



UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UCCD2063 Artificial Intelligence Techniques

JUNE 2024

GROUP ASSIGNMENT


Student	1	2	3
Name	Ng Wei Yu	Ng Wei Hong	Ng Tuck Seng
Student ID	2207448	2106889	2207031
Programme	CS	CS	CS
Contribution	33.33%	33.33%	33.33%
Signature		<i>Wei</i>	<i>ng</i>

Table of Contents

1.0 Introduction	1
1.1 Background	1
1.2 Objectives	4
2.0 Methods.....	5
2.1 Dataset Description:	5
2.2 Data Exploration and Visualization	7
2.3 Data Pre-processing	15
2.3.1 Handling Missing Values.....	15
2.3.2 Dataset Preparation	15
2.3.3 Splitting the Data	15
2.3.4 Separating Numerical and Categorical Data	16
2.3.5 Standardizing Numerical Data	16
2.3.6 Encoding Nominal and Ordinal Data.....	17
2.3.7 Combining Transformed Data	18
2.3.8 Converting Target Variable	18
2.4 Model Selection and Justification	19
2.5 Model Training and Validation.....	22
2.6 Model Tuning and Testing	25
2.6.1 Model Tuning.....	25
2.6.2 Final Testing	27
3.0 Results and Discussion	36
3.1 Summarize the training and testing results	36
3.2 In-depth analysis of the prediction performance and errors of the models.....	38
3.2.1 SVM Model	38
3.2.2 Random Forest Model.....	43
3.2.3 K-Nearest Neighbors Model	48
3.3 Strengths and Weaknesses	52
3.4 Investigate features importance for the prediction.....	53
4.0 Conclusion	58

1.0 Introduction

1.1 Background

Cardiovascular diseases (CVDs) are a major global health concern as they are among the main causes of death. Heart-related disorders must be managed, and their incidence must be decreased, and this requires early detection and prevention. Reducing the prevalence of CVDs and improving health outcomes can result from prompt interventions based on accurate evaluation of cardiovascular risk. The goal of this project is to create predictive models that evaluate cardiovascular risk in individuals by considering a range of health-related variables. The project's goal is to categorize cardiovascular risk into three groups: low, medium, and high—using machine learning techniques. The ultimate objective is to show how machine learning may be used to solve real-world healthcare issues, especially with regard to the detection and prevention of cardiovascular illnesses. This study intends to create predictive models that can effectively measure cardiovascular risk based on these indicators by utilizing machine learning techniques. This research aims to show how machine learning may be applied to solve real-world healthcare problems, namely in the area of cardiovascular disease prevention and diagnosis, through data exploration, pre-processing, and model training.

The dataset does cover several features, including gender, age, height, weight, family history, alcohol, junk food, vege_day, meals_day, snacks, smoking, water_intake, transportation, exercise, TV, income, discipline, and cardiovascular_risk which will be used throughout the entire workflow. Data exploration is performed to go through and understand the characteristics of each of the features in the dataset which includes categorical and numerical data. This can help us to perform the following work easier. At the same time, preprocessing is a vital step in the machine learning process aimed at preparing raw data to be suitable for analysis and modelling. It involves handling missing data, either by filling gaps through techniques like imputation or by removing incomplete records. Preprocessing also addresses outliers that can introduce noise and skew model performance, often by removing or transforming these outliers to reduce their impact. Additionally, data normalisation or standardisation is performed to ensure that features with different scales contribute equally to the model, which is particularly important for algorithms sensitive to feature scaling. Categorical variables, which are non-numeric, are encoded into numerical formats using methods like one-hot encoding, making them compatible with machine learning models. In some cases, dimensionality reduction techniques, such as Principal Component Analysis (PCA), are applied to reduce the number of features, improving model performance and

reducing computational complexity. In short, data preprocessing enhances the quality and relevance of the data, leading to more accurate and reliable predictions by ensuring that the data fed into the model is clean, consistent, and well-structured.

Several machine learning algorithms have been selected and developed, which are comprised of k-nearest neighbors (K-NN), Support Vector Machine (SVM), and Random Forest (RF). Generally, K-NN is abbreviated from k-nearest neighbours. It works by identifying the 'k' nearest data points to a given query point and classifying it based on the majority label of its neighbors. K-NN is intuitive and non-parametric, meaning it doesn't make assumptions about the underlying data distribution. However, it can be computationally expensive, especially with large datasets, as it requires calculating the distance between the query point and all other points in the dataset.

SVM is a powerful class of supervised learning algorithms used for classification and regression tasks. The core idea behind SVM is to find the optimal hyperplane that best separates different classes in the feature space, maximising the margin between the closest data points of each class. This hyperplane is determined by support vectors, which are the data points closest to the decision boundary. SVMs can effectively handle both linear and nonlinear classification problems by employing kernel functions, which transform the feature space to make it possible to find a separating hyperplane in higher dimensions. With their strong theoretical foundation and ability to generalise well to unseen data, SVMs are widely used in various applications, including text classification, image recognition, and bioinformatics.

RF is an ensemble algorithm that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. With the goal to achieve this, random subsets of the data are used to train each tree, and at each node, random subsets of characteristics are chosen for splitting. For classification tasks, it averages the predictions of all the trees, while for regression, it combines the majority vote from all the trees. Robustness, high accuracy, and the capacity to manage big datasets with plenty of attributes are the hallmarks of Random Forest.

In short, all those machine learning algorithms will be developed, experimented and fine-tuned to obtain the best results for the forecasting purpose. The use of these models highlights the value of data-driven strategies in tackling pressing health issues and shows how machine learning can be applied practically. The project's ultimate success will be determined

by its capacity to accurately classify cardiovascular risk and improve health outcomes by facilitating timely interventions and well-informed decision-making.

1.2 Objectives

The major aim of this work is to create and hone machine learning models that can reliably forecast an individual's cardiovascular risk using the given dataset. In order to attain the best possible prediction accuracy, this will entail carrying out data exploration, pre-processing, model selection, training, and tuning. In the end, this will improve the diagnosis and prevention of cardiovascular illnesses.

The objectives of this task can be divided into 3 sub-objectives which are as follows:

- Develop and evaluate machine learning models capable of accurately predicting cardiovascular risk (low, medium, high) based on a diverse range of health-related features in the dataset
- Explore and implement various data preprocessing techniques, including handling missing data and feature selection, to improve the quality and relevance of the input data for the prediction models
- Optimize model performance through experimentation with different hyperparameters, training strategies, and regularisation techniques and validate the models using appropriate evaluation metrics and testing steps.

2.0 Methods

2.1 Dataset Description:

The dataset.csv consists of 2100 samples with 17 input features and 1 output feature, mainly focusing on health and lifestyle factors. The dataset is designed to analyse the risk of cardiovascular issues based on various outputs. It contains both numerical and categorical data, as well as potential complexity due to missing values or noise. Therefore, proper data preprocessing is necessary to address these issues.

Attribute of categorical data:

Column	Attributes
Gender	Male Female
Family_history	Yes No
Alcohol	Yes No
Junk_food	Yes No
Snack	Yes No
Smoking	Yes No
Transportation	Public Private Walk Cycle
TV	Yes No
Discipline	High Moderate Low

Table 2.1.1 Categorical Data

Numerical Data:

Age: Continuous variable representing the age of the individual in years.

Height(cm): Continuous variable representing the height of the individual in centimetres.

Weight(kg): Continuous variable representing the weight of the individual in kilograms.

Vege_day: Integer variable representing the number of days per week the individual consumes vegetables.

Meals_day: Integer variable representing the number of meals the individual consumes per day.

Water_intake(L): Continuous variable representing the individual's daily water intake in litres.

Exercise: Integer variable representing the number of days per week the individual exercises.

Income: Continuous variable representing the income of the individual.

Output:

Cardiovascular_risk(y): Boolean value (yes/no) representing the cardiovascular risk.

