Configuration Manual

MSc Research Project
Data Analytics

Akintomiwa Tomisin Akinyemi
Student ID: x22137149

School of Computing
National College of Ireland

Supervisor: Dr. Anu Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akintomiwa Tomisin Akinyemi
Student ID:	x22137149
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Anu Sahni
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	717
Page Count:	10

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:	
Date:	14th December 2023

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

Akintomiwa Tomisin Akinyemi
x22137149

1 Overview

This manual provides an insight into the implementation phase of the project, including the system specifications, necessary softwares needed to be installed, and environments, needed to conductand execute the research. The research was done to observe the effectiveness of various machine learning models in predicting loan defaults within Nigerian microfinance banks.

2 Hardware/Software Requirements

This section highlights the minimum hardware and software requirements for the execution of this project.

2.1 Hardware Requirements

The hardware details have been provided below:

Spec Name	**Value**
Operating System	Microsoft Windows 11 Home
Processor	Intel(R) Core(TM) i5-1035G1@1.00GHz, 1.19 GHz
RAM	8.00GB (7.60GB usable)
Disk Space	256GB SSD
System Type	Microsoft Surface Laptop Go

Table 1: Hardware Requirements

2.2 Software Requirements

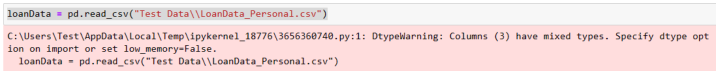
The code implementation was split into two sections; the preliminary processes were done on Jupyter Notebook, and the model implementation was carried out on Google Colab, a cloud-based IDE, using Python programming language. The software details have been provided below:

Spec Name	Value
Programming Language	Python 3.10
IDE	Jupyter Notebook, Google Colab
Browser	Google Chrome, version 119.0.6045.200
RAM	12.7GB
Disk	107GB

Table 2: Software Requirements

3 Data Source

The dataset used for this research was obtained from **QORE** data warehouse. Due to computational and time constraints, only a section of the data was randomly selected for preprocessing and modelling.



```
loanData = pd.read_csv("Test Data\\LoanData_Personal.csv")

C:\Users\Test\AppData\Local\Temp\ipykernel_18776\3656360740.py:1: DtypeWarning: Columns (3) have mixed types. Specify dtype opt
ion on import or set low_memory=False.
  loanData = pd.read_csv("Test Data\\LoanData_Personal.csv")
```

Figure 1: Reading data on Jupyter Notebook

```
# Built in colab with local data upload

from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))

Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving LoanData_Final.csv to LoanData_Final.csv
User uploaded file "LoanData_Final.csv" with length 2499817 bytes
```

Figure 2: Reading data on Google Colab

4 Python Libraries

In order to conduct this research, it was necessary to install and import several Python libraries into Colab. Some of the essential packages include SKlearn, Numpy, and Pandas. Figure 3 displays a comprehensive analysis of the libraries utilised.

```
# Import libraries, features and settings

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import io
%matplotlib inline
import seaborn as sns; sns.set()
from sklearn import preprocessing
plt.rc("font", size = 14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
sns.set(style="white")
sns.set(style="whitegrid", color_codes = True)
```

Figure 3: Imported Libraries on Google Colab

5 Data Preprocessing

The dataset was imported into a pandas dataframe on Jupyter Notebook, after which the records were checked for null and missing values as seen in Figure 4. Columns indicating applicants personal details like *'FirstName'*, *'Surname'*, and *'PhoneNumber'* were removed to protect them. Records with null values for *'CustomerAge'* were removed since it's an important attribute and only a small percentage fell under this category.

Now check for missing or null values

```
# Check for NaN values
nan_values = loanData.isnull().any()

print("\nColumns with NaN values:")
print(nan_values)
```

```
Columns with NaN values:
CustomerID          False
CustomerAge          True
Gender               True
DOB                  True
Address              True
City                 True
MarriageStatus       True
EmploymentStatus     True
BankBalance         False
LoanID              False
LoanAmount          False
LoanTerm (Days)     False
LoanRequestTime      True
LoanStatus           True
dtype: bool
```

