Configuration Manual

MSc Research Project
Data Analytics

Albert Winston
Student ID: X20136331

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…….Albert Winston…………………………………………………………………………………		
Student ID:	…x20136331…………………………………………………………………………………….……		
Programme:	…MSc Data Analytics……………………………	**Year:**	……2022…………
Module:	……………………Research Project…………………………………………………….………		
Lecturer:	…………………Dr Catherine Mulwa…………………………………………………….………		
Submission Due Date:	…………………15th August 2022…………………………………………………….………		
Project Title:	Assessment Methodology for Credit Models to Meet European Regulatory Expectations		
Word Count:	…………5571………………	**Page Count:**	…37……………………….……..………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Albert Winston ……………………………………………………………………………………………………

Date: 8 August 2022 ……………………………………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Albert Winston
X20136331:

1 Introduction

This document details the Project configuration for the MSc research project ‘Assessment Methodology for Credit Models to Meet European Regulatory Expectations’.

2 Hardware and Software Used

2.1 Hardware

The research project was developed on a custom build desktop PC. Additional analysis and documentation were competed using a Lenovo Legion 5 laptop. Specifications for both machines were as follows

Specification	Desktop PC	Laptop
Type	Custom	Lenovo Legion 5
CPU	AMD Ryzen 5 3600	AMD Ryzen 5 4600H
GPU	Nvidia RTX 3070Ti	Nvidia GTX1650
RAM	64GB	32GB
Operating system	Windows 11 Home 64 bit	Windows 11 Home 64 bit

Note that for full functionality a machine with 64GB RAM is the recommended minimum specification at present.

2.2 Software

Software used in the research included the following:

Software	Role
Windows 11 Home (64 bit)	Operating System
PyCharm Community edition	Development of Python code
Microsoft SQL Server	Development and housing of relational database
SQL Server Management Studio	Management of SQL database and testing of generated tables
Microsoft Power BI	Visualization of results in final reporting
MS Excel	Initial documentation of results, generate visuals for report
MS Word	Complete checks of results, verify outcomes
MS Edge	Research, development of report

R	Development of Power BI visualizations
Overleaf	LateX editor for report generation

2.3 Python Libraries

Within Python an extensive range of libraries were used to develop the research. All libraries used are listed below.

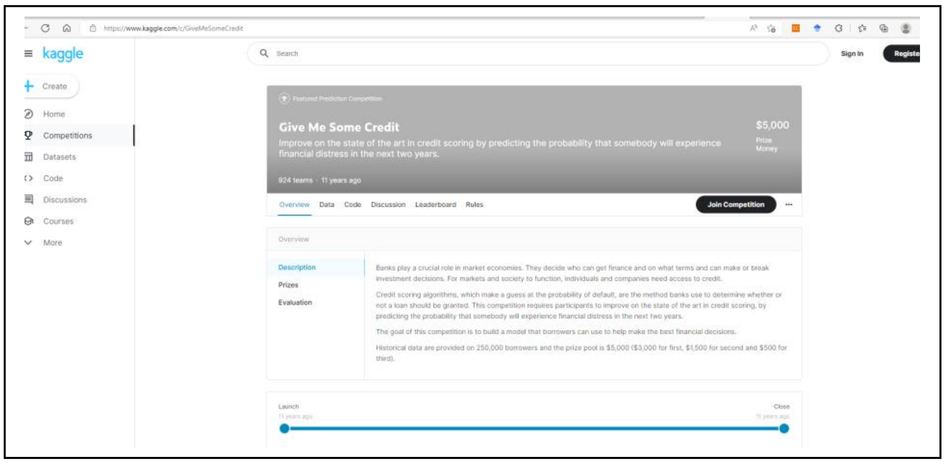
Primary purpose	Library	Modules
Data preparation and manipulation	Pandas	
	Numpy	
	Math	
Visualization	MatPlotlib	Pyplot
	Seaborn	
Modelling	Scikit-Learn	Preprocessing, Model_selection.train_test_split Linear_model.LogisticRegression Ensemble.RandomForestClassifier Ensemble.AdaBoostClassifier tree naïve_bayes.GaussianNB metrics.classification_report metrics.confusion_matrix metrics.roc_curve metrics.roc_auc_score metrics.precision_recall_curve metrics.auc metrics.f1_score metrics.precision_score metrics.recall_score model_selection.RandomizedSearchCV model_selection.GridSaerchCV
	logitboost	LogitBoost
	xgboost	xgb
	TensorFlow	
	Tensorflow_hub	
	Tensorflow.keras	Models, layers, callbacks, BatchNormalization
	Imb_learn	Over_sampling.SMOTE
	Scipy	
	optbinning	BinningProcess, Scorecard, Scorecard.plots
Post model processing	Pickle	
Model explanations	Shap	
Database interaction	Pyodbc	

General file interaction and control activity	Time	
	datetime	date
	os	
	joblib	
	pickle	
	zipfile	
	gc	
	pprint	

3 Methodology

3.1 Initial Data Preparation

Data was sourced from a publicly available credit scoring dataset housed at www.Kaggle.com. The dataset represents credit scoring data used in a Kaggle competition to create a model displaying the greatest possible predictive performance. The website link for the data is: https://www.kaggle.com/c/GiveMeSomeCredit A screenshot of the webpage used is shown in Figure 1



Give Me Some Credit

Improve on the state of the art in credit scoring by predicting the probability that somebody will experience financial distress in the next two years.

924 teams · 11 years ago

Banks play a crucial role in market economies. They decide who can get finance and on what terms and can make or break investment decisions. For markets and society to function, individuals and companies need access to credit.

Credit scoring algorithms, which make a guess at the probability of default, are the method banks use to determine whether or not a loan should be granted. This competition requires participants to improve on the state of the art in credit scoring, by predicting the probability that somebody will experience financial distress in the next two years.

The goal of this competition is to build a model that borrowers can use to help make the best financial decisions.

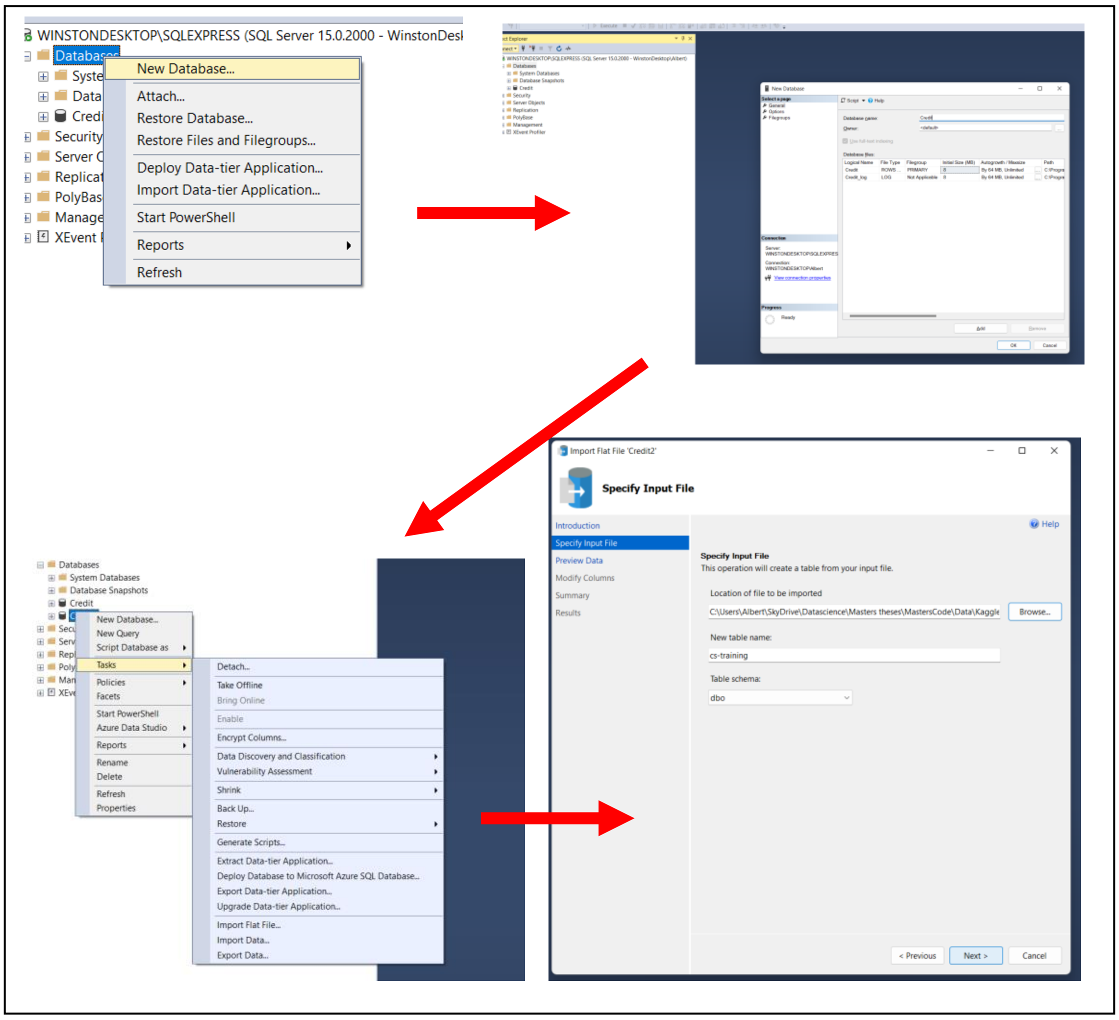
Historical data are provided on 250,000 borrowers and the prize pool is $5,000 ($3,000 for first, $1,500 for second and $500 for third).

Figure 1: Kaggle website for Dataset used in research

The data was sourced in .csv file format. Only the training dataset was used for the research as the test dataset contained on the site does not include the response variable, being intended only for generating predicted values for submission.

Based on this file a SQL server database was created using SQL Server Management Studio (SSMS) to store both the initial data and subsequent tables to be generated by the research as part of developing the credit model database. The initial database (1 table) was generated using the wizards available within MSMS as follows:

1. A new database named Credit was created using the SSMS wizards: The process is illustrated in the screenshots in Figure 2


WINSTONDESKTOP\SQLEXPRESS (SQL Server 15.0.2000 - WinstonDesl

Import Flat File 'Credit2'

Specify Input File

This operation will create a table from your input file.

Location of file to be imported: C:\Users\Albert\SkyDrive\Datascience\Masters theses\MastersCode\Data\Kaggle

New table name: cs-training

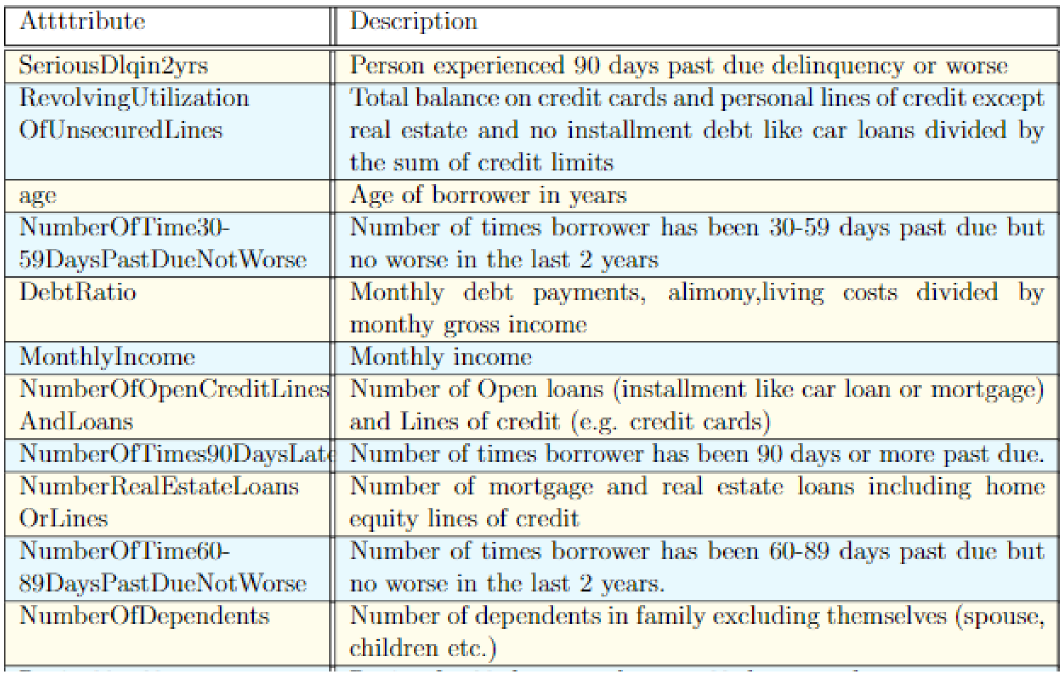
Table schema: dbo

Figure 2: Process to Create initial database

2. Default settings were maintained here to create a basic database table to use to extract the data in a (simple) replication of the process typically seen in industry where large EDW arrangements would be used to store risk data

The data stored in the table contains 10 independent variables and a response variable as detailed in Table 1 below

Table 1: Initial Variables for Credit Modelling dataset

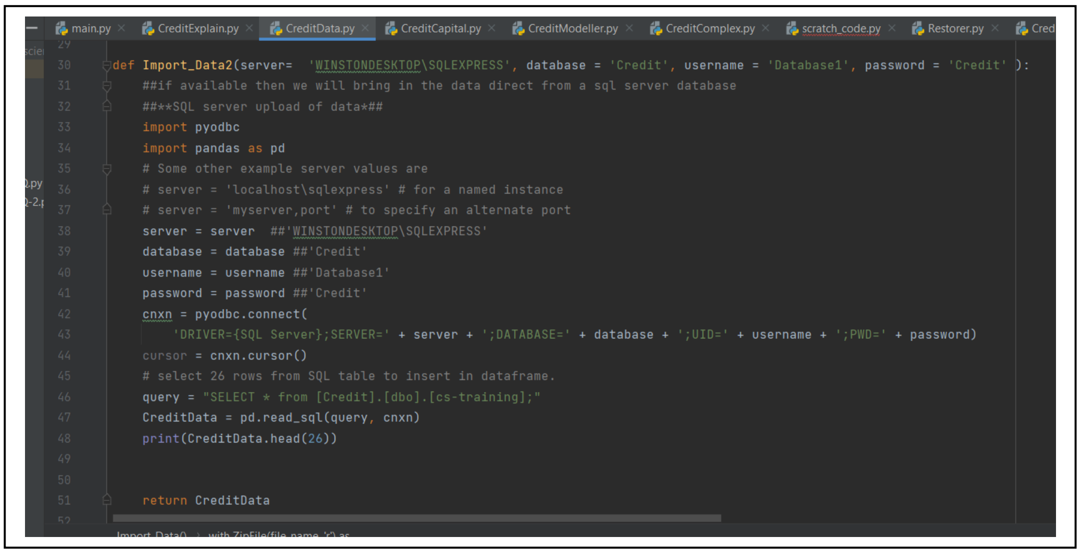


Attttribute	Description
SeriousDlqin2yrs	Person experienced 90 days past due delinquency or worse
RevolvingUtilization OfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits
age	Age of borrower in years
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years
DebtRatio	Monthly debt payments, alimony,living costs divided by monthy gross income
MonthlyIncome	Monthly income
NumberOfOpenCreditLines AndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)
NumberOfTimes90DaysLat	Number of times borrower has been 90 days or more past due.
NumberRealEstateLoans OrLines	Number of mortgage and real estate loans including home equity lines of credit
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)

A unique index column in also provided with the data. This index was used to form a unique identifier for each record within the database, equivalent to a customer number.

3.2 Data Ingestion

The data was ingested into the Python scripts used to carry out the analysis. A function (Import_Data2) was created to connect to the SQL server database using the pyodbc library and select the data from the table containing the dataset. The code function is shown in Figure 3

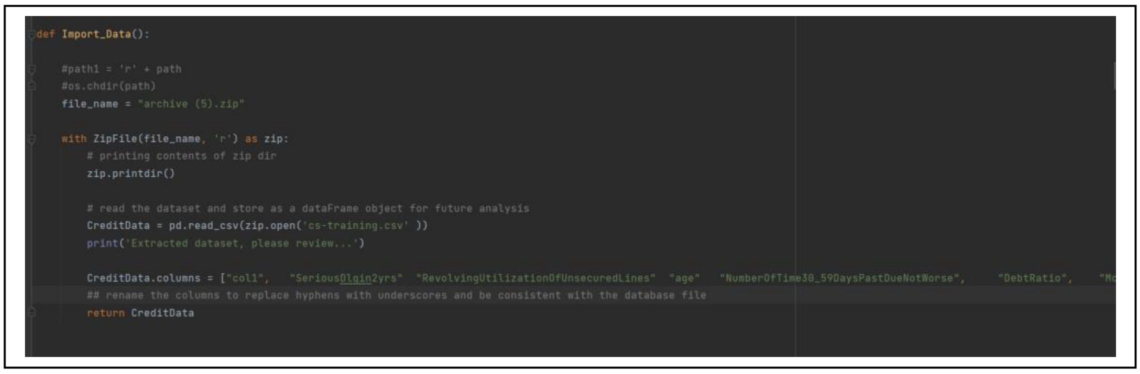


```
def Import_Data2(server=  'WINSTONDESKTOP\SQLEXPRESS', database = 'Credit', username = 'Database1', password = 'Credit' ):
    ##if available then we will bring in the data direct from a sql server database
    ##**SQL server upload of data*##
    import pyodbc
    import pandas as pd
    # Some other example server values are
    # server = 'localhost\sqlexpress' # for a named instance
    # server = 'myserver,port' # to specify an alternate port
    server = server  ##'WINSTONDESKTOP\SQLEXPRESS'
    database = database ##'Credit'
    username = username ##'Database1'
    password = password ##'Credit'
    cnxn = pyodbc.connect(
        'DRIVER={SQL Server};SERVER=' + server + ';DATABASE=' + database + ';UID=' + username + ';PWD=' + password)
    cursor = cnxn.cursor()
    # select 26 rows from SQL table to insert in dataframe.
    query = "SELECT * from [Credit].[dbo].[cs-training];"
    CreditData = pd.read_sql(query, cnxn)
    print(CreditData.head(26))


    return CreditData
```

Figure 3: Python function to bring in data from database

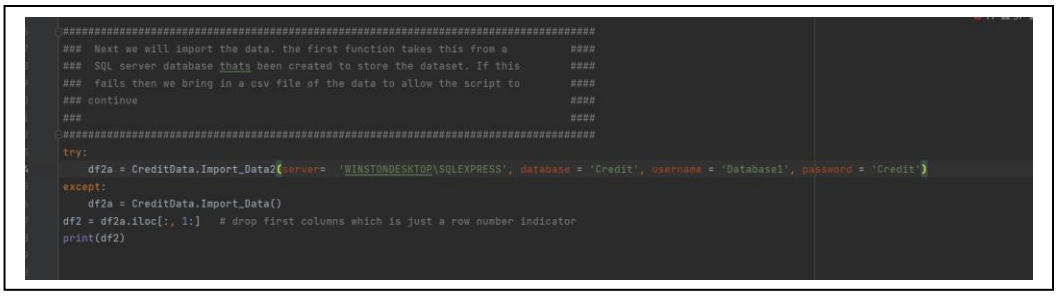
The process is parameterised with the details of the database, including Username and password. In practice this would be an expected level of security. For this database the username is set to ‘Database1’ and the password is ‘credit’.

