion matrix

confusion\_matrix\_results = confusion\_matrix(y\_test, y\_pred)

print("True negatives - correctly classified as not Target: ", confusion\_matrix\_results[0][0])

print("False negatives - wrongly classified as not Target: ",confusion\_matrix\_results[0][1])

print("False positives - wrongly classified as Target: ", confusion\_matrix\_results[1][0])

print("True positives - correctly classified as Target: " ,confusion\_matrix\_results[1][1])

confusion\_matric\_accuracy = (confusion\_matrix\_results[0][0]+confusion\_matrix\_results[1][1])/len(y\_pred)

#just want to make sure program stops if these couts are dirrenent as it mean my accuracy will not be correct

assert len(y\_pred)==(confusion\_matrix\_results[0][0]+confusion\_matrix\_results[0][1]+confusion\_matrix\_results[1][0]+confusion\_matrix\_results[1][1])

print("Confusion Matric - Accuracy: " ,confusion\_matric\_accuracy)

##Test 3 - Coss\_Val\_Score

# evaluate model using each test condition on cross\_val\_score()

#https://scikit-learn.org/stable/modules/cross\_validation.html

scores = cross\_val\_score(model,X,y,scoring='accuracy', cv=kf, n\_jobs=None)

cv\_mean = mean(scores)

# check for invalid results

if isnan(cv\_mean):

continue

# Model performances

model\_name = type(model).\_\_name\_\_

print(str(model\_name)+' Cross Val Score - Accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))

#Append data to dataframe to record results

df\_model\_values = df\_model\_values.append({'CV':CV\_val,'Model':str(model\_name),

'CVS\_Accuracy':round((np.mean(scores)),3),

'CVS\_STD':round((np.std(scores)),3),

'Accuracy\_Score':round(model\_accuracy\_score,3),

'C\_M\_Accuracy':round(confusion\_matric\_accuracy,2),

'True\_Neg':confusion\_matrix\_results[0][0],

'False\_Neg':confusion\_matrix\_results[0][1],

'False\_Pos':confusion\_matrix\_results[1][0],

'True\_Pos':confusion\_matrix\_results[1][1]},ignore\_index = True)

#Sort the values

df\_model\_values.sort\_values(by=['CVS\_Accuracy'], axis=0, ascending=False,inplace=True, kind='quicksort',na\_position='last',ignore\_index=False, key=None)

#save the dafatframe to file

df\_model\_values.to\_csv("ML3\_Loans\_Models\_Results\_on\_Basic\_Data.csv")#use this to see what the data looks like after lateststep

return df\_model\_values

The models I had to choose from where as follows. However I had an issue with my PC running Linear SVC model.

def get\_models():

models = list()

models.append(LogisticRegression())

models.append(RidgeClassifier())

models.append(SGDClassifier())#

models.append(PassiveAggressiveClassifier())

models.append(KNeighborsClassifier())

models.append(DecisionTreeClassifier())

models.append(ExtraTreeClassifier())

##models.append(LinearSVC())# gives low reading but gives fault

models.append(SVC())

models.append(GaussianNB())

models.append(AdaBoostClassifier())

models.append(BaggingClassifier())

models.append(RandomForestClassifier())

models.append(ExtraTreesClassifier())

models.append(GaussianProcessClassifier())

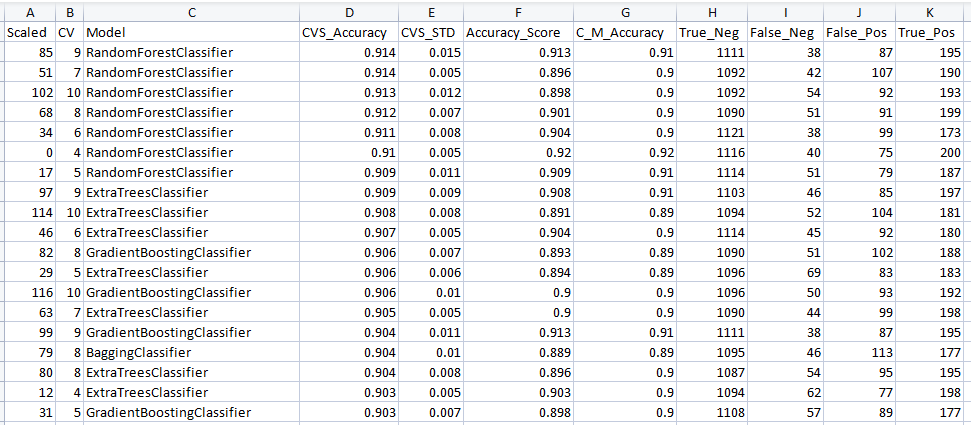
models.append(GradientBoostingClassifier())

models.append(LinearDiscriminantAnalysis())

models.append(QuadraticDiscriminantAnalysis())

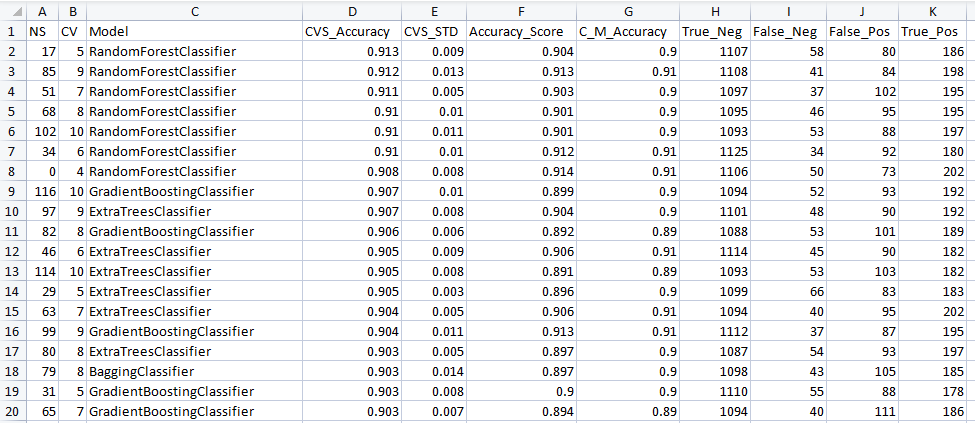
return models

The performance for the top classifiers was as follows and the full detail can be seen in the file “ML3\_Loans\_Models\_Results\_on\_Basic\_Data.csv” . We can see that the Random Classifier perform the best at this point. Results **with** Normalization:



We can see that the Random Classifier perform the best at this point.

Results **without** Normalization:



From the data above we can see that the RandomForestClassifier performed best. It did this with or without Normalization. This is interesting as it confirms what I read, that this model does not require scaling.

Step.4 - Optimise\_Data\_Cleaning

This step used the data from step 2, before the imputed values was completed. The purpose of this step was to try optimize the data further before trying all the classifiers models again. I expected to have an improved model performance.

This step has the following actions;

1. Import data and Perform basic Exploratory Data Analysis (EDA)
2. Perform Basic data cleaning
3. Perform data-frame tidying
4. Optimize data cleaning (including Box plot and Histogram)
5. Scatter plot of single Feature to Target
6. Feature engineering
7. Checked for Multi-collinearity in features
8. Identify most important features

Action 1 - Action 2

These action where covered in step 2 - Basic\_Data\_Cleaning and data was imported in with basic cleaning. Duplicates etc were removed but no imputing for missing values.

Action 3 - Perform data-frame tidying

In this action I dropped the ‘ID’ column and removed rows with many empty cells. I am aware it is nearly a “cardinal sin” to delete data but I found that a row with up to 8 of 9 columns missing would not affect model performance. It appeared to affect performance by 0.05% for the Random Forest model. I ran the model with and without the drop in rows with the data cleaning below. However, this improvement is insignificant as some of standard deviation errors for the models amounted to this amount.

I used this code to complete this. Basically this deletes row up to a number ‘n’ which I set before running the program.

#==========================================================

# Removing rows with many empty features

#==========================================================

n = 8#1#8 #we are allowing rows with up to 7 empty cells

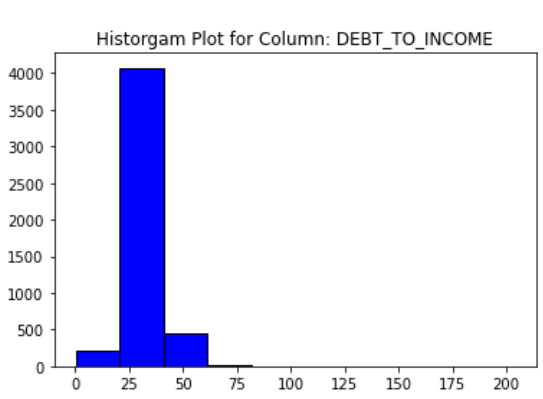
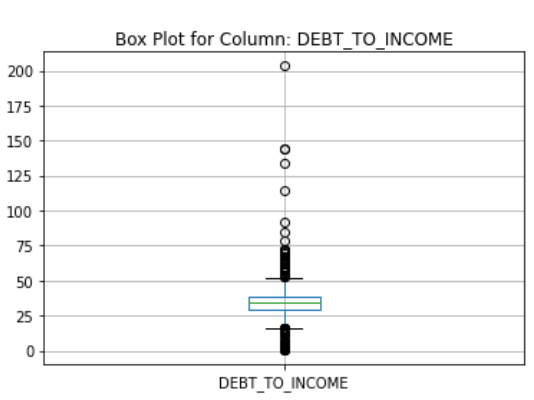
df = df[df.isnull().sum(axis=1) < n]

Action 4 - Optimize data cleaning (including Box plot and Histogram)

In this section I optimized the data based on box or histogram chart. I also measure what % of the data was missing. I will discuss separately below how I imputed missing data.

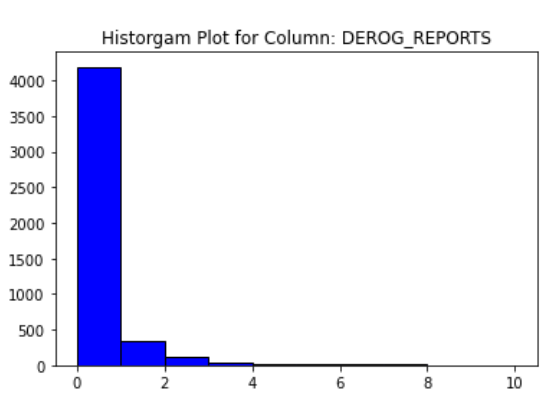
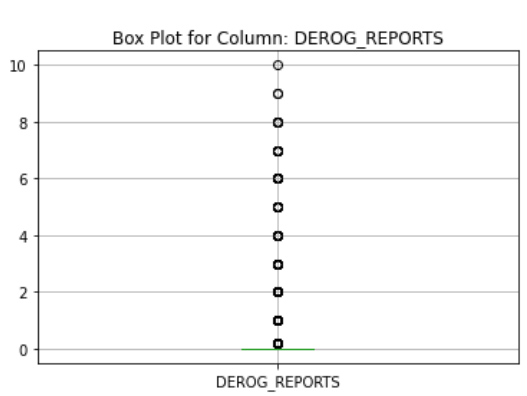
“DEBT\_TO\_INCOME” Total Percent missing 21%

In this chart we can see some outliers but with out an income I was not able to correct or access if it needed to be corrected. I used the KNN\_Impute to resolve the missing data.

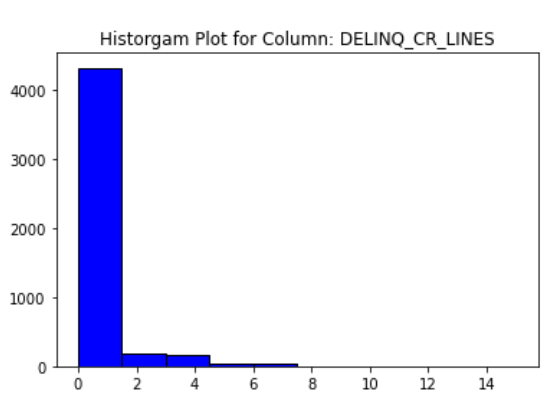
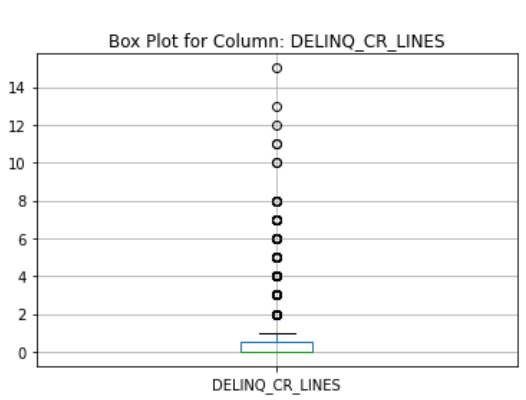
“DEROG\_REPORTS” Total Percent missing 12%

I could not say with any certainty what this should be. The figure has to be taken as correct and I did not want to impute an artificially high or low value as this may affect the models performance. I used the mean to resolve the missing data.

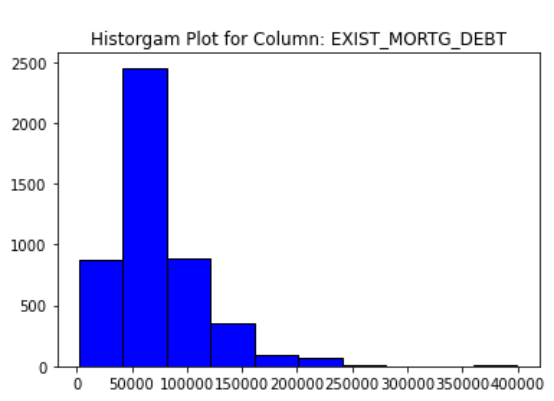
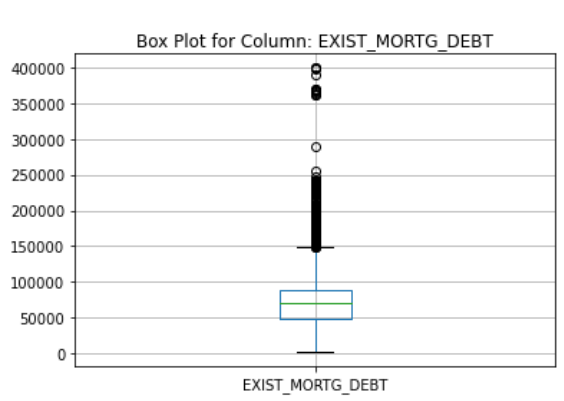
“DELINQ\_CR\_LINES” Total Percent missing 10%

I was not sure what this actually meant. I could not find this on the web. The closet I could find was credit lines. If this is what this is, it would seem a very high number for a single person. But I was not in a position to challenge these figures so I used the mean to resolve the missing data.