Figure 4: Checking for missing values

Transformation of the data also took place in this section as seen in Figure 5. using the *LabelEncoder* method from *sklearn* library, a fit_transform was performed on 'LoanStatus', 'EmploymentStatus', 'MarriageStatus', and 'Gender' attributes. This converted the enum characters to integer.

```python
from sklearn.preprocessing import LabelEncoder

# Instantiate the LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to the categorical attributes
loanData_Cleaned['Loan_Status'] = label_encoder.fit_transform(loanData_Cleaned['LoanStatus'])
loanData_Cleaned['Employment_Status'] = label_encoder.fit_transform(loanData_Cleaned['EmploymentStatus'])
loanData_Cleaned['Marriage_Status'] = label_encoder.fit_transform(loanData_Cleaned['MarriageStatus'])
loanData_Cleaned['Sex'] = label_encoder.fit_transform(loanData_Cleaned['Gender'])

# Display the DataFrame after label encoding
print(loanData_Cleaned.head())
```

```
   CustomerID  CustomerAge  Gender         DOB  \
0        5386         67.0  Female  02/02/1956
1        1608         44.0  Female  12/08/1979
2         901         43.0    Male  21/04/1980
3        1875         46.0  Female  06/09/1977
5        7198         60.0  Female  31/03/1963

                                   Address    City MarriageStatus  \
0                   NO26 FREEDOM ROAD UBIAJA  UBIAJA        Married
1      NO. 22 JUNIOR STAFF QTRS G.R.A UBIAJA  UGBOHA        Married
2  NO.  SACRED HEART LANE EGUARE QTRS UBIAJA     NaN         Single
3                  NO.34 MARKET ROAD UBIAJA.  UBIAJA         Single
5                          EGUSI QTRS UBIAJA  IBADAN        Married

  EmploymentStatus  BankBalance    LoanID  LoanAmount  LoanTerm (Days)  \
0         Employed   210460.105  LP001008    15000000              180
1         Employed   124186.524  LP001024    15000000              180
2         Employed  3178304.000  LP001027    10000000              240
3         Employed   299566.521  LP001028   100000000              300
5         Employed    21414.715  LP001034    60000000              360
```

Figure 5: Encoding the attributes

6 Data Visualization

This section contains the various steps performed during exploratory data analysis (EDA). it shows the relationship of other variables with the target variable (LoanStatus).

1. Figure 6 shows a pie chart of the split between defaulted and repaid loans.

```python
# Count the occurrences of each category in the 'LoanStatus' column
loan_status_counts = finalLoanData['LoanStatus'].value_counts()
custom_labels = ['Defaulted', 'Repaid']

# Plot a pie chart
plt.figure(figsize=(8, 8))
plt.pie(loan_status_counts, labels=custom_labels, autopct='%1.1f%%', startangle=90, colors=['tomato', 'gold'])
plt.title('Distribution of Loan Repayment Status')
plt.show()
```

Figure 6: Distribution of Target variable class

2. The relationship between the applicants gender and outcome of the loan was explored and illustrated in Figure 7.

3. Figure 8 depicts how the applicants' employment status affects the loan status.

```python
sns.countplot(x= finalLoanData['Gender'] , hue= finalLoanData['LoanStatus'])
plt.title('Gender & Loan Status');
```

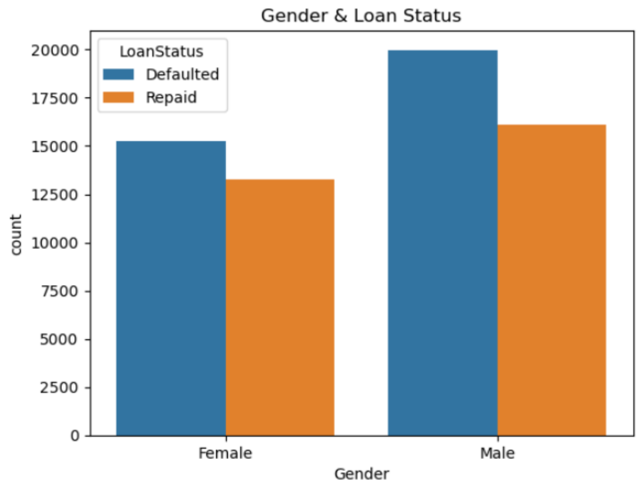


Figure 7: Relationship between Gender and LoanStatus

```python
sns.countplot(x= finalLoanData['EmploymentStatus'] , hue= finalLoanData['LoanStatus'])
plt.title('Employment Status & Loan Status');
```

