National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Avinash Sanjay Gawale
Student ID: x20247303

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Avinash Sanjay Gawale
Student ID:	x20247303
Programme:	MSc in Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Jorge Basilio
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	745
Page Count:	11

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:	19th September 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

Avinash Sanjay Gawale
x20247303

1 Introduction

The configuration manual includes everything needed to replicate the study's findings on a particular environment. A snapshot of the code for data import and pre-processing, exploratory data analysis, all built models, and evaluation is included, along with the necessary tools and hardware.

The report is structured as follows: Information regarding the configuration of the environment is provided in Section 2. Information about data gathering is detailed in Section 3. Data pre-processing and exploratory data analysis are included in Section 4. Information on data splitting for the training and testing phases is provided in Section 5. Details on each model created, along with results and visualizations, are provided in Section 6.

2 Environment

Details about the software and hardware needed to put the research into practice are provided in this section.

2.1 Hardware Requirements

The necessary hardware and software specifications are provided in Figure 1 and Figure 2. Apple M1 chip with 8 GB installed unified RAM memory, 512 GB SSD, and 4 performance and 4 efficiency cores.

MacBook Pro

Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro17,1
Chip:	Apple M1
Total Number of Cores:	8 (4 performance and 4 efficiency)
Memory:	8 GB
System Firmware Version:	6723.140.2
OS Loader Version:	6723.140.2
Serial Number (system):	FVFG15EUQ05F
Hardware UUID:	A4E02BC4-AA75-58F4-A467-1F11069AE047
Provisioning UDID:	00008103-001218D001B9001E
Activation Lock Status:	Enabled

Figure 1: System Hardware Overview

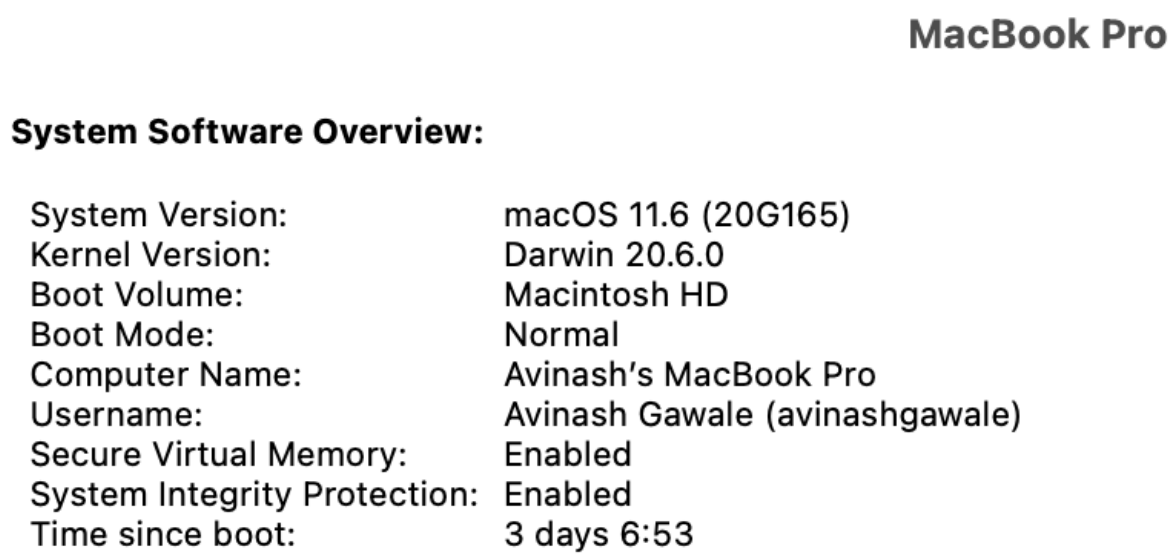

Figure 2: System Software Overview

2.2 Software Requirements

- Python (Version 3.7.13)
- Google Colab : Google Colab, a robust framework for learning and rapidly building machine learning models in Python, is used to carry out the project. Based on Jupyter Notebook, it facilitates team development. Colab is particularly well suited to machine learning, data analysis, and education. It enables anyone to create and execute arbitrary Python script through the browser.

3 Data Collection

The dataset is taken from UCI machine learning repository, which is available on link `https://archive.ics.uci.edu/ml/datasets/wine+quality`. The dataset includes red and white variants of Portuguese "Vinho Verde" wine. The dataset contains 6497 samples of red and white wine. These datasets can be used to perform regression or classification tasks. The dataset consists one dependent variable quality based on sensory data and 11 independent variables based on physicochemical testing.