2.2 Data Exploration and Visualization

In this assignment, we aim to explore the factors influencing cardiovascular risk. The target variable in this dataset is **Cardiovascular_risk(y)**, which indicates whether an individual is at risk for cardiovascular issues. The remaining variables are input features representing various demographic, lifestyle, and health factors.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2100 entries, 0 to 2099
Data columns (total 18 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Gender                                2100 non-null   object
1   Age                                   2100 non-null   int64
2   Height(cm)                           2100 non-null   float64
3   Weight(kg)                           2100 non-null   float64
4   Family_history                        2100 non-null   object
5   Alcohol                              2100 non-null   object
6   Junk_food                            2100 non-null   object
7   Vege_day                             2100 non-null   int64
8   Meals_day                            2100 non-null   int64
9   Snack                                2100 non-null   object
10  Smoking                              2100 non-null   object
11  Water_intake(L)                      2100 non-null   float64
12  Transportation                       2100 non-null   object
13  Exercise                             2100 non-null   int64
14  TV                                    2100 non-null   object
15  Income                               2100 non-null   int64
16  Discipline                           2100 non-null   object
17  Cardiovascular_risk(y)               2100 non-null   object
dtypes: float64(3), int64(5), object(10)
memory usage: 295.4+ KB
```

Figure 2.2.1 Basic Information of the dataset

Figure 2.2.1 shows that the dataset consists of 5 columns with type int64, 3 columns of float64 type, and 10 columns with type object. In total, there are 8 numerical columns and 9 categorical columns. The memory usage of this DataFrame is approximately 295.4 KB, indicating a manageable size for most data processing tasks.

	Age	Height(cm)	Weight(kg)	Vege_day	Meals_day	Water_intake(L)	Exercise	Income
count	2100.000000	2100.000000	2100.000000	2100.000000	2100.000000	2100.000000	2100.000000	2100.000000
mean	24.302381	170.148286	86.561571	2.423333	2.687143	2.007429	1.006190	9432.386190
std	6.342270	9.340941	26.192242	0.584318	0.810088	0.613122	0.894885	5002.350673
min	14.000000	145.000000	39.000000	1.000000	1.000000	1.000000	0.000000	1000.000000
25%	20.000000	163.000000	65.400000	2.000000	3.000000	1.577500	0.000000	4994.500000
50%	23.000000	170.000000	83.000000	2.000000	3.000000	2.000000	1.000000	9226.500000
75%	26.000000	176.800000	107.250000	3.000000	3.000000	2.480000	2.000000	13841.750000
max	61.000000	198.000000	173.000000	3.000000	4.000000	3.000000	3.000000	18000.000000

Figure 2.2.2 Statistical value of numerical variables