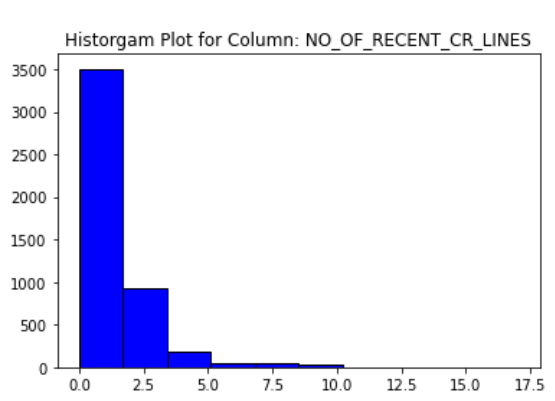
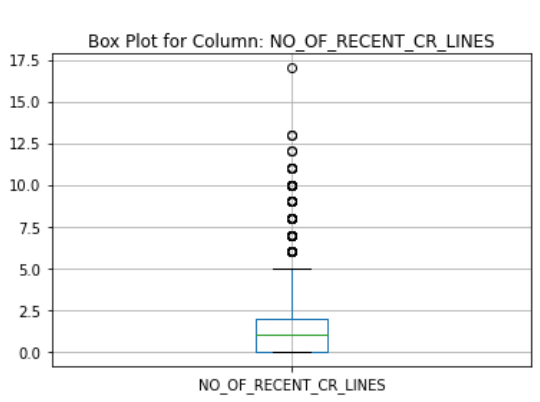
“EXIST\_MORTG\_DEBT” Total Percent missing 10%

This did not seem unreasonable and to be honest, I would need to know the standard house price to have an opinion. For example in Ireland this mean would seem quite low. And the high mortgage debt identified in the box plot may be for a person with multiple mortgages I.e. 2nd or 3rd home? I used the mean to resolve the missing data.

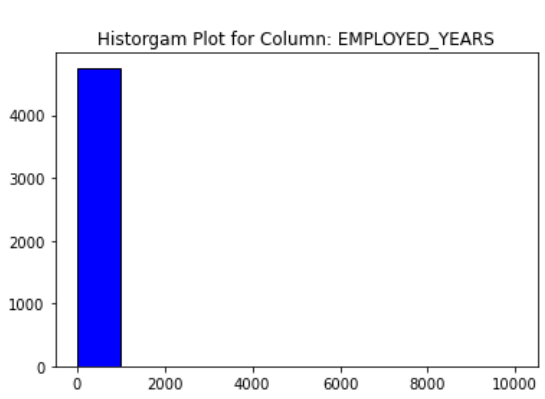
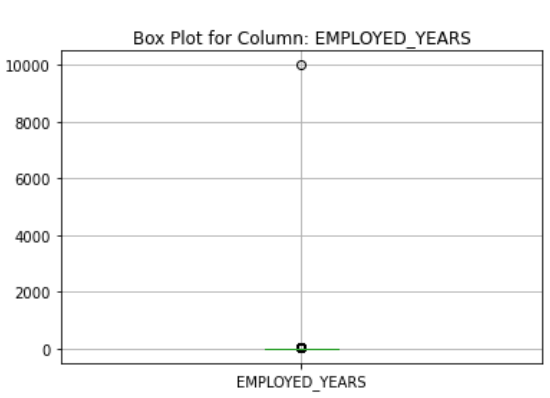
“NO\_OF\_RECENT\_CR\_LINES” Total Percent missing 8%

We did have one extreme outlier but perhaps this is correct and I did not make any correct. I used the mean to resolve the missing data.

“EMPLOYED\_YEARS” Total Percent missing 8%

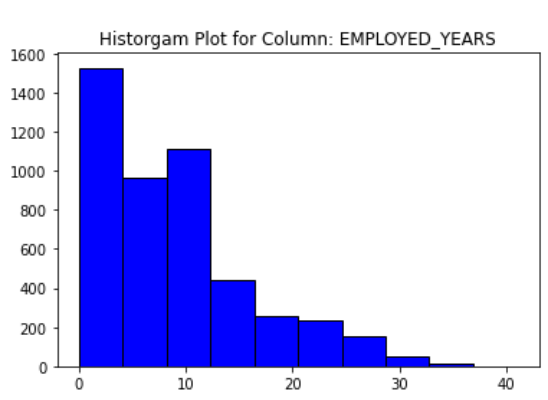
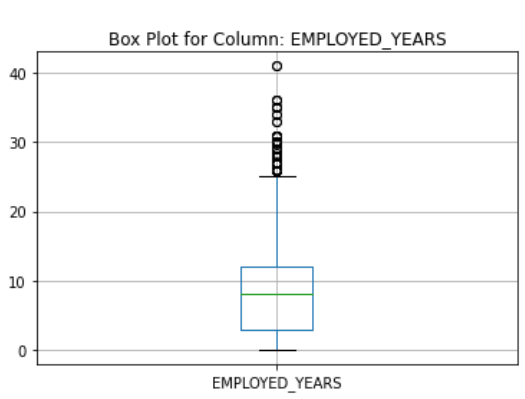
For Column 'EMPLOYED\_YEARS’ The box plot showed an obvious outlier in the data where the employed years was 9999. In theory the max working age would be 40 years but the data does have a 41 year employed. However the box plot shows clearly this to be a 10000 years. I used the mean to resolve the missing data. Its extremely interesting the impact that outlier had on the histogram.

#replacing outlier with mean of column

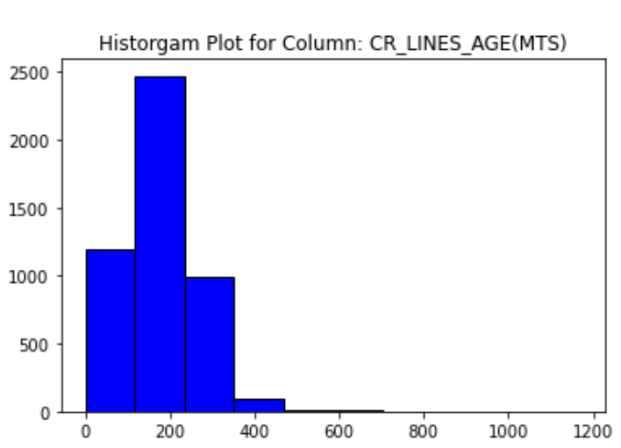
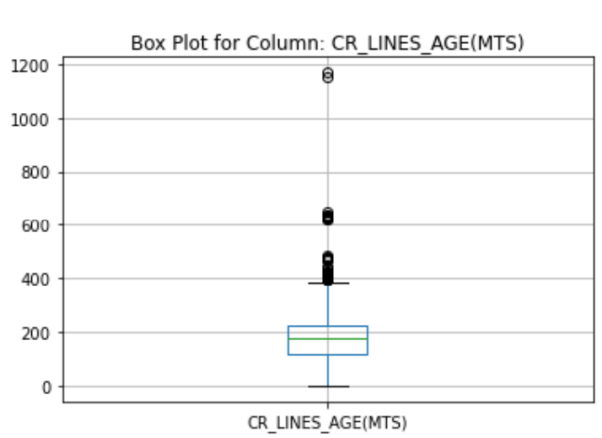
EMPLOYED\_YEARS = round((df['EMPLOYED\_YEARS'].mean()),1)

df['EMPLOYED\_YEARS'].replace(9999,EMPLOYED\_YEARS,inplace=True)

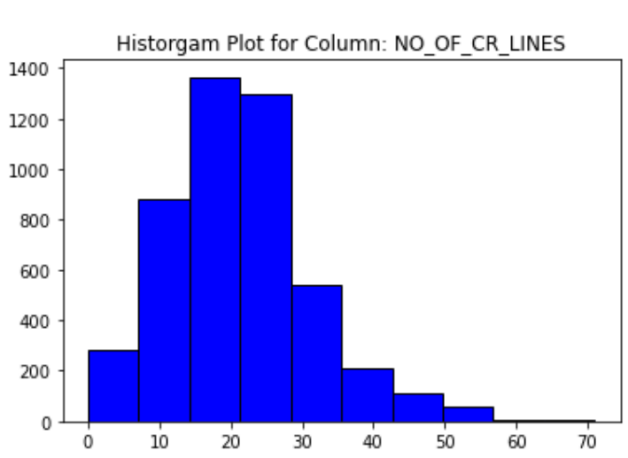
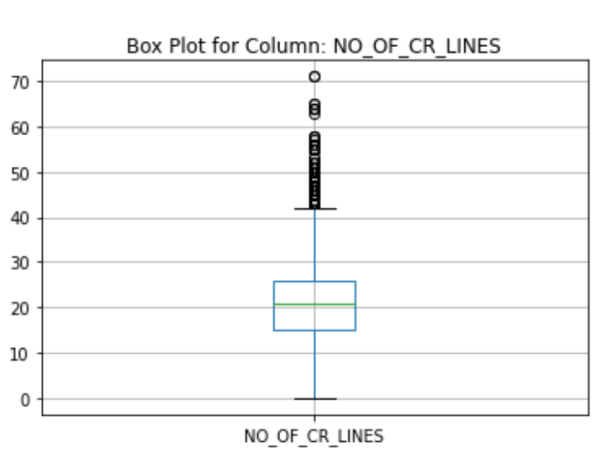
“CR\_LINES\_AGE(MTS)” Total Percent missing 5%

There appears to be something unusual regarding this outlier. The problem I have is if I change it, what do I change it to? I was not sure what this feature actually meant. I imputed the mean of the column for the missing values.

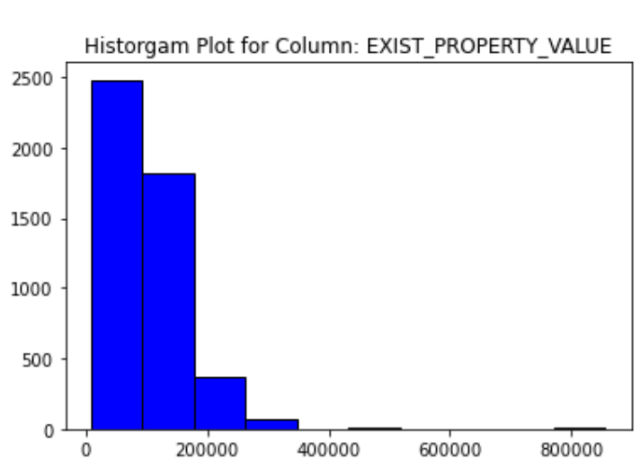
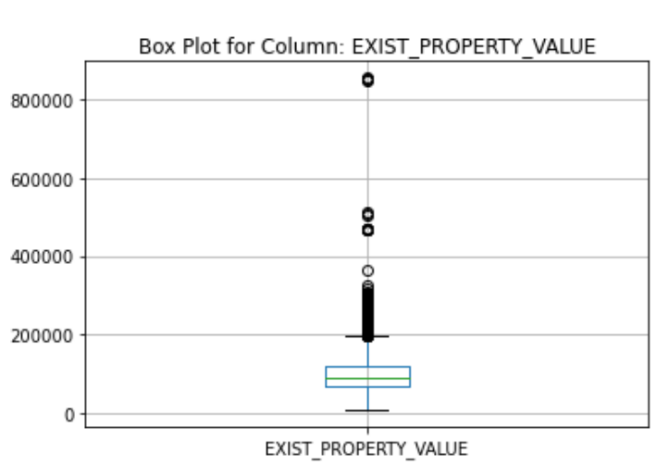
“NO\_OF\_CR\_LINES” Total Percent missing 4%

I used the mean to resolve the missing data. This was because I am not sure what the original column description actually meant, so I was not in a position to challenge these figures. The missing data was relatively small.

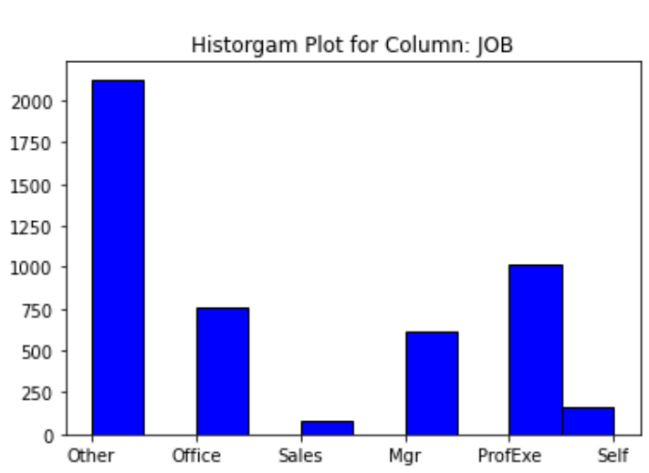
“EXIST\_PROPERTY\_VALUE” Total Percent missing 2%

I used the mean to resolve the missing data. This was because I was not in a position to challenge these figures. The missing data was relatively small.

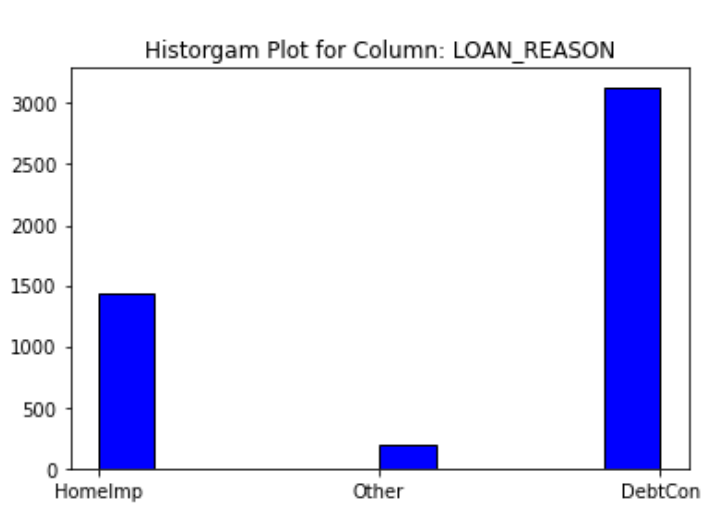
“JOB” Total Percent missing 0%

I replace the “nan” with “Other” so there was no empty features.



“LOAN\_REASON” Total Percent missing 0%

I replace the “nan” with “Other” so there was no empty features



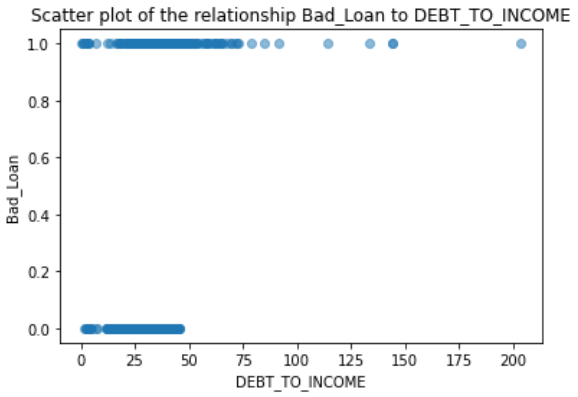
Action 5 - Categorical Scatter plot of single Feature to Target “BAD\_LOAN”

In this section these are the insights I gained by comparing the relationship of the feature to “BAD\_LOAN”. This was completed after the missing cell where imputed with the mean. “DEBT\_TO\_INCOME” feature was imputed with KNN-Impute.