4 Data Exportation

4.1 Importing Libraries

Initially, a few of the common libraries required to build a model for predicting wine quality are installed. Figure 3 shows some of the standard libraries, including NumPy, matplotlib, pandas, and seaborn. The latest versions of these libraries are installed.

```
#Importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from warnings import filterwarnings
filterwarnings(action='ignore')
```

Figure 3: Required Python Libraries

4.2 Importing and Reading Dataset

The dataset file is available in CSV format. The code to import and read the dataset is included in Figure 4.

```
#Importing dataset
from google.colab import files
uploaded = files.upload()
```

```
#Reading csv file
wine_df = pd.read_csv("winequality.csv")
```

Figure 4: Importing and Reading Dataset

The code to check the data head and data shape is included in Figure 5

```
#Checking DataFrame head
wine_df.head()
```

```
#Checking DataFrame shape
wine_df.shape
```

Figure 5: Checking Data Head and Shape

4.3 Exploratory Data Analysis

The code for generating the count plot for our dependent variable, quality, is shown in Figure 6

```
#Countplot of quality attribute
sns.countplot(wine_df['quality'])
plt.show()
```

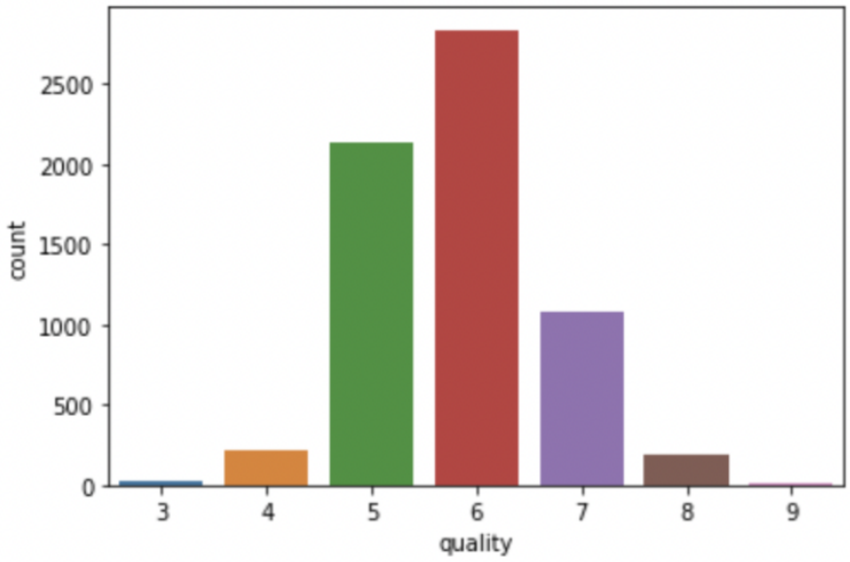

Figure 6: Count Plot of Dependent Variable

4.4 Data Pre-Processing & Transformation

Code for checking the null values is shown in Figure 7

```
#Checking Null values
wine_df.isnull().sum().sort_values(ascending=False)
```

```
fixed acidity           10
pH                       9
volatile acidity         8
sulphates                4
citric acid              3
residual sugar           2
chlorides                2
type                     0
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
alcohol                  0
quality                  0
dtype: int64
```

Figure 7: Checking Null Values

As seen in Figure 8, all null values were replaced with mean values.

```
#Attributes with Null values
missing_val_cols = ["fixed acidity", "pH", "volatile acidity", "sulphates", "citric acid", "residual sugar", "chlorides"]
```

```
#Replacing Null values with mean
for col in missing_val_cols:
    mean = wine_df[col].mean()
    wine_df[col].fillna(mean, inplace=True)
```

Figure 8: Replacing Null Values

The Figure 9 represents the code to check and remove the outliers present in the data if any.

```
#Checking for outliers in the data and removing if any
def remove_outlier(df, col_name):
    plt.figure(figsize=(20,20))
    f, axes = plt.subplots(1, 2,figsize=(12,4))
    sns.boxplot(data = df,x = col_name, ax=axes[0], color='skyblue').set_title("Before Outlier Removal: "+col_name)
    Q1 = df[col_name].quantile(0.25)
    Q3 = df[col_name].quantile(0.75)
    IQR = Q3-Q1
    df[col_name] = df[col_name].apply(lambda x : Q1-1.5*IQR if x < (Q1-1.5*IQR) else (Q3+1.5*IQR if x>(Q3+1.5*IQR) else x))
    sns.boxplot(data = df, x = col_name, ax=axes[1], color='pink').set_title("After Outlier Removal: "+col_name)
    plt.show()
    return df
```

```
for col in X.columns:
    df = remove_outlier(wine_df,col)
plt.show()
```

