# Project Report

Michael Impey - E10569412

This is submission is 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>

Catalog

[Project Report 1](#_Toc29823)

[Abstract 2](#_Toc22664)

[Introduction 2](#_Toc23534)

[Data-set 3](#_Toc19241)

[Implementation Process (Describe your entire process in detail) 4](#_Toc14692)

[Step.0 - Project\_Problem 6](#_Toc607)

[Step.1 - Data\_Gathering 7](#_Toc22034)

[Step.2 - Basic\_Data\_Cleaning 12](#_Toc20990)

[Step.3 - Baseline\_Model\_Testing 17](#_Toc12384)

[Step.4 - Optimise\_Data\_Cleaning 22](#_Toc14653)

[Step.5 - Tune\_Model\_and Select 29](#_Toc9033)

[Step.6 - HyperTune\_model 30](#_Toc17709)

[Step.7 - Optimise\_Data\_Cleaning 31](#_Toc13721)

[Step.8 - Model\_Predict 32](#_Toc16250)

[Results 44](#_Toc14357)

[Insights 45](#_Toc21630)

[References 46](#_Toc20940)

# Abstract

The project attempt to build a supervised machine leaning 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 fullfill the requirement of the course report, I have attached the examples and code that were not used specifically applicable in the project in the project report appendix. These items include WEB scrapping and Regex.

# 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 using predictive modeling.

# Data-set

(Provide a description of your dataset and source. Also justify why you chose this source)

The data file is from <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 defaulted or was seriously delinquent. This adverse outcome occurred in 1,189 cases (20%). For each applicant, 12 input variables were recorded.

I chose this data-set as it looked interesting to me as I am currently applying for a mortgage. 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 is a stock price is more probably to go up or down as opposed to a target price.

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 shown below and 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

(Describe your entire process in detail)

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 remaining elements of the report will be covered in there own section in the appendices.

You will notice that I like to load the data need for the next step and save the outputs of the step. I did this as it was very cumbersome and time consuming to run the project as one whole project and it also allowed me to test the impact of changes on one step on the next.

I used function where possible of where they made sense to me at the time. While this code was for this project I also wanted to make it re-usable and 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 outputs, impact of code and is implemented as just a wait for input command to proceed.

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 build on the previous step. I have attached a chart in the appendicies of how the steps interact.

|  |  |  |
| --- | --- | --- |
| 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 slipt the data to create a Train and Test data-set |
| 2 | Basic\_Data\_Cleaning | Perform EDA, basic data cleaning so that we can create basic data to run through a n umber of models to get initial 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 these trial I selected a model. |
| 6 | HyperTune\_model | On the selected model, Hyper-tune model and assess performance. |
| 7 | Optimise\_Data\_Cleaning | Perform basic and Optimised data cleaning on the Test data set |
| 8 | Model\_Predict | Perform prediction using the test training set on the hyper-tuned model and asses against a number of classifier metrics |

## 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 was trying to predict the clients attributes that would default on their loans. To do this I had to hypothesize what information would be required and what was available. Then there was the quality of the information available.

## Step.1 - Data\_Gathering

This step is where I import the original data, review it and prepare it for the following steps.

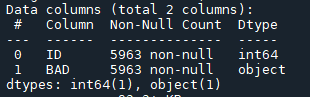
The step has the following actions:

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

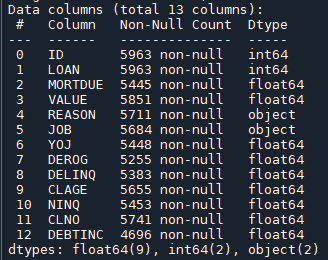
Initially, I saved the csv data as “loan\_bad\_Orignal.csv” to my desk top. This was then modified to create two files 'loan\_bad\_ID\_Target.csv’ and 'loan\_bad\_ID\_Features.csv' . I could then import and merge to make a project file. They were imported and merged on the ‘ID’ column. I created functions to complete the task as I plan to use this program as a temple 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 function. On this level its a little overkill but for me it helped with repeatability and standardizing my 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'

