National College of Ireland

# Configuration Manual

Research Practicum Part 2
MSc Data Analytics

Alan Joseph .
Student ID: 23416068

School of Computing
National College of Ireland

Supervisor: Anderson Simiscuka

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

**Student Name:** ……. Alan Joseph . ……

**Student ID:** ……..23281804……..……

**Programme:** …… MSc Data Analytics …… **Year:** ……2025……..

**Module:** …… Research Practicum Part 2 ……..……

**Lecturer:** …… …… Anderson Simiscuka……..……

**Submission Due Date:** …… Thursday 11th December 2025……..……

**Project Title:** PulseGuard: A Two-Stage Explainable Machine Learning Framework for Heart Disease Detection and Severity Prediction

**Word Count:** ……1096…… **Page Count:** ……6……..……..……

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:** ……11-12-2025……

**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

Alan Joseph.
Student ID: 23281804

## 1 Introduction

The following configuration manual reveals the specific technical specifications required to carry out, setup, and install the two-stage machine-learning system, which was developed under the PulseGuard research project, an assay structure aimed at forecasting whether a heart disease exists during Stage 1 and the severity of the disease during Stage 2. The manual is aimed at allowing the researchers, lecturers or examiners to reproduce the modelling pipeline completely, re-run all experiments, and re-check the results reported in the dissertation.

The system has two machine-learning modules:

- Stage1: Binary classification model trained on the Kaggle Cardiovascular Dataset to determine the presence of a disease.
- Stage 2: Multi-class severity classification model that is trained using the UCI Heart Disease Combined Dataset.

All the experiments were performed in a Python-based analytic environment that uses open-source libraries, therefore, making it reproducible and transparent. All the working process of preprocessing, model training, evaluation, and figure generation are done within a Jupyter Notebook-based environment.

## 2 System Requirements

PulseGuard pipeline was designed and tested on Windows 10 workstation that had an AMD Ryzen-series processor and 16 GB of RAM such that all machine-learning experiments could run smoothly. Though the system has the ability to enable the acceleration of the graphics cards through an NVIDIA RTX-series graphic card, all models in this research were trained and tested on CPU, because the computational requirements were manageable. Both Windows 10 and Windows 11 operating environments were included in the operating environment and are both fully compatible with the Python-based tools. The Python 3.10 or later was used to implement the development workflow, and Jupyter Notebook was used as the main tool of sequential experimentation, visualisation, and debugging. The configuration fits into the methodological framework proposed in the configuration manual and provides a replicable environment to implement the two stage prediction models.

All of the packages in a specific virtual Python environment were installed using pip to prevent dependency conflicts and to guarantee that the libraries would be reproducible.

## 2.1 Software Requirements

**Core Software**

- Python 3.10 or above
- Jupyter Notebook

  **Required Python Libraries**

  Install using:

  pip install <library-name>

| Library | Purpose |
|---|---|
| **pandas** | Data loading, preprocessing, cleaning |
| **numpy** | Numerical operations and array handling |
| **matplotlib** | Plotting and visualisation |
| **seaborn** | Statistical and heatmap visualisations |
| **scikit-learn** | ML models, preprocessing, metrics, scaling |
| **xgboost** | XGBoost classifier (used in both Stage-1 and Stage-2) |
| **imblearn** | SMOTE oversampling for data balancing |
| **joblib** | Saving and loading trained models |
| **shap** | Explainability and feature attribution plots |

It is recommended that all dependencies be installed inside a **virtual environment** or **Conda environment** to prevent version conflicts and ensure consistent execution across systems.

# 3 Dataset

The PulseGuard system is built on two publicly available medical datasets on which the two stages of diagnostic framework are based. In the case of Stage 1, the Kaggle Cardiovascular Disease Dataset is the main source that can be used to predict the presence of a heart disease. These data include a sample of about 70,000 anonymised patient records that are obtained as a result of regular medical check-ups. The records have 12 organized clinical characteristics, such as age, systolic and diastolic blood pressure, and cholesterol level, glucose level measurement, and lifestyle measures, such as smoking status, alcohol consumption, and physical activity. The dependent variable, cardio, is a dichotomous term, which means that the patient has or has not heart disease (1 or 0). The dataset is very appropriate in Stage 1, as it is large and well-balanced and can be used to train binary classification models.

In the case of Stage 2, the UCI Heart Disease Combined Dataset is used to classify at the level of severity. This information is a combination of four previously recognized clinical cohorts; Cleveland, Hungary, Switzerland and Long Beach VA, giving an overall total number of 920 samples of patients. It encompasses 14 clinical features of interest (including: maximum heart rate obtained in exercise, exercise-induced ST depression, chest pain type, rest ECG, and visualisation of major vessels using fluoroscopy). The target variable is the number of diseases in the form of a number (or a number system) of 0 to 4, with 0 being the absence of the disease and 4 being the worst type of coronary artery disease. This data set backs the multi-classes classification activity needed in Stage 2 so that the severity of the disease could be predicted as opposed to whether a disease is present.