To ensure data is successfully brought in a second code script was created to enable the data to be read directly from the original .csv file in cases where there is an error encountered in the database read. This is contained in the Import data function within the CreditData.py module, with the import code within the main.py module branching as required if an error is encountered. (Figure 4, Figure 5.). Following this Nan values are converted to 0’s for certain variables -this is to address cases where a null is used to indicate no arrears and 0 is the appropriate value to apply here.



```
def Import_Data():

    #path1 = 'r' + path
    #os.chdir(path)
    file_name = "archive (5).zip"

    with ZipFile(file_name, 'r') as zip:
        # printing contents of zip dir
        zip.printdir()

        # read the dataset and store as a dataFrame object for future analysis
        CreditData = pd.read_csv(zip.open('cs-training.csv' ))
        print('Extracted dataset, please review...')

        CreditData.columns = ["col1",   "SeriousDlqin2yrs"  "RevolvingUtilizationOfUnsecuredLines"  "age"   "NumberOfTime30_59DaysPastDueNotWorse",     "DebtRatio",    "Mo
        ## rename the columns to replace hyphens with underscores and be consistent with the database file
        return CreditData
```

Figure 4: Code to take data from original .csv file



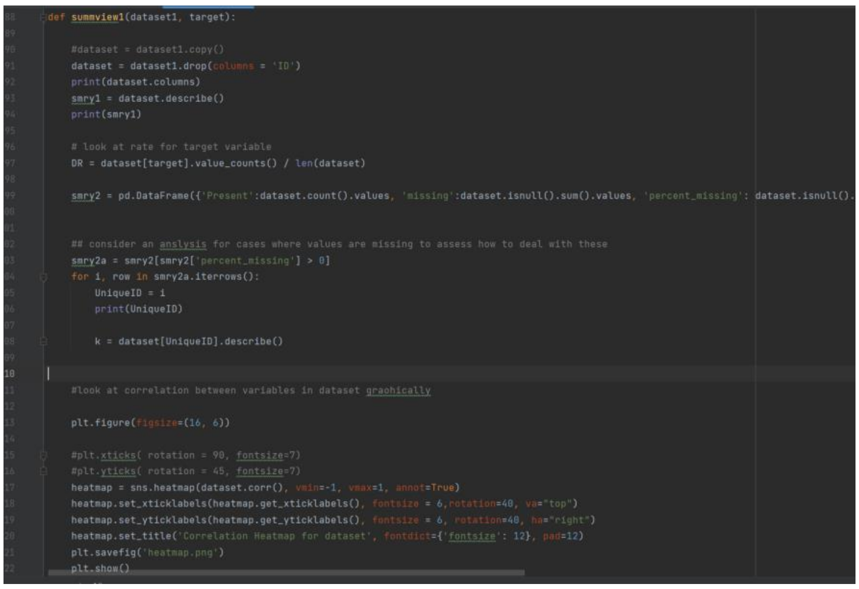
```
##########################################################################################
###  Next we will import the data. the first function takes this from a          ####
###  SQL server database thats been created to store the dataset. If this        ####
###  fails then we bring in a csv file of the data to allow the script to        ####
### continue                                                                      ####
###                                                                               ####
##########################################################################################
try:
    df2a = CreditData.Import_Data2(server=  'WINSTONDESKTOP\SQLEXPRESS', database = 'Credit', username = 'Database1', password = 'Credit')
except:
    df2a = CreditData.Import_Data()
df2 = df2a.iloc[:, 1:]   # drop first columns which is just a row number indicator
print(df2)
```

Figure 5: Code to bring in data within main.py

3.3 Exploratory Analysis and Feature Engineering

3.3.1 Exploratory Analysis of Data

The candidate variables for the model were subjected to Exploratory data analysis (EDA) using the code within the summview function within the CreditData.py module. Summview produces descriptive statistics for the variables in the dataset including producing a correlation heatmap of the variables



```python
def summview1(dataset1, target):

    #dataset = dataset1.copy()
    dataset = dataset1.drop(columns = 'ID')
    print(dataset.columns)
    smry1 = dataset.describe()
    print(smry1)

    # look at rate for target variable
    DR = dataset[target].value_counts() / len(dataset)

    smry2 = pd.DataFrame({'Present':dataset.count().values, 'missing':dataset.isnull().sum().values, 'percent_missing': dataset.isnull().


    ## consider an anslysis for cases where values are missing to assess how to deal with these
    smry2a = smry2[smry2['percent_missing'] > 0]
    for i, row in smry2a.iterrows():
        UniqueID = i
        print(UniqueID)

        k = dataset[UniqueID].describe()


    #look at correlation between variables in dataset graohically

    plt.figure(figsize=(16, 6))

    #plt.xticks( rotation = 90, fontsize=7)
    #plt.yticks( rotation = 45, fontsize=7)
    heatmap = sns.heatmap(dataset.corr(), vmin=-1, vmax=1, annot=True)
    heatmap.set_xticklabels(heatmap.get_xticklabels(), fontsize = 6,rotation=40, va="top")
    heatmap.set_yticklabels(heatmap.get_yticklabels(), fontsize = 6, rotation=40, ha="right")
    heatmap.set_title('Correlation Heatmap for dataset', fontdict={'fontsize': 12}, pad=12)
    plt.savefig('heatmap.png')
    plt.show()
```

Figure 6: The Summview function

The code also gives a visual indication of the predictive power of the candidate variables by producing boxplots and histograms of the variable, as well as creating decile bins for the variables and using these to calculate the weights of evidence and the event rates which are then plotted (Figure 7)

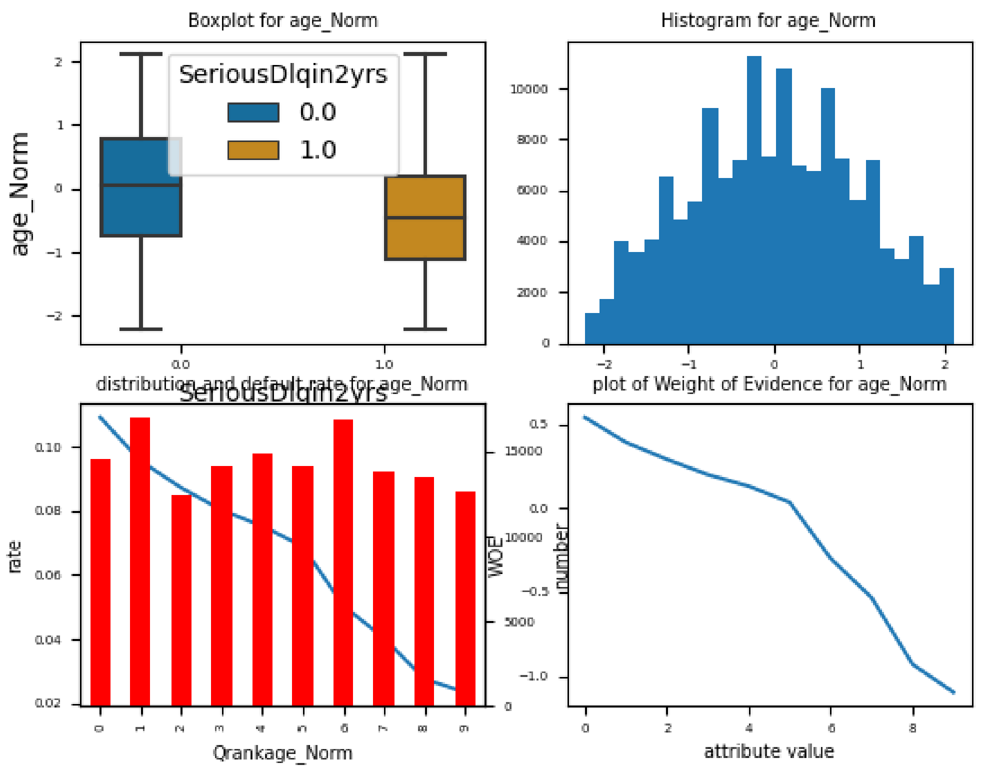


Figure 7: Example of summary plots produced by summview function

Based on this exploratory view of the data a number of issues were identified with the data that were subsequently addressed in the code. A series of rules were applied to the data to ensure that the credit logic was adhered to.

1. age must be between 18 and 80
2. monthly income must be less than 500k
3. NumberOfTime60-89DaysPastDueNotWorse - capped at 25 times
4. debt ratio must be < 200000 - above this is simply too high for any reasonable consideration
6. NumberOfTimes90DaysLate capped at 25 similar to 60-69 factor
7. RevolvingUtilizationOfUnsecuredLines typically should not be much greater than 1 given the ratios definition (cc may allow overdraw etc) - cap at 1.5

Note that in addressing these we have focused on a plausible business interpretation rather the theoretical form of the data – this research is intended to reflect a real-world modelling outcome and thus the guiding principle was plausibility (see Figure 8: Distribution pre and post cleaning).

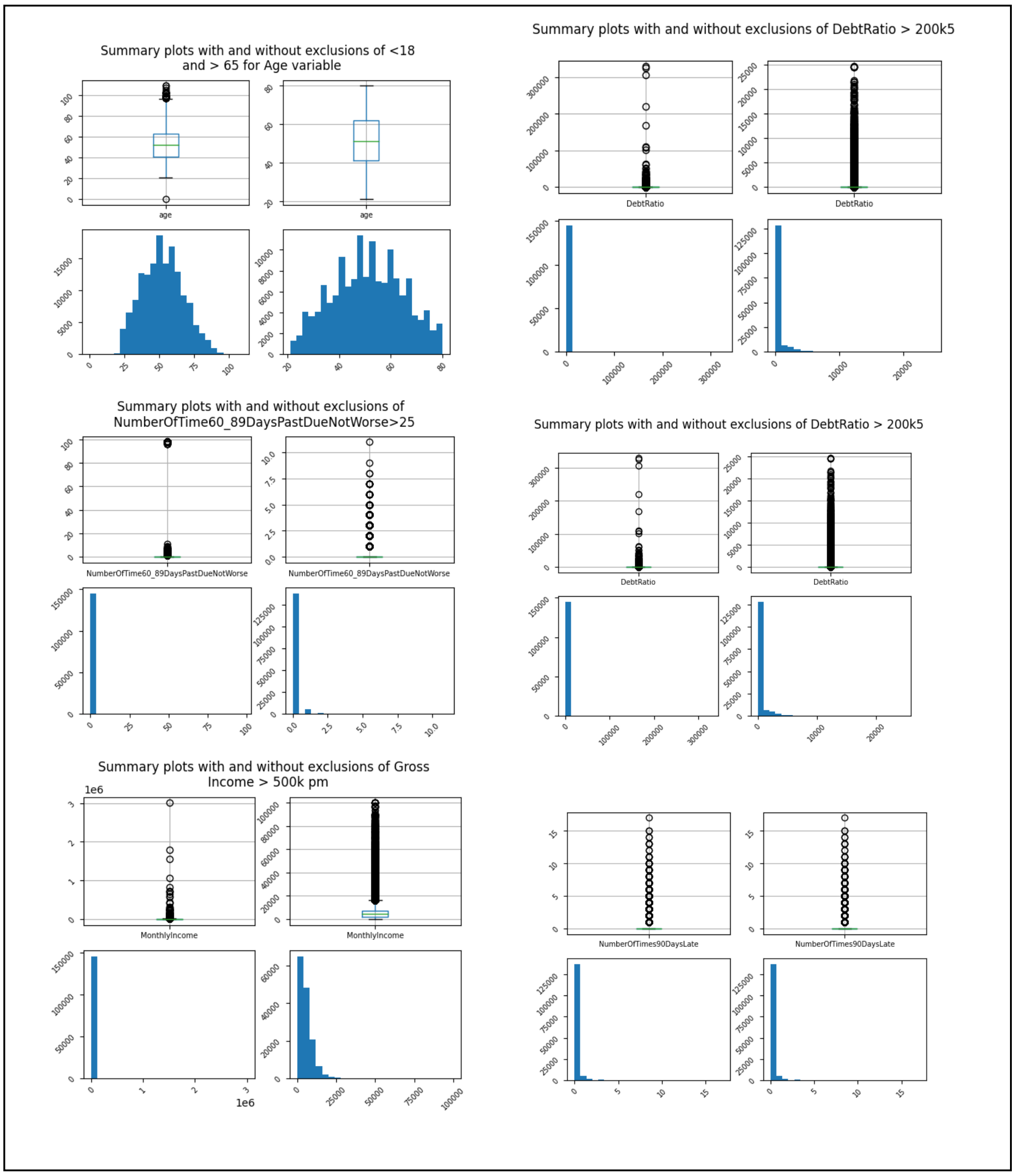
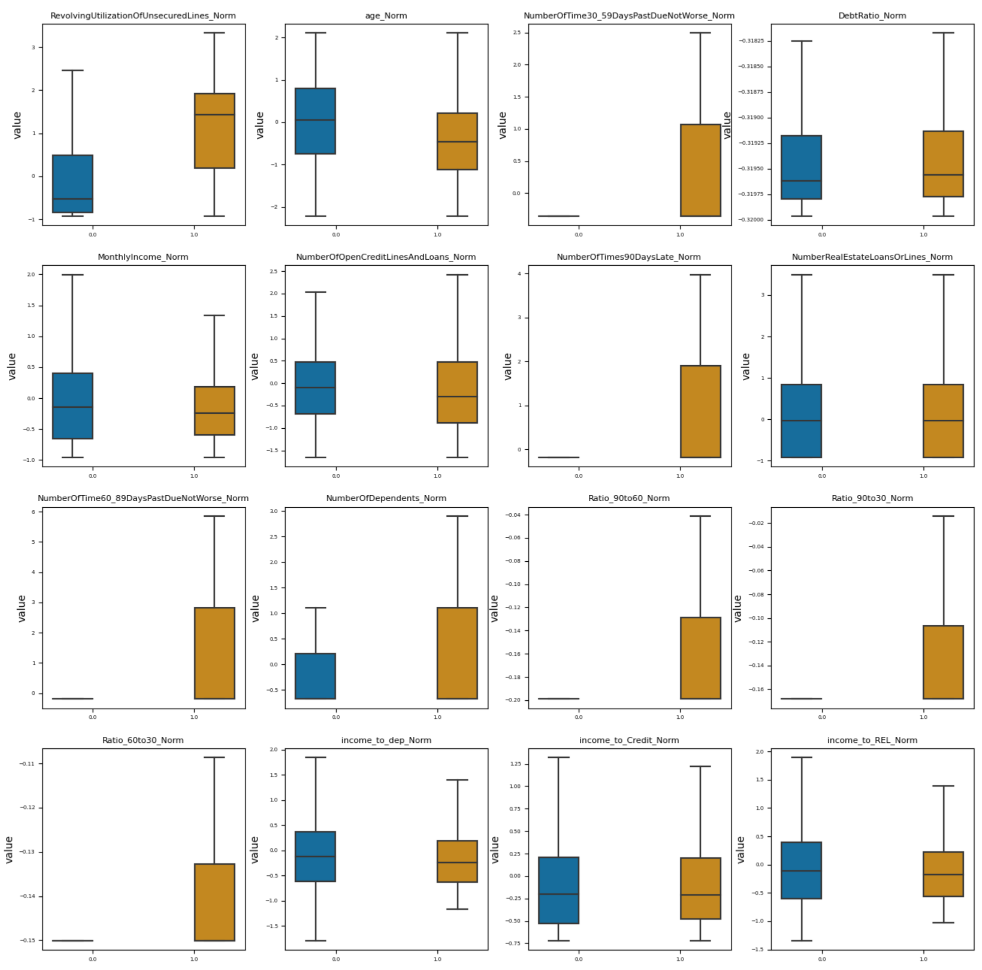


Figure 8: Distribution pre and post cleaning

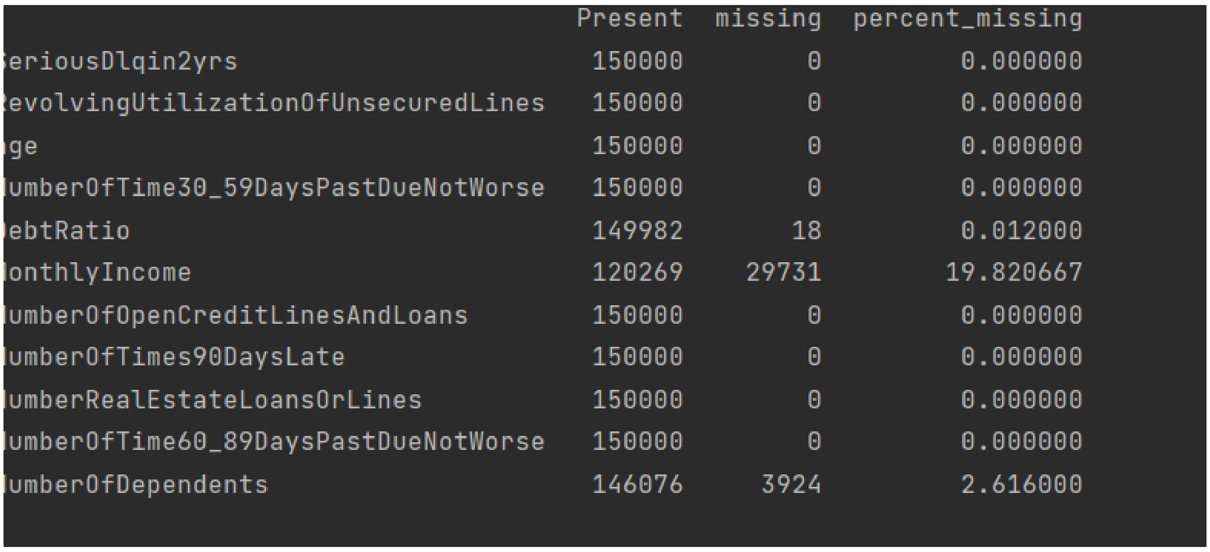
The boxplots of the variables for the two event rates were also considered to assess which factors appeared to be predictive.