Figure 2.2.2 presents a comprehensive summary of the statistical values for the dataset's numerical variables. Upon reviewing the data, it is notable that variables such as "Age", "Weight", and "Meals_day" exhibit significant dispersion, as reflected by their high standard deviations and wide interquartile range. The age and weight data suggest a diverse population, which could significantly influence predictive outcomes. Additionally, the variability in "Meals_day", with several outliers, indicates irregular meal patterns that might affect model robustness. These factors should be carefully considered during model development to avoid potential biases caused by these outliers.

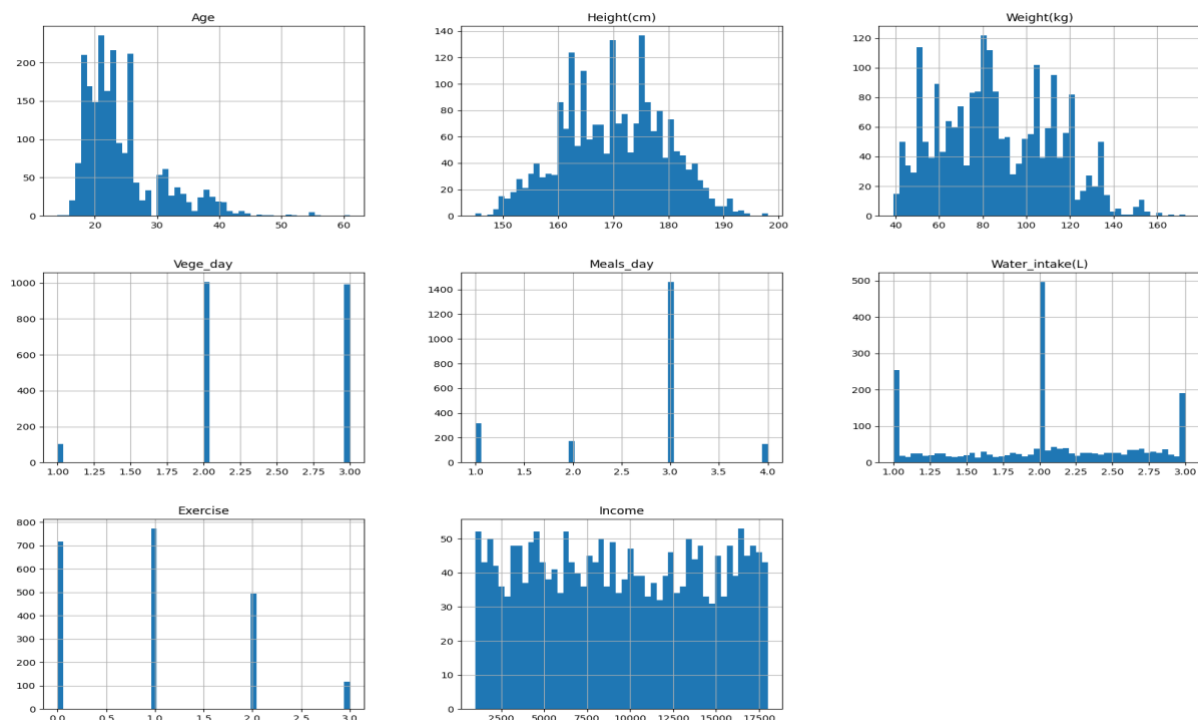
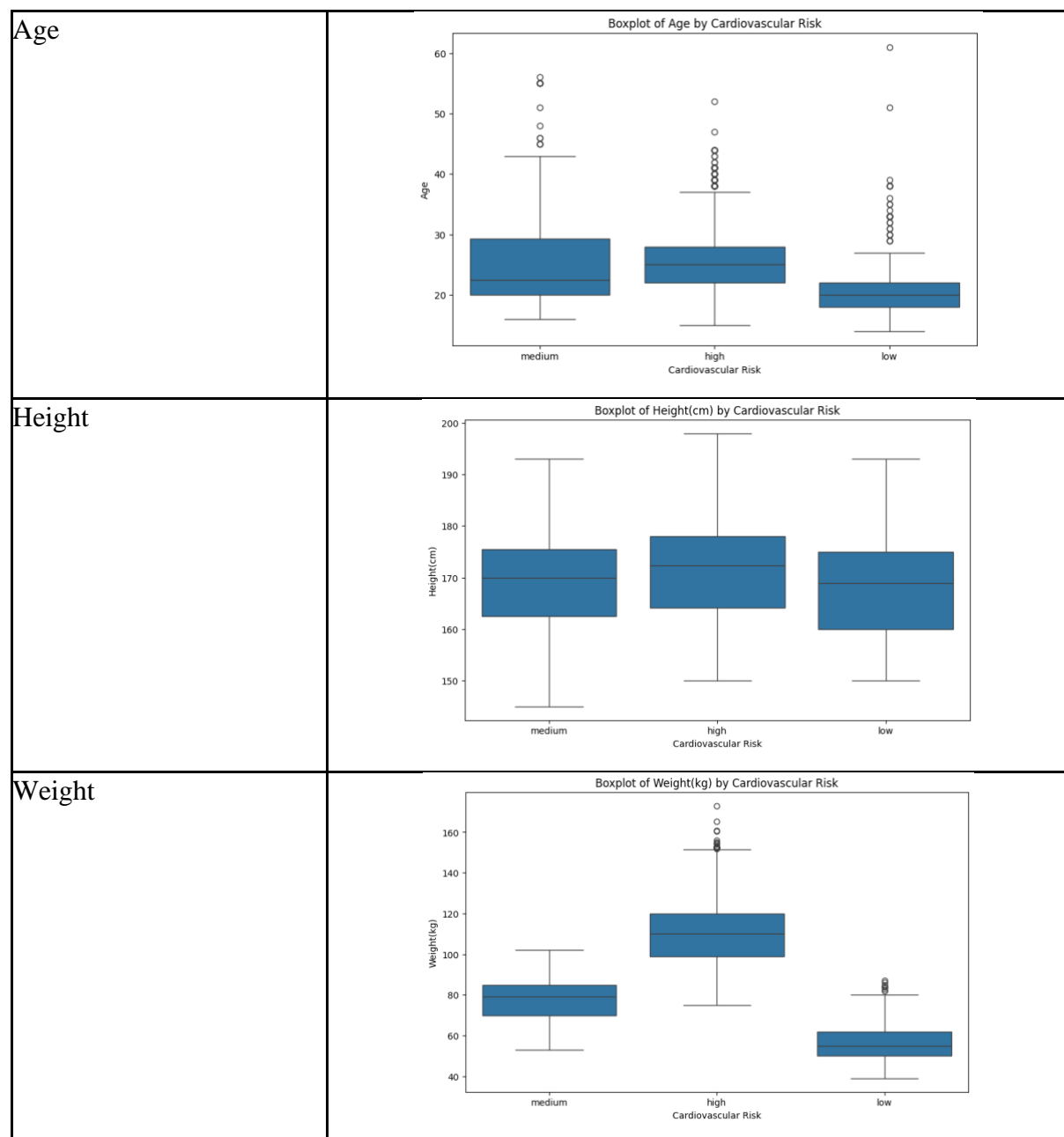


Figure 2.2.3 Histogram of numerical variables

We can observe from Figure 2.2.3 that both the 'Age' and 'Weight(kg)' distributions exhibit right-skewed heavy tails, indicating the presence of older individuals and higher weights in the dataset. Additionally, other issues include the presence of multimodal distributions in 'Weight(kg)' and the concentration of values in discrete categories for variables like 'Vege_day,' 'Meals_day,' and 'Exercise,' which suggests limited variability and potential challenges in analysis.

Boxplots for numerical variables by Output:



Vege_day	<p>Boxplot of Vege_day by Cardiovascular Risk</p>
Meals_day	<p>Boxplot of Meals_day by Cardiovascular Risk</p>
Water_intake	<p>Boxplot of Water_intake(L) by Cardiovascular Risk</p>
Exercise	<p>Boxplot of Exercise by Cardiovascular Risk</p>

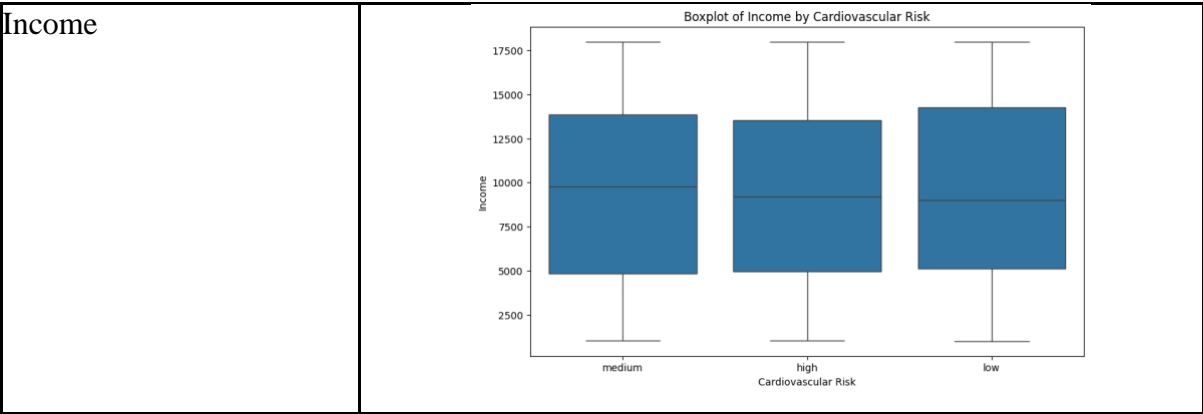
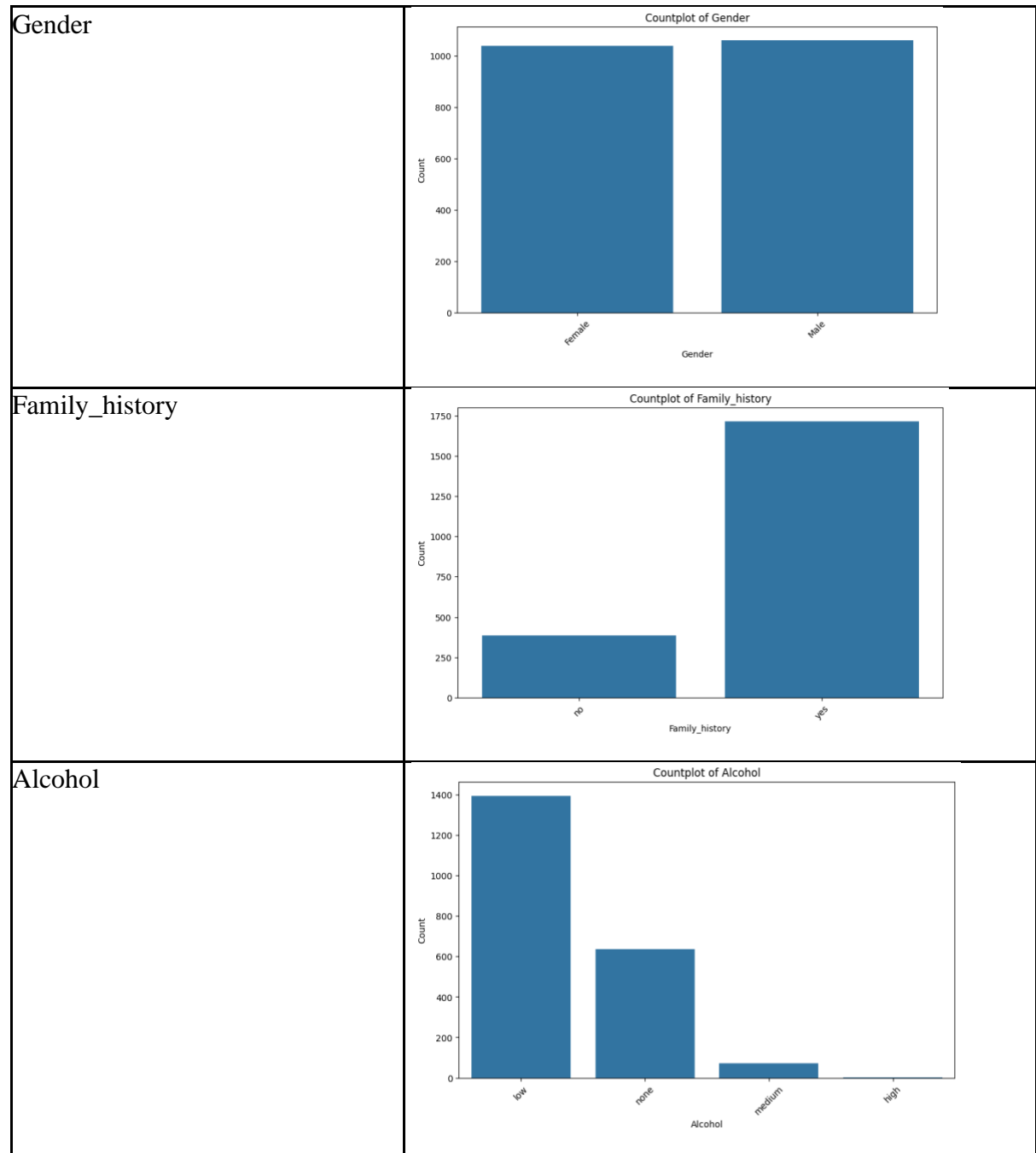


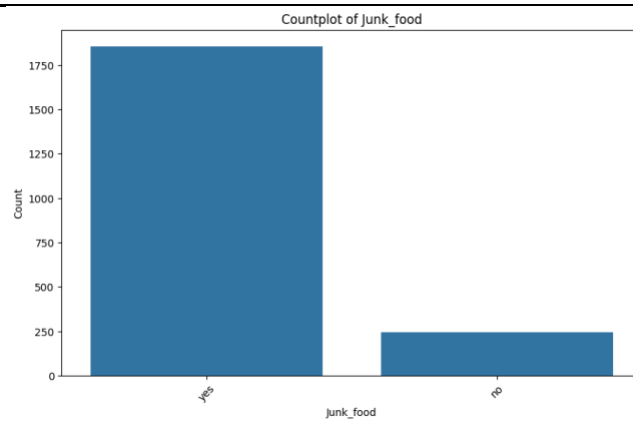
Table 2.2.1 Boxplot of Numerical Data

As shown in the table above, the Age boxplot indeed shows many outliers, especially for the "medium" and "low" cardiovascular risk categories. These outliers appear as individual dots above the upper whisker of each box, indicating ages that are significantly higher than the typical range for each risk group. The "high" risk category seems to have fewer outliers, but its box extends higher, suggesting older ages are more common in this group. The Weight boxplot also displays numerous outliers, particularly in the "high" cardiovascular risk category. These outliers are mostly above the upper whisker, indicating individuals with weights significantly higher than the typical range for their risk group. The "medium" risk category shows outliers on both ends, suggesting more weight variation in this group. Moreover, the Height boxplot does show some outliers, especially in the "medium" and "low" risk categories. These outliers appear both above and below the whiskers, indicating individuals who are either much taller or shorter than the typical range. The Exercise boxplot shows several outliers, particularly in the "medium" and "low" risk categories. These outliers are mostly above the upper whisker, suggesting some individuals exercise significantly more than is typical for their risk group. The boxplot for Meals day looks unusual as well because it shows a very limited range of values, which is between 1 and 4 meals per day.

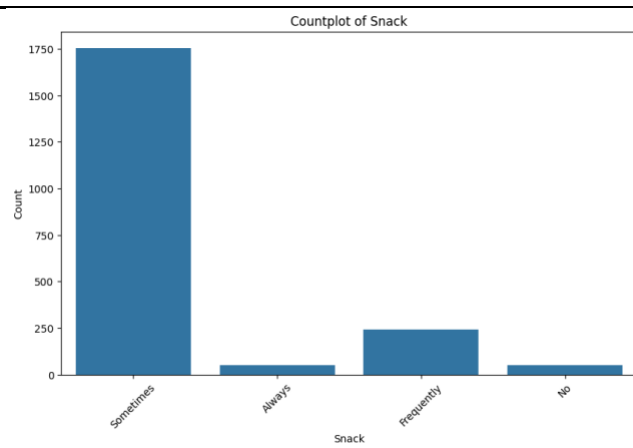
Countplot of categorical variables:



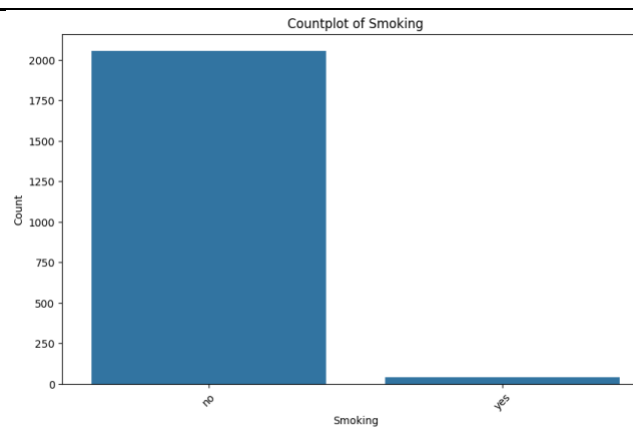
junk_food



Snack



Smoking



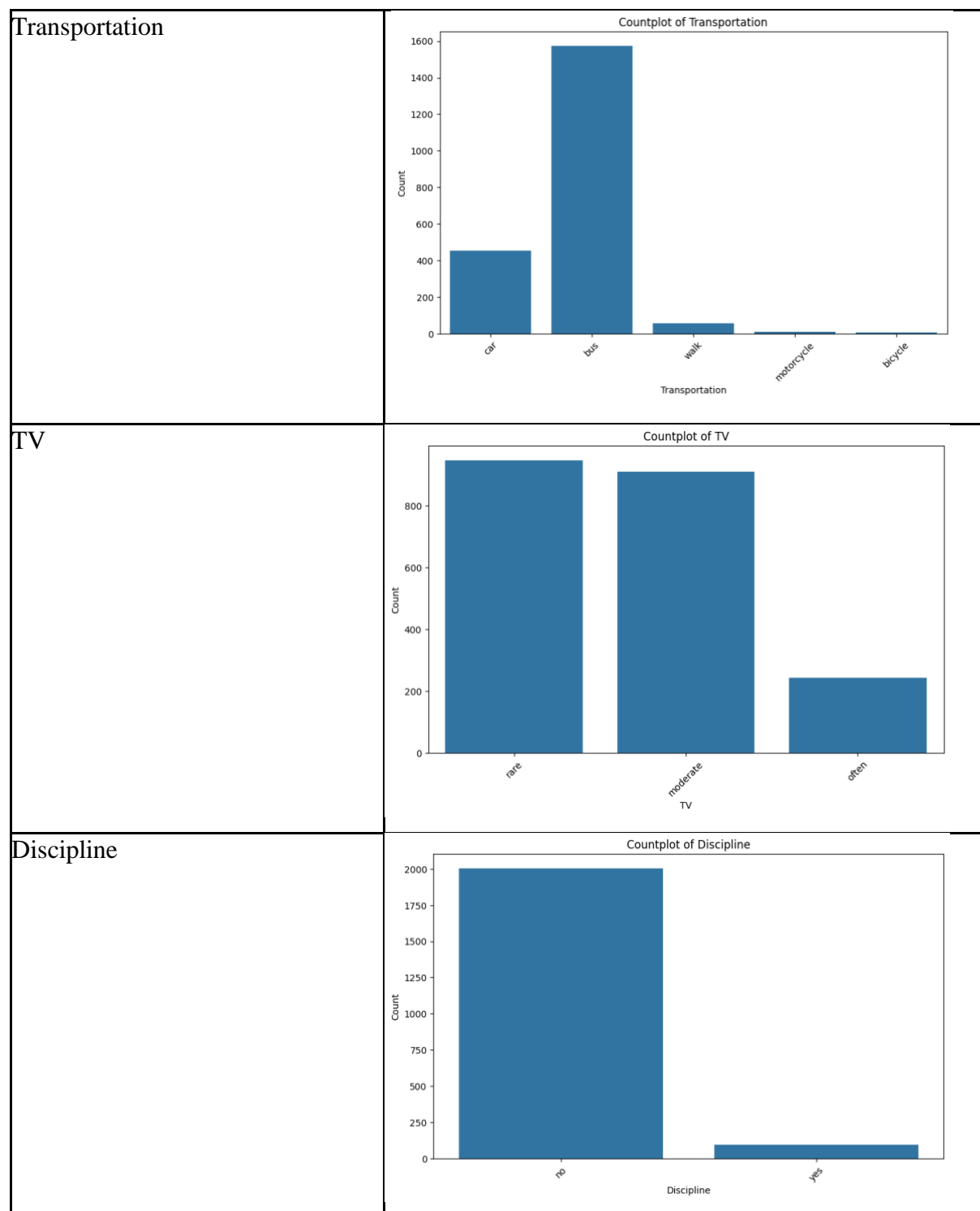


Table 2.2.2 Countplot for categorical data

2.3 Data Pre-processing

2.3.1 Handling Missing Values