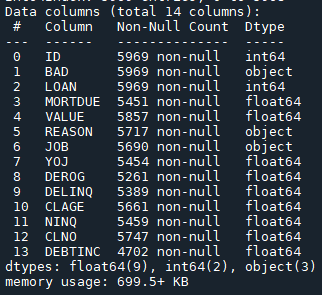
merge\_on = "ID"

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

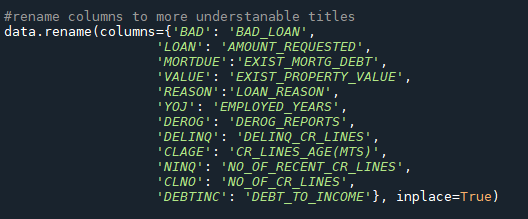
#make a copy in case we ruin orignal dataframe

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 the changes.



I then split this data-frame in to test and split. The reason was use 80% of the data train the model. This 80% of the orignal data was then used in various cross validation methons which we will review later. The other 20% was used to Test the hyper-parameter model. The following code shows how that was achieved and saved. I had an issue saving and reloading the files. The loaded was loading with a new index, to resolved this I use ‘index=False ‘ when saving;

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

#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)

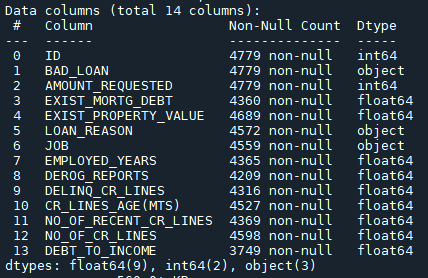
print("\n>>Saved train data.shape: ", train.shape);print(train.info())

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

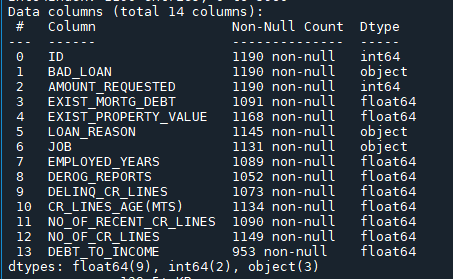
df = pd.read\_csv(filename2)

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

S1\_train\_Loan\_Basic\_Data\_Cleaning.csv



S1\_test\_Loan\_Basic\_Data\_Cleaning.csv



Step.2 - Basic\_Data\_Cleaning

This step used the data from the previous step.The over all aim of this step is to perform basic exploratory Data analysis. Perform basic data cleaning so that I can use this data in the next step to get performance results for a number of classifier ML models.

The step has the following actions;

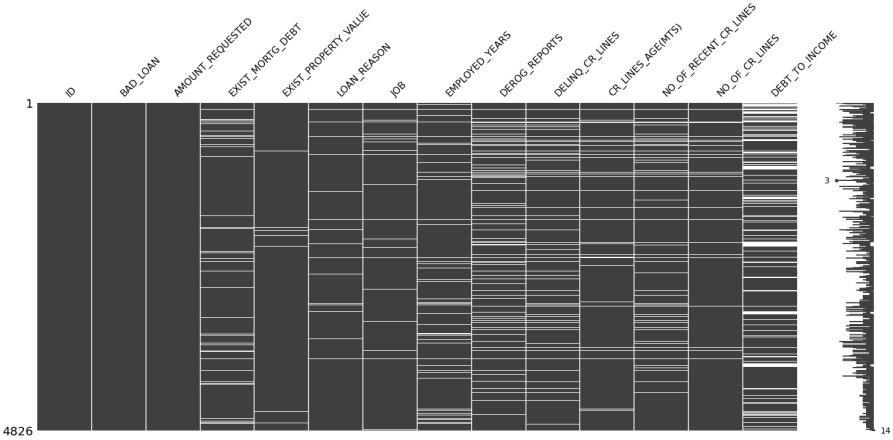
1. Import data and Perform basic Exploratory Data Analysis (EDA)
2. Perform Basic data cleaning
3. Save data-frame to use when Optimize Data Clean
4. Impute data for baseline models and Save data-frame for next step