Boxplots of all factors considered



3.3.2 Feature Imputation

On aspect revealed by the analysis was the presence of missing values for two variables. The level of missing MonthlyIncome information in particular is very high at just under 20%. While setting this to 0 was considered as a conservative approach, ultimately a MICE imputation was applied to the two variables as shown in Figure 9



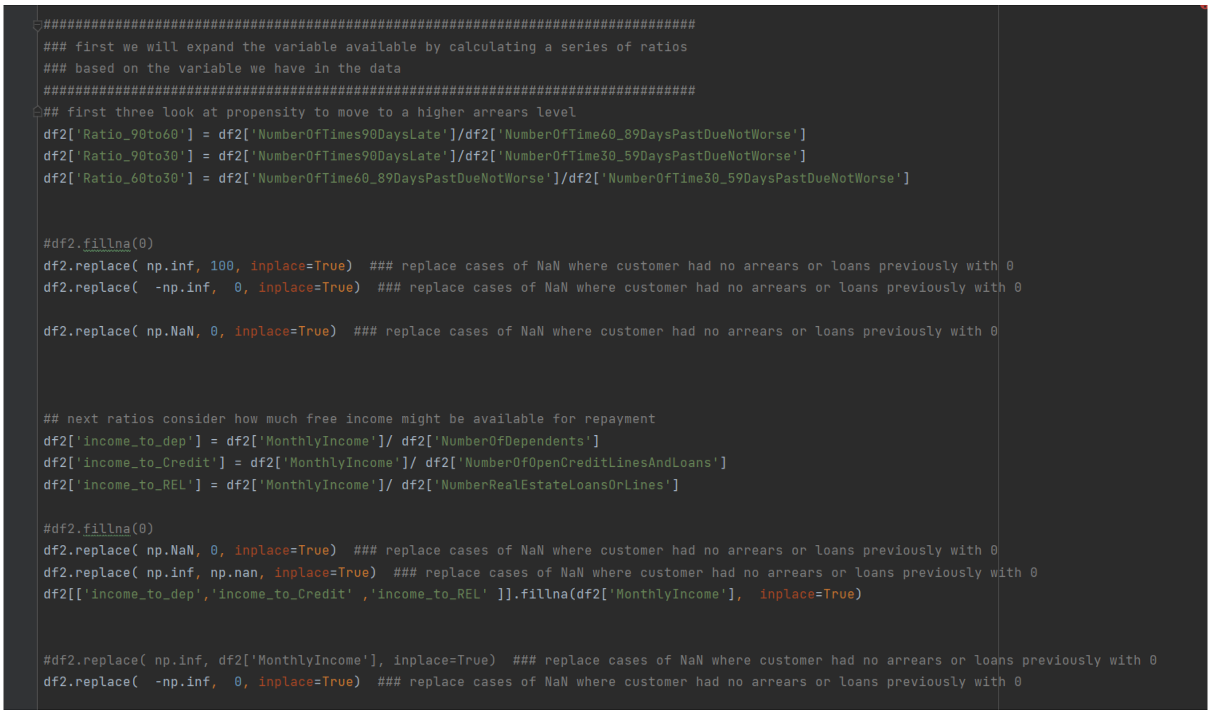
```
                                          Present  missing  percent_missing
eriousDlqin2yrs                           150000        0         0.000000
evolvingUtilizationOfUnsecuredLines       150000        0         0.000000
ge                                        150000        0         0.000000
umberOfTime30_59DaysPastDueNotWorse       150000        0         0.000000
ebtRatio                                  149982       18         0.012000
onthlyIncome                              120269    29731        19.820667
umberOfOpenCreditLinesAndLoans            150000        0         0.000000
umberOfTimes90DaysLate                    150000        0         0.000000
umberRealEstateLoansOrLines               150000        0         0.000000
umberOfTime60_89DaysPastDueNotWorse       150000        0         0.000000
umberOfDependents                         146076     3924         2.616000
```

Figure 9: Imputation code

In a small number of instances this imputation led to negative values for the imputed variables, and in this case the variable was set to zero.

3.3.3 Feature Generation

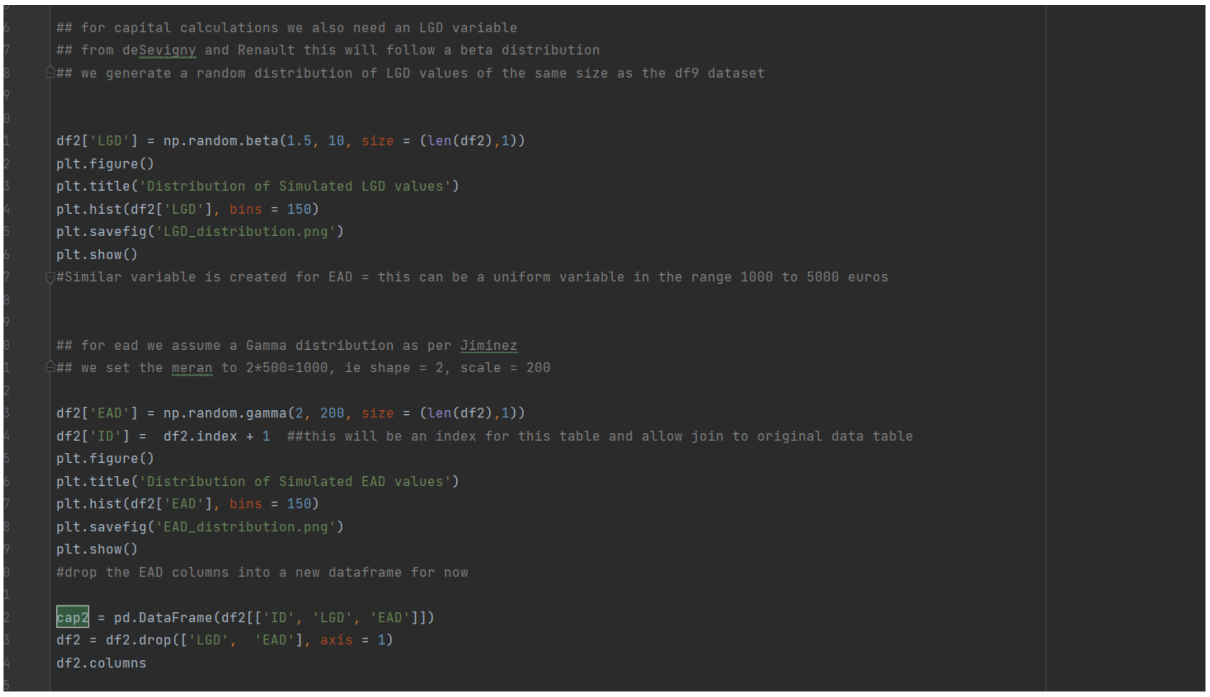
Next additional Ratio variables are created based on the data in the original dataset. These represent additional financial information for case, for example indicating how often a case has gone 90 days overdue relative to how often they went 60 days overdue and are added to see if they add additional value to the models. Some clean-up of the variables is also required here to address specific scenarios. This is shown in Figure 10:



```
##################################################################################
### first we will expand the variable available by calculating a series of ratios
### based on the variable we have in the data
##################################################################################
## first three look at propensity to move to a higher arrears level
df2['Ratio_90to60'] = df2['NumberOfTimes90DaysLate']/df2['NumberOfTime60_89DaysPastDueNotWorse']
df2['Ratio_90to30'] = df2['NumberOfTimes90DaysLate']/df2['NumberOfTime30_59DaysPastDueNotWorse']
df2['Ratio_60to30'] = df2['NumberOfTime60_89DaysPastDueNotWorse']/df2['NumberOfTime30_59DaysPastDueNotWorse']


#df2.fillna(0)
df2.replace( np.inf, 100, inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0
df2.replace(  -np.inf,  0, inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0

df2.replace( np.NaN, 0, inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0



## next ratios consider how much free income might be available for repayment
df2['income_to_dep'] = df2['MonthlyIncome']/ df2['NumberOfDependents']
df2['income_to_Credit'] = df2['MonthlyIncome']/ df2['NumberOfOpenCreditLinesAndLoans']
df2['income_to_REL'] = df2['MonthlyIncome']/ df2['NumberRealEstateLoansOrLines']

#df2.fillna(0)
df2.replace( np.NaN, 0, inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0
df2.replace( np.inf, np.nan, inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0
df2[['income_to_dep','income_to_Credit' ,'income_to_REL' ]].fillna(df2['MonthlyIncome'],  inplace=True)


#df2.replace( np.inf, df2['MonthlyIncome'], inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0
df2.replace(  -np.inf,  0, inplace=True)  ### replace cases of NaN where customer had no arrears or loans previously with 0
```

Figure 10: Generation of additional variables

We also need to simulate additional variables to enable us to calculate the capital requirements for an institution. Loss Given Default (LGD) and Exposure at Default (EAD) variables are therefore calculated using the code in Figure 11



```
## for capital calculations we also need an LGD variable
## from deSevigny and Renault this will follow a beta distribution
## we generate a random distribution of LGD values of the same size as the df9 dataset


df2['LGD'] = np.random.beta(1.5, 10, size = (len(df2),1))
plt.figure()
plt.title('Distribution of Simulated LGD values')
plt.hist(df2['LGD'], bins = 150)
plt.savefig('LGD_distribution.png')
plt.show()
#Similar variable is created for EAD = this can be a uniform variable in the range 1000 to 5000 euros


## for ead we assume a Gamma distribution as per Jiminez
## we set the meran to 2*500=1000, ie shape = 2, scale = 200

df2['EAD'] = np.random.gamma(2, 200, size = (len(df2),1))
df2['ID'] =  df2.index + 1  ##this will be an index for this table and allow join to original data table
plt.figure()
plt.title('Distribution of Simulated EAD values')
plt.hist(df2['EAD'], bins = 150)
plt.savefig('EAD_distribution.png')
plt.show()
#drop the EAD columns into a new dataframe for now

cap2 = pd.DataFrame(df2[['ID', 'LGD', 'EAD']])
df2 = df2.drop(['LGD',  'EAD'], axis = 1)
df2.columns
```

Figure 11: Code to simulate LGD and EAD dataset

The distributions used here follow the approach detailed within the literature, notably (de Servigny & Renault, 2004). These capital variables are maintained in a separate dataset to the modelling data and will be later moved into the modelling database. Distributions for these variables are shown in Figure 12: Distribution of simulated LGD and EAD values

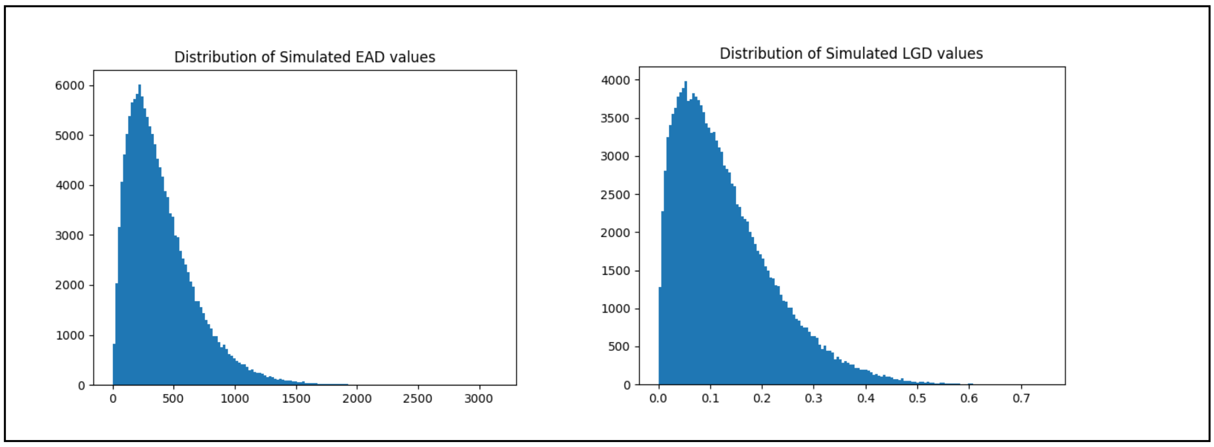
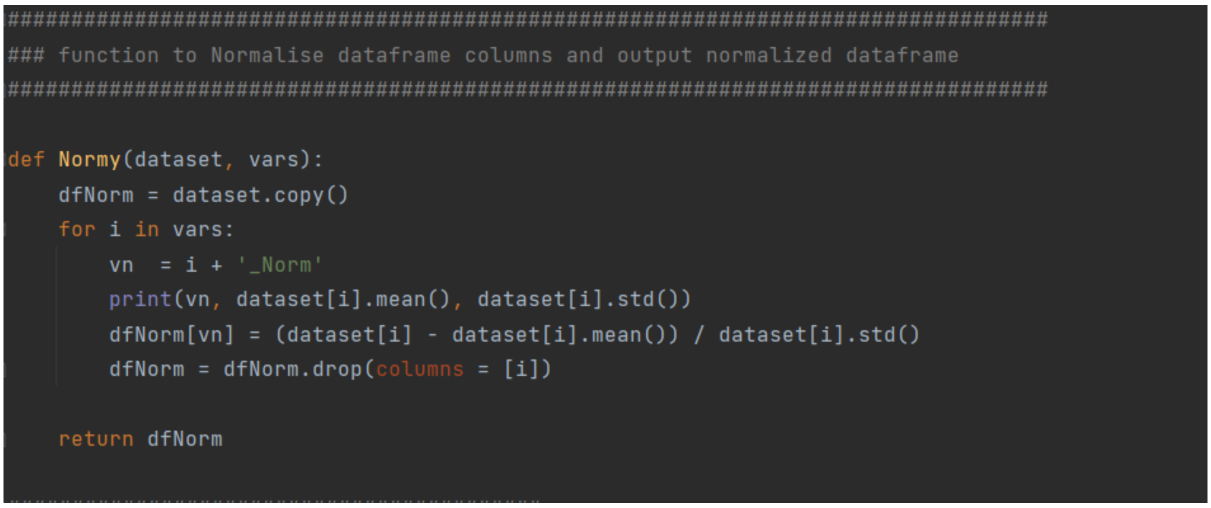


Figure 12: Distribution of simulated LGD and EAD values

Once these steps are completed the data are then standardized so that all variables are centred around 0 with standard deviation of 1. This is completed using the Normy function within the CreditModeller.py module (Figure 13)



```
##################################################################################
### function to Normalise dataframe columns and output normalized dataframe
##################################################################################

def Normy(dataset, vars):
    dfNorm = dataset.copy()
    for i in vars:
        vn  = i + '_Norm'
        print(vn, dataset[i].mean(), dataset[i].std())
        dfNorm[vn] = (dataset[i] - dataset[i].mean()) / dataset[i].std()
        dfNorm = dfNorm.drop(columns = [i])

    return dfNorm
```

Figure 13: Code to standardize variables

This dataset is then passed through the Summview1 function again to repeat the exploratory analysis and ensure that the data is ready to progress to modelling.

4 Data Modelling

4.1 Data Splitting

The dataset created above was split into Train, test and validation samples in a Ratio of 60:20:20. This was completed using the Splitter() function within the CreditModeller.py module (Figure 14).

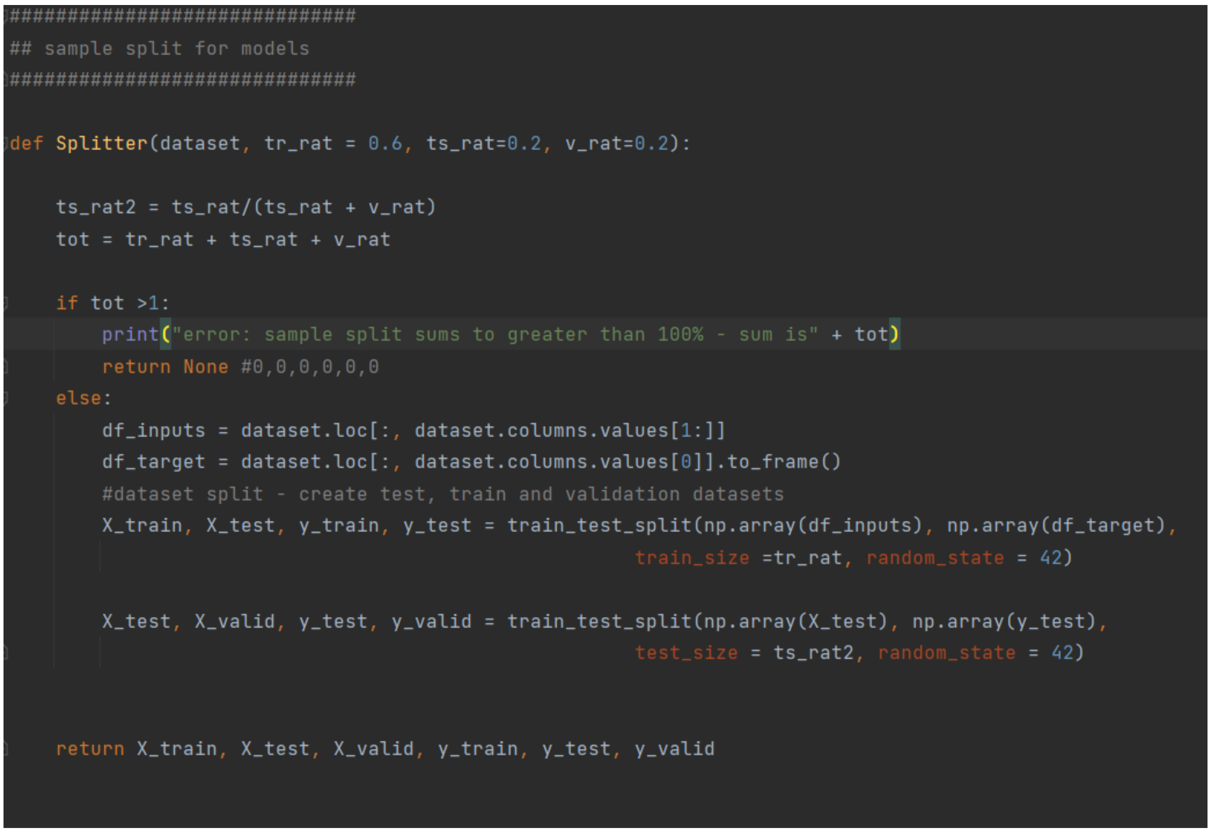

```
##############################
## sample split for models
##############################

def Splitter(dataset, tr_rat = 0.6, ts_rat=0.2, v_rat=0.2):

    ts_rat2 = ts_rat/(ts_rat + v_rat)
    tot = tr_rat + ts_rat + v_rat

    if tot >1:
        print("error: sample split sums to greater than 100% - sum is" + tot)
        return None #0,0,0,0,0,0
    else:
        df_inputs = dataset.loc[:, dataset.columns.values[1:]]
        df_target = dataset.loc[:, dataset.columns.values[0]].to_frame()
        #dataset split - create test, train and validation datasets
        X_train, X_test, y_train, y_test = train_test_split(np.array(df_inputs), np.array(df_target),
                                                            train_size =tr_rat, random_state = 42)

        X_test, X_valid, y_test, y_valid = train_test_split(np.array(X_test), np.array(y_test),
                                                            test_size = ts_rat2, random_state = 42)


    return X_train, X_test, X_valid, y_train, y_test, y_valid
```

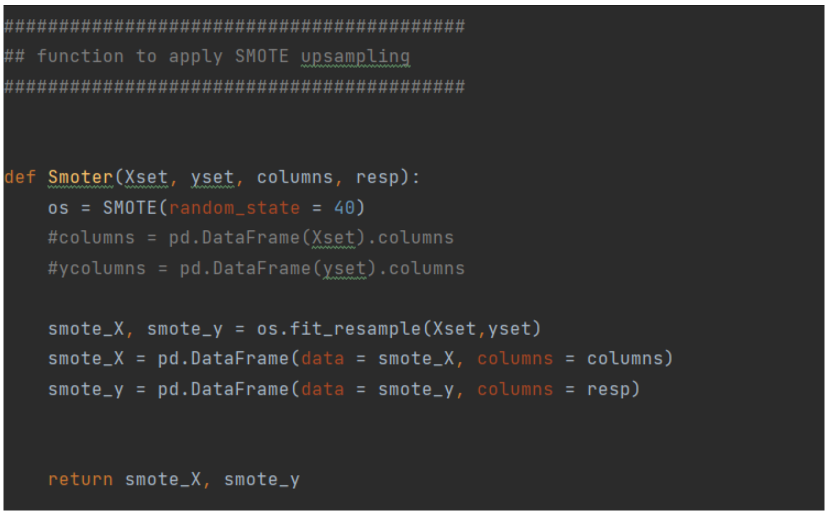
Figure 14: Code for splitting dataset

While the Train/Test /Validation data split would typically be used in a Build / test / hyperparameter tune approach, in this instance the validation dataset will not be used for this. Regulators typically expect that (i) a model will undergo initial review with a separate data sample – this is analogous to the test sample. In addition, they expect that models will be tested on a regular basis using a separate out of time sample. For our research the validation sample will fulfil this role.

