National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Haritha Kutcharlapati
Student ID: x23213248

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Haritha Kutcharlapati

Student ID: X23213248

Programme: MSc Data Analytics **Year:** 2024 - 2025

Module: MSc Research Project

Lecturer: Prof. Furqan Rustam

Submission Due Date: 12/12/2024

Project Title: Enhancing Cardiovascular Risk Prediction Through Machine Learning and Deep Learning

Word Count: 892 **Page Count:** 09

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: Haritha Kutcharlapati

Date: 12/12/24

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

Haritha Kutcharlapati

Student ID: x23213248

1. Introduction

This configuration manual contains the procedure of recreating the experiment as well as the methodology for using machine learning and deep learning in the prediction of cardiovascular diseases. The study uses such as demographic and physiological and lifestyle traits, which include age, sex, blood pressure cholesterol and glucose levels, to identify people with healthy and cardiovascular disease. Algorithms under consideration included the setup models such us Logistic Regression, Random Forest, SVC, Decision Tree, DNN, RNN, and LSTM. feature scaling and encoding methods were used during data preprocessing for accurate model prediction. Among the traditional model, Logistic Regression performed the best, whereas LSTM and DNN are more effective when data shows some more complex pattern.

2. Deployment Environment

The development environment used for this research is the local windows operating systemswith GPU. Both the hardware and the software specification details are mentioned below.

The datasets for this project are obtained from Kaggle and include patient health records with key attributes such as age, cholesterol levels, blood pressure, and This document provides the specifications of the software, packages, and configurations needed so that it can provide similar experimental environment hence similar results.
more.

2.1 Hardware Specification

- Processor: Intel i5 3.60 GHz or equivalent
- RAM: 8.0 GB (15.4 GB usable)
- GPU – NIVIDA RTX 3050

This above-mentioned Hardware Specs based Local System was used to create the environment, to re-run the setup it is not necessary to have the same specification to re-create the environment.

2.2 Software Specification

2.2.1 Operating System: Windows 10/11 or Ubuntu 20.04+

2.2.2 Programming Language: Python version 3.11.5

Figure 1: Python Version

2.2.3 Integrated Development Environment (IDE): Jupyter Notebook 6.5.4 or higherversion. Google Colab.

2.3 Python Libraries Required

Figure 3 display the list of the essential Python Libraries required for the execution of thecode. This mentioned python libraries can be installed using the pip command.

- Numpy
- Pandas
- Scikit-learn
- Matplotlib
- Seaborn
- TensorFlow
- Keras

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler,MinMaxScaler,OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt
from rich import print

# For model training
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier, AdaBoostClassifier
from xgboost import XGBClassifier

# For model performance evaluation
from sklearn.pipeline import Pipeline
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, mean_absolute_error
```

Figure 2: Libraries Imported

3. Data Source

The datasets for this project are obtained from Kaggle and include patient health records with key attributes such as age, cholesterol levels, blood pressure, and more.

- Cardiovascular Disease Dataset: https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset

4. Project Code Files

- **Data Cleaning and Preprocessing Notebook:**

Includes handling missing values, encoding categorical features, and feature scaling.

- **Model Training Notebook:**
  Implements and trains Logistic Regression, Random Forest, ANN, and CNN models.
- **Evaluation Notebook:**
  Evaluates models using accuracy, precision, recall, and F1-score metrics.

5. Data Preparation

5.1 Extracting Data

Load the dataset into Pandas DataFrame from a CSV file.

```
# Reading the data from csv file
df = pd.read_csv("/content/cardio_train.csv",sep=';' , on_bad_lines='skip')
df.drop(['id'],axis=1,inplace=True)
```

Figure 3: Dataset Imported

5.2 Data Preprocessing

- **Handling Missing Values:** Replace missing values with mean or median.
- **Encoding Categorical Features:** Convert categorical variables into numeric using label encoding.
- **Feature Scaling:** Apply standardization for continuous features.