Figure 9: Outliers Check and Removal

Synthetic Minority Oversampling Technique (SMOTE) is applied to balance the data as the data was imbalanced. The quality check is performed in which if wine quality is greater than 6 out of 10 then it's good quality wine if it is less than 6 then is bad quality wine as shown in Figure 10. After quality check we get to know the data is imbalanced in terms of quality. There were 4113 wine samples fall under good quality and 2384 sample fall under bad quality wine as illustrated in Figure 11. After applying SMOTE good and bad quality samples were equalised as can be seen in Figure 12.

```
#Create classification version of target variable
wine_df['goodquality'] = [1 if x >= 6 else 0 for x in wine_df['quality']]
```

Figure 10: Creating Classification Version of Target Variable

```
#Checking proportion of good vs bad wine_dfs
wine_df['goodquality'].value_counts()
```

```
1    4113
0    2384
```

Figure 11: Before Applying SMOTE

```
from imblearn.over_sampling import SMOTE
oversample = SMOTE(k_neighbors=4)
# transform the dataset
X,Y = oversample.fit_resample(X, Y)
```

```
#Checking proportion of good vs bad wine_dfs
Y.value_counts()
```

```
1    4113
0    4113
```

Figure 12: After Applying SMOTE

5 Data Preparation

5.1 Data Splitting

Figure 13 provides the code for splitting the data into training and testing phase in the ratio of 70:30.

```
#Spilliting the dataset in train and test
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3,random_state=7)
```

Figure 13: Splitting Dataset in Train and Test

6 Model Implementation and Evaluation

Evaluation of confusion matrix and classification report is shown in Figure 14

```
#Evaluating confusion matrix and classification report
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report, ConfusionMatrixDisplay, precision_score,recall_score, f1_score
def model_performance(modelName,model,X_test,y_test):
    print("_____")
    print("Model:",modelName)
    y_pred = model.predict(X_test)
    #Accuracy
    print("Accuracy Score:",accuracy_score(Y_test,y_pred))
    #Precision
    print("Precision:",precision_score(Y_test,y_pred))
    #Recall
    print("Recall:",recall_score(Y_test,y_pred))
    #F1 Score
    print("F1 Score:", f1_score(Y_test,y_pred))
    #Confusion Matrix
    cm = confusion_matrix(Y_test,y_pred)
    print("Confusion Matrix:\n",cm)
    cmd = ConfusionMatrixDisplay(cm,display_labels=["good","bad"])
    cmd.plot()
    print("Classification Report:\n",classification_report(y_test,y_pred))
    plt.show()
    print("_____")
```

Figure 14: Function for Model Evaluation

6.1 Decision Tree Implementation

Evaluation of decision tree with classification report and confusion matrix is shown in Figure 15.

```
#Decision Tree implementation
from sklearn.tree import DecisionTreeClassifier
DecisionTree_model = DecisionTreeClassifier(criterion='entropy',random_state=7)
DecisionTree_model.fit(X_train,Y_train)
y_pred = DecisionTree_model.predict(X_test)
```

```
model_performance('DecisionTreeClassifier',DecisionTree_model,X_test,Y_test)
```

```
--------------------------------------------------
Model: DecisionTreeClassifier
Accuracy Score: 0.7925445705024311
Precision: 0.7832278481012658
Recall: 0.8061889250814332
F1 Score: 0.7945425361155698
Confusion Matrix:
 [[966 274]
 [238 990]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.78      0.79      1240
           1       0.78      0.81      0.79      1228

    accuracy                           0.79      2468
   macro avg       0.79      0.79      0.79      2468
weighted avg       0.79      0.79      0.79      2468
```