4.2 Data Class Imbalance

A typical feature of credit default data is significant class imbalance. This is due to the nature of credit risk where the aim is to only lend to those who will ultimately repay the debt. For this dataset we see an event rate of 6.67%, which is heavily imbalanced.

To address the imbalance SMOTE resampling was applied to the data. A function Smoter() was created withing the CreditModeller.py module (Figure 15). This approach ensures that the frequency of event is equalized across the two outcomes through simulating additional data points for the underrepresented class. As a result, the event rate is now increased to 50% in the training dataset. SMOTE was only applied to the Training dataset.



```
###########################################
## function to apply SMOTE upsampling
###########################################


def Smoter(Xset, yset, columns, resp):
    os = SMOTE(random_state = 40)
    #columns = pd.DataFrame(Xset).columns
    #ycolumns = pd.DataFrame(yset).columns

    smote_X, smote_y = os.fit_resample(Xset,yset)
    smote_X = pd.DataFrame(data = smote_X, columns = columns)
    smote_y = pd.DataFrame(data = smote_y, columns = resp)


    return smote_X, smote_y
```

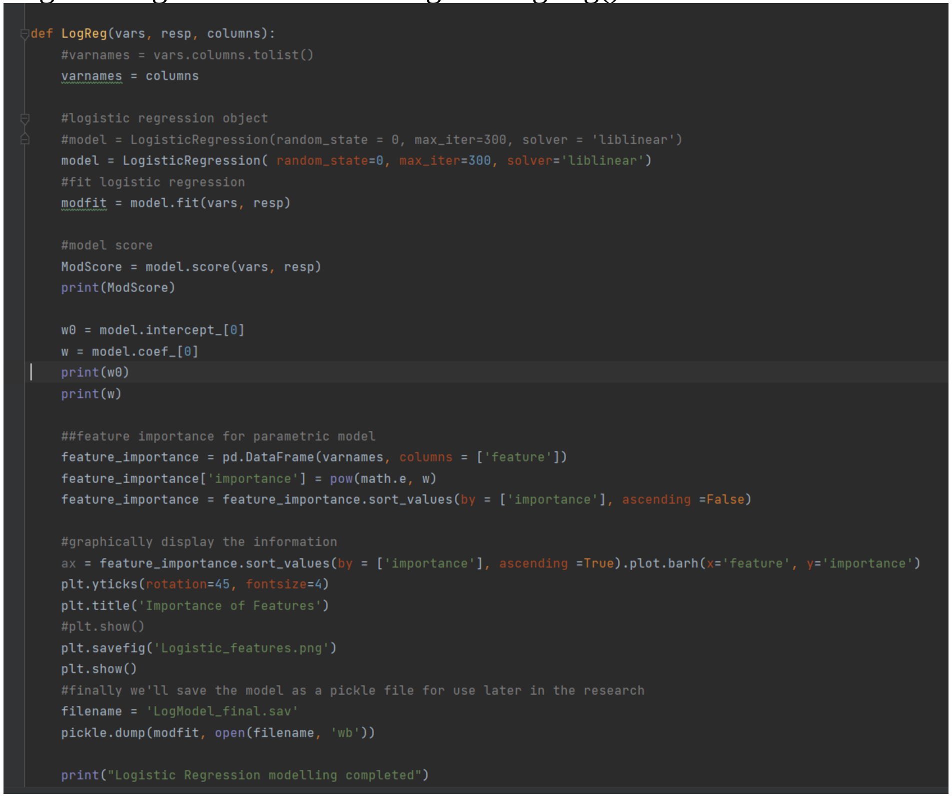
Figure 15: code for smote resampling

4.3 Modelling

A range of Different models are developed as part of the research. The configuration for each will be described below. All models are fitted using functions within the CreditModeller.py module

4.3.1 Logistic Regression

Logistic Regression is fitted using the LogReg() function



```
def LogReg(vars, resp, columns):
    #varnames = vars.columns.tolist()
    varnames = columns

    #logistic regression object
    #model = LogisticRegression(random_state = 0, max_iter=300, solver = 'liblinear')
    model = LogisticRegression( random_state=0, max_iter=300, solver='liblinear')
    #fit logistic regression
    modfit = model.fit(vars, resp)

    #model score
    ModScore = model.score(vars, resp)
    print(ModScore)

    w0 = model.intercept_[0]
    w = model.coef_[0]
    print(w0)
    print(w)

    ##feature importance for parametric model
    feature_importance = pd.DataFrame(varnames, columns = ['feature'])
    feature_importance['importance'] = pow(math.e, w)
    feature_importance = feature_importance.sort_values(by = ['importance'], ascending =False)

    #graphically display the information
    ax = feature_importance.sort_values(by = ['importance'], ascending =True).plot.barh(x='feature', y='importance')
    plt.yticks(rotation=45, fontsize=4)
    plt.title('Importance of Features')
    #plt.show()
    plt.savefig('Logistic_features.png')
    plt.show()
    #finally we'll save the model as a pickle file for use later in the research
    filename = 'LogModel_final.sav'
    pickle.dump(modfit, open(filename, 'wb'))

    print("Logistic Regression modelling completed")
```

Figure 16: Logistic Regression Modelling code

This applies the LogisticRegression module within Scikit-Learn to fit the model. The standard plot of feature importance is produced as part of the analysis and the final model is saved to a file for future use.

Once the model has been fitted by the function, model diagnostics are carried out using the RocPlt() function within the Credit Diagnostic.py module. This generates a series of diagnostics including a graphical view of the confusion matrix, ROC curve, K-S curve and precision recall curve (Figure 17), and a dataframe containing a range of performance metrics for the model.

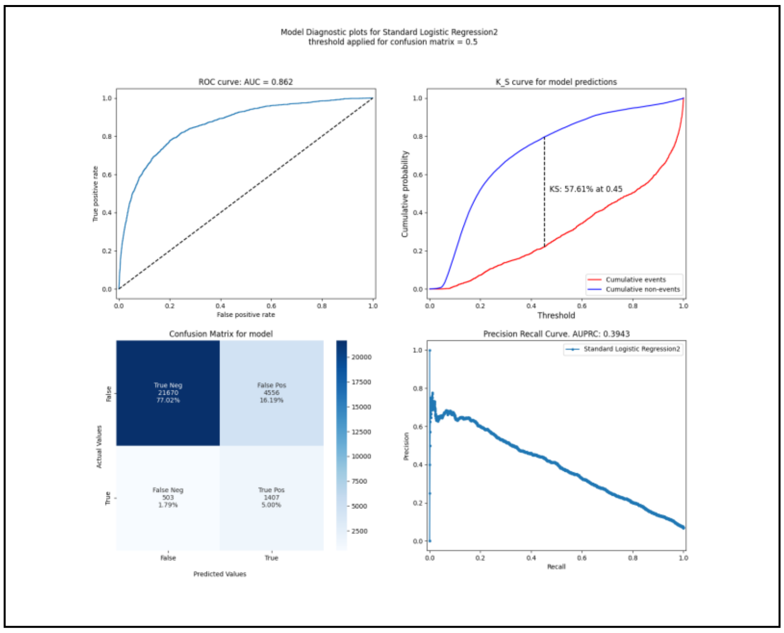


Figure 17: diagnostic output from RocPlt function

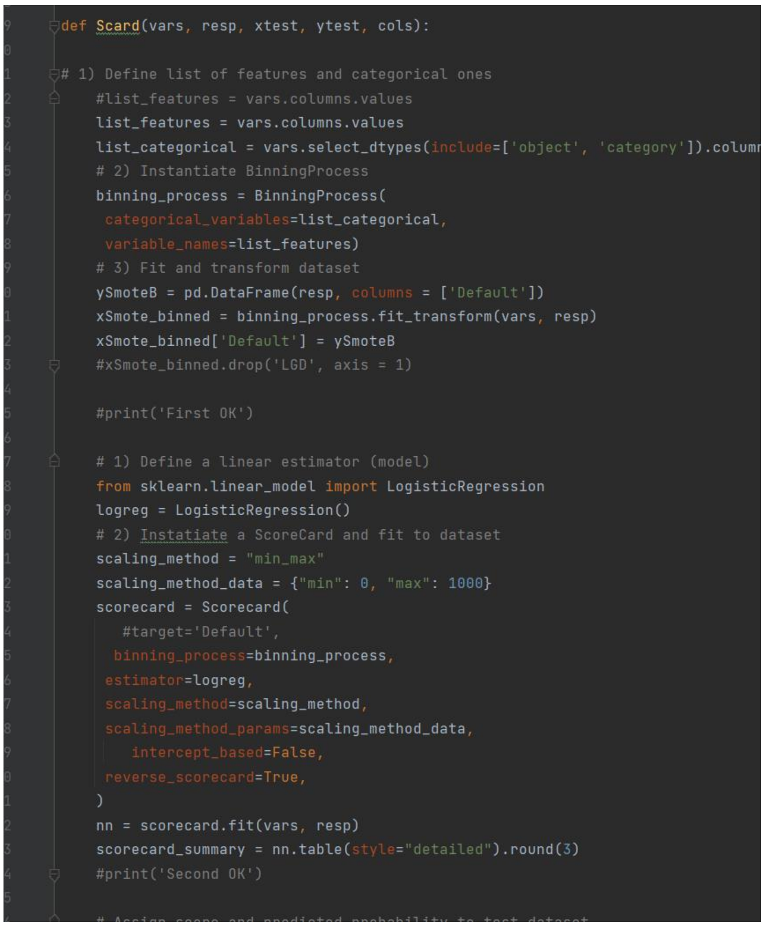
All of the metrics produced will subsequently transferred into the Modelling database for further use. The predicted values for each of the train, test and validated samples under this model are also calculated and saved into dataframes.

As an indicative view of any overfitting the diagnostic information was also produced for the Training dataset. The figures between the two are comparable, giving a degree of comfort that overfit is not an issue here.

The same diagnostic approach is applied to the test dataset for all models. As such we will not repeat the description for this in the sections that follow.

4.3.2 Binned Scorecard Logistic Regression

Binned scorecards developed using weights of evidence are a common approach applied to retailing credit scoring. The advantages of the approach include that it is relatively easy to score and understand the modelling outcomes using this approach.
For this research Pythons OptBinning library was used within the Scard() function. This applies an optimised weight of evidence-based binning to the dataset and uses this to fit a logistic regression via scikit learns LogisticRegression classifier (Figure 18). As such the approach is comparable to the previous, with any difference in performance being down to the additional binning step.

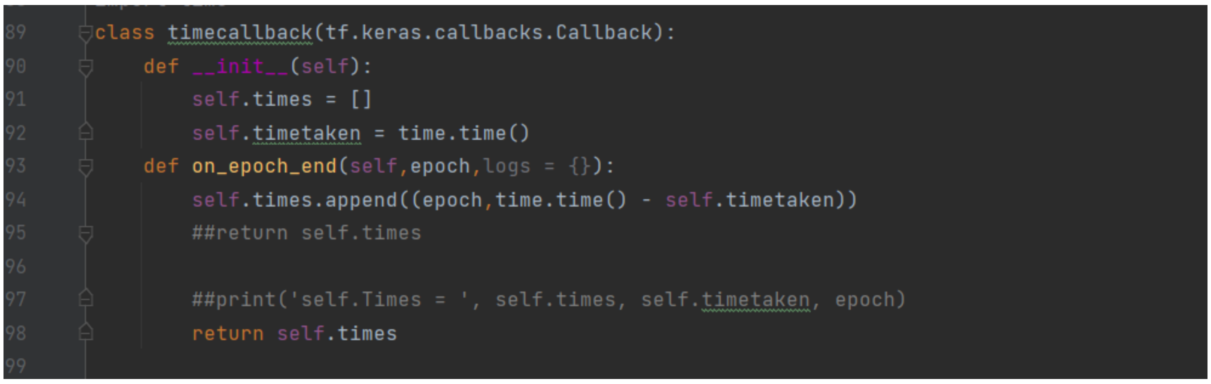


```
def Scard(vars, resp, xtest, ytest, cols):

# 1) Define list of features and categorical ones
    #list_features = vars.columns.values
    list_features = vars.columns.values
    list_categorical = vars.select_dtypes(include=['object', 'category']).colum
    # 2) Instantiate BinningProcess
    binning_process = BinningProcess(
     categorical_variables=list_categorical,
     variable_names=list_features)
    # 3) Fit and transform dataset
    ySmoteB = pd.DataFrame(resp, columns = ['Default'])
    xSmote_binned = binning_process.fit_transform(vars, resp)
    xSmote_binned['Default'] = ySmoteB
    #xSmote_binned.drop('LGD', axis = 1)

    #print('First OK')

    # 1) Define a linear estimator (model)
    from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression()
    # 2) Instatiate a ScoreCard and fit to dataset
    scaling_method = "min_max"
    scaling_method_data = {"min": 0, "max": 1000}
    scorecard = Scorecard(
       #target='Default',
      binning_process=binning_process,
     estimator=logreg,
     scaling_method=scaling_method,
     scaling_method_params=scaling_method_data,
       intercept_based=False,
     reverse_scorecard=True,
    )
    nn = scorecard.fit(vars, resp)
    scorecard_summary = nn.table(style="detailed").round(3)
    #print('Second OK')
```

Figure 18: Scard() function to fit Binned Logistic Regression

4.3.3 Neural Network

A relatively simple Neural Net model was fitted to the data user the Netter() function. One of the aims was to monitor the effect of increasing the number of model epochs on the fit time for the model as an indicator of complexity. To achieve this a custom callback function (Figure 19) was created to monitor this as part of capturing the history for the model:

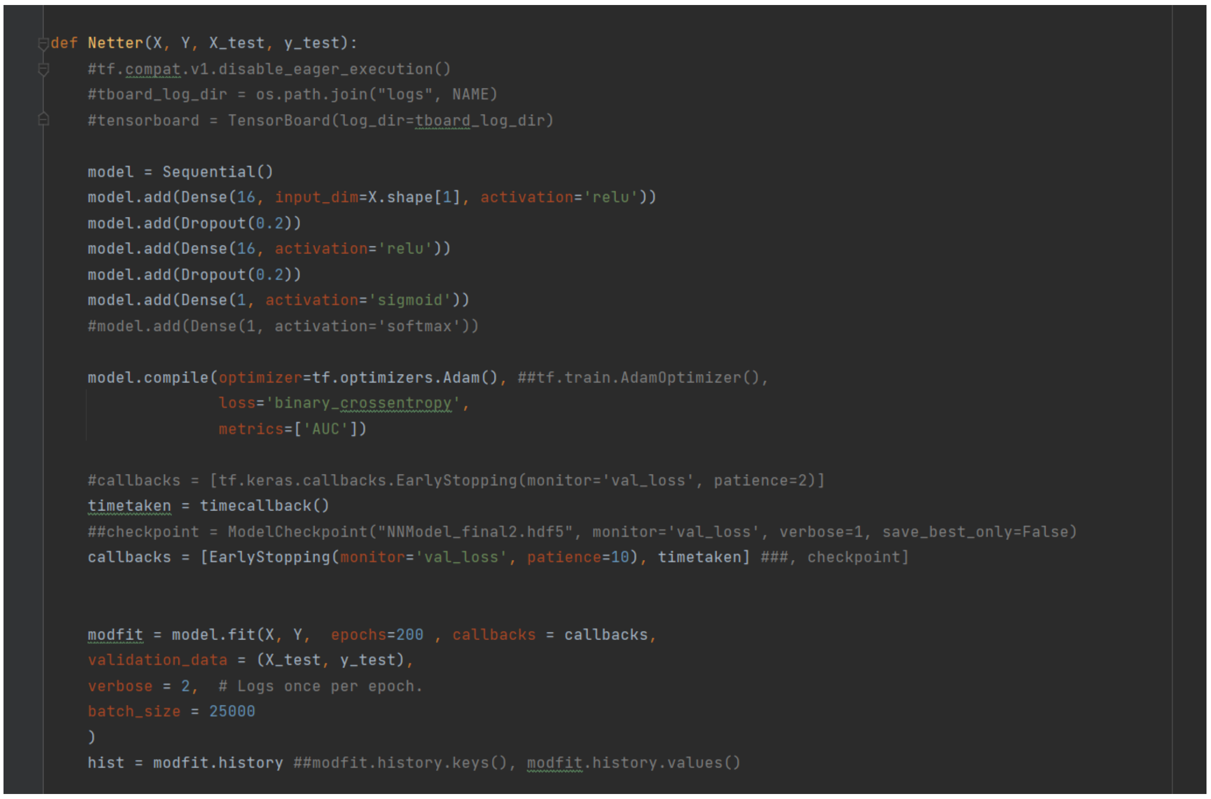


```python
class timecallback(tf.keras.callbacks.Callback):
    def __init__(self):
        self.times = []
        self.timetaken = time.time()
    def on_epoch_end(self,epoch,logs = {}):
        self.times.append((epoch,time.time() - self.timetaken))
        ##return self.times

        ##print('self.Times = ', self.times, self.timetaken, epoch)
        return self.times

```

Figure 19: Callback code applied with Neural Net

A Neural Network was then fitted to the data using Tensorflow.Keras. An input and dense layer was fitted with a relu activation function. A sigmoid activation function was applied to the output layer. A binary cross entropy loss function is applied. The model is designed to run to up to 200 epochs, with a possibility of early stopping applied if 10 successive runs fail to further improve performance.

Once the model is fitted it is saved to a .h5 file for later use. The history data for the model is combined with the information from the custom callback and also output by the function for later assessment.