```
duplicate_count = df.duplicated().sum()
if duplicate_count>0:
  df.drop_duplicates(inplace=True)
  print(f"{duplicate_count} duplicate values removed")
else:
  print("There are no duplicate values")
```

```
24 duplicate values removed
```

Figure 3: Handling duplicate values

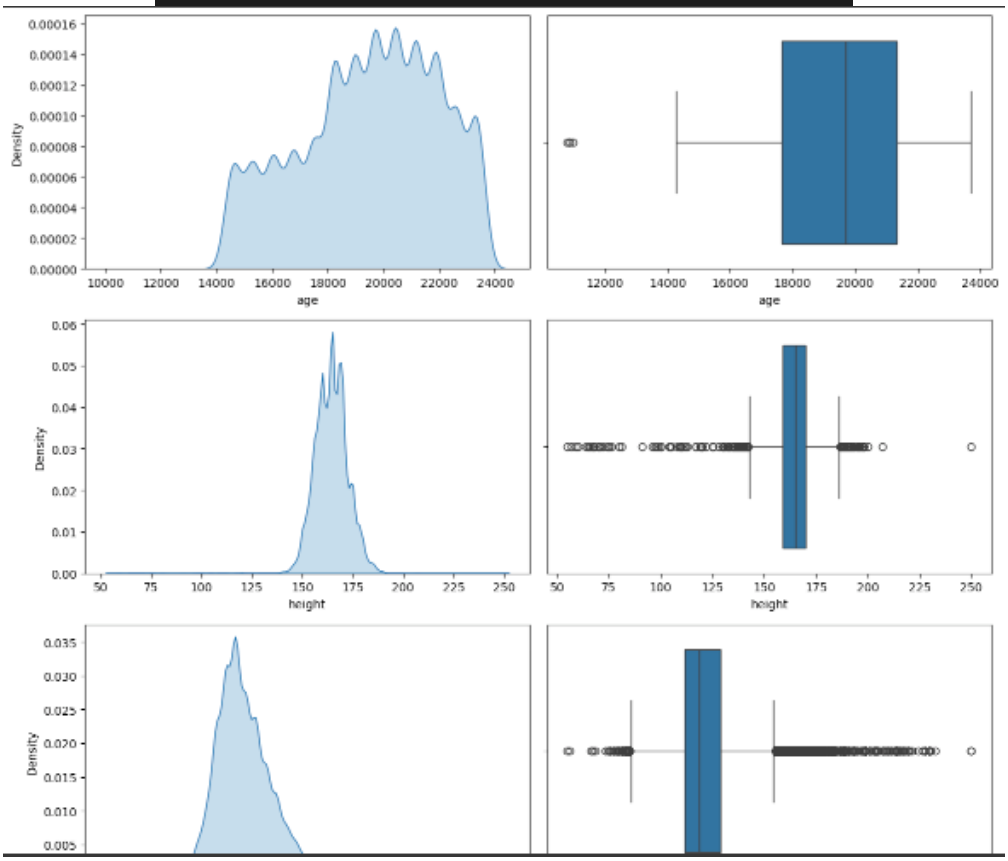
5.3 Feature Engineering

- Extract relevant features such as BMI, cholesterol ratio, and resting heart rate.
- Derive new features using domain knowledge, e.g., heart risk factor.

6. Exploratory Data Analysis (EDA)

- Visualize correlations between features using heatmaps.
- Analyze distribution patterns for key variables like age and cholesterol.
- Detect and handle outliers using boxplots and the IQR method.

```
# Removing the extreme outliers
df = df[(df['ap_hi'] > 0) & (df['ap_hi'] < 250)]
df = df[(df['ap_lo'] > 0) & (df['ap_lo'] < 500)]

fig, axs = plt.subplots(5, 2, figsize=(15, 15))