Action 1 - Perform basic Exploratory Data Analysis

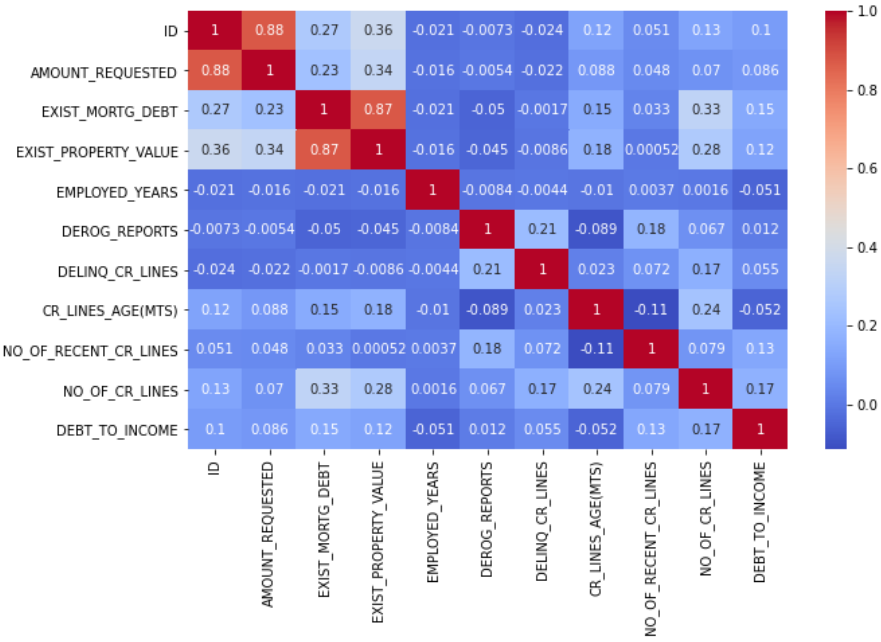
In this action I conducted these three checks;

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 dataframe, info on the dataframe and I completed a for loop where I itereated through the dataframe columns and printed out a description of the column content. I used describe(include=all) so I could get detail 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 missing-ness chart is an excellent way to see how much missing data we have and where and the heat map is an excellent first step in examining any possible correlations in the data. See below

Missing-ness chart:



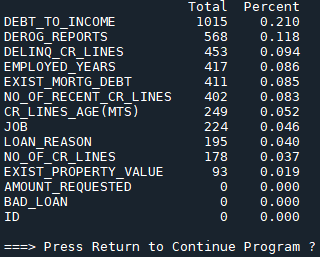
Heat map:



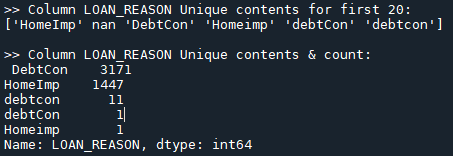
Action 2 - Perform Basic data cleaning

In this action I conducted the follow checks;

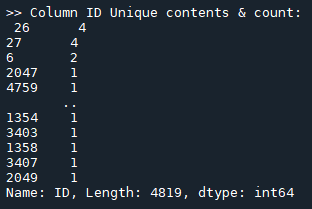
1. Descriptive review of the missing data from the data-frame. I ran a function to table all the columns and their content that was missing. The detail combined and noted for correction in the next actions. The table looked like this;



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 the column. This was especially important as I can see very quickly possible text errors for example miss-spelt “DebtCon” or “debtCon”, missing values “nan” as 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 standard the content of a number of columns. The purpose was to do the most obovious basic data clean. 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

This was the saved to a file 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 input data-frame into the optimize data clean step and I did not want the next input 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 for the baseline ML models to make a prediction with this will form my baseline model performance score. This 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. 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 check the for missing values before and after the change as 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.