**Dataset Links**
**Kaggle Cardiovascular Disease Dataset**
https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset
**UCI Heart Disease Combined Dataset**
https://archive.ics.uci.edu/dataset/45/heart+disease

# 4 Directory Structure

```
pulseguard/
│
├── cardio_train.csv
│
├── pulseguard.ipynb
│
├── uci_heart_combined.csv
│
├──pulseguard_outputs
      │
      ├──  stage1_overall_results.csv
      ├── stage2_overall_results.csv
      ├── stage1_cm.png
      ├── stage2_cm.png
      ├── stage1_roc.png
      ├── stage2_roc.png
      ├── stage1_featimp.png
      ├── stage2_featimp.png
      ├── stage1_shap.png
      ├── stage2_shap.png
      └── per_class_accuracy.csv
```

# 5 Code Execution

### Step 1: Start Jupyter Notebook

To start with, the first thing is to start Jupyter Notebook in the set up Python environment. This is what automatically opens an interactive web interface which makes it easy to implement the complete modelling workflow.

### Step 2: Load the PulseGuard Notebook.

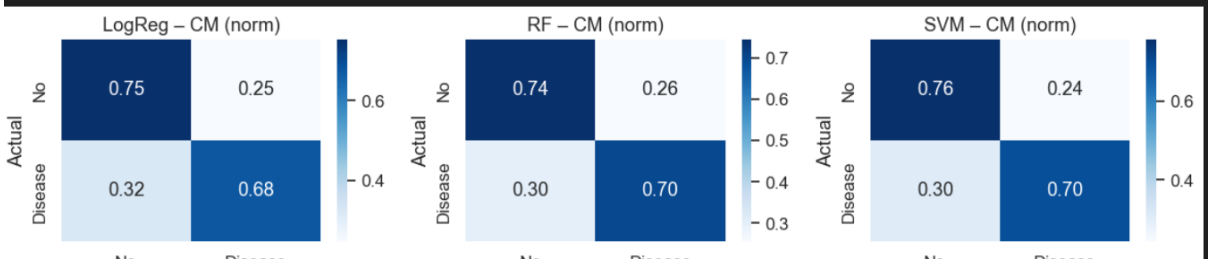
Go to the directory of the project and open the file pulseguard.ipynb. The notebook includes an elaborate sequence of data processing and model construction steps, in an organized manner.

```
import os, time, json, warnings
warnings.filterwarnings("ignore")
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid", font_scale=1.1)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.feature_selection import VarianceThreshold
from sklearn.metrics import (accuracy_score, precision_score, recall_score, f1_score,
                             classification_report, confusion_matrix, roc_curve, auc)

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

try:
    from xgboost import XGBClassifier
except Exception:
    XGBClassifier = None
    print("xgboost not installed; XGBClassifier will be skipped.")
from imblearn.over_sampling import SMOTE
import joblib
import shap
OUT = Path("pulseguard_outputs"); OUT.mkdir(exist_ok=True)
```

### Step 3: Work out every Cell one after the other.

Test the notebook one by one to ensure that the pipeline works properly. The notebook is also customizable and includes all the necessary features such as loading data, preprocessing, SMOTE oversampling, split of the dataset, training Stage 1 and Stage 2 model, confusion matrices, ROC curves, SHAP explainability plots, and preserving the final model.

```
classes1 = [0,1]
plt.figure(figsize=(12,6))
idx = 1
for name, model in trained1.items():
    y_pred = model.predict(X1_te_s)
    cm = confusion_matrix(y1_te, y_pred, labels=classes1)
    cmn = cm.astype(float) / (cm.sum(axis=1, keepdims=True) + 1e-12)
    plt.subplot(2, 3, idx)
    sns.heatmap(cmn, annot=True, fmt=".2f", cmap="Blues",
                xticklabels=["No","Disease"], yticklabels=["No","Disease"])
    plt.title(f"{name} CM (norm)"); plt.xlabel("Predicted"); plt.ylabel("Actual")
    idx += 1
plt.tight_layout();
plt.savefig(OUT/"stage1_all_confusion_matrices.png", dpi=300)
plt.show()
```