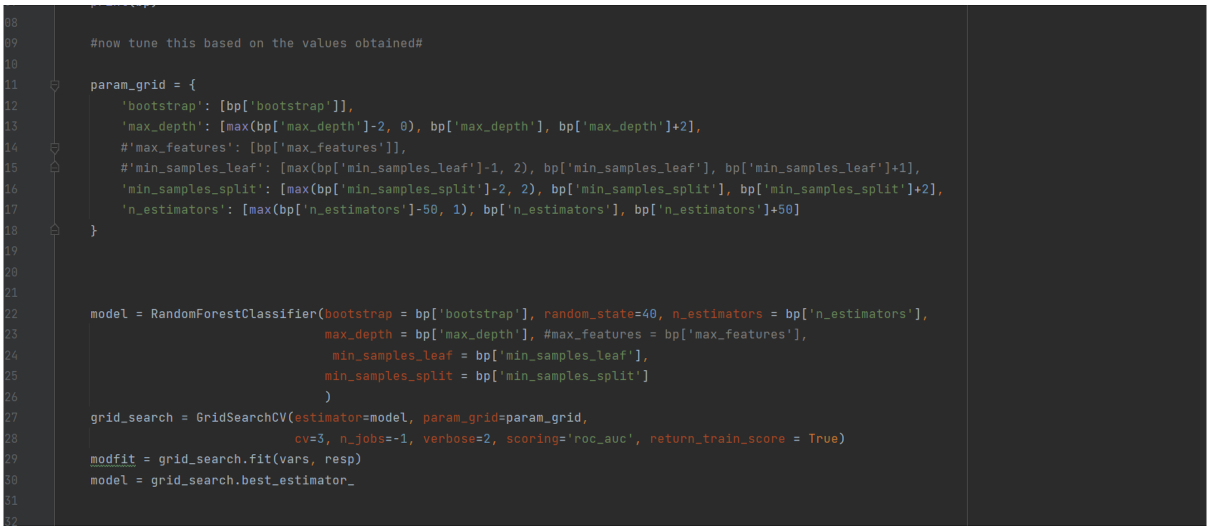
```python
def Netter(X, Y, X_test, y_test):
    #tf.compat.v1.disable_eager_execution()
    #tboard_log_dir = os.path.join("logs", NAME)
    #tensorboard = TensorBoard(log_dir=tboard_log_dir)

    model = Sequential()
    model.add(Dense(16, input_dim=X.shape[1], activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(16, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    #model.add(Dense(1, activation='softmax'))

    model.compile(optimizer=tf.optimizers.Adam(), ##tf.train.AdamOptimizer(),
                  loss='binary_crossentropy',
                  metrics=['AUC'])

    #callbacks = [tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2)]
    timetaken = timecallback()
    ##checkpoint = ModelCheckpoint("NNModel_final2.hdf5", monitor='val_loss', verbose=1, save_best_only=False)
    callbacks = [EarlyStopping(monitor='val_loss', patience=10), timetaken] ###, checkpoint]


    modfit = model.fit(X, Y,  epochs=200 , callbacks = callbacks,
    validation_data = (X_test, y_test),
    verbose = 2,  # Logs once per epoch.
    batch_size = 25000
    )
    hist = modfit.history ##modfit.history.keys(), modfit.history.values()
```

Figure 20: Code for fitting Neural Network

4.3.4 Random Forest

Random Forest was fitted using the RFMod() function. This applies sklearns RandomForestClassifier() to fit the model.

Hyperparameters for the model were tuned using the RandomsearchCV and GridsearchCV modules. Initially 50 randomly selected fits are employed in a randomised search across a range of the hyperparameters. Once this has selected a localised 'best' space, the gridsearchCV approach is then used to refine this further to look for the best possible hyper-parameter set. The final hyper-parameters selected from this are then the ones used in the final fitted model. The search algorithms are both applying a 3-fold cross validation in their search. This will also provide some assurance against overfitting of the model. A range of hyper-parameters are considered in this approach as shown in Figure 21



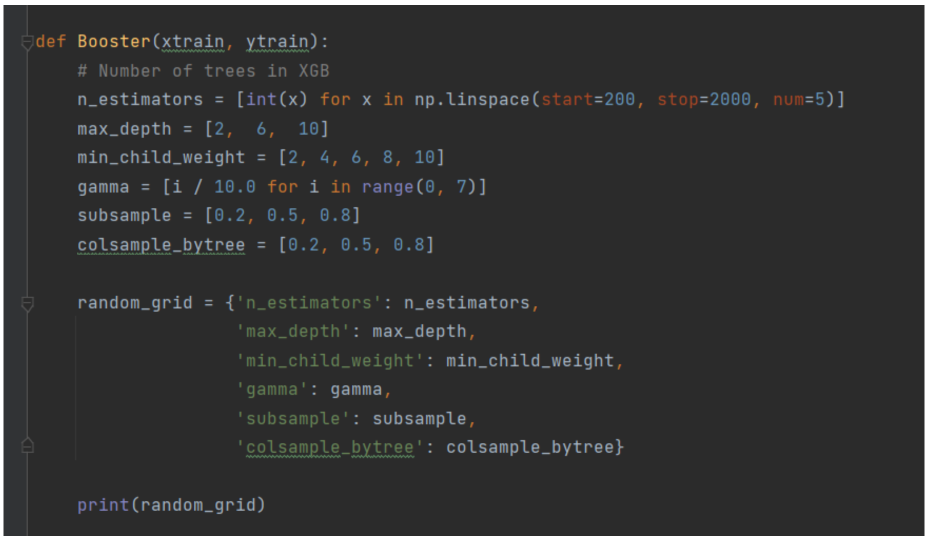
```python
#now tune this based on the values obtained#

param_grid = {
    'bootstrap': [bp['bootstrap']],
    'max_depth': [max(bp['max_depth']-2, 0), bp['max_depth'], bp['max_depth']+2],
    #'max_features': [bp['max_features']],
    #'min_samples_leaf': [max(bp['min_samples_leaf']-1, 2), bp['min_samples_leaf'], bp['min_samples_leaf']+1],
    'min_samples_split': [max(bp['min_samples_split']-2, 2), bp['min_samples_split'], bp['min_samples_split']+2],
    'n_estimators': [max(bp['n_estimators']-50, 1), bp['n_estimators'], bp['n_estimators']+50]
}



model = RandomForestClassifier(bootstrap = bp['bootstrap'], random_state=40, n_estimators = bp['n_estimators'],
                               max_depth = bp['max_depth'], #max_features = bp['max_features'],
                                min_samples_leaf = bp['min_samples_leaf'],
                               min_samples_split = bp['min_samples_split']
                               )
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2, scoring='roc_auc', return_train_score = True)
modfit = grid_search.fit(vars, resp)
model = grid_search.best_estimator_
```

Figure 21:GridSearchCV as applied to RandomForest model

The results of each step of the hyperparameter tuning are also retained during this step to enable a view of the changing complexity of the model as the hyperparameters are adjusted. Fitting time for each model is also captured to assess how the model complexity affects this. The final model is also saved to file or future use.

4.3.5 XGBoost

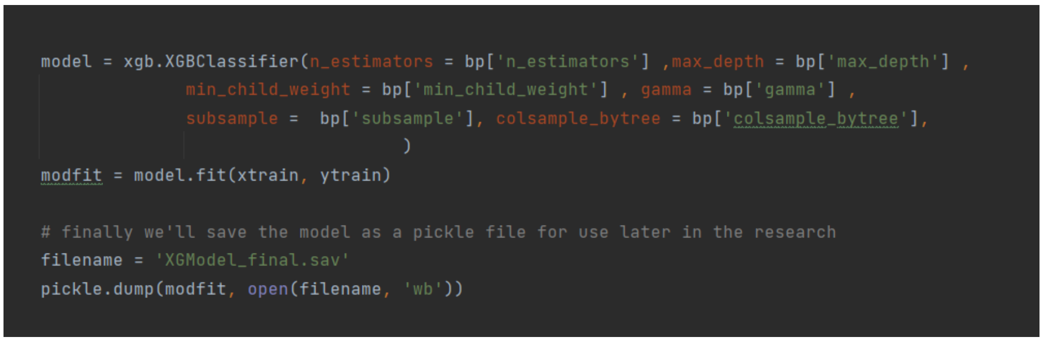
An XGBoost model is fitted using XGBClassifier() from the XGBoost library. This is applied within the Booster() function in CreditModeller.py. Once again RandomSearchCV and GridSearchCV are used to tune the model hyperparameters, with an initial 50 point random grid being refined through Grid search. A 3 folds cross validation approach is again applied here. A range of parameters are tuned (Figure 22):



```
def Booster(xtrain, ytrain):
    # Number of trees in XGB
    n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=5)]
    max_depth = [2,  6,  10]
    min_child_weight = [2, 4, 6, 8, 10]
    gamma = [i / 10.0 for i in range(0, 7)]
    subsample = [0.2, 0.5, 0.8]
    colsample_bytree = [0.2, 0.5, 0.8]

    random_grid = {'n_estimators': n_estimators,
                   'max_depth': max_depth,
                   'min_child_weight': min_child_weight,
                   'gamma': gamma,
                   'subsample': subsample,
                   'colsample_bytree': colsample_bytree}

    print(random_grid)
```

Figure 22: Hyperparameters tuned for XGBoost

Once the gridsearch has completed the best hyperparameters identified by the model are passed to the XGBClassifier to fit the model. Outputs returned by the function include the model, the selected nest parameters and the results from each stage of the tuning process.



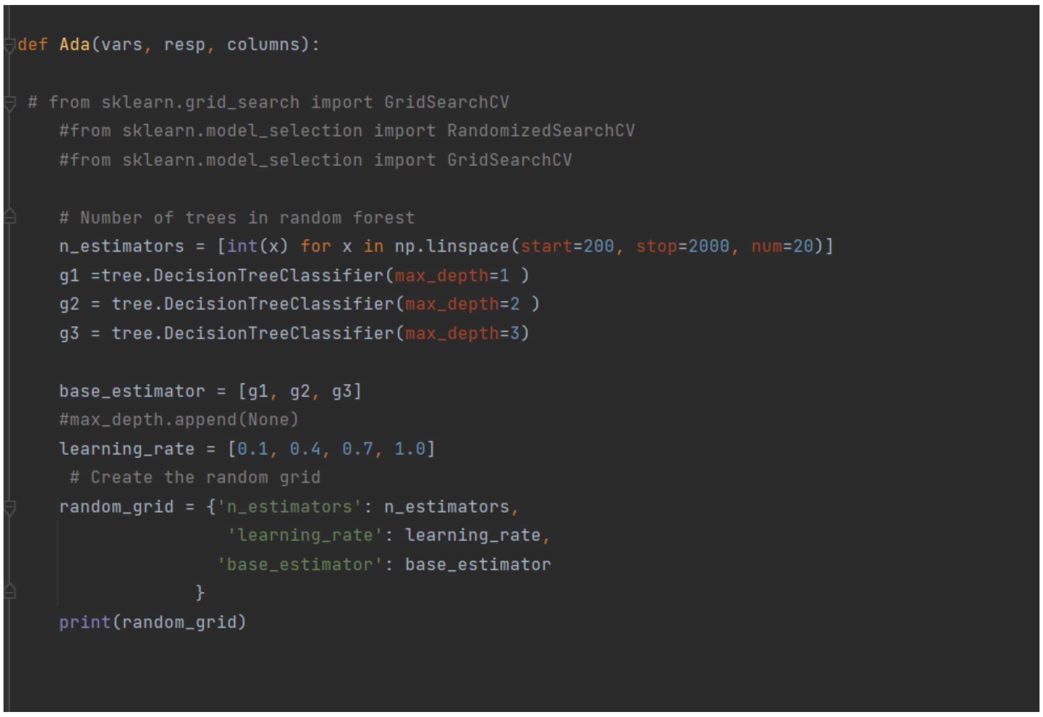
```
model = xgb.XGBClassifier(n_estimators = bp['n_estimators'] ,max_depth = bp['max_depth'] ,
              min_child_weight = bp['min_child_weight'] , gamma = bp['gamma'] ,
              subsample =  bp['subsample'], colsample_bytree = bp['colsample_bytree'],
                            )
modfit = model.fit(xtrain, ytrain)

# finally we'll save the model as a pickle file for use later in the research
filename = 'XGModel_final.sav'
pickle.dump(modfit, open(filename, 'wb'))
```

Figure 23: fitting of XGBoost model

In common with all models fitted, model diagnostics are obtained using the RocPlt() function withinCreditDiagnostic.py. The fitted model is also used to produce predicted probabilities for each of the test, train and validation datasets.

4.3.6 AdaBoost

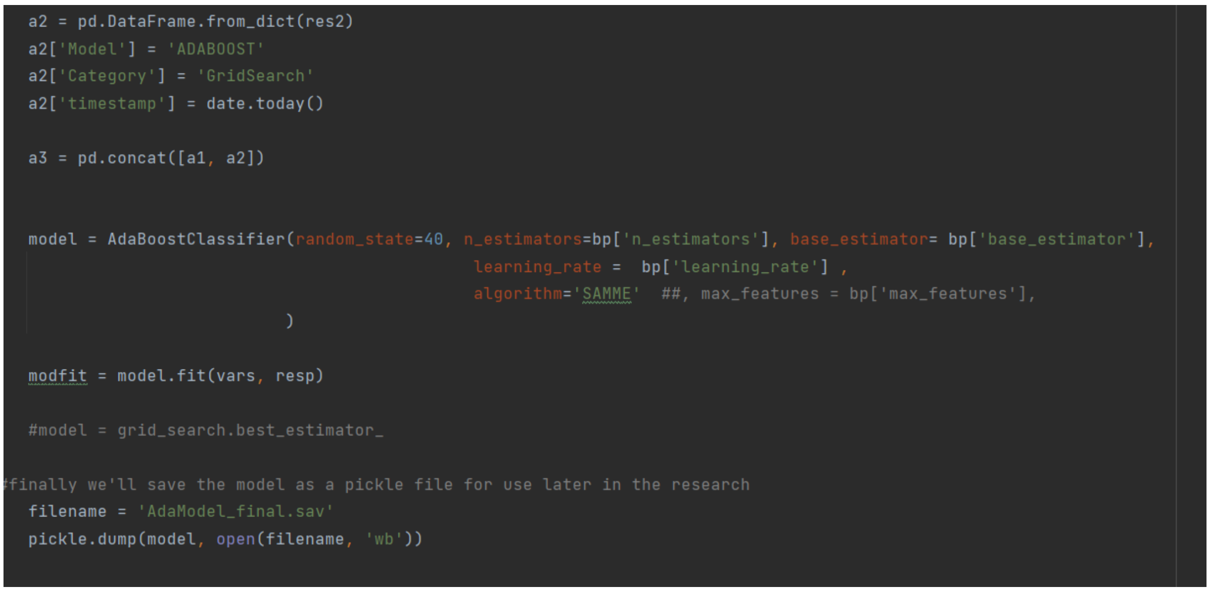
AdaBoost is fitted using scikitlearns AdaBoostClassifier. This is applied within the Ada() function of the CreditModeller.py module. In common with the other tree based methods the hyperparameters are optimised using the randonSearchCV and GridsearchCV routines, with a 3-fold cross validation employed and 50 initial random searches being used to identify an optimal space which is then refined using the grid search approach.



```
def Ada(vars, resp, columns):

 # from sklearn.grid_search import GridSearchCV
    #from sklearn.model_selection import RandomizedSearchCV
    #from sklearn.model_selection import GridSearchCV

    # Number of trees in random forest
    n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=20)]
    g1 =tree.DecisionTreeClassifier(max_depth=1 )
    g2 = tree.DecisionTreeClassifier(max_depth=2 )
    g3 = tree.DecisionTreeClassifier(max_depth=3)

    base_estimator = [g1, g2, g3]
    #max_depth.append(None)
    learning_rate = [0.1, 0.4, 0.7, 1.0]
     # Create the random grid
    random_grid = {'n_estimators': n_estimators,
                    'learning_rate': learning_rate,
                   'base_estimator': base_estimator
                 }
    print(random_grid)
```

Figure 24: Hyperparameter tuning for AdaBoost

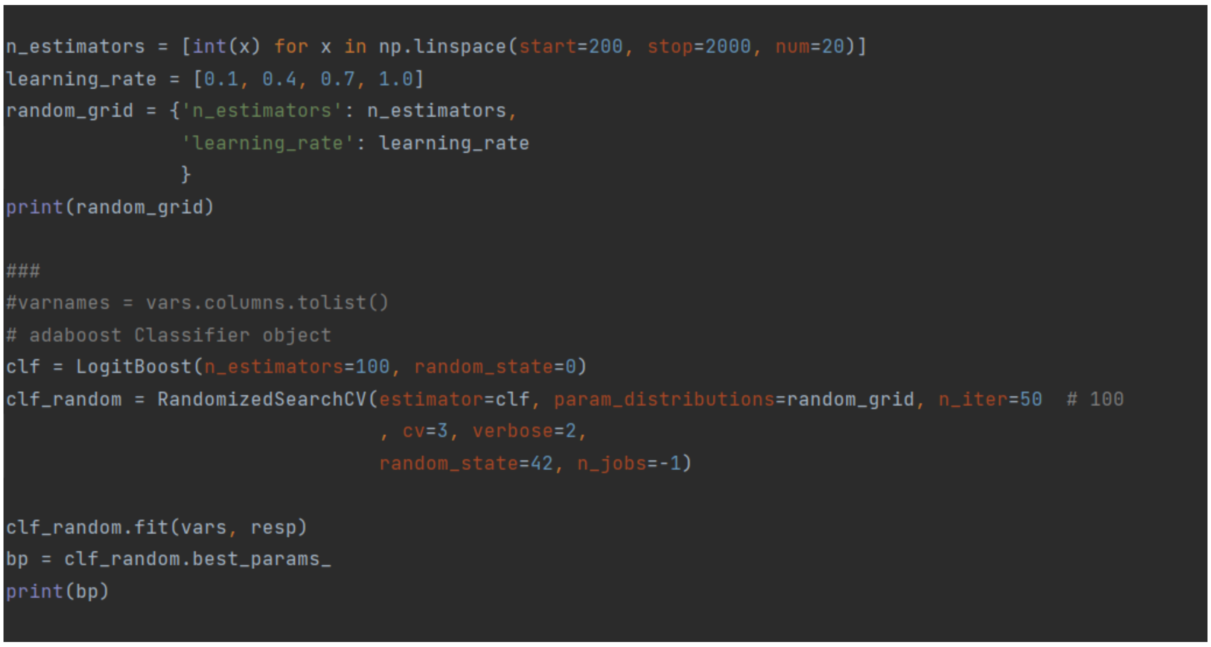
Once again, the model is fitted based on the best parameters produced from this process. For this the AdaBoostClassifier() within sklearn is applied. Results of this exercise, including fit times for the various hyperparameter combinations attempted are ultimately saved to pandas DataFrames, from where they are later uploaded to a SQL Server database. The model is also saved to file during this process.



```
    a2 = pd.DataFrame.from_dict(res2)
    a2['Model'] = 'ADABOOST'
    a2['Category'] = 'GridSearch'
    a2['timestamp'] = date.today()

    a3 = pd.concat([a1, a2])


    model = AdaBoostClassifier(random_state=40, n_estimators=bp['n_estimators'], base_estimator= bp['base_estimator'],
                                             learning_rate =  bp['learning_rate'] ,
                                             algorithm='SAMME'  ##, max_features = bp['max_features'],
                               )

    modfit = model.fit(vars, resp)

    #model = grid_search.best_estimator_

#finally we'll save the model as a pickle file for use later in the research
    filename = 'AdaModel_final.sav'
    pickle.dump(model, open(filename, 'wb'))
```

4.3.7 LogitBoost

Logit Boost is fitted using the LB() function within the CreditModeller.py module. Once again, the hyperparameters of the model are tuned using GridsearchCV and RandomSearchCV with a 3-fold cross validation applied and an initial random search of 50 points.



```
n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=20)]
learning_rate = [0.1, 0.4, 0.7, 1.0]
random_grid = {'n_estimators': n_estimators,
               'learning_rate': learning_rate
               }
print(random_grid)

###
#varnames = vars.columns.tolist()
# adaboost Classifier object
clf = LogitBoost(n_estimators=100, random_state=0)
clf_random = RandomizedSearchCV(estimator=clf, param_distributions=random_grid, n_iter=50  # 100
                                , cv=3, verbose=2,
                                random_state=42, n_jobs=-1)

clf_random.fit(vars, resp)
bp = clf_random.best_params_
print(bp)
```

Figure 25: LogitBost Hyperparameter tuning

Once the hyperparameters are identified these are used to fit a model using the LogitBoost() classifier from the LogitBoost library. Once again, the model and the tuning analysis is returned by the model and stored in pandas dataframes for later use. Model diagnostic is completed using the Rocplt() function and predicted values are generated using the model for the train, test and validation samples.