<https://www.westga.edu/academics/research/vrc/assets/docs/scatterplots_and_correlation_notes.pdf>

BAD\_LOAN: 1 = client defaulted on loan 0 = loan repaid

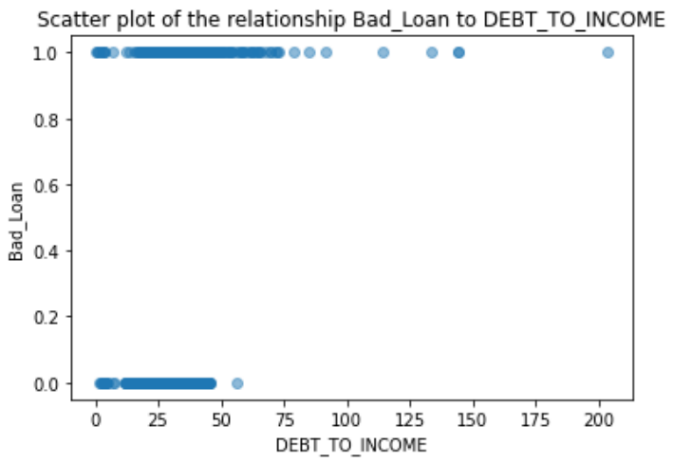
DEBT\_TO\_INCOME **before** KNN-Impute:



**DEBT\_TO\_INCOME:**

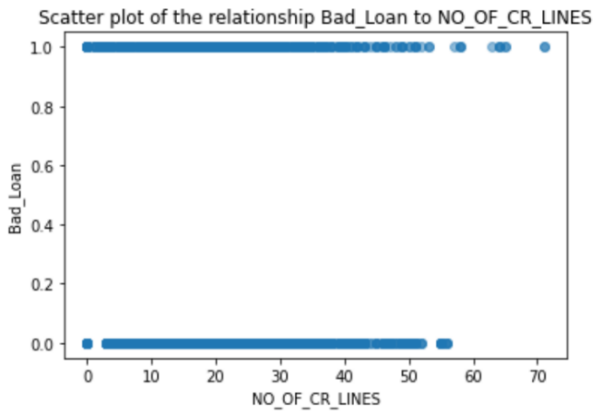
In this scatter plot you could suggest that client defaults increase with “DEBT\_TO\_INCOME”. Which makes sense. The more indebted you are, it could be argued the greater the risk that you will be unable to pay off or struggle to pay your loan.

DEBT\_TO\_INCOME **after** KNN-Impute: No significant visual change



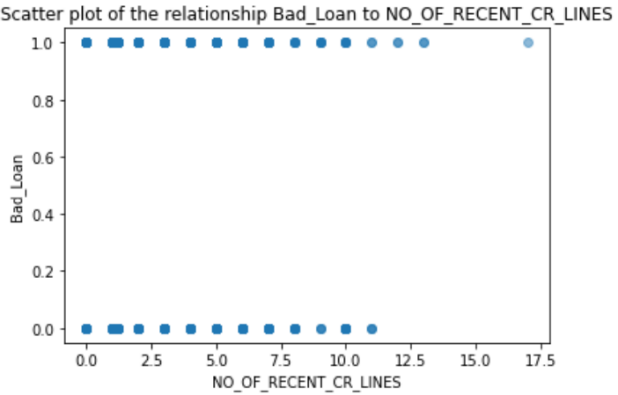
**NO\_OF\_CR\_LINES:**

In this scatter plot you could suggest that client defaults increase very slightly with “NO\_OF\_CR\_LINES”. Which makes sense. Overall they are fairly alike but outliers for default are more prominent in defaults.



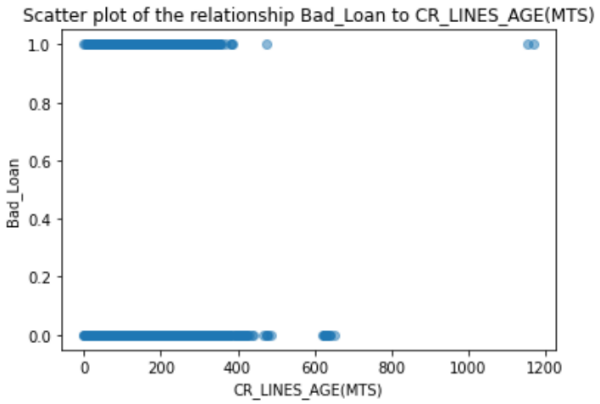
**NO\_OF\_RECENT\_CR\_LINES:**

In this scatter plot you could suggest that client defaults increase very slightly with “NO\_OF\_RECENT\_CR\_LINES”. Which makes sense perhaps that's why they looked for more credit lines. Overall they Default and Repaid are fairly alike.



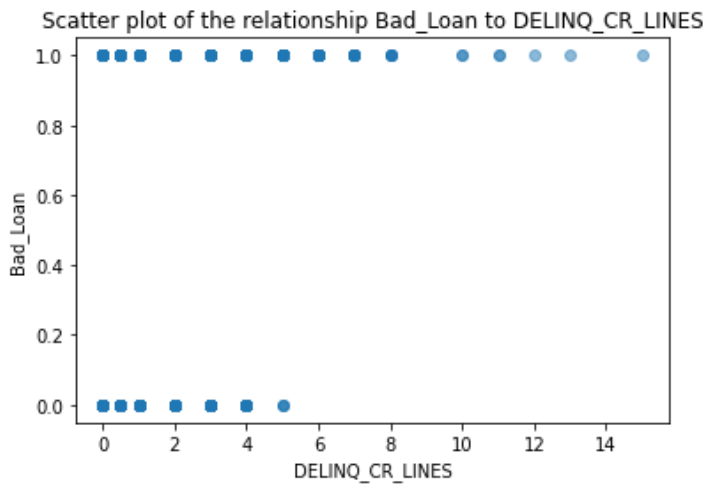
**CR\_LINES\_AGE(MTS):**

you could suggest that client defaults decrease with reduced “CR\_LINES\_AGE(MTS)”. However this could be due to not giving an individual extended time, to allow them to resolve the debt.



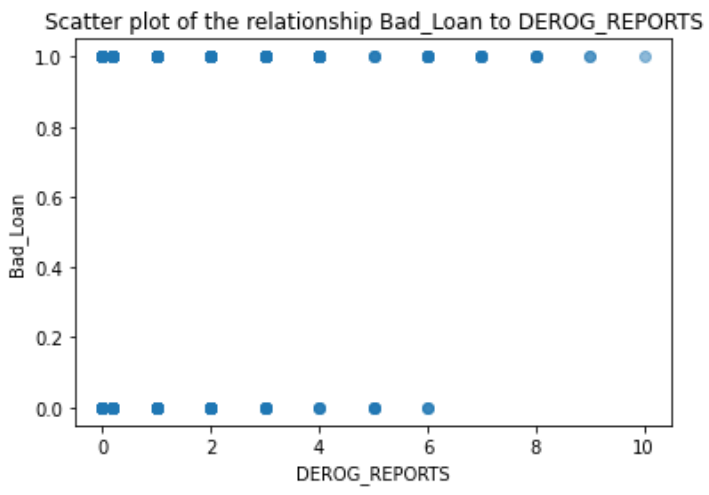
**DELINQ\_CR\_LINES:**

It would appear that the more “Delinquent\_CR\_lines” a person has the more likely to default.



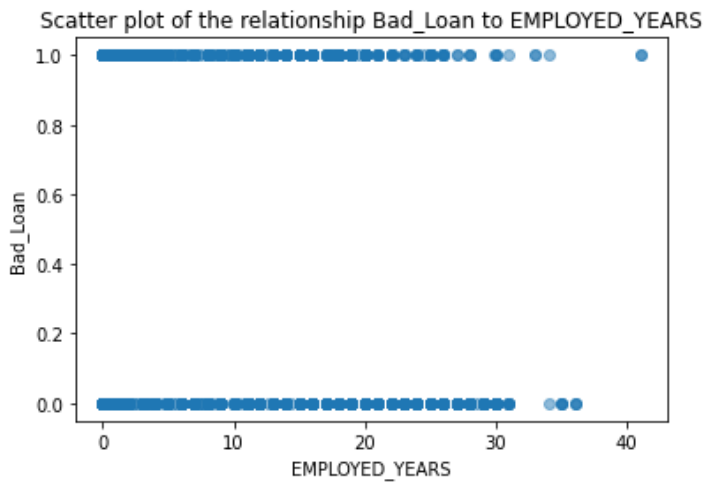
**DEROG\_REPORTS**:

It would appear that the more “Derogatory reports” a person has the more likely to default.



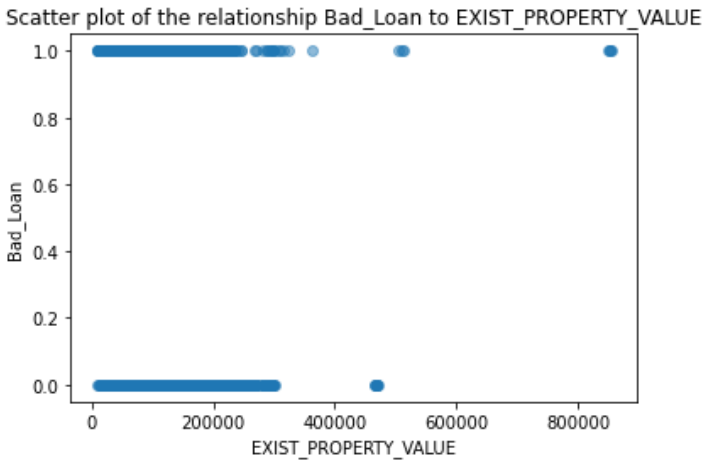
**EMPLOYED\_YEARS:**

It would not appear there is an significant correlation between “years employed” and making BAD\_LOAN.



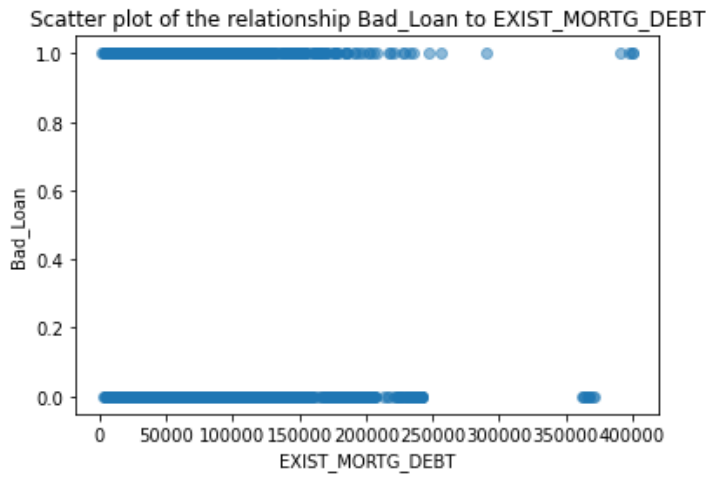
**EXIST\_PROPERTY\_VALUE:**

It would not appear there is an significant correlation between “existing property value” and making BAD\_LOAN. the outliers are two small in number to say for certain.



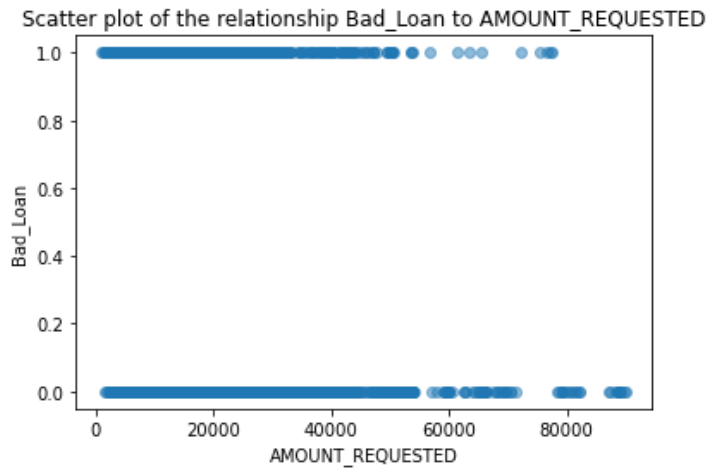
**EXIST\_MORTG\_DEBT:**

It would not appear there is an significant correlation between “existing mortgage debt” and making BAD\_LOAN.



**AMOUNT\_REQUESTED:**

Strangely it would appear that requesting a larger loan would suggest its more slightly more likely to be repaid.



Action 6 - Feature engineering

In this section I used the KNN-IMPUTER to impute missing values to column “DEBT\_TO\_INCOME” which had a total Percent missing of 21%. What made this a difficult column to impute missing values for, was that I could not find any column to indicate an income or any means to back-engineer the “Debt-to-Income” ratio. Scikit learn describe the process as Imputation for completing missing values using k-Nearest Neighbors and can be found at <https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html>

I created optimized data for Impute KNN parameter n= 2, 4, 6, 8, 10 using the code below Then I applied each to the Random Forest Classifier to compute a model performance.

This is the code to KNN-Impute the missing values.

from sklearn.impute import KNNImputer

df = transform\_categorical\_variables(df)

# from sklearn.preprocessing import MinMaxScaler

# scaler = MinMaxScaler()

# df = pd.DataFrame(scaler.fit\_transform(df), columns = df.columns)

imputer = KNNImputer(n\_neighbors=n)

df = pd.DataFrame(imputer.fit\_transform(df),columns = df.columns)

This is the model performance for each value of n, using a KFold CV value of 5. The Random Forest Classifier model had the follow results below. It would appear that n=4 was the best in this very limited trial.

N=2

Accuracy\_score() is: 0.913

True negatives - correctly classified as not Target: 1132

False negatives - wrongly classified as not Target: 9

False positives - wrongly classified as Target: 115

True positives - correctly classified as Target: 171

Confusion Matric - Accuracy: 0.913104414856342

RandomForestClassifier Cross Val Score - Accuracy: 0.924 +/- 0.005

N=4

Accuracy\_score() is: 0.919

True negatives - correctly classified as not Target: 1132

False negatives - wrongly classified as not Target: 9

False positives - wrongly classified as Target: 107

True positives - correctly classified as Target: 179

Confusion Matric - Accuracy: 0.9187105816398038

RandomForestClassifier Cross Val Score - Accuracy: 0.928 +/- 0.006

N=6

Accuracy\_score() is: 0.914

True negatives - correctly classified as not Target: 1136

False negatives - wrongly classified as not Target: 5

False positives - wrongly classified as Target: 118

True positives - correctly classified as Target: 168

Confusion Matric - Accuracy: 0.9138051857042747

RandomForestClassifier Cross Val Score - Accuracy: 0.922 +/- 0.008

N=8

Accuracy\_score() is: 0.916

True negatives - correctly classified as not Target: 1137

False negatives - wrongly classified as not Target: 4

False positives - wrongly classified as Target: 116

True positives - correctly classified as Target: 170

Confusion Matric - Accuracy: 0.9159074982480728

RandomForestClassifier Cross Val Score - Accuracy: 0.926 +/- 0.008

N=10

Accuracy\_score() is: 0.917

True negatives - correctly classified as not Target: 1135

False negatives - wrongly classified as not Target: 6

False positives - wrongly classified as Target: 113

True positives - correctly classified as Target: 173

Confusion Matric - Accuracy: 0.9166082690960056

RandomForestClassifier Cross Val Score - Accuracy: 0.923 +/- 0.007

Action 7 - Checked for Multi-collinearity in features

I want to access the impact of similar features so I completed a multi-collinearity test. I got information on this from

<https://towardsdatascience.com/everything-you-need-to-know-about-multicollinearity-2f21f082d6dc>

And,

<https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/>

Although used for linear regression, it is nice to see if the VIF will indicate if there is high collinearity between features. This is a simple method to detect multi-collinearity in a model is by using something called the Variance Inflation Factor or the VIF for each predicting variable. VIF measures the ratio between the variance for a given regression coefficient with only that variable in the model versus the variance for a given regression coefficient with all variables in the model. A VIF of 1 (the minimum possible VIF) means the tested predictor is not correlated with the other predictors.