## Step.3 - Baseline\_Model\_Testing

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

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

Action 1 - Transform the categorical variables

After importing, in this part I transformed the categorical features to numeric variables. 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 look at feature importance you can see that the Job features are lower at they are categorized.

The data-frame Column was;

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

I use the following code to achieve this 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 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 |

Action 2 - Create Target and Feature variables

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

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. Is this required??

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 in Kaggle. The idea was to send the X and y data and perform KFold cross validation against a number of classifier models. The function tests all the models for a for a specific KFold value, score them using cross\_val\_score and display there accuracy on the data. It would also check there accuracy agaist the know outputs. This was then loaded to a data frame I created and saved to a file "ML3\_Loans\_Models\_Results\_on\_Basic\_Data.csv". The idea in this steps was to get baseline data on the various model for their stand model parameters. In Step 5 of my process I would be repeating this test and wanted to compare the results. I am looking for an improvement in the models performance as it will be tested on Optimised data cleaning or so I hoped.

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:

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

# evaluate each model

for model in models:

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

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

scores = cross\_val\_score(model,X,y,scoring='accuracy', cv=cv, 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)+' CV Accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))

#Implement accuracy\_score() function

model.fit(X,y)

y\_pred = model.predict(X)

#Accuracy score on X

try:

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

print("Accuracy\_score() is: ", round(accuracy\_train, 3))

except:

print("accuracy score is void")

accuracy\_train = 999

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

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

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

'Accuracy\_Score': round(accuracy\_train,3)}, ignore\_index = True)

df\_model\_values.sort\_values(by=['Model\_Accuracy'], axis=0, ascending=False,

inplace=True, kind='quicksort',na\_position='last',

ignore\_index=False, key=None)

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

#Run All classifier model(s) test

#create a dataframe to capture model performance metrics

df\_model\_values = pd.DataFrame(data=None, columns = ['CV', 'Model', 'Model\_Accuracy', 'Model\_STD', 'Accuracy\_Score'])

Kfold\_start = 4

Kfold\_Stop = 11 #fold before thid intiger

df\_model\_values = Classifier\_models\_test(df\_model\_values, Kfold\_start, Kfold\_Stop)

print('\nSorted results for all models .head(20):\n',df\_model\_values.head(20))

The models I had to choose from where as follows. However I had an issue with my PC ruinning 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 appendix. We can se that the Random Classiifier perform the best at this point.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CV | Model | Model\_Accuracy | Model\_STD | Accuracy\_Score |
| 96 | 9 | RandomForestClassifier | 0.913 | 0.015 | 1 |
| 113 | 10 | RandomForestClassifier | 0.911 | 0.017 | 1 |
| 79 | 8 | RandomForestClassifier | 0.91 | 0.017 | 1 |
| 97 | 9 | ExtraTreesClassifier | 0.909 | 0.011 | 1 |
| 29 | 5 | ExtraTreesClassifier | 0.908 | 0.01 | 1 |
| 45 | 6 | RandomForestClassifier | 0.908 | 0.012 | 1 |
| 62 | 7 | RandomForestClassifier | 0.908 | 0.01 | 1 |
| 46 | 6 | ExtraTreesClassifier | 0.907 | 0.011 | 1 |
| 114 | 10 | ExtraTreesClassifier | 0.907 | 0.015 | 1 |
| 63 | 7 | ExtraTreesClassifier | 0.906 | 0.008 | 1 |
| 11 | 4 | RandomForestClassifier | 0.906 | 0.006 | 1 |
| 28 | 5 | RandomForestClassifier | 0.906 | 0.012 | 1 |
| 80 | 8 | ExtraTreesClassifier | 0.905 | 0.015 | 1 |
| 116 | 10 | GradientBoostingClassifier | 0.903 | 0.016 | 0.919 |
| 12 | 4 | ExtraTreesClassifier | 0.903 | 0.01 | 1 |
| 14 | 4 | GradientBoostingClassifier | 0.902 | 0.007 | 0.919 |
| 48 | 6 | GradientBoostingClassifier | 0.902 | 0.008 | 0.919 |
| 99 | 9 | GradientBoostingClassifier | 0.901 | 0.015 | 0.919 |
| 65 | 7 | GradientBoostingClassifier | 0.9 | 0.009 | 0.919 |
| 31 | 5 | GradientBoostingClassifier | 0.9 | 0.009 | 0.919 |
| 82 | 8 | GradientBoostingClassifier | 0.9 | 0.015 | 0.919 |
| 95 | 9 | BaggingClassifier | 0.899 | 0.015 | 0.99 |
| 44 | 6 | BaggingClassifier | 0.898 | 0.01 | 0.99 |
| 60 | 7 | AdaBoostClassifier | 0.897 | 0.012 | 0.904 |
| 78 | 8 | BaggingClassifier | 0.897 | 0.017 | 0.993 |
| 43 | 6 | AdaBoostClassifier | 0.897 | 0.011 | 0.904 |