4.3.8 Naïve Bayes

Naïve Bayes is fitted using the NB() function within the CreditModeller.py module. This fits a Gaussian Naïve Bayes model from the scikit learn library. The fitted model is then saved to file. Model diagnostics are completed using the rocplt() function once again and predicted values are generated for the train, test and validation samples.

Note that the CreditModeller module also contains a script to fit a support vector machine. However due to the extremely long fit times and memory requirements encountered this was not progressed with. This limitation of SVM as dataset size increases is a known limitation of this approach.

5 Model Assessment

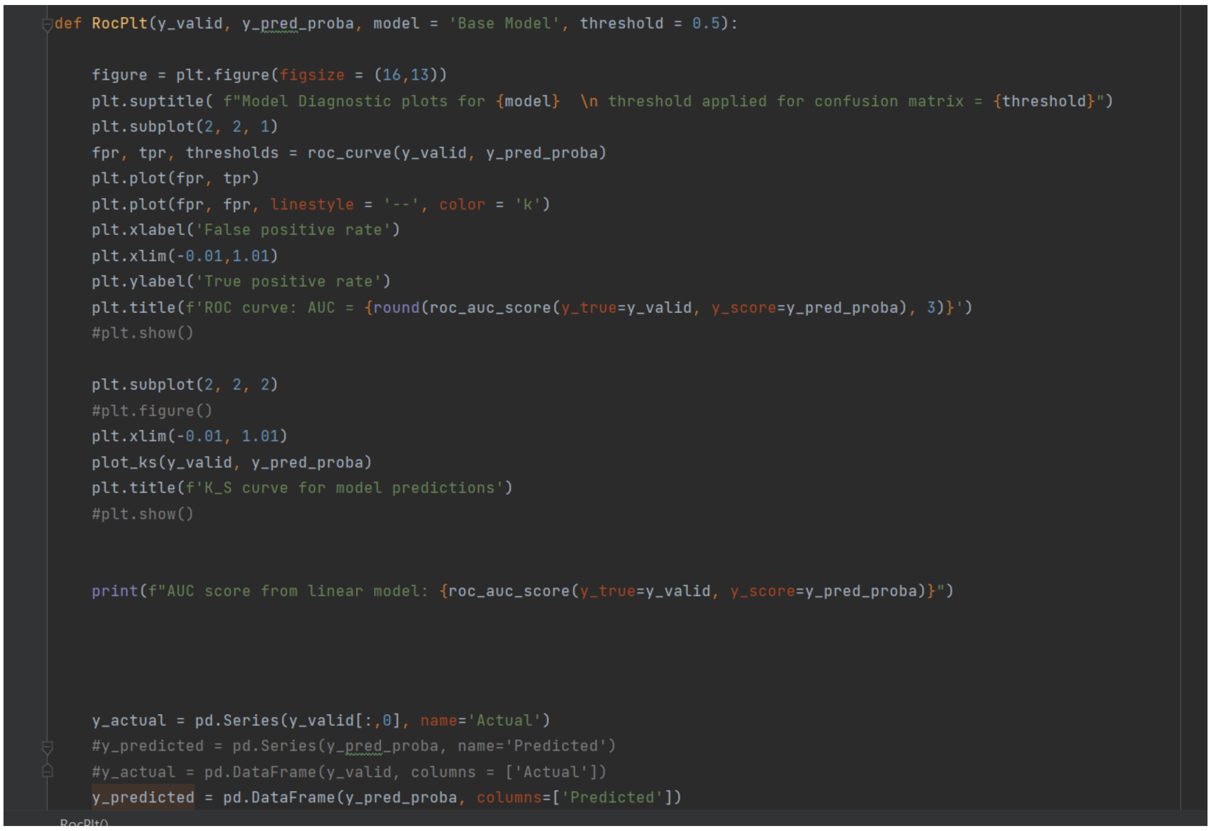
Model Assessment for this research is considered under a number of different headings and addressed using distinct components of the developed code.

5.1 Model Predictive Performance

Under the model performance heading we consider all generated metrics that consider the predictive power of the model. As noted previously, diagnostics on the fitted model are created within the CreditDiagnostic.py module. This module creates a series of Graphical and tabular analysis to help understand the model's performance. The graphic analysis includes:

- ROC plot
- Kolmogorov-Smirnov (KS) plot demonstrating the maximum separation between distributions
- Precision-recall curve for the model
- Graphical view of confusion matrix

The AUC, the area under the ROC curve is the most common measure applied to assess credit default models, and this is produced within the graphic. This metric is typically superior as a measure in cases where class imbalance exists. However, within the literature there are instances of other metrics being used to assess classifier performance. Given this the model also computes a suite of additional metrics: Accuracy, precision, recall and F1, and these are produced as a dataframe when the diagnostic code is run.

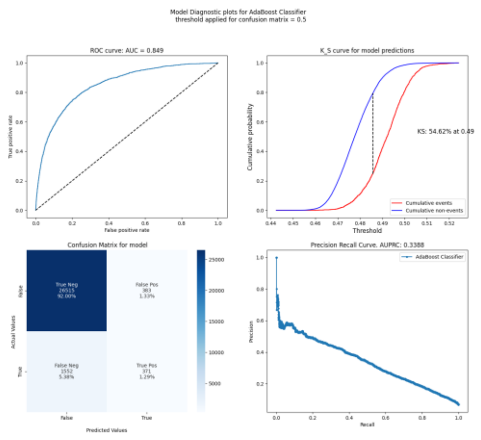


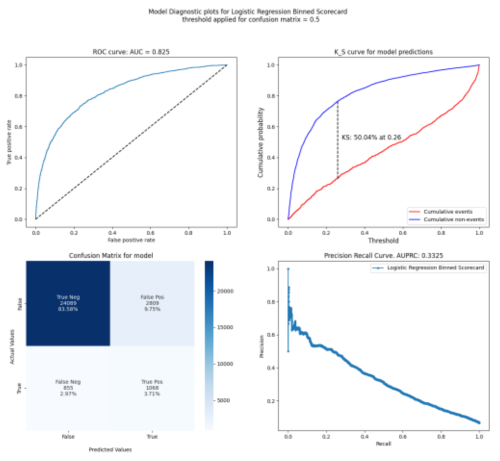
```
def RocPlt(y_valid, y_pred_proba, model = 'Base Model', threshold = 0.5):

    figure = plt.figure(figsize = (16,13))
    plt.suptitle( f"Model Diagnostic plots for {model}  \n threshold applied for confusion matrix = {threshold}")
    plt.subplot(2, 2, 1)
    fpr, tpr, thresholds = roc_curve(y_valid, y_pred_proba)
    plt.plot(fpr, tpr)
    plt.plot(fpr, fpr, linestyle = '--', color = 'k')
    plt.xlabel('False positive rate')
    plt.xlim(-0.01,1.01)
    plt.ylabel('True positive rate')
    plt.title(f'ROC curve: AUC = {round(roc_auc_score(y_true=y_valid, y_score=y_pred_proba), 3)}')
    #plt.show()

    plt.subplot(2, 2, 2)
    #plt.figure()
    plt.xlim(-0.01, 1.01)
    plot_ks(y_valid, y_pred_proba)
    plt.title(f'K_S curve for model predictions')
    #plt.show()


    print(f"AUC score from linear model: {roc_auc_score(y_true=y_valid, y_score=y_pred_proba)}")




    y_actual = pd.Series(y_valid[:,0], name='Actual')
    #y_predicted = pd.Series(y_pred_proba, name='Predicted')
    #y_actual = pd.DataFrame(y_valid, columns = ['Actual'])
    y_predicted = pd.DataFrame(y_pred_proba, columns=['Predicted'])
```

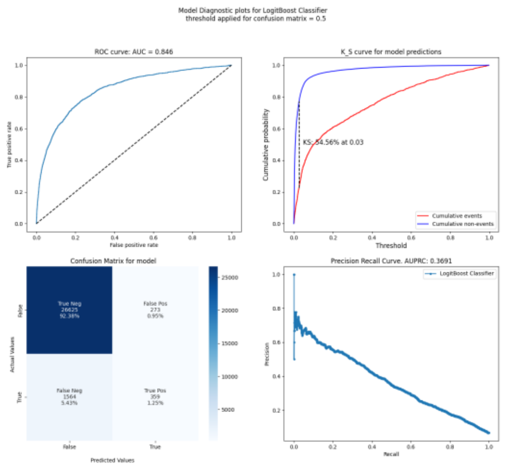
Figure 26: Initial code for RocPlt() function

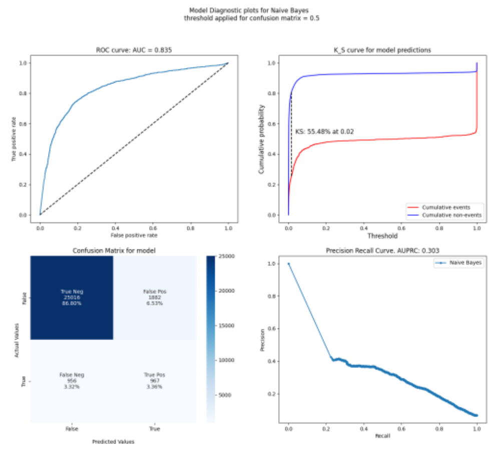
The function assumes a threshold probability of 0.5 in classification, however a user can select a different value if they wish to look at the impact of this on the confusion matrix generated.

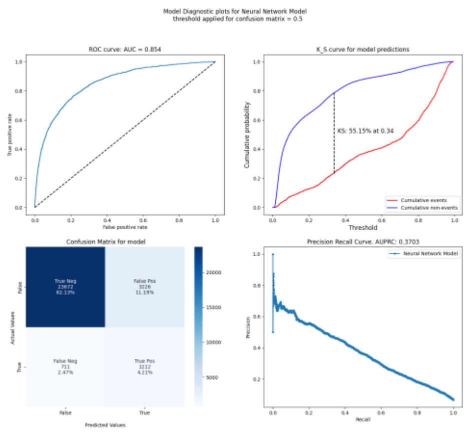
In terms of results for each of the models we can graphically compare the results for each:

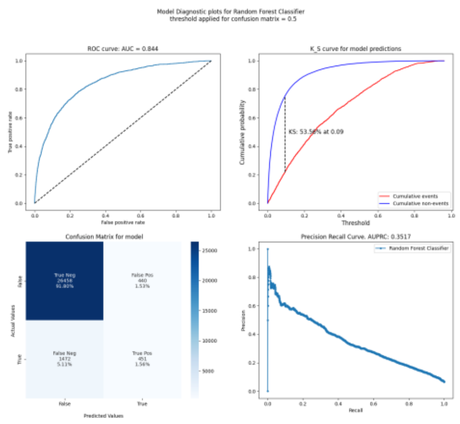


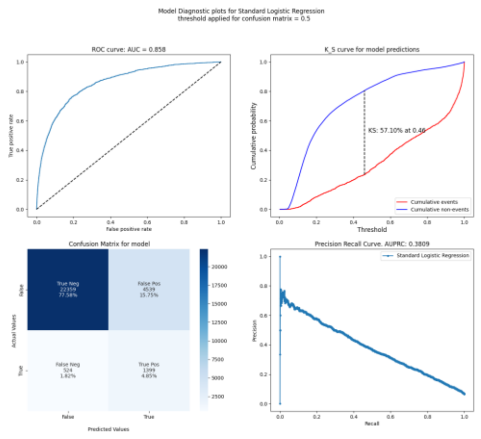


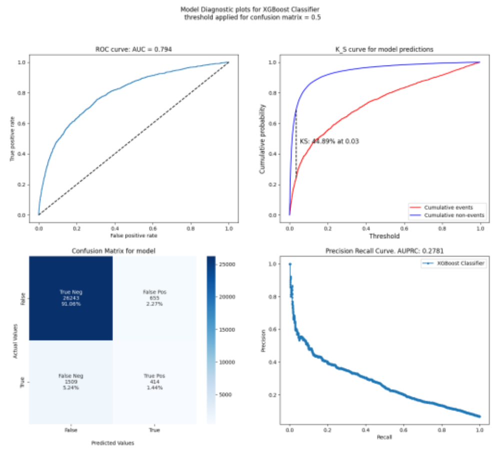












It's clear from the confusion matrices for each that different models take differing approaches, with precision and recall varying greatly across the different models. While classical approaches apply a greater tendency to misclassify goods as bads, tree based approaches appear to be more likely to misclassify bads as goods, which is an undesirable property for a credit default model.

5.2 Model Explainability

Model Explainability at both a global and local level is considered using he SHAP package. This applies an explainer to a model to generate a SHAP value for each entry, combining the features of a number of explainability approaches including SHAP and LIME. While this will differ from the classical approach for linear models and may lead to differing interpretation of most predictive factors, it will allow for a model agnostic approach, and in most cases for a simple logistic model will yield broadly similar interpretations.

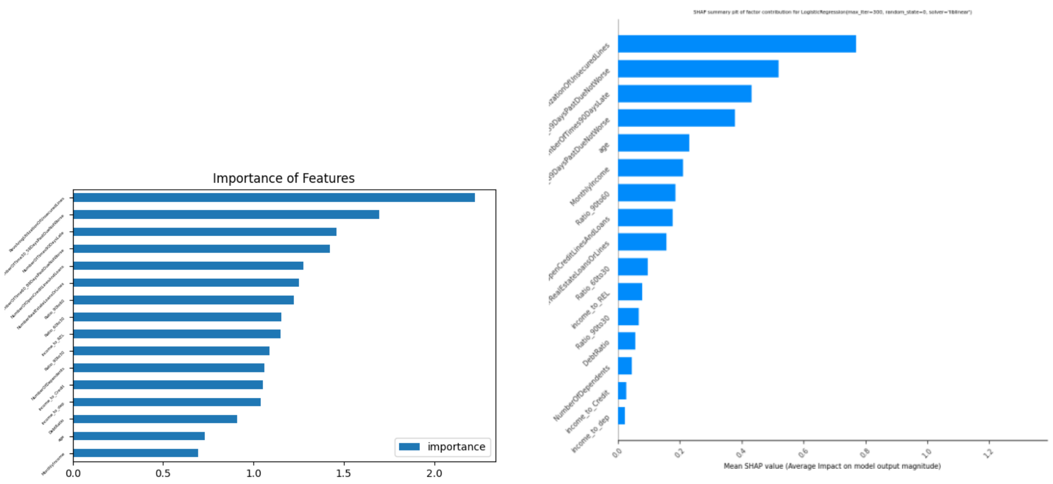


Figure 27:Standard feature importance and SHAP feature importance for the fitted logistic regression model

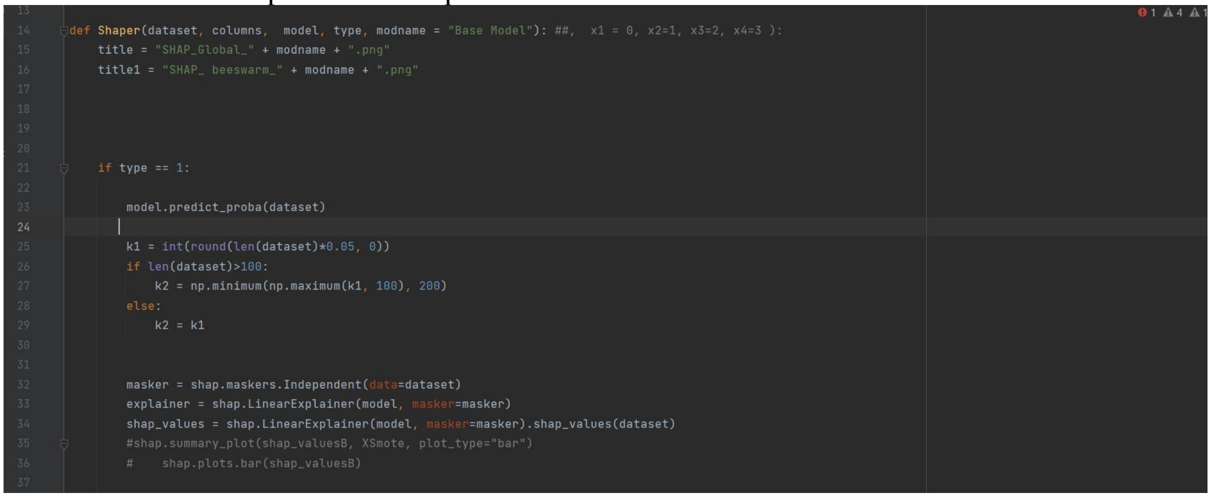
A different explainer is used for different model types. While this approach may lead to some loss of the model agnostic element of SHAP, it conversely will lead to better estimates and significantly faster run times. However, in practice the explainability element of the research is implemented in a manner that allows a user to change their choice of explainer depending on the model. In general however specific types of models will require either a specific explainer (e.g. Linear models require LinearExplainer) or the use of the general KernelExplainer. During the research it became clear that KernelExplainers, with their use of a localized regression fit to explain cases, require significantly longer run times in many cases. As such these are applied to a subset of the data to demonstrate the approach. The explainers used in this research are as follows

Explainer	Models' explainer is applied to
LinearExplainer	Logistic Regression
TreeExplainer	Random Forest XGBoost
DeepExplainer	Neural Network
KernelExplainer	LogitBoost, AdaBoost, Naïve Bayes

Note that not all models were successfully explained. The Logistic Scorecard failed to generate results using SHAP. The issue appears to be related to the way the optbinning package structures the model, which Shap cannot work with. At time of submission this issue had not been satisfactorily resolved and hence explanations for this model are not included. Logisitic scorecard general explainability is a given on the basis of the readily understandable nature of mode coefficients, however this is not model agnostic and would prevent direct comparison of models.

Because Shap can provide both Local and Global explainability it is ideal for satisfying the regulatory requirements around explainability. Specifically, at a global level regulators require senior management and independent validators to be able to explain models and understand what drives predictions. At a local level, both regulatory guidance and GDPR requirements necessitate the ability to understand individual predictions from the model.

SHAP is implemented within the Shaper() function in the CreditExplain.py function. This function takes user parameters that allow different types of models to be explained
For all models SHAP is first used to produce Global explainability, producing a plot of overall feature importance as well as a violin plot that demonstrates how the individual cases contribute to the explainability of the model. The type parameter allows users to toggle between different explainers as required.



```
def Shaper(dataset, columns,  model, type, modname = "Base Model"): ##,  x1 = 0, x2=1, x3=2, x4=3 ):
    title = "SHAP_Global_" + modname + ".png"
    title1 = "SHAP_ beeswarm_" + modname + ".png"




    if type == 1:

        model.predict_proba(dataset)

        k1 = int(round(len(dataset)*0.05, 0))
        if len(dataset)>100:
            k2 = np.minimum(np.maximum(k1, 100), 200)
        else:
            k2 = k1


        masker = shap.maskers.Independent(data=dataset)
        explainer = shap.LinearExplainer(model, masker=masker)
        shap_values = shap.LinearExplainer(model, masker=masker).shap_values(dataset)
        #shap.summary_plot(shap_valuesB, XSmote, plot_type="bar")
        #    shap.plots.bar(shap_valuesB)

```

Figure 28: Initial code for the Shaper() function

For each explainer the function takes a subsample of the dataset to apply as the background distribution. This is then applied with the relevant SHAP explainer to generate the Shap values for the dataset. The code then generates plots of the overall importance of all features and the violin plot showing the contribution of the datapoints to the overall effect for a variable as shown in Figure 29.