The code was as follows;

#==========================================================

#Checking for Multicollinearity

#==========================================================

## https://towardsdatascience.com/everything-you-need-to-know-about-multicollinearity-2f21f082d6dc

## https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/

# creating dummies for vif

df\_cat = transform\_categorical\_variables(df)

# Create datasets for model

target\_column\_name = 'BAD\_LOAN'

X, y = create\_X\_y\_datasets(df\_cat, target\_column\_name)

# VIF dataframe

vif\_data = pd.DataFrame()

vif\_data["feature"] = X.columns

# calculating VIF for each feature

vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i)for i in range(len(X.columns))]

# vif\_data.drop(vif\_data[0], axis=1, inplace = True)

vif\_data.set\_index('feature', inplace = True)

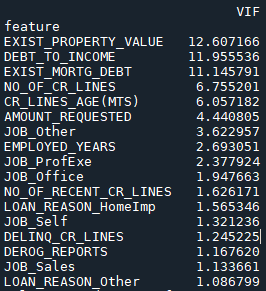
vif\_data.sort\_values(by=['VIF'], inplace=True, ascending=False)

print(vif\_data)

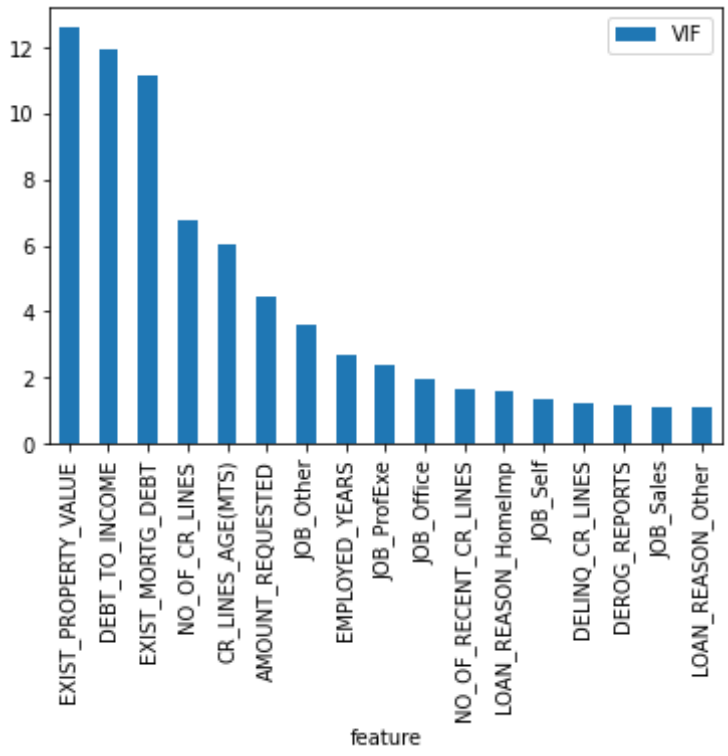
print(vif\_data.info())

vif\_data.plot.bar()

plt.show()



My results would suggest that there is correlated with some of the features. Above 10 was stated as less than desirable in the information I read. “Feature\_property\_Value”, “Debt\_to\_Income” and “Existing\_Mortg\_Debt” appear to be possible correlated. Perhaps having a high mortgage debt is reflected in a high existing property value. Which likely to lead to a higher debt? This is another way to look at the features and the output was as follows;



Action 8 - Identify most important features

In this section I wanted to see what features where having the most significant impact on the output of the model. With this information I remove the lesser important features and assessed model performance. I used the Random Forest Classifier and tested with one less feature each time. With this code;

def most\_important\_features(X,y):

#Feature Importance - https://www.kaggle.com/niklasdonges/end-to-end-project-with-python

random\_forest = RandomForestClassifier(n\_estimators=100)

random\_forest.fit(X, y)

importances = pd.DataFrame({'feature':X.columns,'importance':np.round(random\_forest.feature\_importances\_,3)})

importances = importances.sort\_values('importance',ascending=False).set\_index('feature')

print('\nimportances.head(15):\n',importances.head(15))

#show bar plot on impotant features

importances.plot.bar()

plt.show()

#==========================================================

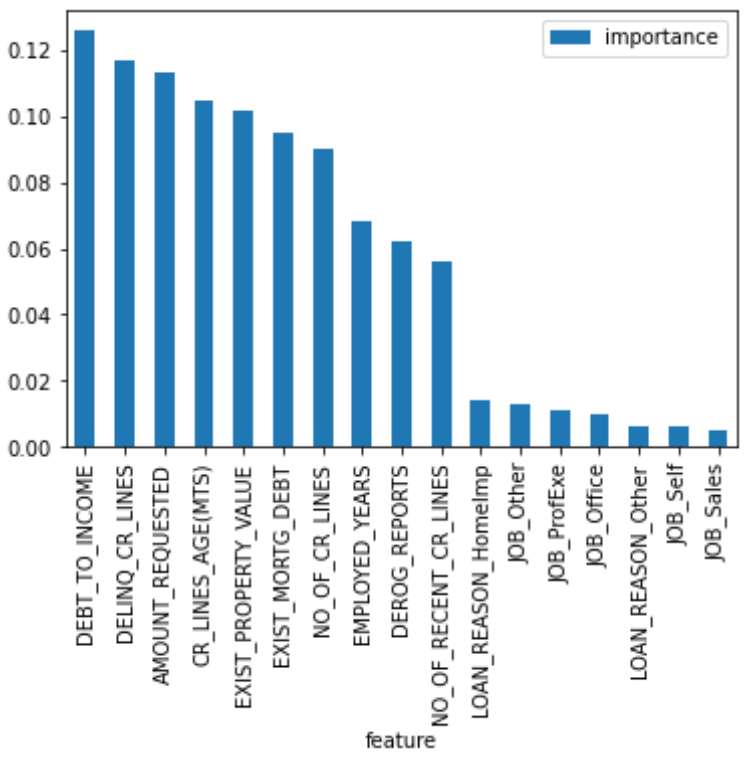
#Identiy important features

#==========================================================

#call function to plot and describe important features

most\_important\_features(X,y)

This is what was found for feature importance;



This is model performance result for N=4, **baseline result.**

Accuracy\_score() is: 0.919

True negatives - correctly classified as not Target: 1132

False negatives - wrongly classified as not Target: 9

False positives - wrongly classified as Target: 107

True positives - correctly classified as Target: 179

Confusion Matric - Accuracy: 0.9187105816398038

RandomForestClassifier Cross Val Score - Accuracy: 0.928 +/- 0.006

This is what I found for when I dropped the least important feature “JOB” .

No significant real change.

Accuracy\_score() is: 0.919

True negatives - correctly classified as not Target: 1135

False negatives - wrongly classified as not Target: 6

False positives - wrongly classified as Target: 109

True positives - correctly classified as Target: 177

Confusion Matric - Accuracy: 0.9194113524877365

RandomForestClassifier Cross Val Score - Accuracy: 0.923 +/- 0.007

This is what I found for when I dropped the next least important feature “Loan\_Reason” . Model Accuracy decreased but not significantly.

Accuracy\_score() is: 0.908

True negatives - correctly classified as not Target: 1132

False negatives - wrongly classified as not Target: 9

False positives - wrongly classified as Target: 122

True positives - correctly classified as Target: 164

Confusion Matric - Accuracy: 0.9081990189208129

RandomForestClassifier Cross Val Score - Accuracy: 0.919 +/- 0.008

This is what I found for when I dropped the next least important feature “NO\_OF\_RECENT\_CR\_LINES” .

Model Accuracy decreased the most but not significantly.

Accuracy\_score() is: 0.901

True negatives - correctly classified as not Target: 1130

False negatives - wrongly classified as not Target: 11

False positives - wrongly classified as Target: 130

True positives - correctly classified as Target: 156

Confusion Matric - Accuracy: 0.9011913104414856

RandomForestClassifier Cross Val Score - Accuracy: 0.912 +/- 0.008

Conclusion: With this finding, that the removal of feature of less importance did not improve the model predictive performance. I decided not to remove any features from the data-frame.

## Step.5 - Tune\_Model\_and Select

This step is similar to Step 3. In Step 3 I ran the models on basically cleaned data but in this step I ran the model on optimized cleaned data as described in step 4. The purpose of this step is to run the Optimized Data through a number of Supervised Classifier Machine Learning Models and assess their performance.

The step has the following actions;

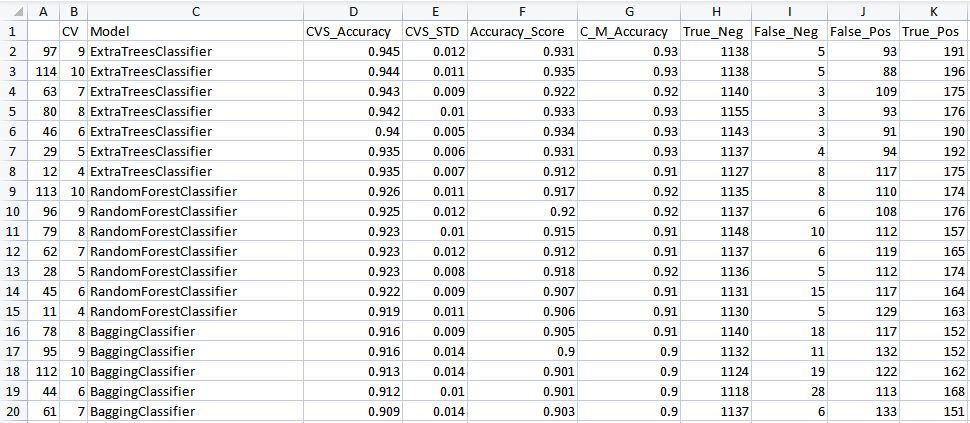
1. Import data and transform the categorical variables.
2. Create a array of Features values ‘X’ and Target array of values called ‘y’.
3. Normalize the data-frame
4. Perform K-Fold cross validation on a number of models

The performance for the top classifiers is below. We can see that the Random Classifier performs excellently but not the best at this point. It has shown an increase in performance of 1%.

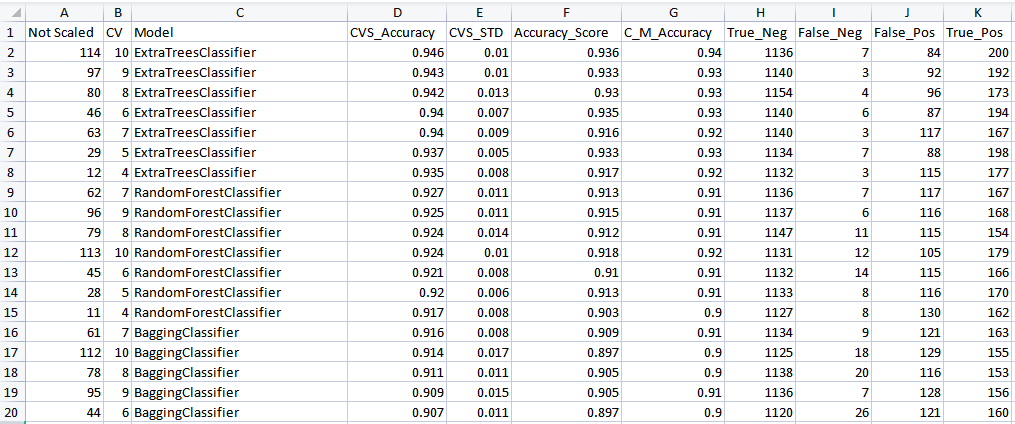
I ran the models on scaled (Normalized) and not scaled data. They a very similar in results and I have since read scaling is not always required.

<https://datascience.stackexchange.com/questions/62031/normalize-standardize-in-a-random-forest>

Top model performances with Optimised\_Data that **was** Normalized.



Top model performances with Optimised\_Data that was **not** Normalized.



## Step.6 - HyperTune\_model

This step is similar to Step 3 and Step 5 in that I am trying to run a model. I have chosen to use the Random Forest Classifier as it performed excellent of the data to date. In this step, I am going to Hyper-tune its parameters. To complete this I am going to use Grid Search Cross Validation. Grid Search Cross-validation is a method for estimating the best parameters for a model. It provides an exhaustive search over specified parameter values for a model.

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>

This step has the following actions;

1. Pre-processing data
2. Hyper-parameter Tuning and Saving best parameters to file
3. Classifier performance after hyper tuning
4. Evaluating final model and saving model

Action 1 - Pre-processing data

This has been described in Step 3 and Step 5 and involves setting up the data so that it can run effectively in the model. This consists of transforming the data, Identifying the Features and Target variables, Scaling (not in this case) the variables and splitting the data into Train and Test sets.

Action 2 - Hyper parameter Tuning

In this action I created a parameter grid which was loaded into the Grid Search Cross Validation function. You can see the dictionary below “param\_grid”. This has the values I will be applying to the model. I created a second test “param\_grid” which is hashed out. This was to allow a more rapid testing as Hyper tuning can take up to 30 minutes on this laptop.

#==========================================================

#Hypertuning - Optimise final model

#==========================================================

print("\nHyperparameter Tuning:")

param\_grid = { "criterion" : ["gini", "entropy"],

"min\_samples\_leaf" : [1, 5, 10, 25, 50, 70],

"min\_samples\_split" : [2, 4, 10, 12, 16, 18, 25, 35],

"n\_estimators": [100, 400, 700, 1000, 1500]}

# ##for quick testing as Hyptuning can take along time to complete

# param\_grid = { "criterion" : ["gini", "entropy"],

# "min\_samples\_leaf" : [1],

# "min\_samples\_split" : [2],

# "n\_estimators": [100]}

The model chosen, the parameters, the number of processors are then set and this is assigned to the variable clf.

# Complete GRID search with various parameters to find best parameters

model = RandomForestClassifier( oob\_score=True, random\_state=1, n\_jobs=-1)

clf = GridSearchCV(estimator=model, param\_grid=param\_grid, n\_jobs=-1)

clf.fit(X\_train, y\_train)

I used the following code to identify the best parameters from the model. These are then displayed.

print("\nBest parameters found as per parama grid")

print("\nclf.best\_params\_", clf.best\_params\_)

Using Pickle, I saved the best model parameter to a file in the project folder. This was so I had a copy of the parameters for Step 8. Making Predictions.