```
# Check for missing values
print("Number of missing values for all columns:")
df.isnull().sum()
```

Figure 2.3.1 Code of Handling Missing Values

The initial step in the code involves checking for the presence of missing values in each column of the DataFrame. Missing data can significantly affect the performance of machine learning models. Checking for missing values is a critical step in data preprocessing because it helps identify potential data quality issues that need to be addressed before proceeding with any analysis or modeling. If the check reveals that there are missing values (null values) in the DataFrame, several strategies can be employed to address these nulls. For example, imputing missing values with the mean, median, or mode.

2.3.2 Dataset Preparation

```
# Separate output vector from the input matrix
X = df.drop('Cardiovascular_risk(y)', axis=1)
y = df['Cardiovascular_risk(y)']
```

Figure 2.3.2 Separation of Features and Target Variable

The dataset preparation involves separating the target variable, Cardiovascular_risk(y), from the features. This step is necessary to clearly define the input (features) and output (target) for the model. Separating the target variable ensures that the features used to train the model do not include the outcome, which could otherwise lead to data leakage and result in an overly optimistic model performance.

2.3.3 Splitting the Data

```
# Split the data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=30)
```

Figure 2.3.3 Data Splitting to Training set and Testing set

The data is split into training and test sets using an 80-20 split ratio. The random state is set to 69 for reproducibility. Splitting the data is crucial for validating the model's

performance on unseen data. An 80-20 split is a commonly accepted practice that balances the need for sufficient training data with the requirement of a reliable test set for evaluation.

2.3.4 Separating Numerical and Categorical Data

```
# Training Set
X_train_num = X_train.drop(['Gender', 'Family_history', 'Alcohol', 'Junk_food', 'Snack', 'Smoking', 'Transportation', 'TV', 'Discipline'], axis=1)

# Testing Set
X_test_num = X_test.drop(['Gender', 'Family_history', 'Alcohol', 'Junk_food', 'Snack', 'Smoking', 'Transportation', 'TV', 'Discipline'], axis=1)

# Training Set
X_train_cat_nominal = X_train[['Gender', 'Family_history', 'Junk_food', 'Smoking', 'Transportation', 'Discipline']]
X_train_cat_ordinal = X_train[['Alcohol', 'Snack', 'TV']]

# Testing Set
X_test_cat_nominal = X_test[['Gender', 'Family_history', 'Junk_food', 'Smoking', 'Transportation', 'Discipline']]
X_test_cat_ordinal = X_test[['Alcohol', 'Snack', 'TV']]
```

Figure 2.3.4 Code of Separating Numerical and Categorical Data

Numerical features are separated from categorical ones, with further distinctions made between nominal and ordinal categorical features. Numerical and categorical data require different preprocessing techniques. Numerical features typically need to be scaled, while categorical features need to be encoded. Further, distinguishing between nominal and ordinal features is essential because they represent different types of categorical data: nominal features have no intrinsic order, whereas ordinal features do. This distinction informs the choice of encoding methods.

2.3.5 Standardizing Numerical Data

```
# Training Set
scaler = StandardScaler(copy = False)
X_train_num_tr = scaler.fit_transform(X_train_num)

# Testing Set
X_test_num_tr = scaler.transform(X_test_num)
```

Figure 2.3.5 Coding of Standardizing Numerical Data

Numerical features are standardized using `StandardScaler`, which transforms the data to have a mean of 0 and a standard deviation of 1. Standardization is important because many machine learning algorithms (e.g., linear models, neural networks) perform better when the input features are on a similar scale. Without scaling, features with larger ranges could disproportionately influence the model, leading to suboptimal performance.

2.3.6 Encoding Nominal and Ordinal Data

```
ordinal_categories = [
    ['none', 'low', 'medium', 'high'], # Custom order for Alcohol
    ['No', 'Sometimes', 'Frequently', 'Always'], # Custom order for Snacks
    ['rare', 'moderate', 'often'] # Custom order for TV Watching Frequency
]

# Initialize the OrdinalEncoder with the defined categories
ordinal_encoder = OrdinalEncoder(categories=ordinal_categories)

# Fit and transform the training data for the categorical ordinal features
X_train_cat_ordinal_encoded = ordinal_encoder.fit_transform(X_train_cat_ordinal)
```

Figure 2.3.6.1 Coding of Ordinal Encoder Utilization

Ordinal features are encoded using OrdinalEncoder, which assigns an integer to each category based on its order. Ordinal encoding is used for features where the categories have a meaningful order. This encoding preserves the ordinal relationships between categories, which can be crucial for the model to understand the progression or intensity implied by the order.

```
# Training Set
onehot_encoder = OneHotEncoder(sparse_output=False)
X_train_cat_nominal_encoded = onehot_encoder.fit_transform(X_train_cat_nominal)

# Testing Set
X_test_cat_nominal_encoded = onehot_encoder.transform(X_test_cat_nominal)
```

Figure 2.3.6.2 Coding of One-hot Encoder

Nominal features are encoded using one-hot encoding, which converts each category into a binary vector. One-hot encoding is appropriate for nominal data because it creates a binary column for each category, ensuring that no unintended ordinal relationship is implied between categories. This prevents the model from incorrectly assuming any order among categories, which could lead to biased predictions.

2.3.7 Combining Transformed Data

```
# Training Set
X_train_cat_encoded = np.hstack([X_train_cat_nominal_encoded, X_train_cat_ordinal_encoded])
X_train_tr = np.hstack([X_train_num_tr, X_train_cat_encoded])
y_train = y_train.values

# Testing Set
X_test_cat_encoded = np.hstack([X_test_cat_nominal_encoded, X_test_cat_ordinal_encoded])
X_test_tr = np.hstack([X_test_num_tr, X_test_cat_encoded])
y_test = y_test.values
```

Figure 2.3.7 Code of Combining Transformed Data

The transformed nominal and ordinal categorical data are combined into a single array, which is then merged with the standardized numerical data to form the complete training set. Combining the transformed data ensures that all features are appropriately preprocessed and ready for input into the machine learning model. This integration is crucial for maintaining the integrity of the dataset, allowing the model to leverage both numerical and categorical information in its predictions.

2.3.8 Converting Target Variable