# Plotting price
sns.kdeplot(df['age'], ax=axs[0, 0], fill=True)
sns.boxplot(data=df, x='age', ax=axs[0, 1])

# Plotting height
sns.kdeplot(df['height'], ax=axs[1,0], fill=True)
sns.boxplot(data=df, x='height', ax=axs[1, 1])

# Plotting weight
sns.kdeplot(df['weight'], ax=axs[2, 0], fill=True)
sns.boxplot(data=df, x='weight', ax=axs[2, 1])

# Plotting ap_hi
sns.kdeplot(df['ap_hi'], ax=axs[3, 0], fill=True)
sns.boxplot(data=df, x='ap_hi', ax=axs[3, 1])

# Plotting ap_lo
sns.kdeplot(df['ap_lo'], ax=axs[4, 0], fill=True)
sns.boxplot(data=df, x='ap_lo', ax=axs[4, 1])

plt.tight_layout()
plt.show()
```



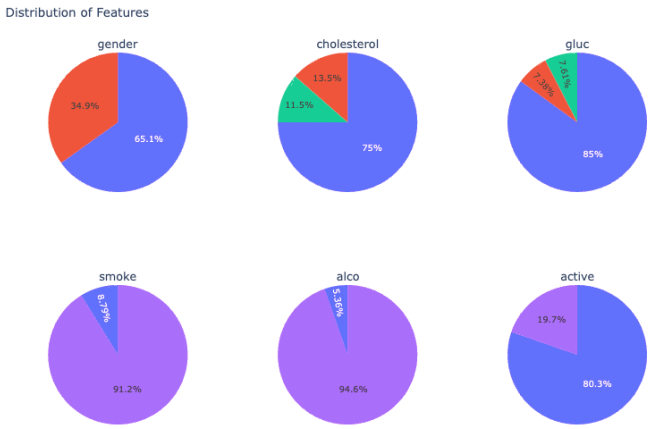


Figure 4: Removing Outliers and visualizations

7. Model Building

7.1 Splitting Data

Split data into training (80%) and testing (20%) subsets.

```
X = df.drop(['cardio'],axis=1)
y = df['cardio']

X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.8)
print(X_train.shape)
print(X_test.shape)
```

```
(55168, 10)
(13793, 10)
```

Figure 5: Splitting training and testing data

7.2 Model Training

- Train models including Logistic Regression, Random Forest, ANN, and CNN.
- Fine-tune hyperparameters using GridSearchCV.

```python
# Dictionary of classification models with tuning for faster training
model_dict = {
    'logistic_regression': LogisticRegression(solver='lbfgs', max_iter=100, random_state=42),
    'svc': SVC(probability=True, kernel='linear', random_state=42),
    'decision_tree': DecisionTreeClassifier(max_depth=3, random_state=42),
    'random_forest': RandomForestClassifier(n_estimators=5, max_depth=5, random_state=42)
}

output = []

# Iterate through models
for model_name, model in model_dict.items():

    # Training the model
    model.fit(X_train_processed, y_train)

    # Making predictions
    y_pred = model.predict(X_test_processed)
```

Figure 6: Model training code snippet

7.3 Evaluation

Evaluate models using metrics:

- **Accuracy, Precision, Recall, and F1-score**

```python
# Create a DataFrame from the output
results_df = pd.DataFrame(output, columns=['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])