#=================================================

# Save Hypertuned parameters

#=================================================

# Import pickle Package

import pickle

# Save the Modle to file in the current working directory

model\_filename = "6\_Best\_Model\_Params.pkl"

with open(model\_filename, 'wb') as file:

pickle.dump(clf.best\_params\_, file)

Action 3 - Classifier performance after hyper tuning

I used the following code to test the models performance

#=================================================

# Perfom Classification Report after Hypertuning

#=================================================

from sklearn.metrics import classification\_report

print("\nDetailed classification report for HyperTuned model:")

print("Train scores:")

y\_pred = clf.predict(X\_train)

print(classification\_report(y\_train, y\_pred))

print("Test scores:")

y\_pred = clf.predict(X\_test)

print(classification\_report(y\_test, y\_pred));pause()

Output:

Detailed classification report for HyperTuned model:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test scores:** |  |  |  |  |
|  | precision | recall | f1-score | support |
| 0.0 - Repaid | 0.88 | 0.98 | *0.93* | *2665* |
| 1.0 - Default | 0.86 | 0.47 | *0.61* | *674* |
|  |  |  |  |  |
| accuracy |  |  | 0.88 | 3339 |
| macro avg | 0.87 | 0.73 | 0.77 | 3339 |
| weighted avg | 0.88 | 0.88 | 0.86 | 3339 |

Action 4 - Evaluating final model and save

I then re-loaded the Hyper-Tuned parameters into the Random Forest Classifier model. I did this to practice setting up the model.

#=================================================

# RandomForestClassifier set up parameters after Hypertuning

#=================================================

#Test new best paramters: Random Forest with TEST data

model = RandomForestClassifier(criterion = "gini",

min\_samples\_leaf = 1,

min\_samples\_split = 2,

n\_estimators=700,

max\_features='auto',

oob\_score=True,

random\_state=1,

n\_jobs=-1)

##clf.best\_params\_ {'criterion': 'gini', 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 700}

#Check model parameters loaded

print('\nConfirm parameters currently in use:\n')

pprint(model.get\_params());pause()

#get oob score

model.fit(X\_train, y\_train)

print("\nHypertuned - oob score:", round(model.oob\_score\_, 2)\*100, "%")

#Implement accuracy\_score() function on model

y\_pred = model.predict(X\_test)

accuracy\_test = accuracy\_score(y\_test, y\_pred)

print("Accuracy\_score() is: ", round(accuracy\_test, 3));pause()

#Confusion matrix

confusion\_matrix\_results = confusion\_matrix(y\_test, y\_pred)

print("True negatives - correctly classified as not Target: ", confusion\_matrix\_results[0][0])

print("False negatives - wrongly classified as not Target: ",confusion\_matrix\_results[0][1])

print("False positives - wrongly classified as Target: ", confusion\_matrix\_results[1][0])

print("True positives - correctly classified as Target: " ,confusion\_matrix\_results[1][1])

confusion\_matric\_accuracy = (confusion\_matrix\_results[0][0]+confusion\_matrix\_results[1][1])/len(y\_pred)

#just want to make sure program stops if these couts are dirrenent as it mean my accuracy will not be correct

assert len(y\_pred)==(confusion\_matrix\_results[0][0]+confusion\_matrix\_results[0][1]+confusion\_matrix\_results[1][0]+confusion\_matrix\_results[1][1])

print("Confusion Matric - Accuracy: " ,confusion\_matric\_accuracy)

These are the cross val (n=7) scores from Steps 5 with no scaling

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CV | CVS\_Accuracy | CVS\_STD | Accuracy\_Score | C\_M\_  Accuracy | True\_Neg | False\_Neg | False\_Pos | True\_Pos |
| 7 | 0.927 | 0.011 | 0.913 | 0.91 | 1136 | 7 | 117 | 167 |

I got the following results for the Random Forest Classifier after Hyper-tuning: These are the model performance figures for the Test data. The test data has a different number of rows (30% of the rows). This is a good example where accuracy score should not be used in isolation. Notice the high number of False Positives but to look at the accuracy score alone you would be mistaken in thinking the model perform excellently.

You can see the drop in predictive performance on the Test data. The result went from 91.3% to 89.2%.

Accuracy\_score() is: 0.892

True negatives - correctly classified as not Target: 2638

False negatives - wrongly classified as not Target: 45

False positives - wrongly classified as Target: 315

True positives - correctly classified as Target: 330

Confusion Matric - Accuracy: 0.8918269230769231

I then saved the model to the project file. The reason I did this is because I was testing each component of the project individually and wanted to be able to re-load the working model when required.

This was saved as follows;

#==========================================================

#Save model to file

# https://www.kaggle.com/prmohanty/python-how-to-save-and-load-ml-models

#==========================================================

# Import pickle Package

import pickle

# Save the Modle to file in the current working directory

model\_filename = "6\_Loan\_UCD\_ML\_Model.pkl"

with open(model\_filename, 'wb') as file:

pickle.dump(model, file)

## Step.7 - Optimise\_Validation\_Data\_Cleaning

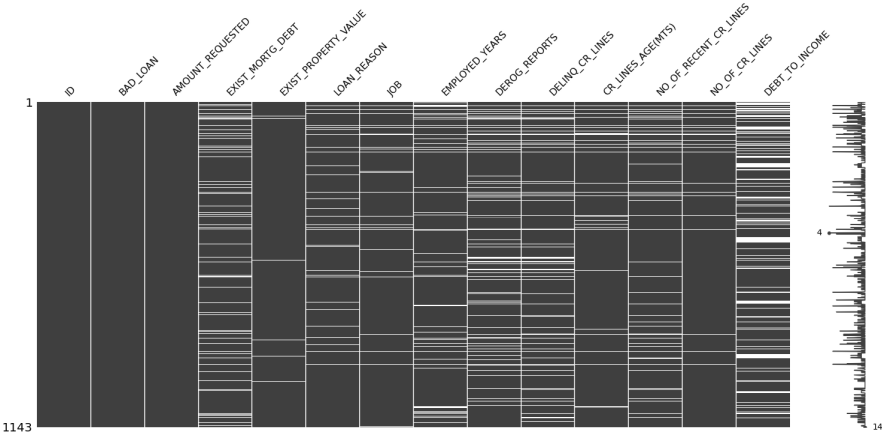
In this section, I perform the basic and optimized data cleaning techniques in one program on the validation data-set.The purpose of the Validation data-frame was to test the model on unseen data. This is so that it is in-biased and we can get a true indication of how well the model will work. I created this validation data-frame at the start of the project. This data will then be used to make the final predictions and check the performance of the model.

This is an important step as there may have been anomalies that were not in the Training data. For example there may have been different text issues, miss spellings.

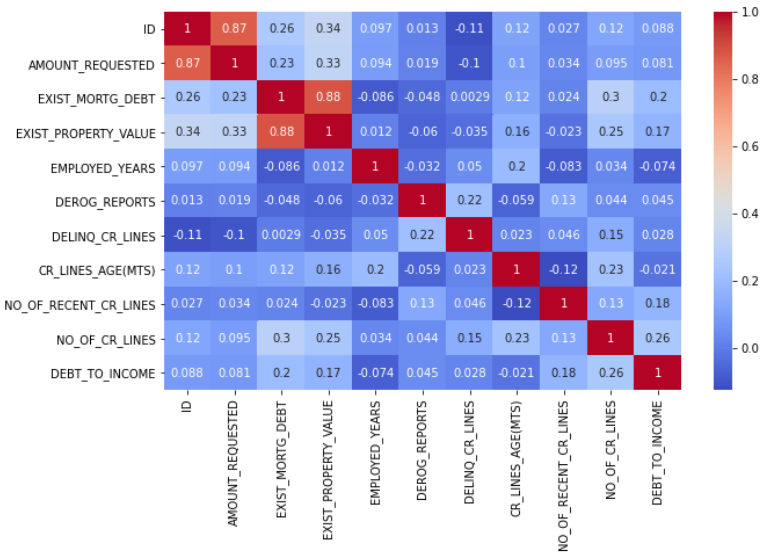
This step consisted of some of the following but no additional anomalies were found that had not been seen already;

* Data checking the columns dTypes
* EDA Descriptive: shape was (1143, 14)
* EDA Exploratory

**Missingness**



**Heat Map**

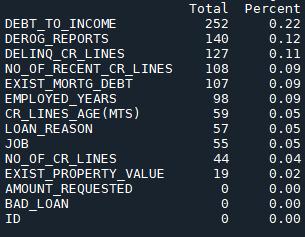
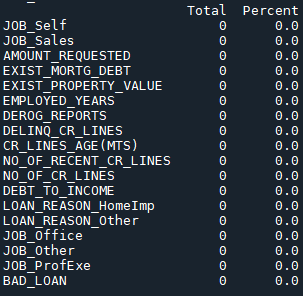


* Drop duplicate rows

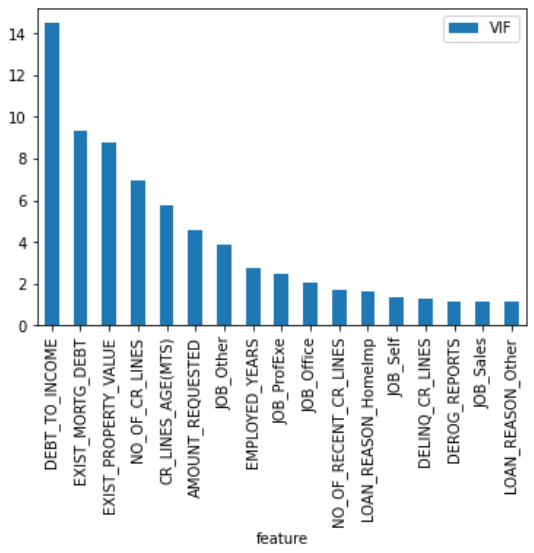


* Standardized contents in “BAD\_LOAN” and “JOB” columns
* Check scatter plots - these will be reviewed in the insight section
* Imputed missing values
* imputed values for DEBT\_TO\_INCOME using KNNImputer
* Descriptive missing data check;

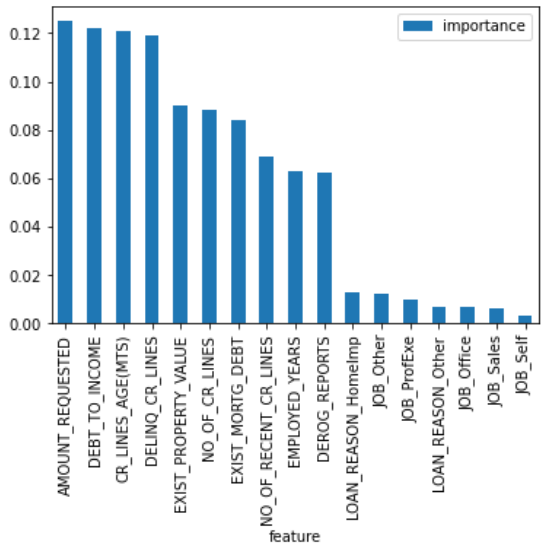
Data frame missing before clean and after clean

 to 

* Then I checked the data-frame for multi-collinearity



* Then I checked the data-frame for feature importance



Conclusion: This data-frame was saved to **“S7\_Loan\_Optimised\_Data\_Cleaning.csv”**. After screening the scatter plots and graphs above the data-frame was similar to the Optimized data-frame. I did not have to perform any additional changes. But this should not be taken for granted. After completing this step, when doing a project again, I would probably do this activity for the entire data-set together, before splitting in to Train-Test and Validation data-sets.

## Step.8 - Model\_Prediction

This step uses the validation data-set generated in “Step 1 - Data\_Gathering”. It was then optimized in “Step 7 - Optimise\_Data\_Cleaning”. The purpose of creating this new cleaned data-set was to see how accurately the model predict on clean data. In this step, we will also review a number of different ways to evaluate the model. The three main metrics used to evaluate a classification model are accuracy, precision, and recall.

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

Precision is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class.

Recall is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class.

This step has the following actions;

1. Import Hyper tuned Model and get parameters
2. Import the Optimized data and pre-process for model
3. Import Hyper tuned model parameters and examine
4. Evaluate model predictions

Action 1 - Import Hyper-tuned Model and get parameters

In this action I imported the model code using pickel. I used the following guide to complete this activity. <https://www.kaggle.com/prmohanty/python-how-to-save-and-load-ml-models>

I imported the model and checked the best parameters. I was checking to see were they the same from “Step 6 - Hyper tuning”. I was interested to see if any info was lost in the save / load process.

Action 2. Import the Optimized data and pre-process for model

I will not spend a lot of time describing this action as its mostly a repeat of pre-processing in Step 6. In this step I imported the Optimised data, checked its shape to make sure the columns matched what the model was expecting.

df shape was : (1246, 18)

X shape is : (1246, 17)

y shape is: (1246,)

This has been an issue for me on other occasions where the column count to the model was different to when it was prepared. This can happen when its loaded from a saved file as sometimes creates a new index column.

Action 3 - Import Hyper tuned model parameters and examine

For my own interest I imported the hyper tuned best parameters which I had saved and created a model Random Forest Classifier with these parameters. The purpose was just to compare how would a saved model that was reloaded performance compare to a model that was directly saved and re-loaded.

Action 4 - Evaluate model predictions

Below are the a number of evaluation metric I used to evaluate the model. While I did not need all of these I thought it would be interesting to see how they scored the models predictive ability.

Evaluation methods used

* Method 1 - Model accuracy
* Method 2 - oob score
* Method 3 - Confusion Matrix
* Method 4 - Precision and Recall / Precision Recall Curve
* Method 5 - F-Score
* Method 6 - Classification\_report
* Method 7 - ROC AUC Curve and ROC AUC Score

Method 1 - Models accuracy

This checks the models prediction accuracy. Classification Accuracy is what we mean when we say accuracy. It is the ratio of number of correct predictions to the total number of input samples. You must have equal number of samples belonging to each class. A class being the potential predictions (Loan Repaid / Loan Defaulted). If one class is very high, say 98%. Then our model can easily get 98% training accuracy by simply predicting every training sample belonging to that class. This is very important learning.

<https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score>

Code:

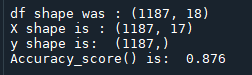
from sklearn.metrics import accuracy\_score

y\_pred = model.predict(X\_test)

prediction\_accuracy = accuracy\_score(y\_test, y\_pred) # fraction of correctly classified samples

print("prediction\_accuracy:", round(prediction\_accuracy,2))

Output Accuracy\_score for this model:



Method 2 - oob score

The Random Forest Classifier is trained using bootstrap aggregation. The out-of-bag (OOB) error is the average error for each calculated using predictions from the trees that do not contain in their respective bootstrap sample. This allows the Random Forest Classifier to be fit and validated whilst being trained. The OOB error can be measured at the addition of each new tree during training. The result allows a practitioner to approximate a suitable value of n\_estimators at which the error stabilizes. <https://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html>

Code:

#get oob score

model.fit(X\_train, y\_train)

print("\nHypertuned - oob score:", round(model.oob\_score\_, 2)\*100, "%") ;pause()

Output oob for this model:



Method 3 - Confusion Matrix

A confusion matrix gives you a lot of information about how well your model does. When performing classification predictions, there's four types of outcomes that could occur.

* True positives are when you predict an observation belongs to a class and it actually does belong to that class.
* True negatives are when you predict an observation does not belong to a class and it actually does not belong to that class.
* False positives occur when you predict an observation belongs to a class when in reality it does not.
* False negatives occur when you predict an observation does not belong to a class when in fact it does.

These four outcomes are plotted on a confusion matrix binary classification.