Step.4 - Optimise\_Data\_Cleaning

This step used the data from step 2 before the imputed values where included. The purpose of this step was to try optimize the data further before trying the classifiers again where I hope to have an improved performance.

The 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 actyion where covered in step 2 - Basic\_Data\_Cleaning and data was imported in with basic cleaning

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 dat but I found that a row with up to 8 of 9 columns missing would not affect the results and would improve model performance

>> did it

I used this code to complete this. Basically 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 optimised the data based on box or historgram chart. I also measure what % of the data was missing.

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

I will discuss separately below how I imputed missing data.

>> it As this is a ration I was hoping link to profession.

“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

>> did it

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

I imputed the mean of the column.

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

I imputed the mean of the column.

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

I imputed the mean of the column.

“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.

#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%

I imputed the mean of the column.

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

I imputed the mean of the column.

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

I imputed the mean of the column.

“JOB” Total Percent missing 0%

I replace the nans with other so there was no empty features

“LOAN\_REASON” Total Percent missing 0%

I replace the nans with other so there was no empty features

Action 5 - Scatter plot of single Feature to Target

In this section these are the insights I gained

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 value for was that I could not find and column to indicate an income or was of back engineering the Debt-to-Income ratio. Used the following code and tested for n = 2 in a for loop????

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=9)

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

Action 7 - Checked for Multi-collinearity in features

I want to access the impact of similar feature 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/>

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)

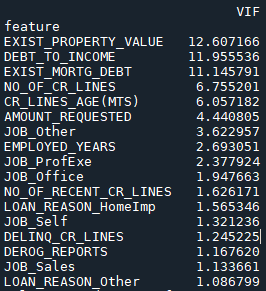
print(vif\_data)

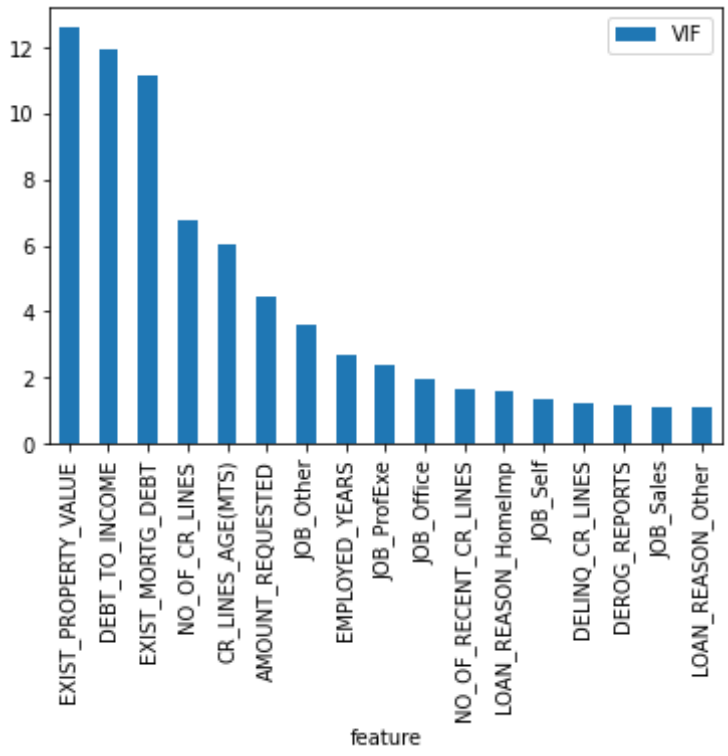
print(vif\_data.info())