```
y_train = y_train.values
```

Figure 2.3.8 Code of Converting Target Variable

The target variable `y_train` is converted into a NumPy array. Converting the target variable into a NumPy array ensures compatibility with various machine learning libraries and algorithms that require the input data in this format. This step is necessary for seamless integration into the modelling pipeline.

2.4 Model Selection and Justification

Random Forest

The selection of a Random Forest Classifier for cardiovascular risk prediction is well-justified due to several key factors. First, its ability to handle non-linear relationships is essential, as suggested by the weak to moderate correlations observed in the dataset, indicating complex interactions between features. Random Forest's capacity to provide feature importance aligns with the need to identify the key predictors of cardiovascular risk among the various lifestyle factors. Additionally, the model's robustness to outliers is advantageous, especially for features such as Age and Weight, where outliers are likely present.

Another benefit is Random Forest's ability to process both categorical and numerical data without requiring extensive preprocessing, making it particularly suited for this mixed dataset. Its ensemble learning approach helps reduce overfitting, which is crucial for health-related predictions, where generalization to unseen data is paramount. Even though the dataset size is unknown, Random Forest is known to perform well on small to medium-sized datasets, which is common in healthcare studies. Its ability to handle missing data, although not directly evident in the visualizations, adds another layer of versatility.

While less interpretable than individual decision trees, Random Forest offers more transparency than many other machine learning models, which is important in healthcare settings where interpretability is valued. The model effectively balances bias and variance, making it well-suited to model the complex relationships between lifestyle factors and cardiovascular risk. Finally, Random Forest's independence from assumptions about feature relationships, such as the observed correlation between height and weight, makes it an appropriate and powerful choice for this task.

Support Vector Machine (SVM)

Support Vector Machines (SVMs) are often selected for predicting cardiovascular risk due to several key advantages that make them well-suited for this type of classification task. One of the primary reasons is their ability to handle high-dimensional data. Cardiovascular risk prediction typically involves multiple clinical, demographic, and biological factors, such as age, gender, lifestyle habits, and other biomarkers. SVM excels in high-dimensional spaces, making it ideal when dealing with a wide range of features. Additionally, SVM's use of kernel

functions enables it to handle non-linearly separable data, which is often the case with health data that has complex, intertwined relationships.

Another advantage of SVM is its robustness to overfitting. In medical data, overfitting can be a significant concern due to relatively small sample sizes or noisy data. SVM, particularly with a properly tuned regularization parameter (C), helps to avoid overfitting, ensuring the model captures true patterns in the data rather than noise. This robustness is crucial for making accurate cardiovascular risk predictions. Furthermore, SVM is well-suited to handle nonlinear relationships between risk factors and outcomes. The risk of cardiovascular events is often influenced by nonlinear interactions among multiple factors. SVM's kernel functions, such as the radial basis function (RBF), can capture these complex relationships, making it a valuable tool for accurate risk predictions.

Margin maximization is another strength of SVM. By finding a hyperplane that maximizes the margin between different classes (e.g., low risk and high risk), SVM enhances the model's generalization capabilities. This is important for medical applications, where predicting cardiovascular risk with a high degree of accuracy is critical for informed decision-making. In addition, SVM is effective at handling imbalanced data, a common issue in medical applications where high-risk cases are fewer compared to low-risk ones. SVM offers class-weighting techniques that assign higher importance to minority classes, ensuring that high-risk individuals are accurately detected and the model does not become biased toward the majority class.

SVM is also highly versatile and can be adapted to different risk models through the selection of appropriate kernel functions. This flexibility makes SVM a powerful tool across various risk prediction models. In conclusion, SVM is an excellent choice for cardiovascular risk prediction due to its ability to handle high-dimensional and nonlinear data, robustness against overfitting, and ability to manage imbalanced datasets.

KNN

The K-Nearest Neighbors (KNN) algorithm is a simple yet effective machine learning technique well-suited for tasks like cardiovascular risk prediction. It functions based on the similarity principle, which holds that similar data points are probably members of the same class. KNN can be used in this assignment to forecast an individual's cardiovascular risk based on their eating habits, demographic data, and other pertinent criteria.

KNN initially determines the distance between each training data point and a fresh, unknown data point before making predictions. The algorithm's performance can be greatly affected by the chosen distance metric. Minkowski, Manhattan, and Euclidean distances are examples of common distance measures. The K neighbors who are closest to the new spot after the distances have been determined are chosen. The new point is given the class label of most of these neighbors. For example, the algorithm would forecast a "high" cardiovascular risk for the new individual if most of the K's nearest neighbors have a high cardiovascular risk.

A key component of KNN is the selection of K or the number of neighbors taken into account. When K is too small, the algorithm may overfit and become overly sensitive to data noise. Underfitting occurs when the algorithm is overly insensitive to the underlying patterns in the data, and this can be caused by a high value of K. Methods such as cross-validation can be used to find the ideal value of K. Furthermore, KNN may exhibit sensitivity to the feature scale. Enhancing the performance of the algorithm can be achieved by standardizing or normalizing the features.

KNN is particularly well-suited for this task because it doesn't require explicit model training. Instead, it relies on the training data itself to make predictions. This method is adaptable and simple to use, particularly in situations where the underlying relationships between characteristics and the target variable are unclear or complex. KNN is appropriate for the variety of features in the dataset since it can handle both numerical and categorical data.

2.5 Model Training and Validation

RandomForest classifier

The Random Forest Classifier was initially set up with 42 trees (`n_estimators=42`) and a maximum depth of 10 (`max_depth=10`), using a fixed random seed (`random_state=69`) to ensure consistent results across runs. After training the model on the training dataset using the `fit` method, predictions were generated on the same dataset with the `predict` method. The model's performance was evaluated using key metrics such as accuracy, precision, recall, and F1 scores, with a macro average applied to ensure fair treatment of all classes, even in the presence of class imbalances. The model demonstrated high accuracy, indicating that the Random Forest classifier performed very well in predicting the outcomes on the training dataset.

To further assess the model's robustness, 3-fold cross-validation was conducted, with accuracy serving as the scoring metric. This helped to evaluate the model's performance across different subsets of the training data, and the results, including the average cross-validation score, were printed. Additionally, a confusion matrix was generated, offering a detailed breakdown of true positives, true negatives, false positives, and false negatives, giving a clearer view of how well the model performed across the different classification tasks.