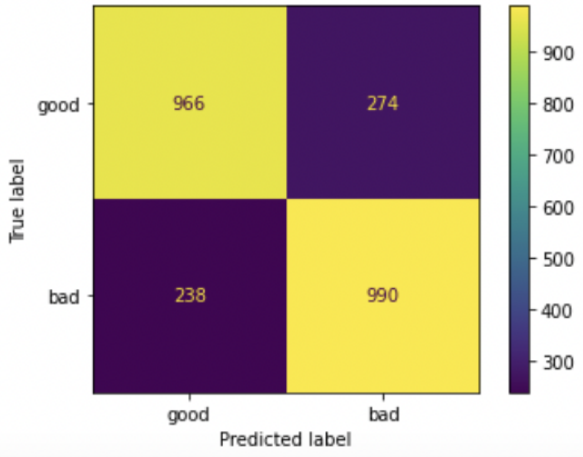

Figure 15: Decision Tree Implementation

6.2 Random Forest Implementation

Evaluation of random forest with classification report and confusion matrix is shown in Figure 16.

```python
#Random Forest implementation
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(random_state=1)
model2.fit(X_train, Y_train)
y_pred2 = model2.predict(X_test)
```

```python
model_performance('RandomForestClassifier',model2,X_test,Y_test)
```

```
--------------------------------------------------
Model: RandomForestClassifier
Accuracy Score: 0.8557536466774717
Precision: 0.8682432432432432
Recall: 0.8371335504885994
F1 Score: 0.8524046434494196
Confusion Matrix:
 [[1084  156]
 [ 200 1028]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.87      0.86      1240
           1       0.87      0.84      0.85      1228

    accuracy                           0.86      2468
   macro avg       0.86      0.86      0.86      2468
weighted avg       0.86      0.86      0.86      2468
```

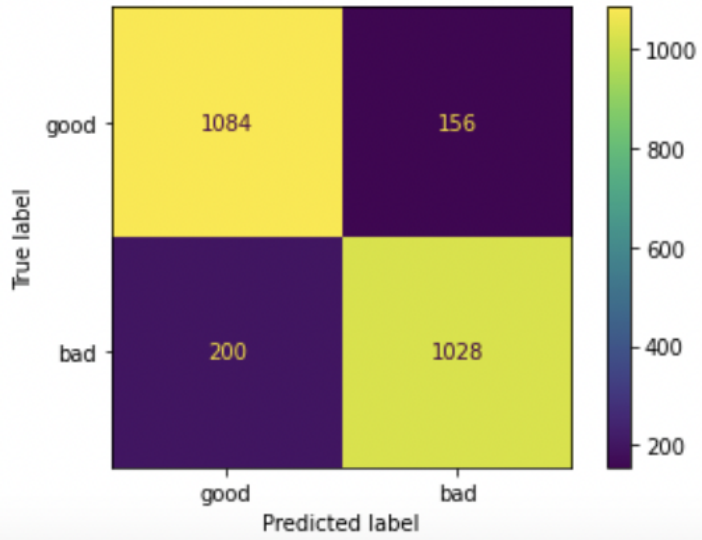

Figure 16: Random Forest Implementation

6.3 XGBoost Implementation

Evaluation of XGBoost with classification report and confusion matrix is shown in Figure 17.

```
#XGBoost implementation
import xgboost as xgb
model5 = xgb.XGBClassifier(random_state=1)
model5.fit(X_train, Y_train)
y_pred5 = model5.predict(X_test)
```

```
model_performance('xgboost',model5,X_test,Y_test)
```

```
_____________________________________________________
Model: xgboost
Accuracy Score: 0.7807941653160454
Precision: 0.7953568357695615
Recall: 0.753257328990228
F1 Score: 0.7737348389795065
Confusion Matrix:
 [[1002  238]
 [ 303  925]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.81      0.79      1240
           1       0.80      0.75      0.77      1228

    accuracy                           0.78      2468
   macro avg       0.78      0.78      0.78      2468
weighted avg       0.78      0.78      0.78      2468
```