vif\_data.plot.bar()

plt.show()

And the output was as follows;





Action 8 - Identify most important features

In this section I want to see what feature where having the most significant impact on the out of the model. 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()

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

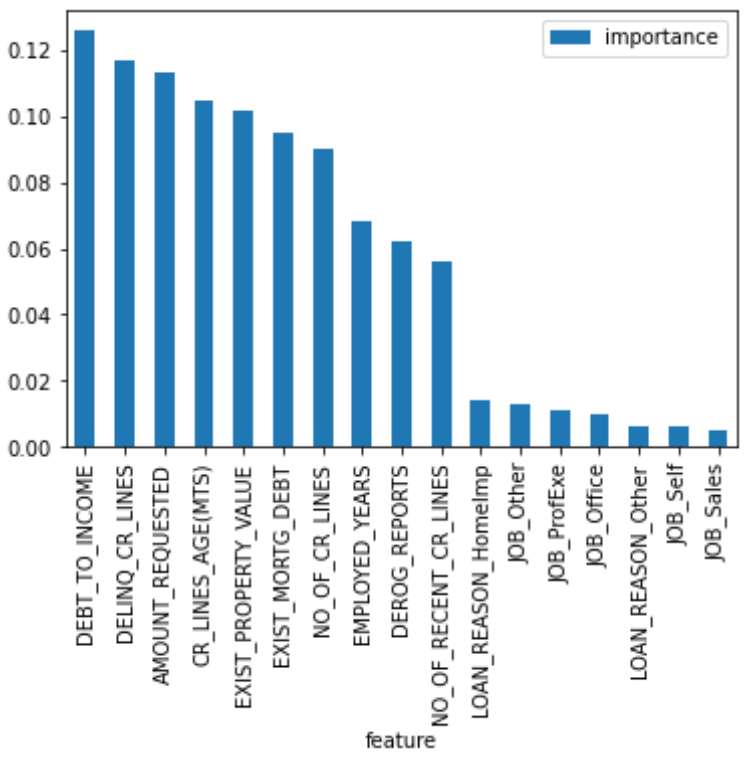
#Identiy important features

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

#call function to plot and describe important features

most\_important\_features(X,y)

This is what was found;



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

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

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 was as follows and the full detail can be seen in the appendix. We can see that the Random Classifier performs excellently but not the best at this the best at this point. It has shown an increase in performance of 1%.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CV | Model | Model\_Accuracy | Model\_STD | Accuracy\_Score |
| 80 | 8 | ExtraTreesClassifier | 0.939 | 0.011 | 1 |
| 114 | 10 | ExtraTreesClassifier | 0.939 | 0.012 | 1 |
| 46 | 6 | ExtraTreesClassifier | 0.938 | 0.011 | 1 |
| 97 | 9 | ExtraTreesClassifier | 0.937 | 0.013 | 1 |
| 29 | 5 | ExtraTreesClassifier | 0.936 | 0.009 | 1 |
| 63 | 7 | ExtraTreesClassifier | 0.936 | 0.012 | 1 |
| 12 | 4 | ExtraTreesClassifier | 0.931 | 0.013 | 1 |
| 113 | 10 | RandomForestClassifier | 0.925 | 0.013 | 1 |
| 62 | 7 | RandomForestClassifier | 0.924 | 0.013 | 1 |
| 96 | 9 | RandomForestClassifier | 0.923 | 0.013 | 1 |
| 28 | 5 | RandomForestClassifier | 0.922 | 0.008 | 1 |
| 79 | 8 | RandomForestClassifier | 0.921 | 0.012 | 1 |
| 45 | 6 | RandomForestClassifier | 0.919 | 0.011 | 1 |
| 11 | 4 | RandomForestClassifier | 0.917 | 0.011 | 1 |
| 95 | 9 | BaggingClassifier | 0.916 | 0.013 | 0.993 |
| 44 | 6 | BaggingClassifier | 0.91 | 0.014 | 0.991 |
| 112 | 10 | BaggingClassifier | 0.909 | 0.015 | 0.993 |
| 78 | 8 | BaggingClassifier | 0.907 | 0.014 | 0.992 |
| 10 | 4 | BaggingClassifier | 0.905 | 0.009 | 0.994 |