Furthermore, cross-validation was performed using predicted probabilities with `'cross_val_predict'` and `'predict_proba'`, allowing for AUC (Area Under the Curve) curves to be plotted for each class. The AUC scores were computed to measure the model's effectiveness in distinguishing between classes. High AUC indicates strong discriminatory power, showing that the RandomForest is capable of distinguishing between different classes effectively.

The model was also tested on unseen data, where similar performance metrics were calculated, including accuracy, precision, recall, and F1 scores. A classification report was printed, and a confusion matrix was generated for the test set. This provides insights into the model's predictive ability when applied to new data. The model maintained its high accuracy on the test set, demonstrating strong predictive ability and generalization to new, unseen data. This consistency between the training and test sets highlights the model's robustness.

Support Vector Machine (SVM)

The Support Vector Machine (SVM) model was initialized using a linear kernel with a random state of 69 to classify the target variable based on the provided features. The model was wrapped with the `OneVsRestClassifier` for multi-class classification, ensuring it could handle multiple classes efficiently. During the training phase, the model was fitted to the

training data, allowing it to learn the decision boundaries between different classes. The model achieved a strong training accuracy, indicating its ability to classify a significant portion of the training instances correctly. This high accuracy reflects the model's capability to capture complex relationships in the data using the margin-maximization approach characteristic of SVM.

To comprehensively evaluate the model's performance, precision, recall, and F1 scores were computed using a macro average. This approach ensured that the evaluation was balanced across all classes, regardless of their frequency in the dataset. Additionally, the confusion matrix for the training set was generated, providing a detailed breakdown of correct and incorrect classifications. Cross-validation was conducted using three folds to validate the model's generalization ability further. The cross-validation scores offered a robust measure of the model's consistency across different data subsets, ensuring its stability when applied to new data. The average cross-validation score indicated that the model maintained solid performance across these different splits.

In addition to accuracy metrics, the SVM model's classification capabilities were further assessed using the Area Under the Curve (AUC) for each class. AUC scores were calculated based on the prediction probabilities from the cross-validation process, with the resulting curves providing a visual and quantitative measure of how well the model distinguished between different classes. The macro-average ROC AUC and Precision-Recall AUC scores highlighted the model's effectiveness, with higher values indicating strong performance across all classes.

When applied to the test set, the SVM model demonstrated excellent accuracy, confirming its ability to generalize effectively to unseen data. The model's precision, recall, and F1 scores for the test set were again computed using macro averaging, providing a balanced view of its performance across all classes. The confusion matrix for the test set offered further insights into the actual versus predicted labels, highlighting any remaining misclassifications. These evaluation metrics, combined with solid test accuracy, indicate that the SVM model is well-suited for this classification task, effectively handling both training and unseen test data with strong predictive performance.

KNN

The K-Nearest Neighbors (KNN) model was initialized with $k=5$ neighbors, which is a commonly used default setting for this classifier. During training, the model achieved strong

accuracy, demonstrating its ability to classify most instances correctly by leveraging the distances between data points. This high training accuracy reflects the model's proficiency in identifying similar patterns and grouping instances with comparable features into the correct classes.

To evaluate the KNN model's performance more thoroughly, precision, recall, and F1 scores were calculated using a macro average. A confusion matrix was also generated, providing a detailed view of correct and incorrect classifications and highlighting areas of both strong performance and misclassification. Cross-validation with three folds was employed to assess the model's generalization ability. This method provided a robust evaluation across multiple subsets of the training data, ensuring that the results were not biased by any particular split. The average cross-validation score indicated consistent performance across different data splits, reinforcing the model's reliability.

Further evaluation included prediction probabilities and multiclass ROC AUC and PR AUC curves. These curves, generated through cross-validation, offered both visual and quantitative measures of the model's ability to differentiate between classes. The macro-average ROC AUC and PR AUC scores provided a comprehensive view of the model's performance, with higher values indicating better classification ability across all classes.

When applied to the test set, the KNN model demonstrated solid accuracy, confirming its effectiveness in generalizing to unseen data. The confusion matrix for the test set provided a final check of actual versus predicted labels, revealing any remaining misclassifications and offering a detailed breakdown of true positives, true negatives, false positives, and false negatives. Overall, the KNN model performed well in both the training and testing phases, accurately classifying data points based on their proximity to known instances and proving to be a robust tool for classification tasks.

2.6 Model Tuning and Testing

2.6.1 Model Tuning

After completing training and validating using different models, the next move is to fine-tune the model to find out which model has the highest potential to obtain the highest accuracy. For this purpose, the Grid Search has been chosen to look for the best hyperparameters for each model. The table below shows the hyperparameters that are used to tune the model in different models respectively.

Model	Hyperparameters
Random Forest Classifier	'n_estimators': [50, 100, 200] 'max_depth': [None, 10, 20, 30] 'min_samples_split': [2, 5, 10] 'min_samples_leaf': [1, 2, 4]
Support Vector Machine (SVM)	C: [0.1, 1, 10, 100] kernel: ['linear', 'rbf'] gamma: ['scale', 'auto']
KNN	n_neighbors: [3, 5, 7, 9, 11], leaf_size: [10, 20, 30, 40, 50], algorithm: ['auto', 'ball_tree', 'kd_tree', 'brute'],

Table 2.6.1 Model Hyperparameters

Hyperparameters for Each Model:

- **Random Forest Classifier:**

- **n_estimators:** Number of trees in the forest. Tuning this helps control the model's complexity.
- **max_depth:** Maximum depth of each tree. Controls how complex the individual trees can be. Deeper trees can model more complex relationships but are prone to overfitting.
- **min_samples_split:** Minimum number of samples required to split an internal node. This helps prevent the tree from learning too much from noise.
- **min_samples_leaf:** Minimum number of samples required to be at a leaf node. This also helps in controlling overfitting by smoothing the model.

- **Support Vector Machine (SVM):**

- **C:** Regularization parameter that controls the trade-off between achieving a low error on the training data and minimizing the model complexity. A smaller C encourages a simpler model.
- **kernel:** Function used to transform the data into another dimension where it can be linearly separated. Common choices are:
 - linear: No transformation; a linear model is fitted.
 - rbf: Radial Basis Function, a non-linear transformation.
- **gamma:** Defines how far the influence of a single training example reaches. Low values mean 'far' and high values mean 'close'. This parameter is critical when using rbf kernel.

- **K-Nearest Neighbors (KNN):**

- **n_neighbors:** Number of neighbors to consider when making a classification. A lower number makes the model more sensitive to noise.