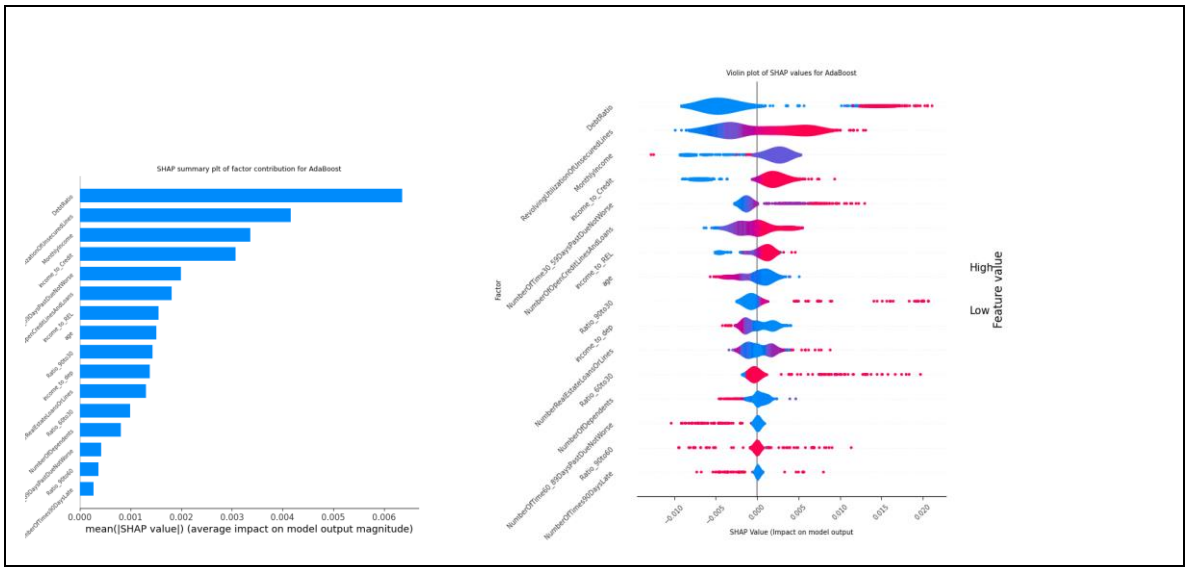


Figure 29: Graphical Global output for Shaper() function

The function also considers the Partial dependency for each factor in the model, producing a series of partial dependency plots for these (Figure 30)

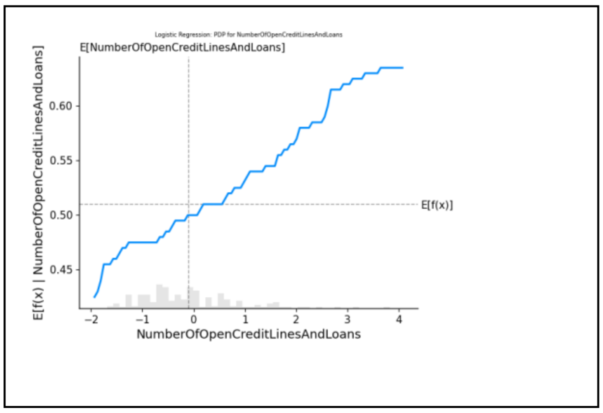
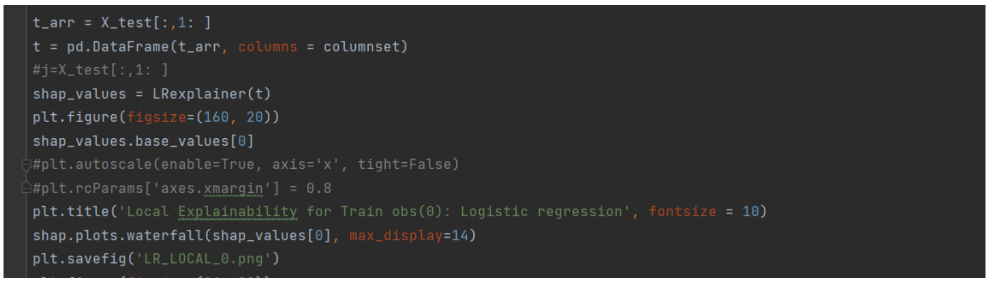


Figure 30: Partial Dependency Plot from Shaper()

The Shaper functions returns the final explainer used as well as the shap values for the data. This can then be used to obtain a local explanation specific to individual observations in a given dataset. A slightly different manipulation is carried out in Main,py depending on the explainer used, However the general sequence remains similar –

- Obtain shap values for a subset of values from a dataset using the explainer
- Get the base values for the analysis to obtain the expected value
- Generate a waterfall plot of the explanation for the individual observations.



```
t_arr = X_test[:,1: ]
t = pd.DataFrame(t_arr, columns = columnset)
#j=X_test[:,1: ]
shap_values = LRexplainer(t)
plt.figure(figsize=(160, 20))
shap_values.base_values[0]
#plt.autoscale(enable=True, axis='x', tight=False)
#plt.rcParams['axes.xmargin'] = 0.8
plt.title('Local Explainability for Train obs(0): Logistic regression', fontsize = 10)
shap.plots.waterfall(shap_values[0], max_display=14)
plt.savefig('LR_LOCAL_0.png')
```

Figure 31: Generation of waterfall plots for individual values using SHAP

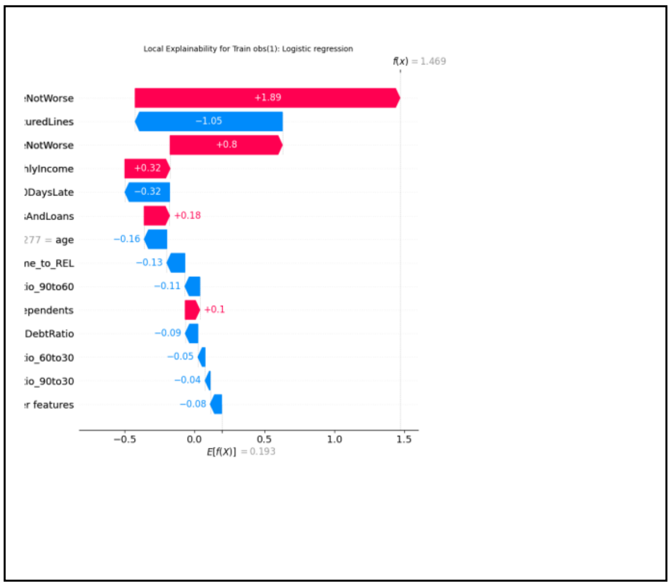


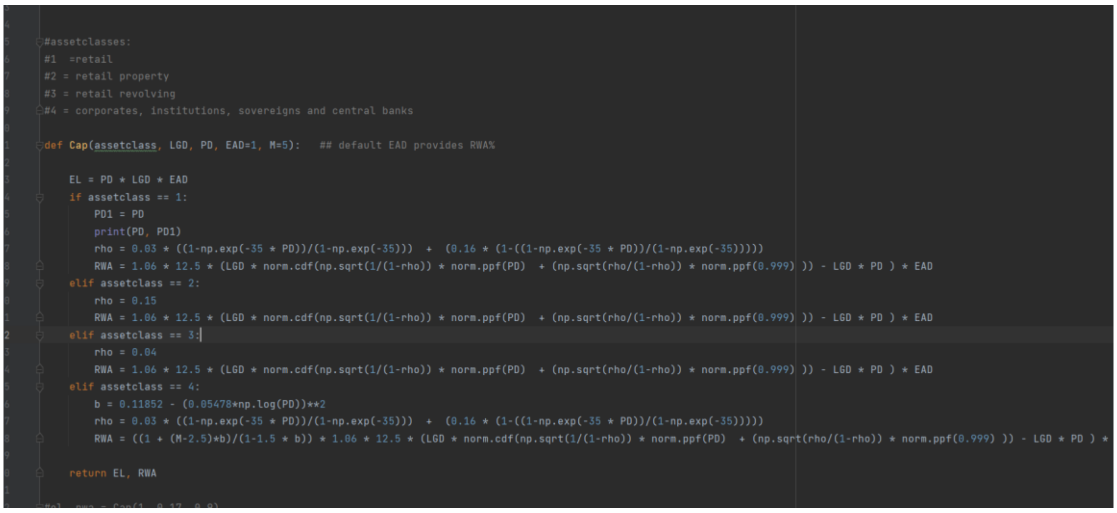
Figure 32: Waterfall plot example using SHAP

The interpretation of the waterfall plots in Figure 32 is the straightforward, with the overall estimate for the observation being calculates as the expected value E(X) plus the sum of the contributions for the individual factors.

5.3 Model Capital Assessment

A key aspect of models used for credit default management is the contribution they make to the overall capital holdings for a bank. The probability of Default (PD) is one component of the calculation of the Risk Weighted Assets (RWA) for an institution

We use the predicted probabilities generated by each model, combined with the LGD and EAD values we simulated earlier in the code (section 3.3.3) to calculate the RWA associated with each model. Given that LGD and EAD are constant for a given observation, this approach enables us to consider the impact of each model on the projected capital for the institution. This analysis is completed in the CreditCapital.py module using the Cap() function.



```
#assetclasses:
#1  =retail
#2 = retail property
#3 = retail revolving
#4 = corporates, institutions, sovereigns and central banks

def Cap(assetclass, LGD, PD, EAD=1, M=5):   ## default EAD provides RWA%

    EL = PD * LGD * EAD
    if assetclass == 1:
        PD1 = PD
        print(PD, PD1)
        rho = 0.03 * ((1-np.exp(-35 * PD))/(1-np.exp(-35)))  +  (0.16 * (1-((1-np.exp(-35 * PD))/(1-np.exp(-35)))))
        RWA = 1.06 * 12.5 * (LGD * norm.cdf(np.sqrt(1/(1-rho)) * norm.ppf(PD)  + (np.sqrt(rho/(1-rho)) * norm.ppf(0.999) )) - LGD * PD ) * EAD
    elif assetclass == 2:
        rho = 0.15
        RWA = 1.06 * 12.5 * (LGD * norm.cdf(np.sqrt(1/(1-rho)) * norm.ppf(PD)  + (np.sqrt(rho/(1-rho)) * norm.ppf(0.999) )) - LGD * PD ) * EAD
    elif assetclass == 3:
        rho = 0.04
        RWA = 1.06 * 12.5 * (LGD * norm.cdf(np.sqrt(1/(1-rho)) * norm.ppf(PD)  + (np.sqrt(rho/(1-rho)) * norm.ppf(0.999) )) - LGD * PD ) * EAD
    elif assetclass == 4:
        b = 0.11852 - (0.05478*np.log(PD))**2
        rho = 0.03 * ((1-np.exp(-35 * PD))/(1-np.exp(-35)))  +  (0.16 * (1-((1-np.exp(-35 * PD))/(1-np.exp(-35)))))
        RWA = ((1 + (M-2.5)*b)/(1-1.5 * b)) * 1.06 * 12.5 * (LGD * norm.cdf(np.sqrt(1/(1-rho)) * norm.ppf(PD)  + (np.sqrt(rho/(1-rho)) * norm.ppf(0.999) )) - LGD * PD ) *

    return EL, RWA
```

Figure 33: Cap() function for calculating regulatory capital

The Cap function applies the regulatory capital calculation specified within the Capital Requirements Regulation (CRR). The function allows the user to specify different exposure classes given that slightly different rules apply for each. Given the variables in our dataset we have assumed a retail exposure class for this research.

Note that in calculating the RWA we have not imposed certain regulatory floors (for example some exposure classes place a lower bound on PD). This is to allow the research to demonstrate only the effects of the model choice.

In terms of how a model contributes to RWA, the distribution of PD is critical due to the nonlinear relationship between PD and Unexpected Loss, which is the key element of the RWA calculation. This relationship is shown in Figure 34 while the underlying risk function applied is shown in Figure 35

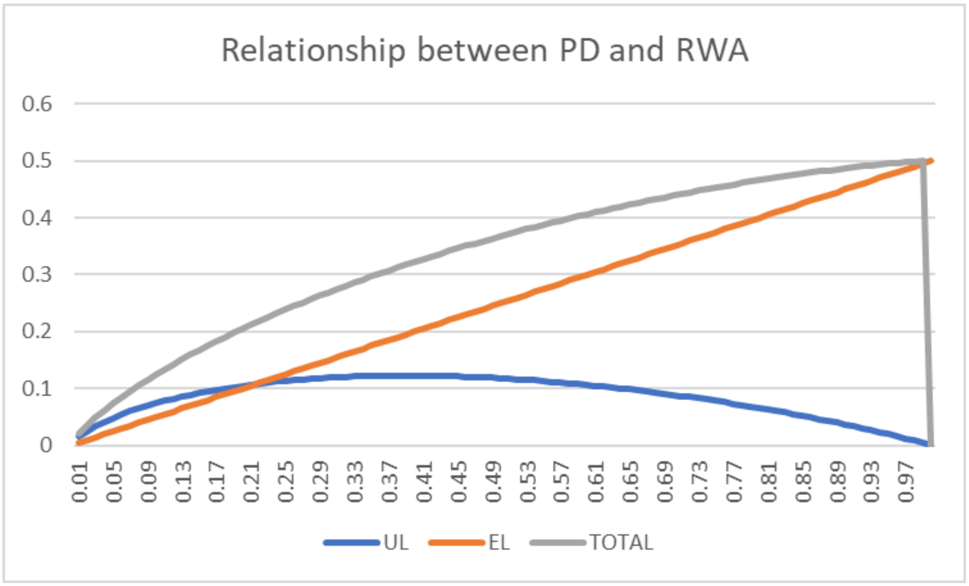


Figure 34: Relationship between PD and Loss

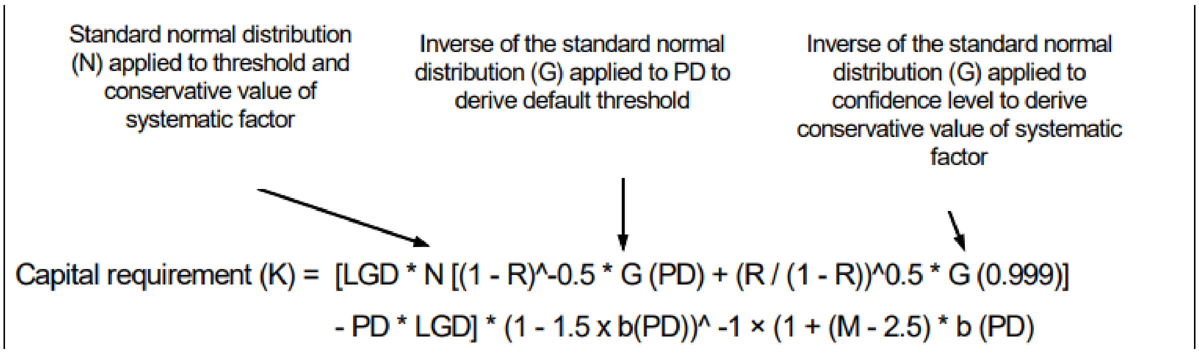

Standard normal distribution (N) applied to threshold and conservative value of systematic factor
Inverse of the standard normal distribution (G) applied to PD to derive default threshold
Inverse of the standard normal distribution (G) applied to confidence level to derive conservative value of systematic factor
Capital requirement (K) = [LGD * N [(1 - R)^-0.5 * G (PD) + (R / (1 - R))^0.5 * G (0.999)]
- PD * LGD] * (1 - 1.5 x b(PD))^ -1 × (1 + (M - 2.5) * b (PD)

Figure 35: Basel Loss function

As a result we see that the lowest RWA will be obtained when PD is either extremely low or extremely high. This is due to the loss at high PD values largely being expected loss. This means that a classifier that produces an extremely well separated distribution, such as Logitboost, is likely to generate very favourable capital outcomes, while one that is less definite, e.g. AdaBoost in our example, while require relatively high capital regardless of predictive power. Given the Underlying approach to capital assumes the other capital parameters LGD and EAD are independent of PD we also get no benefit when we subsequently model these. However excessively low capital is likely to lead to regulatory concerns, therefore there is a trade-off between the two extremes.

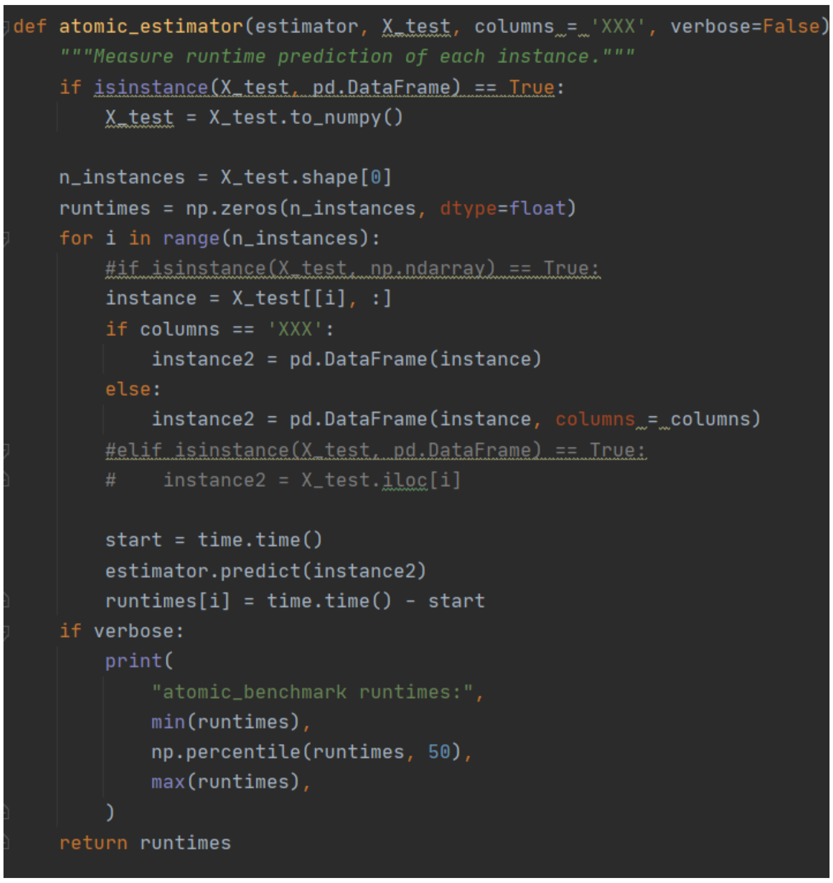
5.4 Model complexity

In assessing model complexity, we use predictive latency as proxy for complexity. Predictive latency is seen as being affected by both model complexity and model size and as such should serve as a good proxy for the overall complexity of the model. A number of variables could skew the results and therefore the research attempts as far as possible to fix a number of elements associated with the complexity estimation, for example:

The same pandas dataframe structure is fitted to each model: this ensures that (i) the models are all tested on the exact same data and (ii) the data structure used is the same to ensure none of the models benefit from simply having more optimised data.