## Step.6 - HyperTune\_model

This step is similar to Step 3 and 5 ion that I am trying to run a model. Thsi time however I am only running 1 model and am going to try Hypertune its parameters. To colmete it I am going to use Grid Search Corss Validation. This wroks by….

The step has the following actions;

1. Prepocessing data
2. Hyper parameter Tuning
3. Saving best parameters
4. Classiifier performance after hyper tuning
5. Evaluating final model

The performance for the top clas

## Step.7 - Optimise\_Data\_Cleaning

This section I perform the basic and optimised data cleaning techniques to the Test dataframe that I created at the start of the project. ~this data will then be used to make the final prediction and check the accuracy of the model.

They hjave been previous;y been described but consists of the following

## Step.8 - Model\_Predict

This step uses the Test generated form Step 1 “Data\_Gathering” and optimised in Step 7 “Optimise\_Data\_Cleaning”. The purpose of doing this was to see how accurately can the model predict on clean data from the same data set. In this step we will alos review a number different away to evalust the model

The 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 Hypertuning. I was interested to see if any info was lost in the save / load processs.

The output from the code was;

{'criterion': 'gini', 'min\_samples\_leaf': 1, 'min\_samples\_split': 2, 'n\_estimators': 700}

===> Press Return to Continue Program ?

1.Parameters currently in use:

{'bootstrap': True,

'ccp\_alpha': 0.0,

'class\_weight': None,

'criterion': 'gini',

'max\_depth': None,

'max\_features': 'auto',

'max\_leaf\_nodes': None,

'max\_samples': None,

'min\_impurity\_decrease': 0.0,

'min\_impurity\_split': None,

'min\_samples\_leaf': 1,

'min\_samples\_split': 10,

'min\_weight\_fraction\_leaf': 0.0,

'n\_estimators': 100,

'n\_jobs': -1,

'oob\_score': True,

'random\_state': 1,

'verbose': 0,

'warm\_start': False}

And the code was;

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

#load model from file

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

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

# Model file name - Modle from current working directory

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

# Load the Model back from file

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

model = pickle.load(file)

#load best parameters from step 6 - GRID serach CV hypertuning

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

# Load the Model back from file

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

best\_params = pickle.load(file)

print(best\_params);pause()

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

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

# RandomForestClassifier parameters

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

#lets see parameters of model

print('\n1.Parameters currently in use:\n')

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

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 itys 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 and the saved or loaded file creates a new index column.

Action 3 - Import Hyper tuned model parameters and examine

For my own interest I imported the hyper tuned best\_params which I had saved and created a model RandomForestClassifier with these parameters. I used the following code and the purpose was just to compare how would a saved model that was reloaded compare to a model that was directly saved and re-loaded.

>> how did it comapre

Action 4 - Evaluate model predictions

Below are the a number of evaluation metric I used to acssess 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 model checks the prediction

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

print("y\_pred.shape: ",y\_pred.shape)

y\_test;print("y\_true.shape: ",y\_test.shape)

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

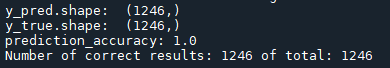
print("prediction\_accuracy:", round(prediction\_accuracy,2))

#normalize bool, default=True - #If False, return the number of correctly classified samples. Otherwise, return the fraction of correctly classified samples.