- **leaf_size:** Affects the speed of the algorithm and impacts the structure of the underlying tree used to store data.
- **algorithm:** Determines the algorithm used to compute the nearest neighbours:
 - auto: Automatically selects the most appropriate algorithm based on the input data.
 - ball_tree, kd_tree: Tree-based methods that store and search for neighbors.
 - brute: A straightforward search algorithm that considers every point.

2.6.2 Final Testing

Comparison of Scores Before and After Tuning

Classification Algorithm

Model	Before Tuning (Testing Set)	After Tuning (Testing Set)
Random Forest Classifier	Accuracy: 0.9548	Accuracy: 0.9714
	Precision: 0.9492	Precision: 0.9686
	Recall: 0.9485	Recall: 0.9667
	F1 score: 0.9483	F1 score: 0.9676
Support Vector Machine	Accuracy: 0.9548	Accuracy: 0.9667
	Precision: 0.9513	Precision: 0.9657
	Recall: 0.9447	Recall: 0.9600
	F1 score: 0.9473	F1 score: 0.9623
KNN	Accuracy: 0.8833	Accuracy: 0.8905
	Precision: 0.8809	Precision: 0.8874
	Recall: 0.8639	Recall: 0.8765
	F1 score: 0.8677	F1 score: 0.8785

Table 2.6.2: Score of classification algorithms before and after tuning

After tuning, the Random Forest Classifier showed the most significant improvements across all metrics—accuracy, precision, recall, and F1 score. The accuracy increased from 0.9548 to 0.9714, demonstrating that the model became better at correctly classifying instances overall. Precision and recall also saw substantial gains, improving from 0.9492 to 0.9686 and from 0.9485 to 0.9667, respectively. This indicates that after tuning, the model not only became more accurate but also more reliable in identifying true positives without a significant increase in false positives. The F1 score, which balances precision and recall, improved from 0.9483 to 0.9676, underscoring the model's overall effectiveness in handling the trade-off between precision and recall.

SVM also showed notable improvements across all metrics after tuning, though slightly less dramatic than those seen with the Random Forest. The accuracy increased from 0.9548 to 0.9667, reflecting a general improvement in the model's ability to classify instances correctly. Precision and recall both improved significantly, moving from 0.9513 to 0.9657 and from 0.9447 to 0.9600, respectively. These improvements suggest that SVM became more consistent in making accurate identifications and in correctly identifying actual positives. The F1 score, which improved from 0.9473 to 0.9623, highlights that SVM became more balanced in managing precision and recall, making it a reliable choice for this classification task.

While KNN did improve across all metrics after tuning, the improvements were more modest compared to Random Forest and SVM. The accuracy increased slightly from 0.8833 to 0.8905, indicating a minor improvement in overall classification performance. Precision and recall saw slight increases, moving from 0.8809 to 0.8874 and from 0.8639 to 0.8765, respectively. These metrics suggest that while KNN became somewhat better at identifying true positives and reducing false positives, the improvements were not as pronounced as with the other models. The F1 score, improving from 0.8677 to 0.8785, indicates that KNN, while more balanced after tuning, still lags in its ability to handle the precision-recall trade-off effectively.

AUC Score

Model	Before Tuning (Testing Set)	After Tuning (Testing Set)
Random Forest Classifier	Macro Average ROC AUC = 0.9925 Macro Average PR AUC = 0.9825	Macro Average ROC AUC = 0.9980 Macro Average PR AUC = 0.9951
Support Vector Machine (SVM)	Macro Average ROC AUC = 0.9822 Macro Average PR AUC = 0.9639	Macro Average ROC AUC = 0.9932 Macro Average PR AUC = 0.9884
KNN	Macro Average ROC AUC = 0.9139 Macro Average PR AUC = 0.8369	Macro Average ROC AUC = 0.9585 Macro Average PR AUC = 0.9318

Table 2.6.3: AUC score before and after tuning

Random Forest's AUC scores were the highest both before and after tuning, showing that it was the best at distinguishing between classes and handling the precision-recall trade-off. SVM also saw significant improvements in AUC scores, particularly in the PR AUC, indicating better performance in scenarios with class imbalance or where precision is critical. KNN's AUC scores improved the most in terms of percentage, especially PR AUC. Still, they remained lower overall compared to the other models, indicating that even with tuning, KNN lagged in overall discriminative ability.

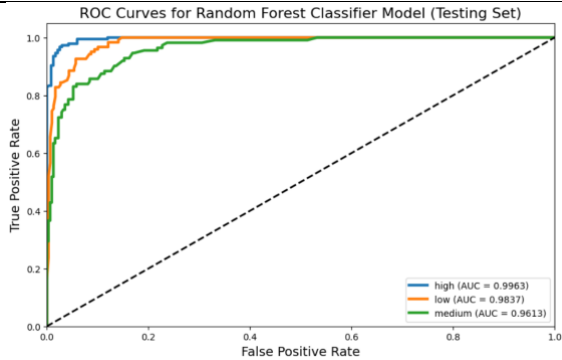
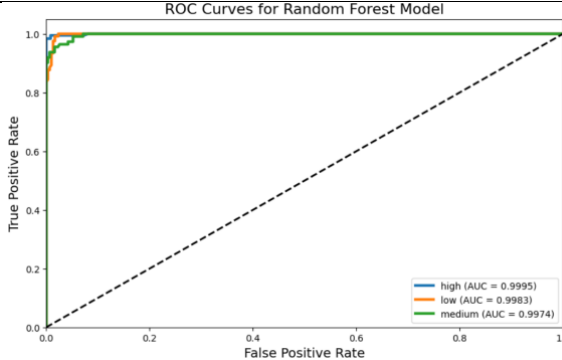
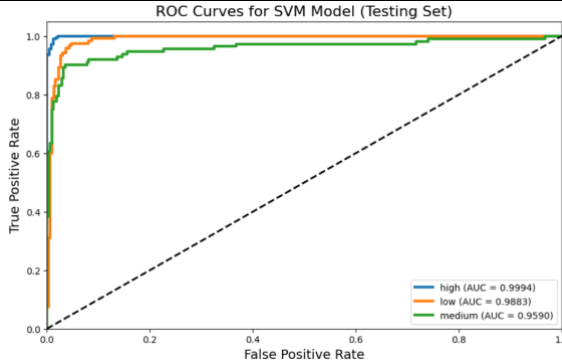
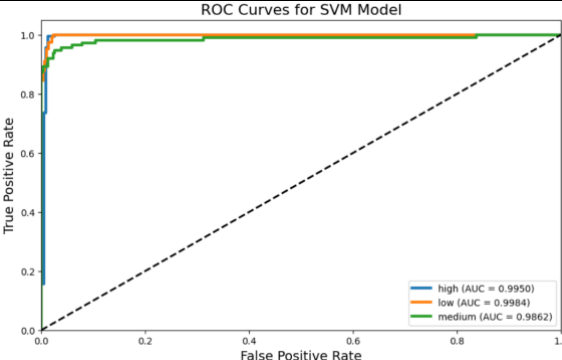
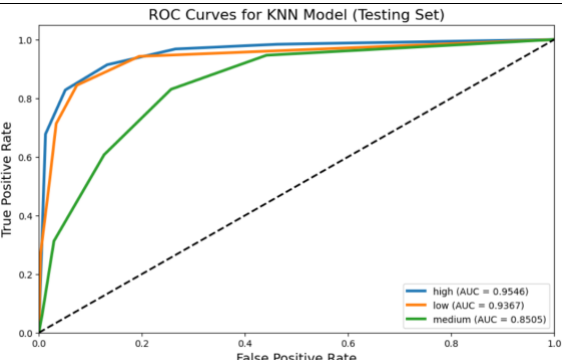
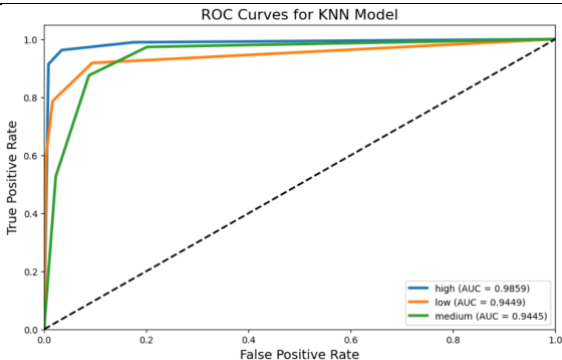
Model	ROC Curve (before tuning)	ROC Curve (after tuning)
Random Forest Classifier		
SVM		
KNN		

Table 2.6.4 ROC Curve (before and after tuning)

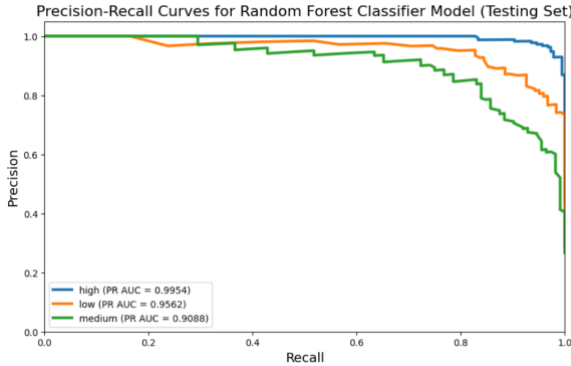
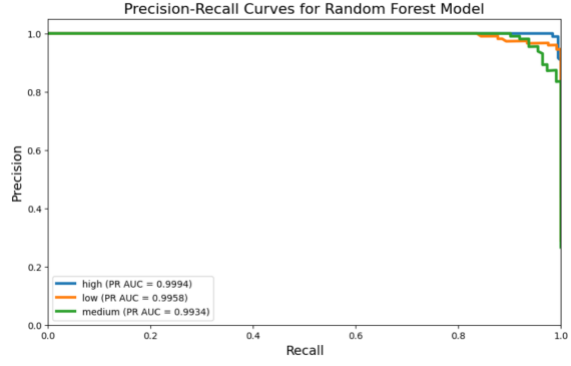
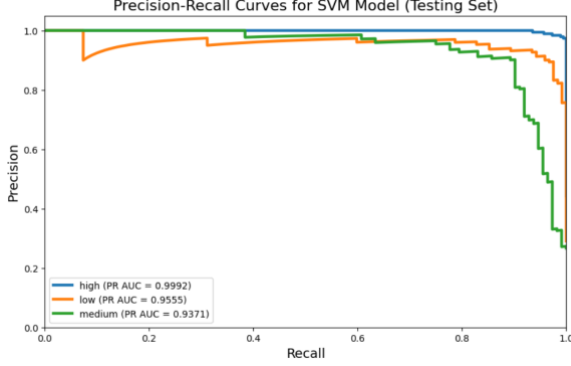
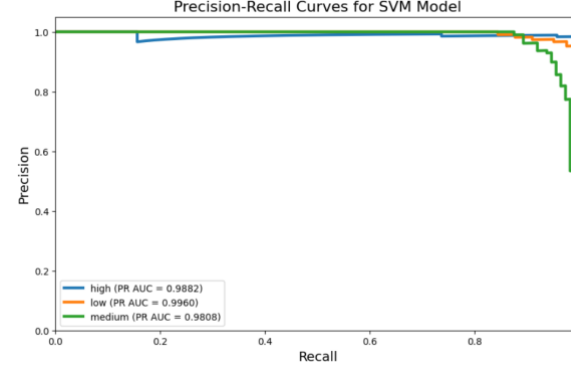
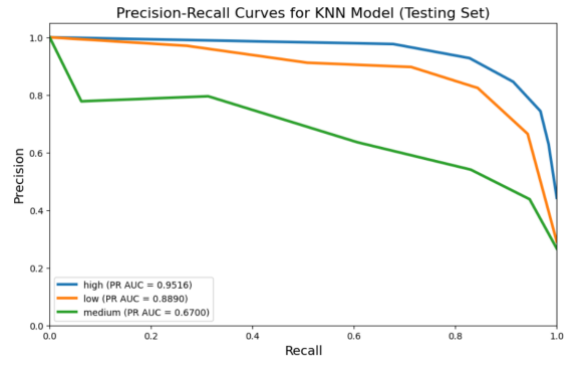
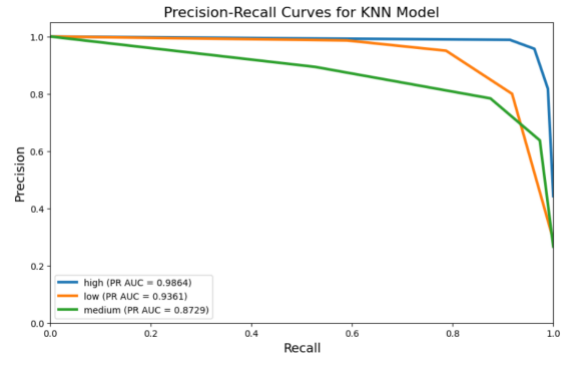
Model	PR Curve (before tuning)	PR Curve (after tuning)
Random Forest Classifier		
SVM		
KNN		

Table 2.6.5 PR Curve (before and after tuning)

1. Random Forest Classifier:

- ROC Curve:
 - Before Tuning: The ROC curve for Random Forest before tuning might already be close to the top-left corner, reflecting the model's inherent ability to separate classes well. The AUC (Area Under the Curve) should be relatively high.
 - After Tuning: After tuning, you would expect the ROC curve to move even closer to the top-left corner, indicating improved performance. The AUC should increase slightly, showing better discrimination between positive and negative classes.
- PR Curve:
 - Before Tuning: The PR curve before tuning might show a good balance between precision and recall, with the curve approaching the top-right corner of the plot. However, there might be some trade-offs, especially if the dataset is imbalanced.
 - After Tuning: After tuning, the PR curve should show improvement, with the curve moving closer to the top-right corner. This would indicate that the model is achieving higher precision without sacrificing recall or vice versa. The area under the PR curve should increase, suggesting better handling of positive predictions.

2. Support Vector Machine (SVM):

- ROC Curve:
 - Before Tuning: SVM's ROC curve before tuning might be slightly less optimal than that of Random Forest, depending on the kernel and regularization parameters used. The curve should still be relatively close to the top-left corner, but there might be room for improvement.
 - After Tuning: After tuning, the ROC curve for SVM should move closer to the top-left corner, reflecting an increase in the model's ability to distinguish between classes. The AUC should increase, indicating better overall performance.
- PR Curve:

- Before Tuning: The PR curve for SVM before tuning might show some variability, especially if the initial parameters are not well-suited to the data. The curve might not reach as close to the top-right corner as desired, particularly if precision or recall is lacking.
- After Tuning: Tuning should improve the PR curve, moving it closer to the top-right corner. This would indicate better precision-recall balance and the area under the PR curve should increase, showing that the model is better at making accurate positive predictions without increasing false positives.

3. K-Nearest Neighbors (KNN):

- ROC Curve:
 - Before Tuning: KNN's ROC curve before tuning might not be as close to the top-left corner as those of Random Forest or SVM. The curve might reflect a model that struggles to distinguish between classes effectively with a lower AUC.
 - After Tuning: After tuning, the ROC curve for KNN might improve, but the improvement may be less dramatic compared to the other models. The curve should move closer to the top-left corner, but the AUC might still be lower than that of Random Forest or SVM, indicating that KNN is less effective at class separation.
- PR Curve:
 - Before Tuning: The PR curve for KNN before tuning might show a significant trade-off between precision and recall, with the curve potentially falling short of the top-right corner. This would indicate that KNN might struggle with either false positives or false negatives, depending on the class distribution.