# Display the DataFrame
model_df_sorted = results_df.sort_values(['Accuracy'], ascending=False)
model_df_sorted
```

	Model	Accuracy	Precision	Recall	F1 Score
1	svc	0.728703	0.737206	0.727921	0.697543
0	logistic_regression	0.727325	0.730411	0.726838	0.707929
3	random_forest	0.726455	0.736431	0.725609	0.692276
2	decision_tree	0.725513	0.728752	0.725012	0.705461

Figure 7: Results for ML models

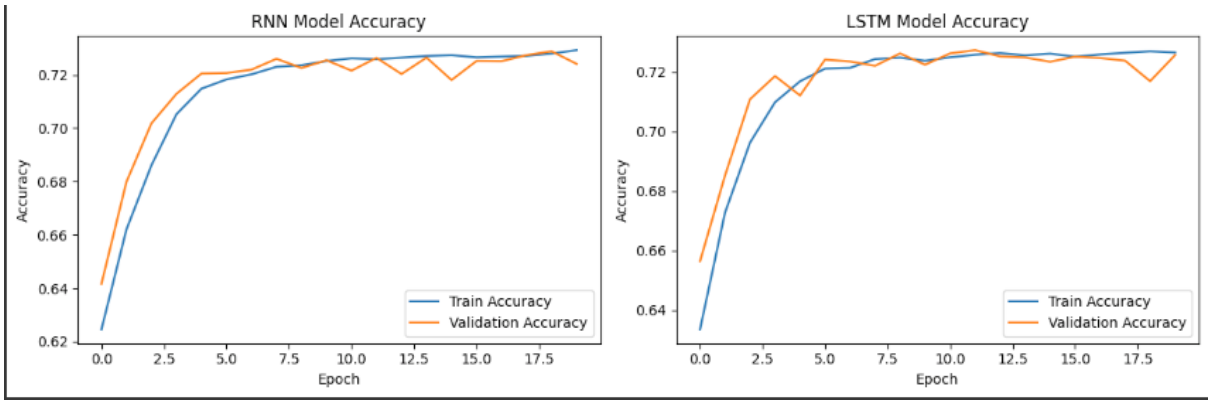


Figure 8: Comparison plot for RNN and LSTM

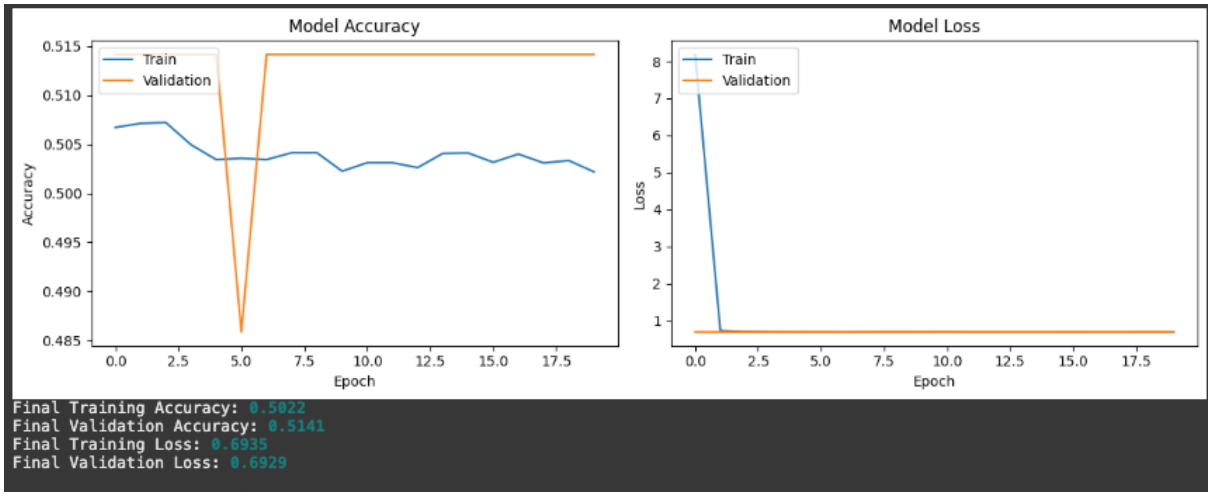


Figure 9: DNN Results

8. Hyperparameter Tuning

- Perform hyperparameter optimization for Random Forest (e.g., n_estimators, max_depth).
- Apply learning rate and epoch tuning for ANN and CNN.

Hyper-parameter Tuning

```
[ ]  # Import necessary libraries
     from sklearn.model_selection import GridSearchCV
     from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score

     # Define parameter grid for Logistic Regression
     param_grid = {
         'C': [0.001, 0.01, 0.1, 1, 10, 100],
         'penalty': ['l1', 'l2'],
         'solver': ['liblinear', 'saga'],
         'max_iter': [100, 200, 300, 500],
         'class_weight': [None, 'balanced']
     }

     # Create a new Logistic Regression model
     lr_model = LogisticRegression(random_state=42)

     # Define scoring metrics
     scoring = {
         'accuracy': make_scorer(accuracy_score),
         'precision': make_scorer(precision_score, average='macro'),
         'recall': make_scorer(recall_score, average='macro'),
         'f1': make_scorer(f1_score, average='macro')
     }

     # Create GridSearchCV object
     grid_search = GridSearchCV(
         estimator=lr_model,
         param_grid=param_grid,
         cv=5,
         scoring=scoring,
         refit='accuracy',
         verbose=2,
         n_jobs=-1
     )
```

Figure 10: Hyper-parameter Tuning