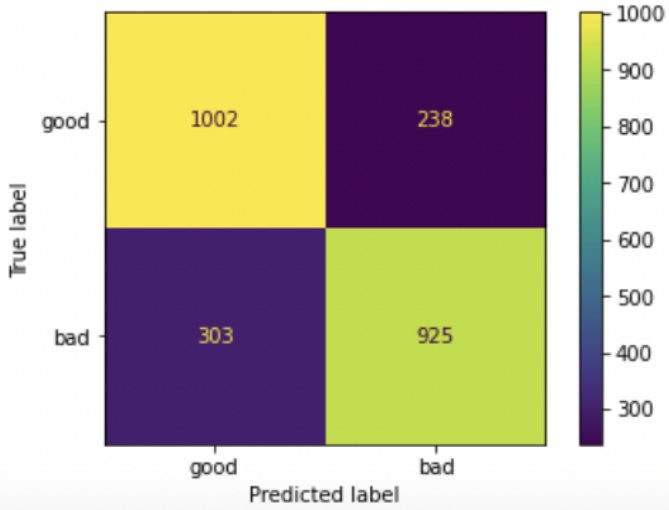

Figure 17: XGBoost Implementation

6.4 Hybrid Model

To build the hybrid model, five instances of Decision Tree, Random Forest, and XGBoost are created and the best result from each of the 15 instances is chosen as the final output, as illustrated in Figure 18.

```
#Defining Hybrid Ensemble Learning Model
#create the sub-models
estimators = []
```

```
#Defining 5 Decision Tree Classifiers
from sklearn.tree import DecisionTreeClassifier
model1 = DecisionTreeClassifier(max_depth = 3)
estimators.append(('cart1', model1))
model2 = DecisionTreeClassifier(max_depth = 4)
estimators.append(('cart2', model2))
model3 = DecisionTreeClassifier(max_depth = 5)
estimators.append(('cart3', model3))
model4 = DecisionTreeClassifier(max_depth = 2)
estimators.append(('cart4', model4))
model5 = DecisionTreeClassifier(max_depth = 3)
estimators.append(('cart5', model5))
```

```
#Defining 5 XGBoost classifiers
model11 = xgb.XGBClassifier(random_state=42)
estimators.append(('xgb1', model11))
model12 = xgb.XGBClassifier(random_state=45)
estimators.append(('xgb2', model12))
model13 = xgb.XGBClassifier( random_state=40)
estimators.append(('xgb3', model13))
model14 = xgb.XGBClassifier( random_state=46)
estimators.append(('xgb4', model14))
model15 = xgb.XGBClassifier(random_state=48)
estimators.append(('xgb5', model15))
```

```
#Defining 5 Random Forest Classifiers
model21 = RandomForestClassifier(max_depth = 3,random_state=1)
estimators.append(('rfc1', model21))
model22 = RandomForestClassifier(max_depth = 4,random_state=1)
estimators.append(('rfc2', model22))
model23 = RandomForestClassifier(max_depth = 5,random_state=1)
estimators.append(('rfc3', model23))
model24 = RandomForestClassifier(max_depth = 6,random_state=1)
estimators.append(('rfc4', model24))
model25 = RandomForestClassifier(max_depth = 7,random_state=1)
estimators.append(('rfc5', model25))
```

Figure 18: Defining Hybrid Model

Evaluation of hybrid model with classification report and confusion matrix is shown in Figure 19.

```
#Hybrid Machine Learning Model implementation
from sklearn.ensemble import VotingClassifier
ensemble = VotingClassifier(estimators)
ensemble.fit(X_train, Y_train)
y_pred = ensemble.predict(X_test)
```

```
model_performance('HybridModel',ensemble,X_test,Y_test)
```

```
_____________________________________________________
Model: HybridModel
Accuracy Score: 0.7771474878444085
Precision: 0.79073756432247
Recall: 0.750814332247557
F1 Score: 0.7702589807852965
Confusion Matrix:
 [[996 244]
 [306 922]]
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.80      0.78      1240
           1       0.79      0.75      0.77      1228

    accuracy                           0.78      2468
   macro avg       0.78      0.78      0.78      2468
weighted avg       0.78      0.78      0.78      2468
```

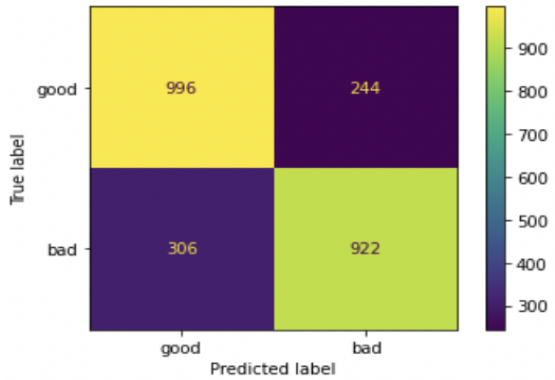

Figure 19: Hybrid Model Implementation