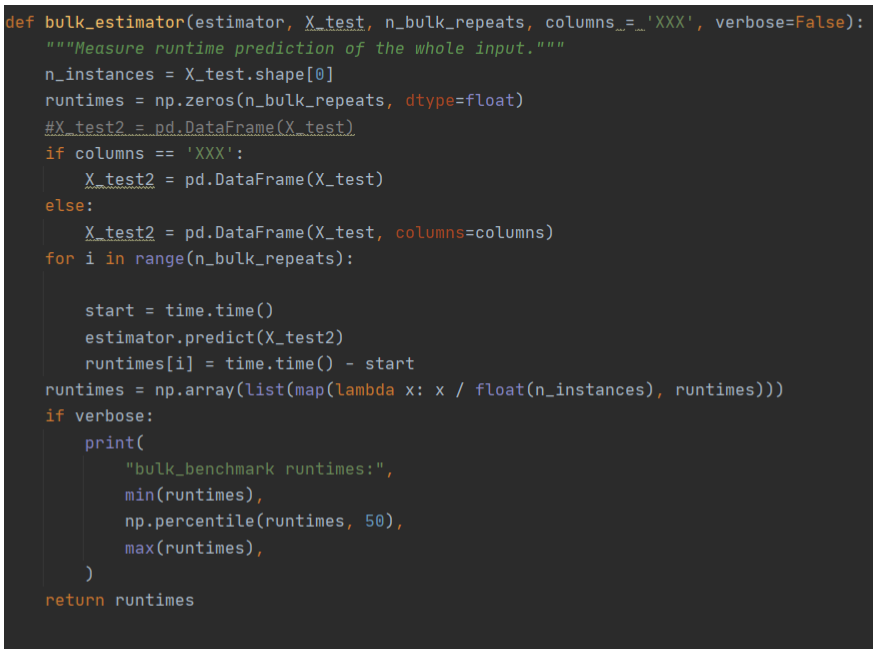
The same machine is used to fit the data to the models. This ensures that differences in hardware do not influence the performance of the models.

A common Python function is applied to both the atomic and batch predictive latency across all models considered. This is to ensure that any differences due to the coding approach are not a factor. The code applied is illustrated in Figure 36 and Figure 37



```python
def atomic_estimator(estimator, X_test, columns = 'XXX', verbose=False):
    """Measure runtime prediction of each instance."""
    if isinstance(X_test, pd.DataFrame) == True:
        X_test = X_test.to_numpy()

    n_instances = X_test.shape[0]
    runtimes = np.zeros(n_instances, dtype=float)
    for i in range(n_instances):
        #if isinstance(X_test, np.ndarray) == True:
        instance = X_test[[i], :]
        if columns == 'XXX':
            instance2 = pd.DataFrame(instance)
        else:
            instance2 = pd.DataFrame(instance, columns = columns)
        #elif isinstance(X_test, pd.DataFrame) == True:
        #    instance2 = X_test.iloc[i]

        start = time.time()
        estimator.predict(instance2)
        runtimes[i] = time.time() - start
    if verbose:
        print(
            "atomic_benchmark runtimes:",
            min(runtimes),
            np.percentile(runtimes, 50),
            max(runtimes),
        )
    return runtimes
```

Figure 36: Atomic latency code



```python
def bulk_estimator(estimator, X_test, n_bulk_repeats, columns = 'XXX', verbose=False):
    """Measure runtime prediction of the whole input."""
    n_instances = X_test.shape[0]
    runtimes = np.zeros(n_bulk_repeats, dtype=float)
    #X_test2 = pd.DataFrame(X_test)
    if columns == 'XXX':
        X_test2 = pd.DataFrame(X_test)
    else:
        X_test2 = pd.DataFrame(X_test, columns=columns)
    for i in range(n_bulk_repeats):

        start = time.time()
        estimator.predict(X_test2)
        runtimes[i] = time.time() - start
    runtimes = np.array(list(map(lambda x: x / float(n_instances), runtimes)))
    if verbose:
        print(
            "bulk_benchmark runtimes:",
            min(runtimes),
            np.percentile(runtimes, 50),
            max(runtimes),
        )
    return runtimes
```

Figure 37: Batch latency Code

6 Database Preparation

A SQL Server database of modelling results is generated based on the results of the analysis. This is to enable (i) a permanent record of the modelled outcomes as well as (ii) the capacity to augment this database with future modelling information. Such information would potentially help inform an organisations future modelling choices and (iii)a detailed record of the data used to make modelling decisions as typically required by regulators during on-site inspections and considered a key element of the model's documentation.

All information is written to the database using PYODBC. The Python scrip generates new database tables for:

- The Test, train and validation datasets
- The fit estimates from each stage of the gridsearch approach for the ensemble approaches and for the epochs of the Neural Network.
- The predicted values from each model.
- The capital parameters and EL and RWA estimates.
- The timings for the complexity calculations for each model.

Each of these forms a history of the modelling experience that can be leveraged to justify modelling choices to regulators against one or more of the criteria contained in the regulatory guidance. The database tables are also related through various fields, for example the capital and test data are elated through the unique identifier. This is to allow potentially more complex queries to be created from the history database.
.

7 Power BI dashboards

Power BI is used to generate dashboards to both visualize the results of the model developments and assessment and to provide monitoring and validation against the separate validation sample. This ability to effectively monitor and validate is a key regulatory requirement.

For the purposes of demonstrating this aspect, the validation split (20%) from the original train/ test/validate split has been employed as an Out of Sample (OOS) dataset. This is typically the dataset that regulators would expect to see monitoring and validation carried out against, including backtesting and model performance.

8 Installing and Running the Project

8.1 Files included in submission

The submitted files include:

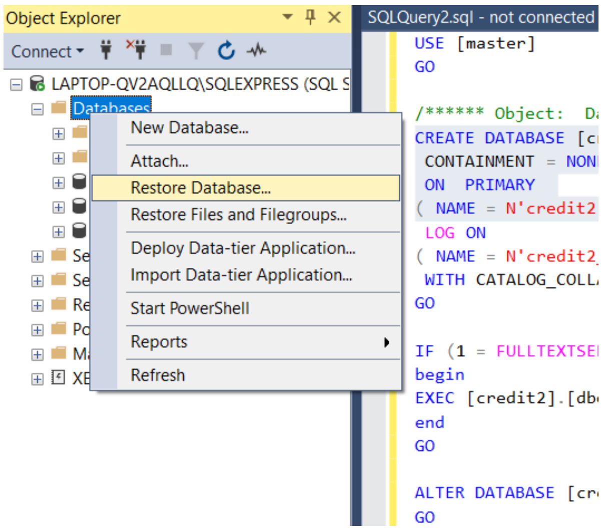
- The final database backup file. This includes the original dataset table used in the analysis
- Python code to generate the results of the research. Note that the original dataset splits used are contained in the database.
- The PowerBI dashboard generated for the database results
- The original CSV dataset downloaded from Kaggle.com

These are stored within separate subfolders in the submission files and need to be restored to a single folder location that the code can be pointed to. The exception is the database backup restore, which should be restored to the appropriate location on the user's system. Filepath will again need to be pointed to this.

To install and run the project it is necessary to restore the database containing the data, and then modify the code to point to this. Steps are as follows:

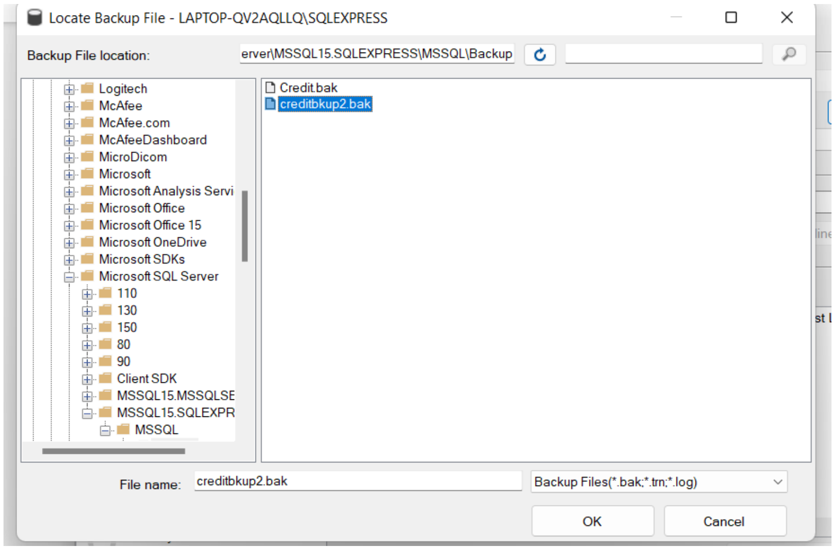
8.2 Restore Database

Open SQL Server Management Studio
Right click on databases and select 'Restore Database'


Object Explorer
Connect
LAPTOP-QV2AQLLQ\SQLEXPRESS (SQL S
Databases
New Database...
Attach...
Restore Database...
Restore Files and Filegroups...
Deploy Data-tier Application...
Import Data-tier Application...
Start PowerShell
Reports
Refresh
SQLQuery2.sql - not connected

```
USE [master]
GO

/****** Object:  Da
CREATE DATABASE [c
 CONTAINMENT = NON
 ON  PRIMARY
( NAME = N'credit2
 LOG ON
( NAME = N'credit2_
 WITH CATALOG_COLL
GO

IF (1 = FULLTEXTSE
begin
EXEC [credit2].[db
end
GO

ALTER DATABASE [cr
GO
```

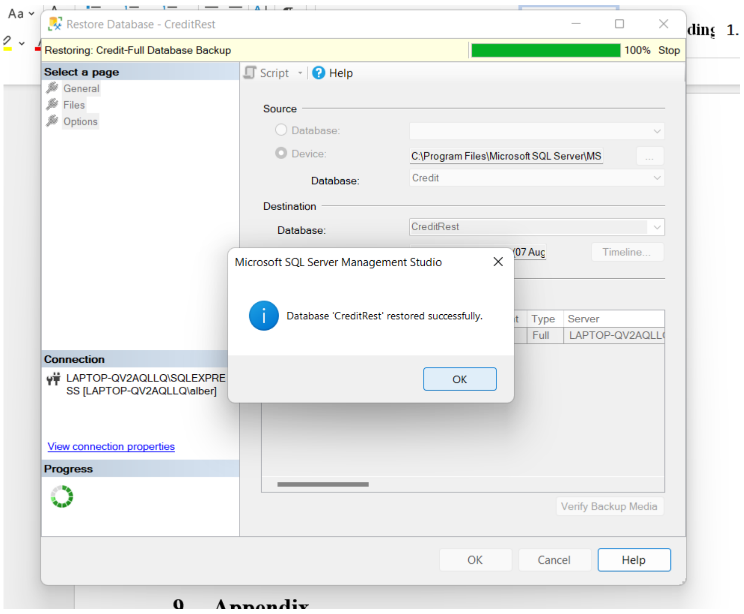
Select Device and search for the database to be restored.



Backup File location: erver\MSSQL15.SQLEXPRESS\MSSQL\Backup

Credit.bak
creditbkup2.bak

File name: creditbkup2.bak
Backup Files(*.bak;*.trn;*.log)


Restore Database - CreditRest

Restoring: Credit-Full Database Backup 100% Stop

Select a page
- General
- Files
- Options

Source
- Database:
- Device: C:\Program Files\Microsoft SQL Server\MS
- Database: Credit

Destination
- Database: CreditRest

Type	Server
Full	LAPTOP-QV2AQLL

Microsoft SQL Server Management Studio

Database 'CreditRest' restored successfully.

OK

Connection

LAPTOP-QV2AQLLQ\SQLEXPRESS [LAPTOP-QV2AQLLQ\alber]

View connection properties

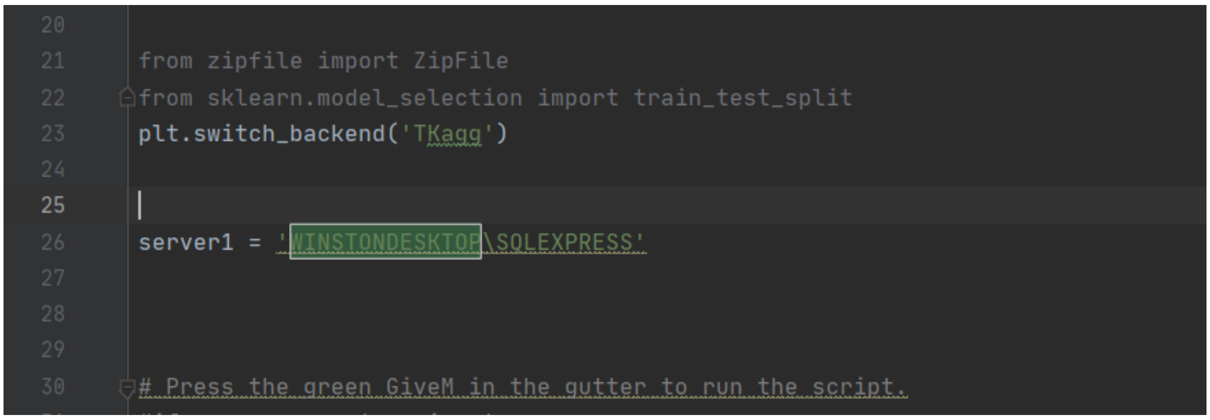
Progress

8.3 Files required

There is a standard dependency in the files used – the power BI requires a database in place to draw data from, the Python code requires appropriate file path structure to be defined at the start of the code etc. Details of setup are provided below.

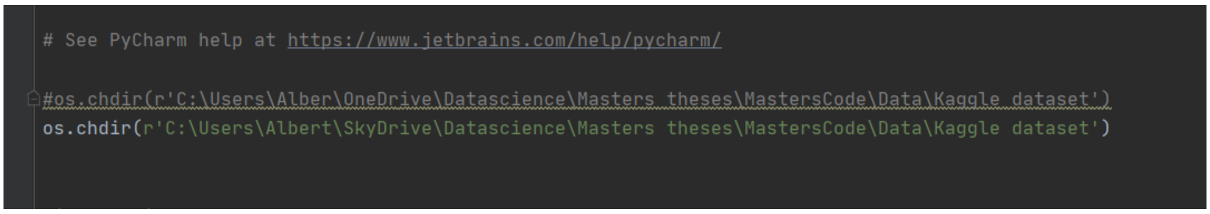
8.4 Running Python code

To run the code the script should point to the database restored in section 8.1. Line 26 of the _main_ .py module therefore needs to be modified to point to the server location for this database.



```
20
21   from zipfile import ZipFile
22   from sklearn.model_selection import train_test_split
23   plt.switch_backend('TKagg')
24
25
26   server1 = 'WINSTONDESKTOP\SQLEXPRESS'
27
28
29
30   # Press the green GiveM in the gutter to run the script.
```

The script also needs to point to the directory where the python code is stored. Line 36 can be modified to achieve this.



```
# See PyCharm help at https://www.jetbrains.com/help/pycharm/

#os.chdir(r'C:\Users\Alber\OneDrive\Datascience\Masters theses\MastersCode\Data\Kaggle dataset')
os.chdir(r'C:\Users\Albert\SkyDrive\Datascience\Masters theses\MastersCode\Data\Kaggle dataset')
```

8.5 Possible issues and known bugs

TensorFlow's latest version is known to have some issues interacting with SHAPs DeepExplainer. The SHAP explainer for this project was run against TensorFlow 2.5.0. However some issues were identified when attempting to run using TensorFlow 2.9.1

References

Brownlee (2020) Repeated k-Fold Cross-Validation for Model Evaluation in Python Available at Repeated k-Fold Cross-Validation for Model Evaluation in Python (machinelearningmastery.com) (Accessed 18 June 2022)
Brownlee (2020) SMOTE for imbalanced classification with Python, Available at: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
(Accessed: 16 May 2020)

Brownlee (2021) How to Develop Your First XGBoost Model in Python, Available at https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/ (Accessed 28 May 2022)

Brownlee (2022) How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras (machinelearningmastery.com) Available at https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/ (Accessed 20 June 2022)

Builtin (2021) Classification Models in Python: A Beginner's Guide, Available at https://builtin.com/data-science/evaluating-classification-models (Accessed 11 June 2022)

cmdlinetips (2019) How To Make Grouped Boxplots in Python with Seaborn? - Python and R Tips AZ Available at https://cmdlinetips.com/2019/03/how-to-make-grouped-boxplots-in-python-with-seaborn/#:~:text=Boxplots%20are%20one%20of%20the%20most%20common%20ways,want%20to%20visualize%20such%20data%20using%20grouped%20boxplots. (Accessed 13 June 2022)

Datacamp (2022) Using XGBoost in Python Tutorial, Available at Using XGBoost in Python Tutorial | DataCamp, (Accessed 28 May 2022)

de Servigny, A., & Renault, O. (2004). *Measuring and Managing Credit Risk.* McGraw-Hill.

Google (2019) Minimizing real-time prediction serving latency in machine learning | Cloud Architecture Center | Google Cloud Available at https://cloud.google.com/architecture/minimizing-predictive-serving-latency-in-machine-learning (Accessed 14 June 2022)

Keras.io (2021) Evaluating and exporting scikit-learn metrics in a Keras callback Available at https://keras.io/examples/keras_recipes/sklearn_metric_callbacks/ (Accessed 30 June 2022)

Koehrsen (2018) Hyperparameter Tuning the Random Forest in Python Available at https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74 (Accessed 15 June 2022)

Koehrsen (2018) A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning Available at https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (Accessed 19 June 2022)

Mavrim (2019) LogitBoost — logitboost 0.7 documentation Available at https://logitboost.readthedocs.io/ (Accessed 19 June 2022)

Martins (2021) XGBoost: A Complete Guide to Fine-Tune and Optimize your Model Available at https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663 (Accessed 22 June 2022)
Okamura (2020) GridSearchCV for Beginners. Available at GridSearchCV for Beginners. It is somewhat common knowledge in the… | by Scott Okamura | Towards Data Science (Accessed 18 June 2022)

O'Reilly.com (2019) Local Interpretable Model-Agnostic Explanations (LIME): An Introduction – O'Reilly (oreilly.com) Available at https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/ (Accessed 30 June 2022)
Stack Vidhya (2021) How To Plot Confusion Matrix In Python And Why You Need To? - Stack Vidhya Available at https://www.stackvidhya.com/plot-confusion-matrix-in-python-and-why/ (Accessed 11 June 2022)

Palencia, G (2020) developing-scorecards-in-python-using-optbinning, Available at https://towardsdatascience.com/developing-scorecards-in-python-using-optbinning-ab9a205e1f69 (Accessed 28 May 2022)

Statology (2021) How to Create a Confusion Matrix in Python - Statology Available at https://www.statology.org/confusion-matrix-python/#:~:text=To%20create%20a%20confusion%20matrix%20for%20a%20logistic,matrix%20for%20a%20logistic%20regression%20model%20in%20Python. (Accessed 11 June 2022)

The Programming Foundation Module 4 - Logistic Regression | The Programming Foundation
Available at:
https://learn.theprogrammingfoundation.org/getting_started/intro_data_science/module4/?gclid=CjwKCAjw7IeUBhBbEiwADhiEMWuMLd8ZAbvOg9VTPSATHM4XN9jp9g2ADIzCwexa1pLGAo5c5gQuZxoCLawQAvD_BwE (Accessed 17 May 2022)