## Step 4: Allow Processing Time

Some of the stages can take long to execute based on the performance of the system. Stage 1 models are quicker to finish in comparison to SVM-based stages, which can take longer. Multi-class classification and SMOTE balancing of Stage 2 require more computation.

```
models1 = {
    "LogReg": LogisticRegression(max_iter=4000, class_weight="balanced"),
    "RF": RandomForestClassifier(n_estimators=400, random_state=42, class_weight="balanced_subsample"),
    "SVM": SVC(probability=True, kernel="rbf", class_weight="balanced", random_state=42),
    "MLP": MLPClassifier(hidden_layer_sizes=(64,32), activation="relu", solver="adam", max_iter=800, random_state=42),
}
if XGBClassifier is not None:
    models1["XGB"] = XGBClassifier(
        n_estimators=600, max_depth=4, learning_rate=0.08,
        subsample=0.9, colsample_bytree=0.9,
        objective="binary:logistic", random_state=42
    )

trained1, rows1 = {}, []
for name, mdl in models1.items():
    t0 = time.time(); mdl.fit(X1_tr_sm, y1_tr_sm); t1 = time.time()
    yhat = mdl.predict(X1_te_s)
    acc  = accuracy_score(y1_te, yhat)
    prec = precision_score(y1_te, yhat, zero_division=0)
    rec  = recall_score(y1_te, yhat, zero_division=0)
    f1   = f1_score(y1_te, yhat, zero_division=0)
    rows1.append({"Model":name, "Accuracy":acc, "Precision":prec, "Recall":rec, "F1":f1, "TrainSec":t1-t0})
    trained1[name] = mdl
    joblib.dump(mdl, OUT/f"stage1_{name}.pkl")

stage1_overall = pd.DataFrame(rows1).sort_values(["F1","Accuracy"], ascending=False)
display(stage1_overall)
stage1_overall.to_csv(OUT/"stage1_overall_results.csv", index=False)
print("Saved:", OUT/"stage1_overall_results.csv")
```

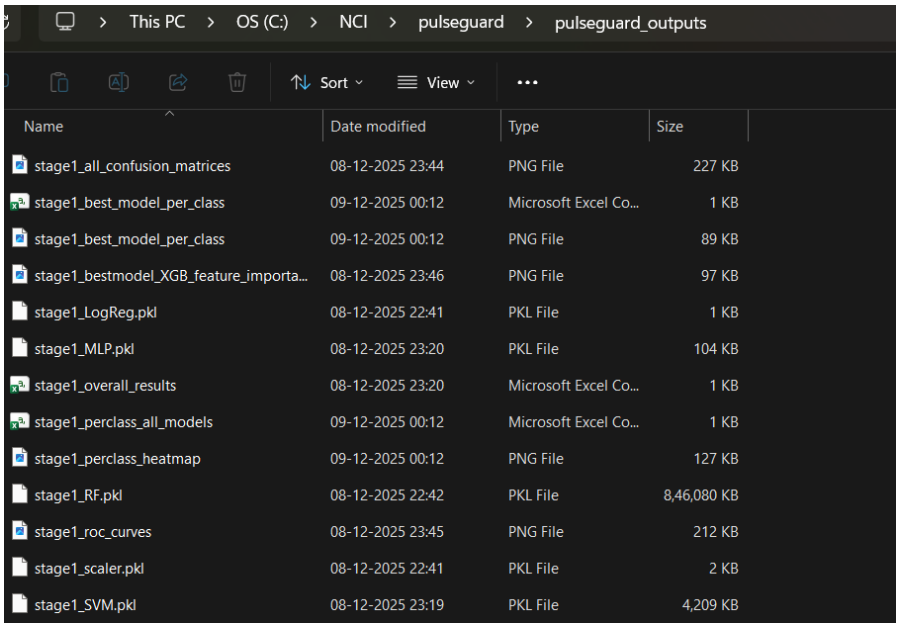
## Step 5: Examine Generation of Output.

In the course of pipeline execution, a range of evaluation metrics become obtainable, including accuracy, precision, recall, F1-scores, instance -class forecasts, confusion matrices, ROC-AUC curves and SHAP visualisations. These findings will give an in-depth evaluation of every step of the process.

| | Model | Accuracy | Precision | Recall | F1 | TrainSec |
|---|---|---|---|---|---|---|
| 4 | XGB | 0.733286 | 0.750384 | 0.698685 | 0.723612 | 2.030784 |
| 3 | MLP | 0.723429 | 0.733483 | 0.701401 | 0.717083 | 98.692965 |
| 2 | SVM | 0.725357 | 0.739620 | 0.695111 | 0.716675 | 2161.312956 |
| 1 | RF | 0.722000 | 0.732127 | 0.699686 | 0.715539 | 45.423391 |
| 0 | LogReg | 0.714000 | 0.731436 | 0.675815 | 0.702526 | 0.217108 |

**Step 6: Locate Exported Files**

All the output files such as CSV summaries and visualization are automatically saved in the /pulseguard_outputs folder. One can view performance tables, plots and exported models by accessing this directory.




**Step 7: Repetition or Revision of Experiments.**

The notebook may be re-run to add new settings of hyperparameters, algorithm changes, or even more evaluation techniques. Existing executions update automatically all results and all files exported in the pipeline.