<https://www.jeremyjordan.me/evaluating-a-machine-learning-model/>

Code:

from sklearn.model\_selection import cross\_val\_predict

from sklearn.metrics import confusion\_matrix

predictions = cross\_val\_predict(model, X\_train, y\_train, cv=5)

confusion\_matrix\_results = confusion\_matrix(y\_train, predictions)

print("\nConfusion Matrix: \n",confusion\_matrix\_results)

print("\nConfusion Matrix: \nThe first row is about the not-target-predictions:")

print("True negatives - correctly classified as not Target: ", confusion\_matrix\_results[0][0])

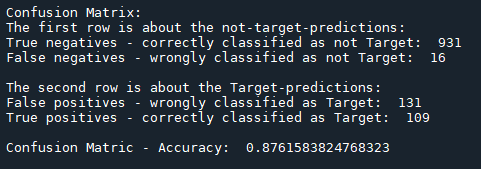
print("False negatives - wrongly classified as not Target: ",confusion\_matrix\_results[0][1])

print("\nThe second row is about the Target-predictions:")

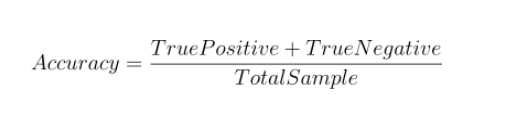
print("False positives - wrongly classified as Target: ", confusion\_matrix\_results[1][0])

print("True positives - correctly classified as Target: " ,confusion\_matrix\_results[1][1]);pause()

Confusion Matrix for this model:



Accuracy for the confusion matrix can be calculated by taking average of the values lying across the “main diagonal” i.e



Confusion Matrix forms the basis for the other types of metrics. I go the following accuracy score using this formulae;



Method 4 - Precision and Recall / Precision Recall Curve

Precision is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class. The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0.

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score>

from sklearn.metrics import precision\_score

precision\_score(y\_true, y\_pred)

Recall is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class. The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.The best value is 1 and the worst value is 0.

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score>

from sklearn.metrics import recall\_score

recall\_score(y\_true, y\_pred)

Code:

from sklearn.metrics import precision\_score, recall\_score

print("\nPrecision and Recall results:")

print("Precision - predicts % of the time, a Target correctly:", round(**precision\_score(y\_train, predictions)**, 2))

print("Recall- tells us that it predicted the Target % of the correct:", round(**recall\_score(y\_train, predictions**),2))

Output



Precision Recall Curve: For each person the Random Forest algorithm computes a probability based on a function and it classifies the person as defaulted (when the score is bigger the than threshold) or as not defaulted (when the score is smaller than the threshold). That's why the threshold plays an important part.

from sklearn.metrics import precision\_recall\_curve

# getting the probabilities of our predictions

y\_scores = model.predict\_proba(X\_train)

y\_scores = y\_scores[:,1]

precision, recall, threshold = precision\_recall\_curve(y\_train, y\_scores)

#function

def plot\_precision\_and\_recall(precision, recall, threshold):

plt.plot(threshold, precision[:-1], "r-", label="precision", linewidth=5)

plt.plot(threshold, recall[:-1], "b", label="recall", linewidth=5)

plt.xlabel("threshold", fontsize=19)

plt.legend(loc="upper right", fontsize=19)

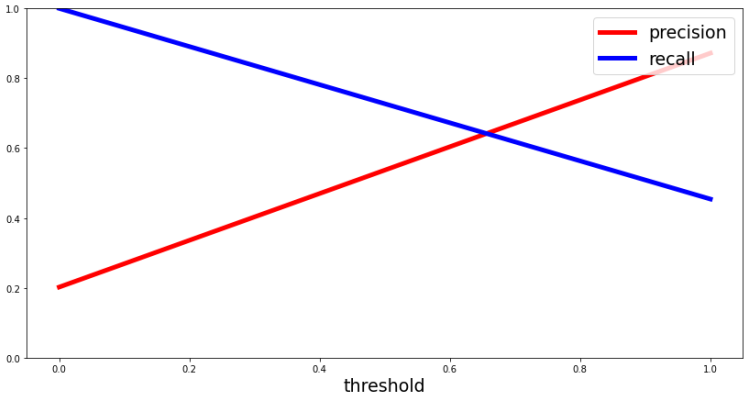
plt.ylim([0, 1])

plt.figure(figsize=(14, 7))

plot\_precision\_and\_recall(precision, recall, threshold)

plt.show();pause()

Output:



Method 5 - F1-Score

Compute the F1 score, also known as balanced F-score or F-measure. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 \* (precision \* recall) / (precision + recall)

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score>

from sklearn.metrics import f1\_score

f1\_score(y\_true, y\_pred)

F1 Score is used to measure a test’s accuracy. F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model.

<https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>

The F1-Score combines precision and recall into one score. The F-score is computed with the harmonic mean of precision and recall. Note that it assigns much more weight to low values. As a result of that, the classifier will only get a high F-score, if both recall and precision are high.

Code:

from sklearn.metrics import f1\_score

print("\nF1-score - combine precision and recall into one score")

f1\_score(y\_train, predictions)

Output:



Method 6 - Classification\_report

The Classification report is a text report showing the main classification metrics. It computes the Precision, Recall, F-measure and support for each class. The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The support is the number of occurrences of each class in y\_true.

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1', 'class 2']

print(classification\_report(y\_true, y\_pred, target\_names=target\_names))

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html#sklearn.metrics.precision_recall_fscore_support>

Code:

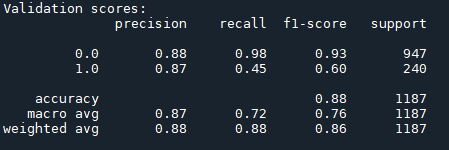
#Classification report

from sklearn.metrics import classification\_report

print("Test scores:")

y\_pred = model.predict(X\_test)

print(classification\_report(y\_test, y\_pred));pause()



Method 7 - ROC AUC Curve and ROC AUC Score

Area Under Curve(AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.

<https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>

ROC AUC Curve - Another way to evaluate and compare your binary classifier . This curve plots the true positive rate (also called recall) against the false positive rate (ratio of incorrectly classified negative instances), instead of plotting the precision versus the recall.

Code:

from sklearn.metrics import roc\_curve

# compute true positive rate and false positive rate

false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_train, y\_scores)

# plotting them against each other

def plot\_roc\_curve(false\_positive\_rate, true\_positive\_rate, label=None):

plt.plot(false\_positive\_rate, true\_positive\_rate, linewidth=2, label=label)

plt.plot([0, 1], [0, 1], 'r', linewidth=4)

plt.axis([0, 1, 0, 1])

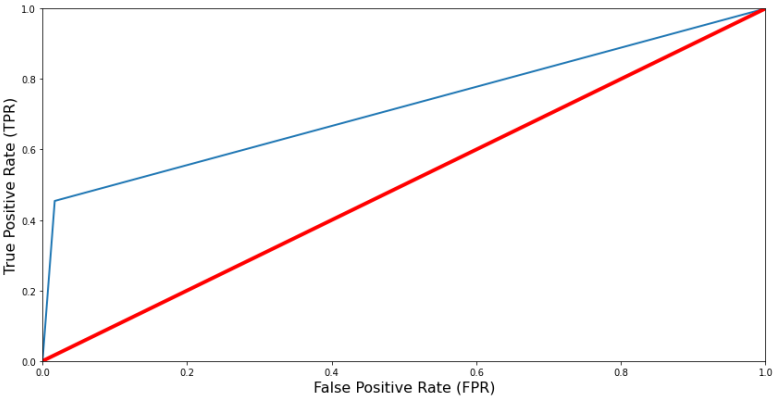
plt.xlabel('False Positive Rate (FPR)', fontsize=16)

plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(14, 7))

plot\_roc\_curve(false\_positive\_rate, true\_positive\_rate)

plt.show();pause()



ROC AUC Score - The ROC AUC Score is the corresponding score to the ROC AUC Curve. It is simply computed by measuring the area under the curve, which is called AUC. A classifiers that is 100% correct, would have a ROC AUC Score of 1 and a completely random classifier would have a score of 0.5.

Code:

from sklearn.metrics import roc\_auc\_score

r\_a\_score = roc\_auc\_score(y\_train, y\_scores)

print("\nROC-AUC-Score - A classifiers that is 100% correct, \nwould have a ROC AUC Score of 1, score is :", r\_a\_score);pause()

Output:

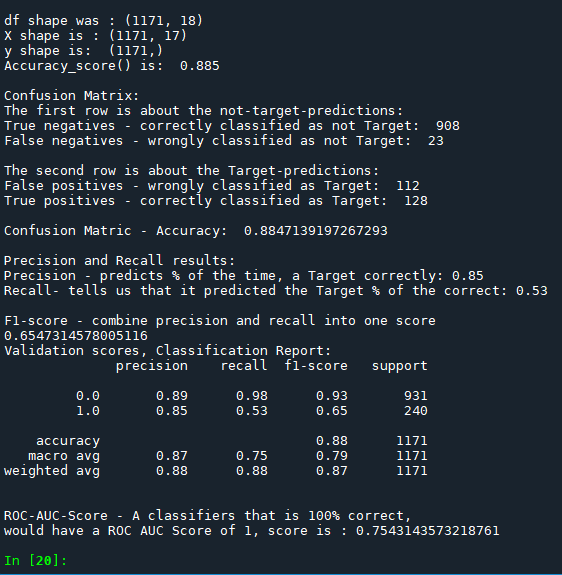


# Results

(Include the charts and describe them)

Below are the predicted final results using the Hyper-tuned model.

The model selected was a Random Forest Classifier.



# Insights

Many of the insights are captured in the project but significant these are captured below.

* Project plan

The importance of a plan was critical for me in completing this project in a systematic way. Before I constructed the plan I was more randomly completing actions and not really achieving any thing. The problem statement formed the foundation of the project and really helped guide my analysis.

* Data set

When I started to do this project my focus was on running & optimizing the model. In retrospect while this is very important my focus may have been miss guided. Looking back, I now think that examining the data visually, descriptively and feature engineering is perhaps more important.

My change of mind is based on the fact that no matter how much I ran the data in the model while changing model parameters etc, the predictions only marginally changed. It was only when I optimized the data that the model improved. So I am not saying that tuning - hyper tuning the model is not important. What I am suggesting is that this can be relatively automated, if you have the time and computational power you can check any parameter combinations which is available to every data analysis but its optimizing the data, feature engineering that appear to be key to your model gaining an edge.

* Quality of data-set

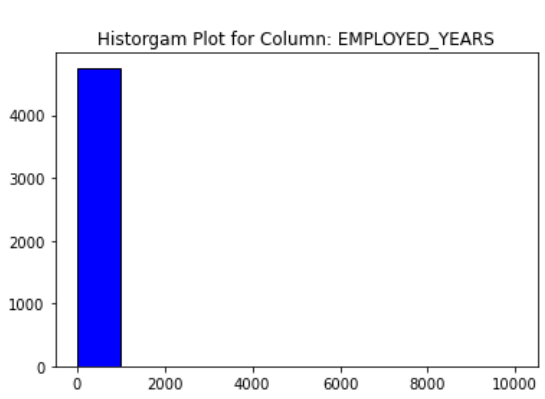
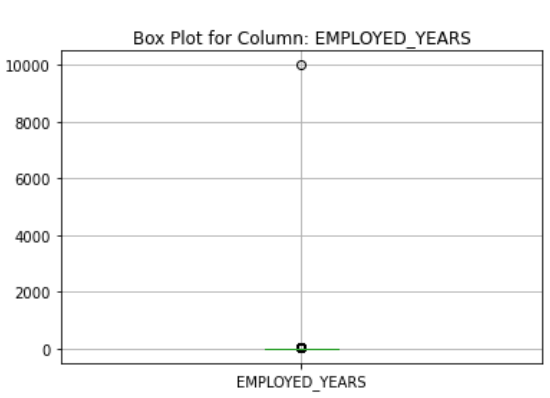
As noted above this is very important. Getting the right data-set is key. The considerations are;

1. Does this satisfy your problem statement?
2. Does it have the appropriate number of rows?
3. Does it have an adequate number of columns?
4. How much missing data is there and where?
5. Can you get detail on the feature title to aid your understanding?
6. Is it possible to feature engineer?

If I was running this project again and wanted to do a loan prediction project I am not confident that this data-set is complete enough to do a project like this. I found it hard to get detail from the WEB that would allow me understand the features adequately enough. This made it difficult to make meaningful imputations and feature engineering. For example there was no income feature to back-engineer the Debt-to-Income feature for missing data. I could not get any information on some of the features, so could not say with any certainty some of box-plot outliers were actually outliers?

* Interpreting Features

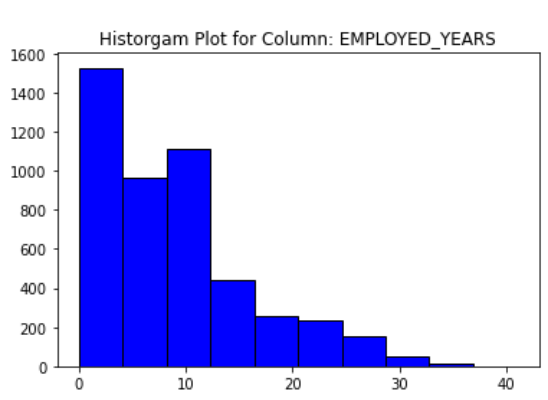
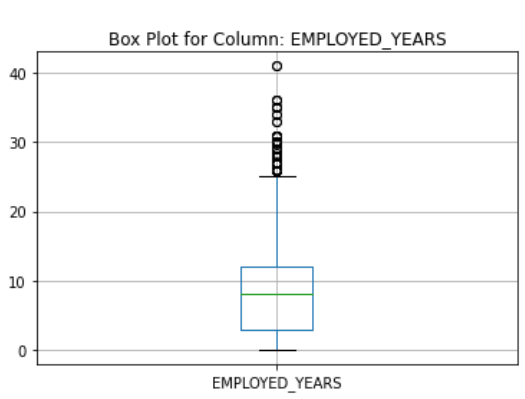
Understanding the importance of a feature to a project is very important and I used a number of approaches. I used scatter plots, box plots, histograms etc but it was only after completing a set of histogram and scatter plots I realized how misleading these plots can actually be. For example, when I looked at the employed years charts below I initially missed how the outlier was affecting the chart. It was only with some experience that I realized that this must be an outlier and when I changed it for the mean, you can see how chart changed to display more valuable information.

#replacing outlier with mean of column

EMPLOYED\_YEARS = round((df['EMPLOYED\_YEARS'].mean()),1)

df['EMPLOYED\_YEARS'].replace(9999,EMPLOYED\_YEARS,inplace=True)

Although with a bit more experience I now find that these charts are also a bit limited and now use a combination of Seaborn scatter plots, violin and strip plots. With these I used hue="BAD\_LOAN". This give a better clue of what is actually happening the feature.Using this code below I will briefly describe any insights or observations on the relationships using a strip plot for Bad Loans against the other features. I will comment on the significant charts.