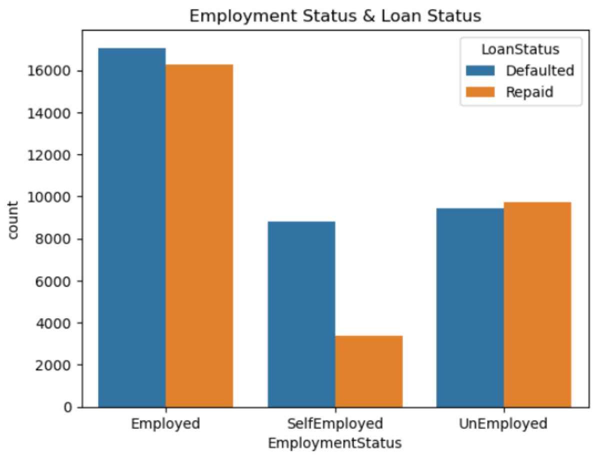


Figure 8: Relationship between EmploymentStatus and LoanStatus

7 Final Data Save

After data preprocessing, transformation, and exploration, the final data was saved to a csv file using the Python command as seen in Figure 9. A method from the Pandas library was used to facilitate this.

```
# Write final dataset into csv file to be used for Colab analysis

finalLoanData.to_csv('Test Data\\LoanData_Final.csv', index=False)
```

Figure 9: Saving final data to csv

8 Split Data into Train and Test

The final data was imported into Google Colab[1] to perform model analysis and evaluation. Before the machine learning models were fitted, the dataset was split into training and test data on a 80/20 basis.

```
# Split data into train test sets

from sklearn.model_selection import train_test_split
trainingSet, testSet = train_test_split(df, test_size=0.2)
```

Figure 10: Splitting dataset into train and test

9 Model Implementation and Evaluation

Both the Keras and Sklearn Python libraries were utilised in the process of implementing the models, and the sklearn.metrics library was utilised in order to compute the findings. The details of the implementation of each of the models are discussed in the following sections.

9.1 Logistic Regression

Figure 11 shows the code snippet of the Logistic Regression classifier model.

```
#import python library
from sklearn.linear_model import LogisticRegression


logreg = LogisticRegression()

#fit the model
logreg.fit(X_train,y_train)

#predict using the trained model
y_pred = logreg.predict(X_test)
```

Figure 11: Logistic Regression Model

[1] https://colab.google/

9.2 Random Forest

The random forest model was built using default parameters as seen in Figure 12 and tuned using the code illustrated in Figure 13.

```
from sklearn.ensemble import RandomForestClassifier
ranfor = RandomForestClassifier(n_estimators = 100)

ranfor.fit(X_train,y_train)
y_pred = ranfor.predict(X_test)
```

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Figure 12: Random Forest Model

Adjusting the n_estimators value to see if there'll be a difference

```
ranfor1 = RandomForestClassifier(n_estimators = 200)

ranfor1.fit(X_train,y_train)
y_pred = ranfor1.predict(X_test)
```

Figure 13: Tuned Random Forest Model

9.3 Decision Trees

The decision tree model was built with various hyperparameters, as shown in Figure 14

```
from sklearn.tree import DecisionTreeClassifier
dTree1 = DecisionTreeClassifier(criterion = "entropy", random_state = 100, max_depth=3, min_samples_leaf=5)

dTree1.fit(X_train,y_train)
y_pred = ranfor.predict(X_test)
```

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[5576, 1355],
       [1246, 4754]])
```

```
dTree2 = DecisionTreeClassifier(criterion = "entropy", random_state = 300, max_depth=10, min_samples_leaf=5)

dTree2.fit(X_train,y_train)
y_pred = ranfor.predict(X_test)

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[5576, 1355],
       [1246, 4754]])
```

```
dTree3 = DecisionTreeClassifier(criterion = "entropy", random_state = 300, max_depth=10, min_samples_leaf=10)

dTree3.fit(X_train,y_train)
y_pred = ranfor.predict(X_test)