Num\_correct\_samples = accuracy\_score(y\_test, y\_pred, normalize=False)

print("Number of correct results: "+ str(Num\_correct\_samples) + " of total: " + str(len(y\_pred)));pause()

Output:



Method 2 - oob score

This score is …

#get oob score

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

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



Method 3 - Confusion Matrix

A confusion matrix gives you a lot of information about how well your model does.

The first row is about the not-survived-predictions:

779 passengers were correctly classified as not BAD\_LOAN (called true negatives) and

225 where wrongly classified as not BAD\_LOAN (false negatives).



The second row is about the survived-predictions:

123 passengers where wrongly classified as BAD\_LOAN (false positives) and

119 where correctly classified as BAD\_LOAN (true positives).

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])

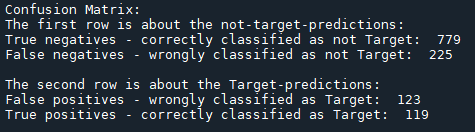
print("False negatives - wrongly classified as not Target: ",confusion\_matrix\_results[0][1])

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()

Method 1 - output



Method 4 - Precision and Recall / Precision Recall Curve

This Evaluation method says the model predicts 35% of the time, an individuals BAD\_LOAN correctly (precision).

The recall tells us that it predicted the BAD\_LOAN of 49 % of the people who actually Repaid the loan (Recall).

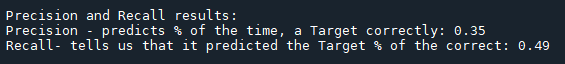
from sklearn.metrics import precision\_score, recall\_score

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 has to classify,

#it computes a probability based on a function

#and it classifies the person as survived (when the score is bigger the than threshold)

#or as not survived (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)

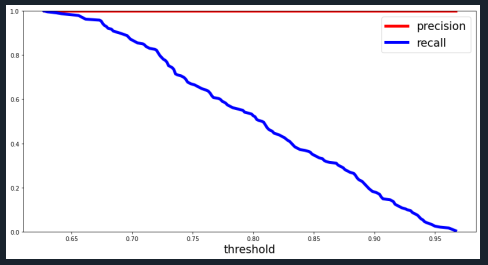
plt.ylim([0, 1])

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

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

plt.show();pause()

Output:



Method 5 - F-Score

The F-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.

from sklearn.metrics import f1\_score

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

f1\_score(y\_train, predictions);pause()

Output:



Method 6 - Classification\_report

This report…

#Classification report

from sklearn.metrics import classification\_report

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

print("Train scores:")

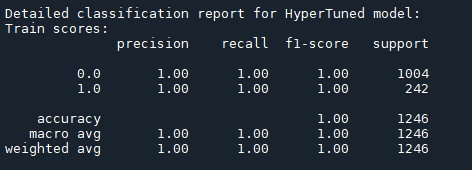
y\_pred = model.predict(X\_train)

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

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

#ROC AUC Curve - Another way to evaluate and compare your binary classifier

#is provided by the ROC AUC Curve. 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.

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])

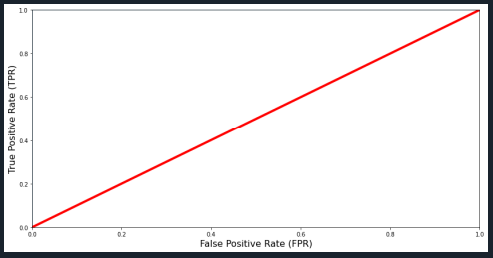
plt.xlabel('False Positive Rate (FPR)', fontsize=16)

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 classiffier would have a score of 0.5.

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)

# Insights

(Point out at least 5 insights in bullet points)

# References

(Include any references if required)