Code:

#list of features o be examined

Examine = ['BAD\_LOAN']

#list of all fetaure i desired to do relationship against

Examine2 =['BAD\_LOAN', 'AMOUNT\_REQUESTED', 'EXIST\_MORTG\_DEBT', 'EXIST\_PROPERTY\_VALUE', 'EMPLOYED\_YEARS', 'DEROG\_REPORTS', 'DELINQ\_CR\_LINES', 'CR\_LINES\_AGE(MTS)', 'NO\_OF\_RECENT\_CR\_LINES', 'NO\_OF\_CR\_LINES', 'DEBT\_TO\_INCOME', 'LOAN\_REASON\_HomeImp', 'LOAN\_REASON\_Other', 'JOB\_Office', 'JOB\_Other', 'JOB\_ProfExe', 'JOB\_Sales', 'JOB\_Self']

# loop1 to go through list of features

for col in df[Examine]:

#loop2 to go through list of features

for col2 in df[Examine2]:

#create a relationship plot

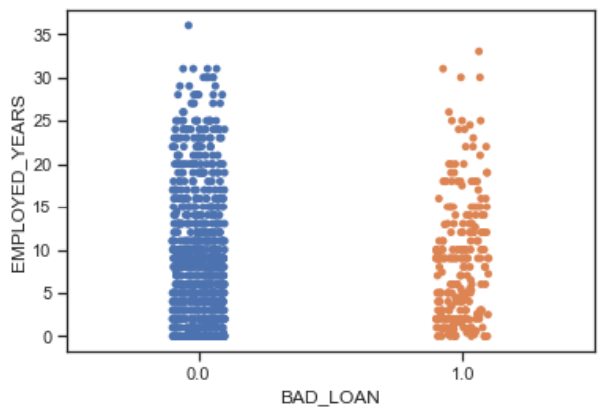
sns.catplot(x=col, y=col2, hue="BAD\_LOAN", data=df)

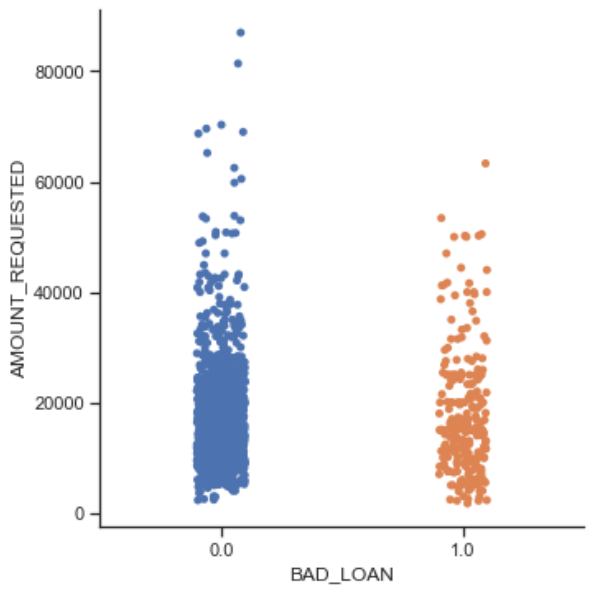
#show plot

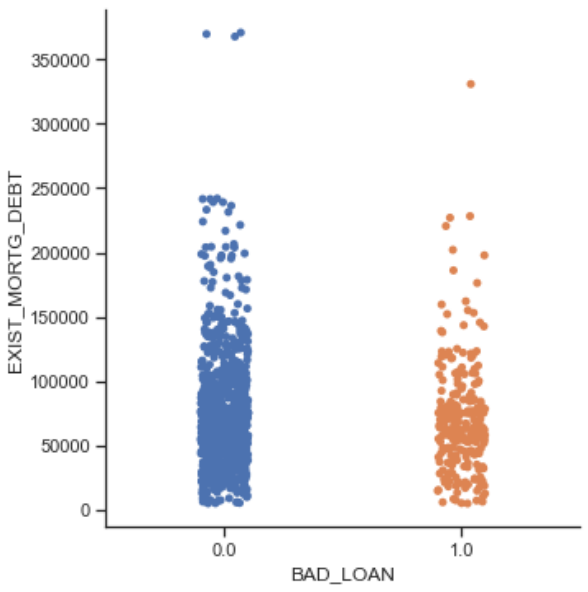
plt.show()

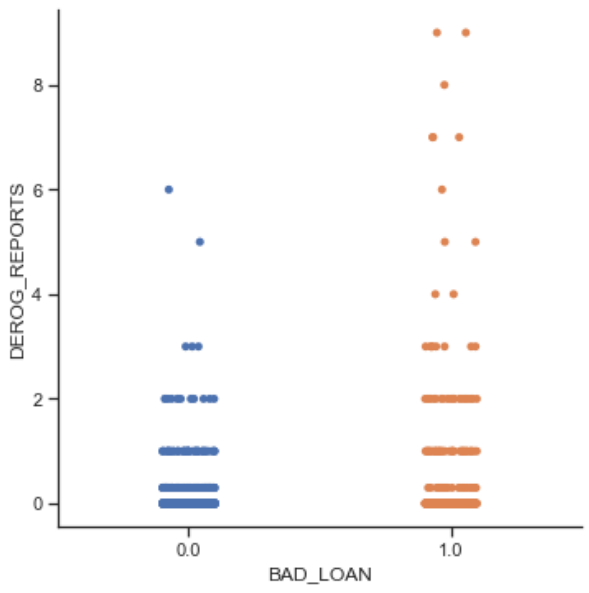
Bad loans - “1” is the loan was defaulted on. “0” is the loan was repaid.

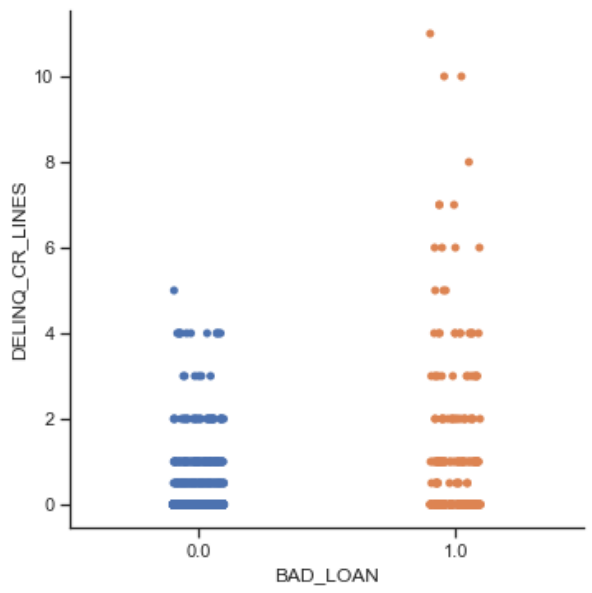
EMPLOYED\_YEARS: Below is an example for employed years. You can see that there is a slight correlation with years employed and loan repaid “0”. probably not that surprising. I had to use a strip-plot as the data in the features was overlapping and causing an error or loss of graphic.

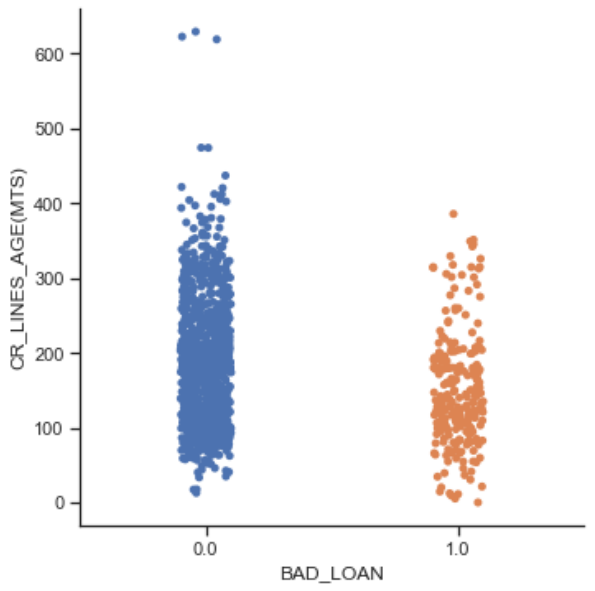


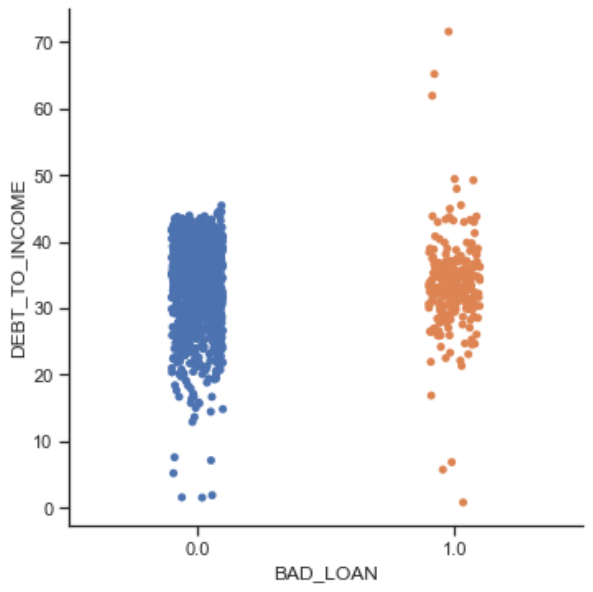












For the next part of the discussion I created nested for-loops, where I looped through each column against each column to examine relationships. Code:

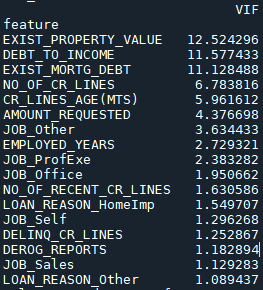
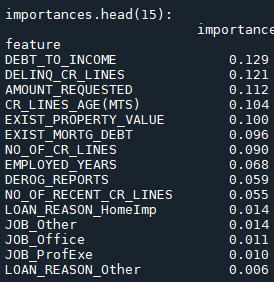
for col in df.columns:

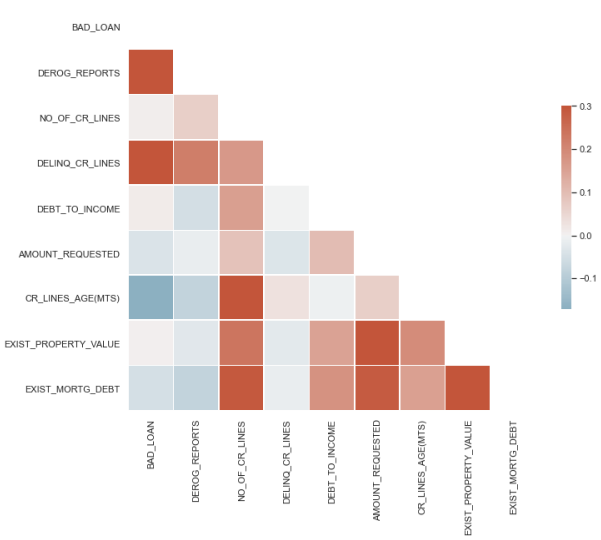
for col2 in df.columns:

sns.relplot(x=col, y=col2, hue="BAD\_LOAN", data=df)

plt.show()

In order to focus the discussion I will use the following two pieces of information (feature VIF, Importance), I will take the top 5 items from each list and review their relationship to each other and the target in a heat map below.



Model evaluation - accuracy alone// saving a model is evaluating in case the crash

Takes time, energy and error crahs

# References

1. k-fold-cross-validation

<https://www.askpython.com/python/examples/k-fold-cross-validation>

1. train-test-split

<https://www.bitdegree.org/learn/train-test-split>

1. cross\_validation

<https://scikit-learn.org/stable/modules/cross_validation.html>

1. KNNImputer

<https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html>

1. detecting-multicollinearity

<https://towardsdatascience.com/everything-you-need-to-know-about-multicollinearity-2f21f082d6dc>

<https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/>

1. Feature Importance

<https://www.kaggle.com/niklasdonges/end-to-end-project-with-python>

1. how-to-save-and-load-ml-models

<https://www.kaggle.com/prmohanty/python-how-to-save-and-load-ml-models>

1. Evaluation metrics

<https://www.jeremyjordan.me/evaluating-a-machine-learning-model/>

1. Model evaluation

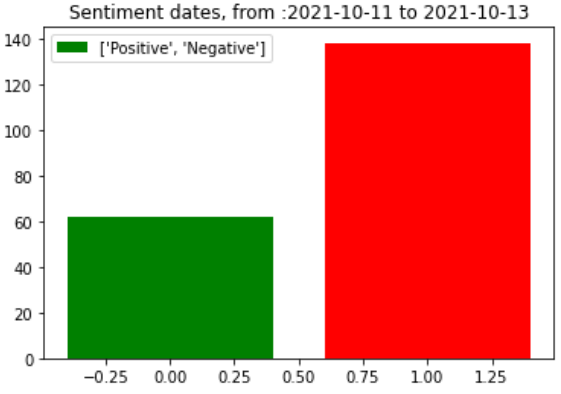
<https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>

1. Model evaluation from scikit

<https://scikit-learn.org/stable/modules/model_evaluation.html>

# Appendix 1 - API - Twitter

Below is a very interesting code that I have used in other projects. I return the sentiment which can be positive or Negative against a key work. In this case I put in **“AI”** and the search dates where start = '2021-10-11' to end = '2021-10-13'. Twitter only allow you go back 7 days of data and collect a max of 3500 calls in a certain period. I am currently using this to look at Bitcoin sentiment and used it in another project I did with UCD introduction to data Analytic. I am collecting sentiment data but did not have enough to use in this machine learning project. The program produces a plot of Positive versus Negative sentiment. The API code is quite simple and it is amazing how much you can get done in python with two lines of code. Blow is the out for the search on “AI” and comment in the code of what it is doing.



This is where I got the code from DogeCoi8n Sentiment anaylsis.

<https://www.youtube.com/watch?v=dSOUd9Sm1gI>

import tweepy

import textblob

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import re

I got keys by logging onto Twitter and creating a developers account.

# keys and tokens from the Twitter Dev Console

consumer\_key = ''

api\_key = consumer\_key

consumer\_secret = 'b'

api\_key\_secret = consumer\_secret

access\_token = '9'

access\_token\_secret = 'tI'

I used this to hide my keys which I put in a file in the same location as I opened the python file from

# all\_keys = open('twitterKeys','r').read().splitlines()

# api\_key = all\_keys[0]

# api\_key\_secret = all\_keys[1]

# access\_token = all\_keys[2]

# access\_token\_secret = all\_keys[3]

This handle opening the API link and handshakes

#Connecting to API

authenticator = tweepy.OAuthHandler(api\_key,api\_key\_secret)

authenticator.set\_access\_token(access\_token, access\_token\_secret)

This opens the communication channel

#Create the API

TwitterAPI = tweepy.API(authenticator,wait\_on\_rate\_limit=(True))

I set the Topic to search for here and the look back peroid

#Topic to check for sentiment

topic = 'AI'

search = f'#{topic} - filter:retweets'

#search peroid - look back max 7 days

start = '2021-10-11'

end = '2021-10-13'

We make the search here and the number of tweets to search - you have a max search so you have to spread of the look back peroid

# createing cursor

tweet\_cursor = tweepy.Cursor(TwitterAPI.search, q=search, lange='en', until=end, since=start, tweet\_mode='extended').items(200)#since goes back 7 days

This collects the Tweets

#getting tweets

tweets = [tweet.full\_text for tweet in tweet\_cursor]

Now we are putting them in to a data-frame and using REGEX to sub out characters (@, #) we do not want for the Tweets data-frame.