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[5576, 1355],
       [1246, 4754]])
```

No difference in the output of the model even after modifying it's attributes

Figure 14: Decision Tree Model

9.4 K Nearest Neighbor

The KNN model was built with default parameters, as shown in Figure 15

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()

knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Figure 15: KNN Model

9.5 Naive Bayes

Default parameters were used to build the Naive Bayes model as shown in Figure 16

```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB

# instantiate the model
gnb = GaussianNB()

# fit the model
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

#Evaluate model prediction
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Figure 16: Gaussian Naive Bayes Model

9.6 XGBoost

The model was built using default hyperparameters in Figure 17 while Figure 18 shows how the hyperparameters were tuned in search for a better result.

```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)

# fit the model
xgb_model.fit(X_train, y_train)
y_pred = xgb_model.predict(X_test)

#Evaluate model prediction
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Figure 17: XGBoost Model

```
xgb_model1 = xgb.XGBClassifier(
 learning_rate =0.1,
 n_estimators=200,
 max_depth=4,
 min_child_weight=6,
 gamma=0.3,
 subsample=0.8,
 colsample_bytree=0.8,
 reg_alpha=0.005,
 objective= 'binary:logistic',
 nthread=4,
 scale_pos_weight=1,
 seed=27)


# fit the model
xgb_model1.fit(X_train, y_train)
y_pred = xgb_model1.predict(X_test)

#Evaluate model prediction
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Figure 18: Tuned XGBoost Model

9.7 Deep Neural Networks

The keras package was used to build this model, Figure 19 and Figure 20 show the code used to build the models.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# define the keras model
model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=10, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

Figure 19: DNN Model

```
# define the keras model
model1 = Sequential()
model1.add(Dense(24, input_shape=(8,), activation='relu'))
model1.add(Dense(18, activation='relu'))
model1.add(Dense(12, activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
# compile the keras model
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model1.fit(X_train, y_train, epochs=20, batch_size=10)
# evaluate the keras model
_, accuracy = model1.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

Figure 20: Tuned DNN Model

9.8 Model Evaluation

Each model underwent evaluation using an agreed set of metrics, which included Accuracy, Precision, Recall, and AUC score. Figure 21 shows the confusion matrix and evaluation results for XGBoost, the best performing model.

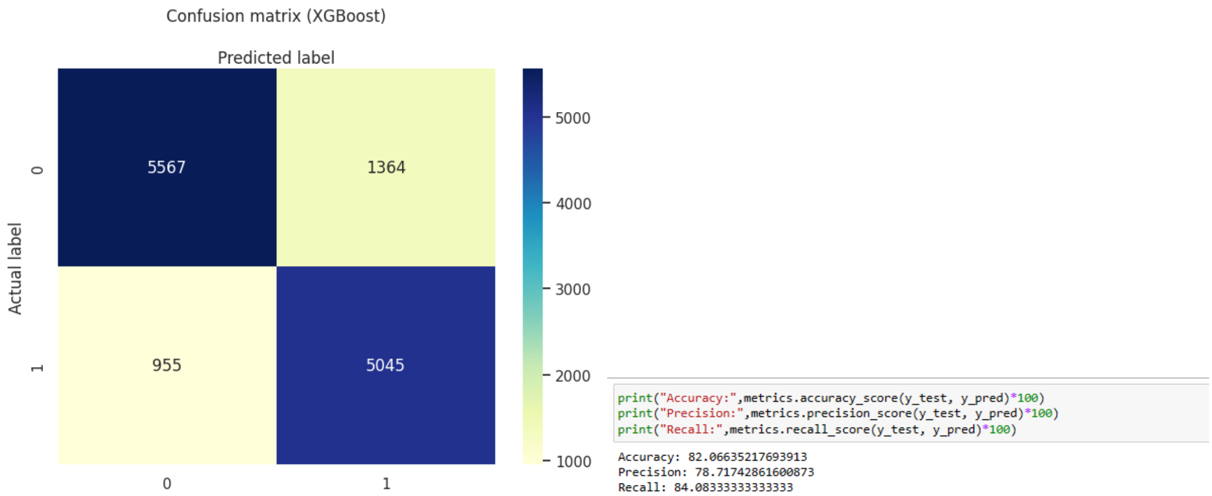


Figure 21: XGBoost Model Evaluation