#turn in to data frame and get polarity

tweets\_df = pd.DataFrame(tweets, columns=['Tweets'])

#clean data frame - remove uneccary part of tweets that dont help with sentiment analysis

Using a for loop we iterate through the rows

for \_,row in tweets\_df.iterrows():

row['tweets'] = re.sub('http\s+', '', row['Tweets'])#substitute out

row['tweets'] = re.sub('#\s+', '', row['Tweets'])#substitute out

row['tweets'] = re.sub('@\s+', '', row['Tweets'])#substitute out

row['tweets'] = re.sub('\\n', '', row['Tweets'])#substitute out

Using textblob we identify the sentiment of the tweets

#identift the number of positive and negative tweets with detail above substituted out

tweets\_df['Polarity'] = tweets\_df['Tweets'].map(lambda tweet:textblob.TextBlob(tweet).sentiment.polarity)

tweets\_df['Result'] = tweets\_df['Polarity'].map(lambda pol: '+' if pol > 0 else'-')

I am creating the plot of the sentiment below, first get the counts and second plot it

#Count number of negative and positive tweets

positive = tweets\_df[tweets\_df.Result == '+'].count()['Tweets']

negative = tweets\_df[tweets\_df.Result == '-'].count()['Tweets']

#Create plot

plt.bar([0,1], [positive, negative], label=['Positive', 'Negative'], color=['green', 'red']) # colort positive = green and neg = red

plt.legend()

plt.title (f'Sentiment dates, from :{start} to {end}')

plt.show()

# Appendix 2 - Regex

The files associated with this are;

* 'S1\_Data\_Gathering\_Titanic.csv' is input file
* Regex test.py is the project file
* 'S1\_Data\_Gathering\_Regex\_output.csv

Below is an example of code I created to use REGEX to get the prefix of the passengers for the TITANIC data. This is so that I so that I could assign an age base on the prefix for missing rows. REGEX was not required in this project so I am using this as an example for the project point required.

1st I imported the file and set the strings to upper case to remove any errors with capitalization.

I used REGEX to get the Prefix. First I subtituted the string with “” up to the comm and the substittued the string with “” after the full stop. This was then save to a list called prefix. I then use the set function to set the unique contents of the list

2nd I created a new column for each of the prefixs. I used the contains function to put a true where the prefix was contained in the “Name” column.

3rd I created a for loop to iterate through the list of prefixs and if the prefix was the same as a preset prefix like “Mr” or “Mrs” I loaded the data in the “Age” column at that row to the prefix column.   
I used <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html> to help with this. I created different if conditional statement for each prefix. This would be better achieved with a function as the code is repeated had I more time.

4th I then got the mean age for each of the prefixs. Basically the mean of the column. This was my output

# Mean age on Titanic: 29.7

# imputed\_age\_Mr: 32.4

# imputed\_age\_Mrs: 35.9

# imputed\_age\_Miss: 21.8

# imputed\_age\_Don: 40.0

# imputed\_age\_Master: 4.8

5th I This step was basically looping through the rows of “Age” and where they were empty inserting the mean for that prefix. If the prefix was not available I used the mean of the column. I had to reset the True and False of the prefix column as this was used in the conditional check. I also had to set the column location by the name

col\_no\_ID = df.columns.get\_loc('PassengerId')

As when the program would activate the columns would shift etc and this would put the program in error so I made them dynamic. There is not doubth in my mind that tere is an easire way to achieve what I have achieve in much less code and on relection I could have used function to reduce the number of rows required.

Output of changes to missing agaes, you can see that it reads in the name, location and empty age cell “nan” and repeats with the mean age for that prefix in place of the “nan”.

1111111: 693 LAM, MR. ALI nan

Changed to: 693 LAM, MR. ALI 32.4

3333333: 698 MULLENS, MISS. KATHERINE "KATIE" nan

change to: 698 MULLENS, MISS. KATHERINE "KATIE" 21.8

5555555: 710 MOUBAREK, MASTER. HALIM GONIOS ("WILLIAM GEORGE") nan

change to: 710 MOUBAREK, MASTER. HALIM GONIOS ("WILLIAM GEORGE") 4.8

Code:

import pandas as pd

import numpy as np

import re

#import files for project - Load df from file

filename = 'S1\_Data\_Gathering\_Titanic.csv'

df = pd.read\_csv(filename)

print("\nLoaded df.shape: ", df.shape);print("\ndf.info(): ",df.info())

#to convert all strings in "Name"to upper case just case there is a mix in the data

df['Name'] = df['Name'].str.upper()

#=============================================================================

# 1st we make a list fo prefixs

#=============================================================================

# fill in missing ages using MR, Master, Mrs, Ms, Don, other. This is the use of REGEX for the Titanic data set to

# get a mena age appropiate to the prefix.

#find prefixs in Name column

prefix = [] # list to hold prefixes

#iterate through rows of dataframe column

for row in df['Name']:

# strip after comma - start of string

row = re.sub(r'^.\*,\s','',row)

# strip after full stop - end of string

row = re.sub(r'(?<=\.)[^.]\*$','',row)

#append prefix to last prefix in list - prefix

prefix.append(row)

#used for testing

# mylist = ['nowplaying', 'PBS', 'PBS', 'nowplaying', 'job', 'debate', 'thenandnow', 'impet', 'impet','impet']

# myset = set(mylist)

# print(myset)

#set() -> new empty set object set(iterable) -> new set object

#Build an unordered collection of unique elements.

unique\_prefix = set(prefix)

print(unique\_prefix)

#output was:

#{'JONKHEER.', 'MAJOR.', 'THE COUNTESS.', 'MR.', 'MISS.', 'MRS. MARTIN (ELIZABETH L.',

#'COL.', 'MME.', 'DR.', 'MS.', 'SIR.', 'LADY.', 'MASTER.', 'MRS.', 'CAPT.', 'REV.', 'DON.', 'MLLE.'}

# I am only selecting the following from the list for convienance - issue with would need to be resolved

list\_of\_prefixs = ('Mr','Mrs','Miss','Don','Master')

#=============================================================================

# 2nd we have to find the title and make a colum where its True or false for each prefix

#=============================================================================

#create a new column in dataframe with prefixs - find rows in `df` which contain r'\,\sTEXT.\s'

# df['Mr'] = (df[df['Name'].str.contains(r'\w,\sMR.\s\w')])# if upper() set

df['Mr'] = df['Name'].str.contains(r'\,\sMR.\s')

df['Mrs'] = df['Name'].str.contains(r'\,\sMRS')

df['Miss'] = df['Name'].str.contains(r'\,\sMISS.\s')

df['Don'] = df['Name'].str.contains(r'\,\sDON.\s')

df['Master'] = df['Name'].str.contains(r'\,\sMASTER\.\s')

#=============================================================================

#3rd interate through the list of prefixs and piut age in to the associated column prefix

#=============================================================================

for prefix in list\_of\_prefixs:

# if iterate is the same as prefix do code

if prefix == 'Mr':

# find the column index no for prefix

col\_no = df.columns.get\_loc(prefix)

#f find the column number for Age

col\_no\_age = df.columns.get\_loc('Age')

#using key and value take number for column with this prefix

#interate through the prefix column rows and is the value is true do condition

for index, value in df[prefix].items():

if value == True:

#https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html

#assign the value in the age column at that location to the prefix column

df.iloc[index, col\_no] = df.iloc[index, col\_no\_age]

if prefix == 'Mrs':

col\_no = df.columns.get\_loc(prefix)

col\_no\_age = df.columns.get\_loc('Age')

for index, value in df[prefix].items():

if value == True:

df.iloc[index, col\_no] = df.iloc[index, col\_no\_age]

if prefix == 'Miss':

col\_no = df.columns.get\_loc(prefix)

col\_no\_age = df.columns.get\_loc('Age')

for index, value in df[prefix].items():

if value == True:

df.iloc[index, col\_no] = df.iloc[index, col\_no\_age]

If prefix == 'Don':

col\_no = df.columns.get\_loc(prefix)

col\_no\_age = df.columns.get\_loc('Age')

for index, value in df[prefix].items():

if value == True:

df.iloc[index, col\_no] = df.iloc[index, col\_no\_age]

if prefix == 'Master':

col\_no = df.columns.get\_loc(prefix)

col\_no\_age = df.columns.get\_loc('Age')

for index, value in df[prefix].items():

if value == True:

df.iloc[index, col\_no] = df.iloc[index, col\_no\_age]

#=============================================================================

# 4th Get mean of Ages in prefix column

#=============================================================================

age\_mean = round((df["Age"].mean()), 1)

print("\nMean age on Titanic: ",age\_mean)

#problem with nan skewing results - Get ages for prefixs

df['Mr\_Age\_Numeric'] = pd.to\_numeric(df['Mr'], errors='coerce')

df['Mr\_Age\_Numeric'] = df['Mr\_Age\_Numeric'].replace(0, np.NaN)

imputed\_age\_Mr = round((df['Mr\_Age\_Numeric'].mean()), 1)

print("imputed\_age\_Mr: ", (imputed\_age\_Mr))

df['Mrs\_Age\_Numeric'] = pd.to\_numeric(df['Mrs'], errors='coerce')

df['Mrs\_Age\_Numeric'] = df['Mrs\_Age\_Numeric'].replace(0, np.NaN)

imputed\_age\_Mrs = round((df['Mrs\_Age\_Numeric'].mean()), 1)

print("imputed\_age\_Mrs: ", (imputed\_age\_Mrs))

df['Miss\_Age\_Numeric'] = pd.to\_numeric(df['Miss'], errors='coerce')

df['Miss\_Age\_Numeric'] = df['Miss'].replace(0, np.NaN)

imputed\_age\_Miss = round((df['Miss\_Age\_Numeric'].mean()), 1)

print("imputed\_age\_Miss: ", (imputed\_age\_Miss))

df['Don\_Age\_Numeric'] = pd.to\_numeric(df['Don'], errors='coerce')

df['Don\_Age\_Numeric'] = df['Don'].replace(0, np.NaN)

imputed\_age\_Don = round((df['Don\_Age\_Numeric'].mean()),1)

print("imputed\_age\_Don: ", (imputed\_age\_Don))

df['Master\_Age\_Numeric'] = pd.to\_numeric(df['Master'], errors='coerce')

df['Master\_Age\_Numeric'] = df['Master'].replace(0, np.NaN)

imputed\_age\_Master = round((df['Master\_Age\_Numeric'].mean()), 1)

print("imputed\_age\_Master: ", (imputed\_age\_Master))

#output:

# Mean age on Titanic: 29.7

# imputed\_age\_Mr: 32.4

# imputed\_age\_Mrs: 35.9

# imputed\_age\_Miss: 21.8

# imputed\_age\_Don: 40.0

# imputed\_age\_Master: 4.8

#=============================================================================

# 5th Loop through rows, if df['Age'] is empty insert age depending on prefix

#=============================================================================

#get column locations for

col\_no\_age = df.columns.get\_loc('Age')

col\_no\_Name = df.columns.get\_loc('Name')

col\_no\_ID = df.columns.get\_loc('PassengerId')

#iterate through the dataframe, row by row

for i in range(len(df)):

#if the age in the row is empty

if str(df.iloc[i, col\_no\_age]) == str(np.nan):

#assign True to each prefix and assin to variable

df['Mr'] = df['Name'].str.contains(r'\,\sMR.\s')

#assign prefix column location to variable

col\_no\_mr = df.columns.get\_loc('Mr')

df['Mrs'] = df['Name'].str.contains(r'\,\sMRS.\s')

col\_no\_mrs = df.columns.get\_loc('Mrs')

df['Miss'] = df['Name'].str.contains(r'\,\sMISS.\s')

col\_no\_miss = df.columns.get\_loc('Miss')

df['Don'] = df['Name'].str.contains(r'\,\sDON.\s')

col\_no\_don = df.columns.get\_loc('Don')

df['Master'] = df['Name'].str.contains(r'\,\sMASTER\.\s')

col\_no\_master = df.columns.get\_loc('Master')

#if the prefix colum is True and the age was empty

if df.iloc[i, col\_no\_mr] == True:

print("1111111: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

df.iloc[i, col\_no\_age] = imputed\_age\_Mr

print("Changed to: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

elif df.iloc[i, col\_no\_mrs] == True:

print("2222222: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

df.iloc[i, col\_no\_age] = imputed\_age\_Mrs

print("change to: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

elif df.iloc[i, col\_no\_miss] == True:

print("3333333: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

df.iloc[i, col\_no\_age] = imputed\_age\_Miss

print("change to: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

elif df.iloc[i, col\_no\_don] == True:

print("4444444: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

df.iloc[i, col\_no\_age] = imputed\_age\_Don

print("change to: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

elif df.iloc[i, col\_no\_master] == True:

print("5555555: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

df.iloc[i, col\_no\_age] = imputed\_age\_Master

print("change to: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

else:

#Catch any prefixs that are not above

print("Other: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

df.iloc[i, col\_no\_age] = age\_mean

print("change to: ", df.iloc[i, col\_no\_ID], df.iloc[i, col\_no\_Name], df.iloc[i, col\_no\_age])

print()

#save to file so that we can see the changes

filename1 = 'S1\_Data\_Gathering\_Regex\_output.csv'

df.to\_csv(filename1)

# Appendix 3 - Deep Learning

This section looks at how a deep learning model would perform at prediction on this data-set. To complete this I built a test program to find the optimal parameters and then used these to create my final model.

I used the clean data set to find the model parameters from this data set saved from Step 4;

'S4\_Loan\_Optimised\_Data\_Cleaning.csv'

and used the validation data from Step 7, to test the model on clean data.

'S7\_Loan\_Validation\_Optimised\_Data\_Cleaning.csv'

I was surprised how hard it was to create a working model. I was able to get the accuracy to about 80% but it was only looking at the classification report and Confusion Matrix it was obvious some thing was not working. Data set used contained 931 repaid 240 defaulted. So really the model was just saying every one repaid which made the model approximately 80% correct.

Accuracy\_score() is: 0.795

Confusion Matrix:

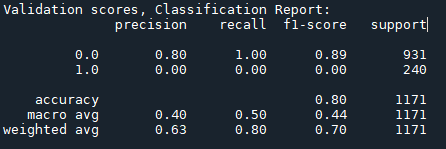
True negatives - correctly classified as not Target: 931

False negatives - wrongly classified as not Target: 0

False positives - wrongly classified as Target: 240

True positives - correctly classified as Target: 0

Confusion Matric - Accuracy: 0.7950469684030743



Why was I not getting any True Positive results? This led me to writing a program to loop through various parameters of Learning rate, Batch size and Hidden Layers.

# Appendix 4 - Seven points and page numbers

This is the page locations for the seven points below;

1. Real-world scenario

○ The project should use a real-world dataset and include a reference of their source in the report (Page)

1. Importing data

○ Your project should make use of one or more of the following: Relational database, API or web scraping (Twitter API - Page)

○ Import a CSV file into a Pandas DataFrame (Page)

3. Analysing data

○ Your project should use Regex to extract a pattern in data (Page)

○ Replace missing values or drop duplicates (Page)

○ Make use of iterators (Page)

○ Merge DataFrames (Page)

4. Python

○ Define a custom function to create reusable code (Page)

○ NumPy (Page)

○ Dictionary or Lists (Page)

5. Machine Learning

○ Predict a target variable with Supervised algorithm

○ Perform hyper parameter tuning (Page)

6. Visualise

○ Present two charts with Seaborn or Matplotlib (Page)

1. Generate valuable insights (Page)