 - After Tuning: Tuning could help improve the PR curve, but like with the ROC curve, the improvement might be modest. The curve might move closer to the top-right corner, but the area under the PR curve could remain lower than that of Random Forest or SVM, reflecting KNN's challenges with maintaining precision and recall balance.

In summary, after tuning, both Random Forest and SVM typically show significant improvements in their ROC and PR curves, indicating better overall model performance. KNN, while improved, often remains less effective than the other two models, reflecting its limitations in certain classification tasks.

Confusion Matrix

Model	Before Tuning (Testing Set)	After Tuning (Testing Set)
Random Forest Classifier	$\begin{bmatrix} 184 & 1 & 1 \\ 0 & 111 & 11 \\ 3 & 3 & 106 \end{bmatrix}$	$\begin{bmatrix} 185 & 1 & 0 \\ 0 & 117 & 5 \\ 3 & 3 & 106 \end{bmatrix}$
SVM	$\begin{bmatrix} 186 & 0 & 0 \\ 0 & 117 & 5 \\ 5 & 9 & 98 \end{bmatrix}$	$\begin{bmatrix} 185 & 0 & 1 \\ 0 & 120 & 2 \\ 5 & 6 & 101 \end{bmatrix}$
KNN	$\begin{bmatrix} 184 & 1 & 1 \\ 4 & 92 & 26 \\ 13 & 4 & 95 \end{bmatrix}$	$\begin{bmatrix} 180 & 1 & 5 \\ 4 & 96 & 22 \\ 10 & 4 & 98 \end{bmatrix}$

Table 2.6.4 Confusion Matrix before and after tuning

After tuning, the confusion matrix of the random forest classifier showed a reduction in FP and FN values, indicating fewer misclassifications. The increase in TP and TN suggests that the model became better at correctly identifying both positive and negative instances. Next, the confusion matrix for the SVM model shows improvements, with a reduction in both FP and FN values. This means that the model made fewer errors after tuning, resulting in more accurate predictions. Finally, for KNN, although there was some reduction in FP and FN values after tuning, the improvement was modest. The confusion matrix still reflected more errors compared to the other models, with relatively higher instances of FP and FN.

In summary, both Random Forest and SVM emerged as strong performers after hyperparameter tuning, with Random Forest having a slight edge in overall performance. KNN, although improved, was less effective in handling the classification task. For tasks where balanced precision, recall, and F1 score are crucial, Random Forest and SVM are the preferred models, with Random Forest slightly leading in overall robustness. But, when considering the model's ability to consistently perform well across different datasets, minimize errors, and

provide reliable predictions, Random Forest stands out as the best model overall, making it the optimal choice for this classification task."

3.0 Results and Discussion

3.1 Summarize the training and testing results

Classification Algorithms (Training Set) (Before Tuning)	Accuracy	Precision	Recall	F1-score
Random Forest Classifier	0.9988	0.9986	0.9985	0.9985
SVM	0.9530	0.9505	0.9433	0.9454
KNN	0.9095	0.9086	0.8915	0.8983

Table 3.1.1 Classification Algorithm (Training Set)

The Random Forest Classifier achieved almost perfect metrics on the training set, with an accuracy of 0.9988, precision of 0.9986, recall of 0.9985, and an F1-score of 0.9985. This high performance indicates that the model was extremely effective in learning the patterns within the training data. Random Forests are composed of multiple decision trees, and by averaging the results of these trees, the model reduces variance and overfitting while capturing complex patterns in the data. The high performance on the training set suggests that the model was able to classify almost all instances it was trained on accurately.

SVM also performed well on the training set, with an accuracy of 0.9530, precision of 0.9505, recall of 0.9433, and an F1-score of 0.9454. SVM works by finding the optimal hyperplane that best separates the classes in the feature space. The relatively high metrics indicate that SVM effectively learned the structure of the training data, although it didn't capture the patterns as thoroughly as the Random Forest model. This could be due to the nature of SVM, which might not fit the training data as tightly as Random Forest, especially if the data is not perfectly linearly separable.

KNN showed the lowest performance on the training set, with an accuracy of 0.9095, precision of 0.9086, recall of 0.8915, and an F1-score of 0.8983. KNN classifies a data point based on the majority class among its nearest neighbors. The lower performance on the training set suggests that KNN may have struggled with the complexity of the data or with distinguishing between classes when data points from different classes were close to each other. KNN's reliance on distance metrics can make it sensitive to noisy data and irrelevant features, which could explain the lower metrics compared to the other models.

Classification Algorithms (Testing Set) (Before Tuning)	Accuracy	Precision	Recall	F1-score
Random Forest Classifier	0.9548	0.9492	0.9485	0.9483
SVM	0.9548	0.9492	0.9485	0.9483
KNN	0.8833	0.8809	0.86	0.8677

Table 3.1.2 Classification Algorithm (Testing Set)

On the test set, the Random Forest Classifier maintained a high level of performance, with an accuracy of 0.9548, precision of 0.9492, recall of 0.9485, and an F1-score of 0.9483. The slight drop in performance compared to the training set is typical and suggests that the Random Forest model generalizes well to new, unseen data. The model's ability to maintain high metrics on the test set indicates that it did not overfit the training data and can effectively handle real-world scenarios.

SVM also performed strongly on the test set, with metrics very similar to those of Random Forest—accuracy of 0.9548, precision of 0.9513, recall of 0.9447, and an F1-score of 0.9483. The SVM's performance on the test set, closely matching its training set results, indicates that it has a good balance between fitting the training data and generalizing to new data. The model's ability to maintain high precision and recall on the test set suggests that it effectively manages the trade-off between sensitivity and specificity.

KNN exhibited the lowest performance on the test set, with an accuracy of 0.8833, precision of 0.8809, recall of 0.8600, and an F1-score of 0.8677. The more significant drop in performance for KNN from the training set to the test set indicates that the model may have overfitted the training data. KNN's reliance on the nearest neighbors makes it sensitive to outliers and variations in the data that are not representative of the overall population. The lower test set metrics suggest that KNN struggled more with generalization, making it less reliable for predicting new data compared to Random Forest and SVM.

3.2 In-depth analysis of the prediction performance and errors of the models

3.2.1 SVM Model

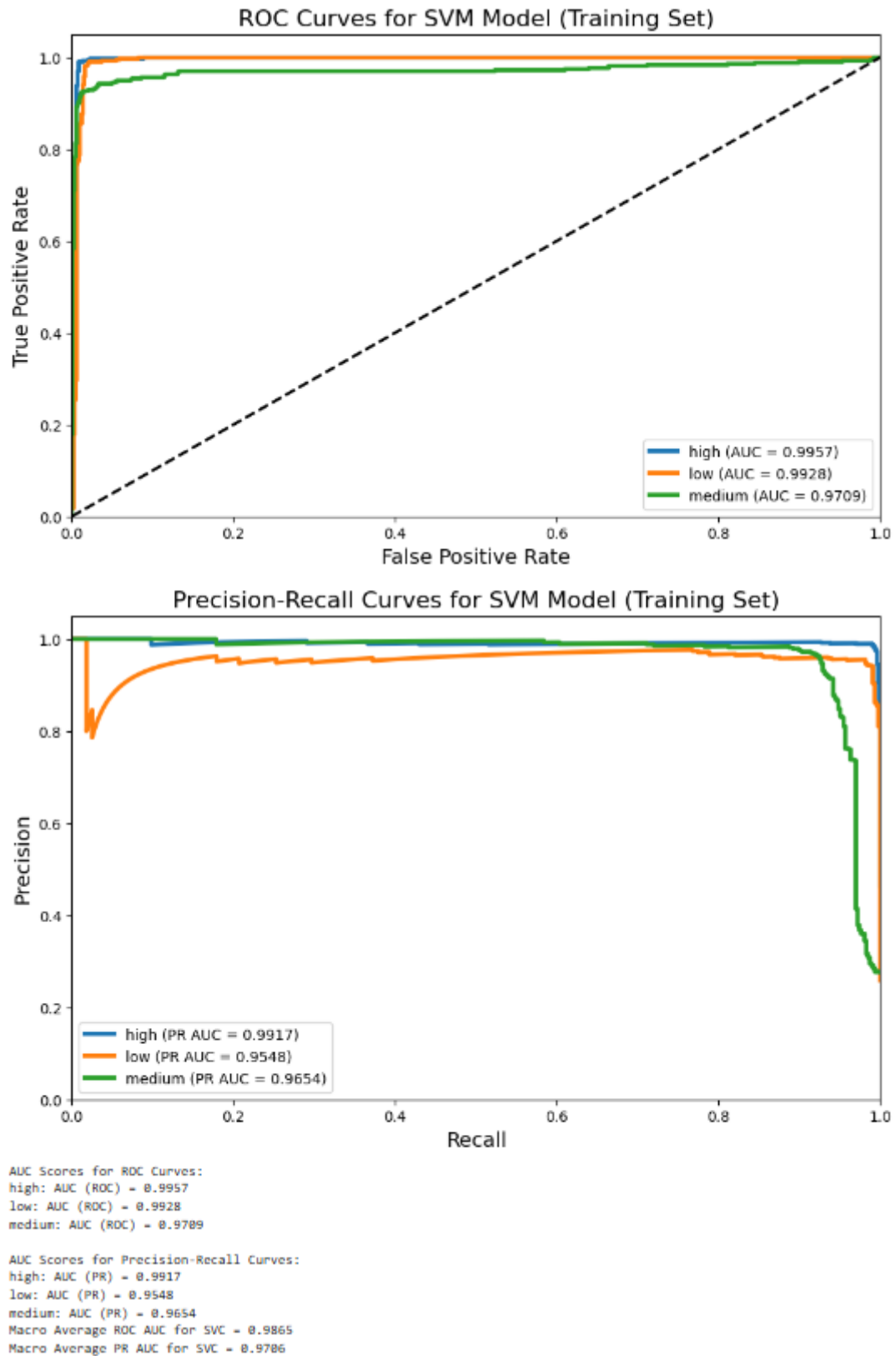


Figure 3.2.1.1 ROC and Precision-Recall curves for SVM Model (Training Set)

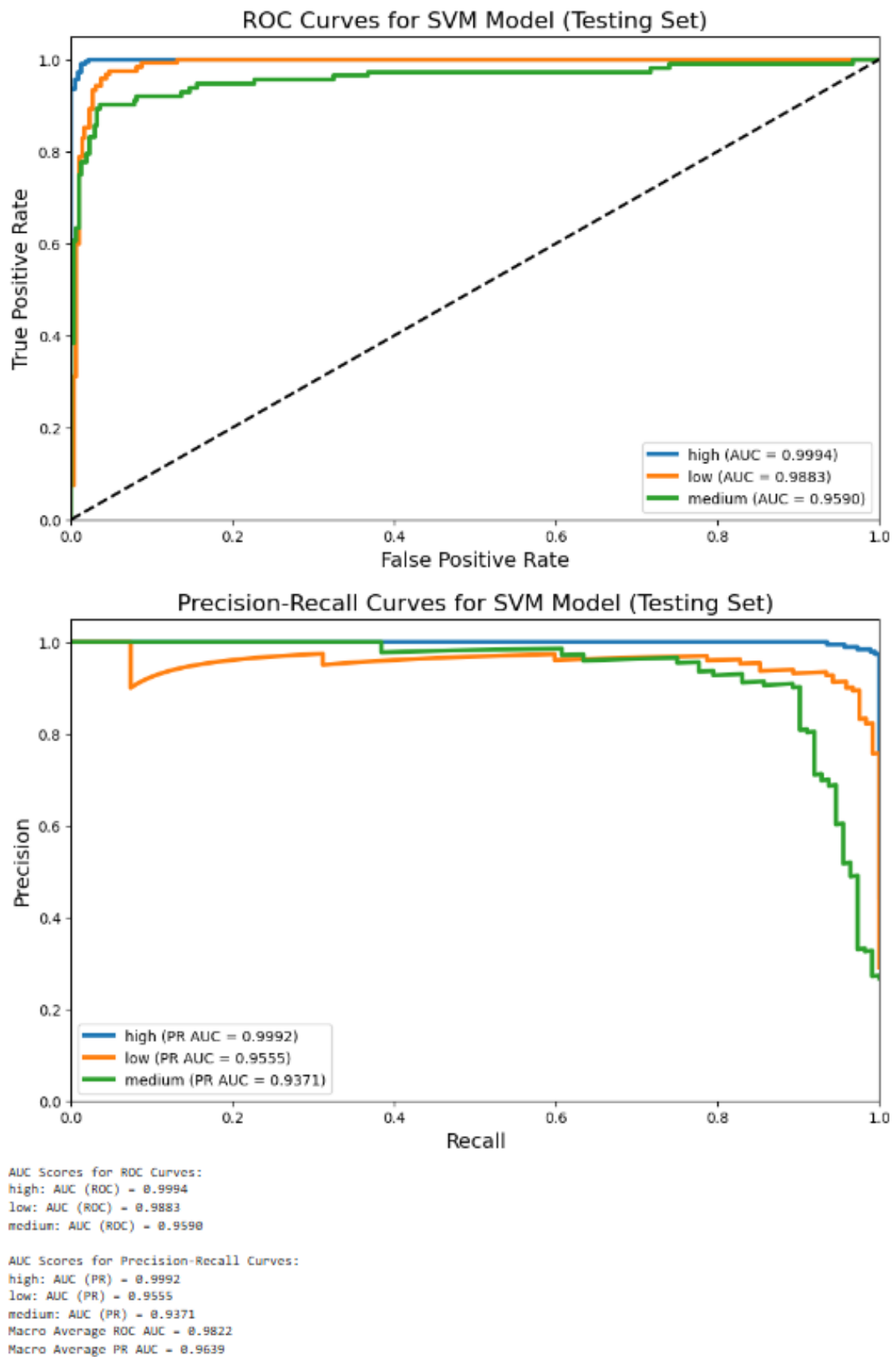


Figure 3.2.1.2 ROC and Precision-Recall curves for SVM Model (Testing Set)

Based on Figure 3.2.1 and Figure 3.2.2, the SVM model demonstrated strong performance on both the training and test sets, with accuracy values of 0.9530 and 0.9548, respectively. The macro-average precision, recall, and F1 scores remained consistent across both sets, with only slight variations—indicating that the model is robust and capable of accurately classifying data. The slight drop in cross-validation scores, averaging 0.9274, compared to the training accuracy, suggests some variability when applied to different subsets of the data. This could indicate that while the model generalizes well, there is still potential for overfitting, particularly in edge cases or specific class distributions.

In terms of error analysis, the confusion matrices reveal where the model struggles. In the training set, class 3 had notable misclassifications, with 38 instances incorrectly predicted as class 2. A similar trend is observed in the test set, where 9 instances of class 3 were misclassified as class 2. This pattern suggests that the model may struggle to differentiate between these two classes, possibly due to overlapping feature distributions. Addressing this issue could involve further fine-tuning the model, such as adjusting the kernel function or using additional feature engineering to separate these classes better.

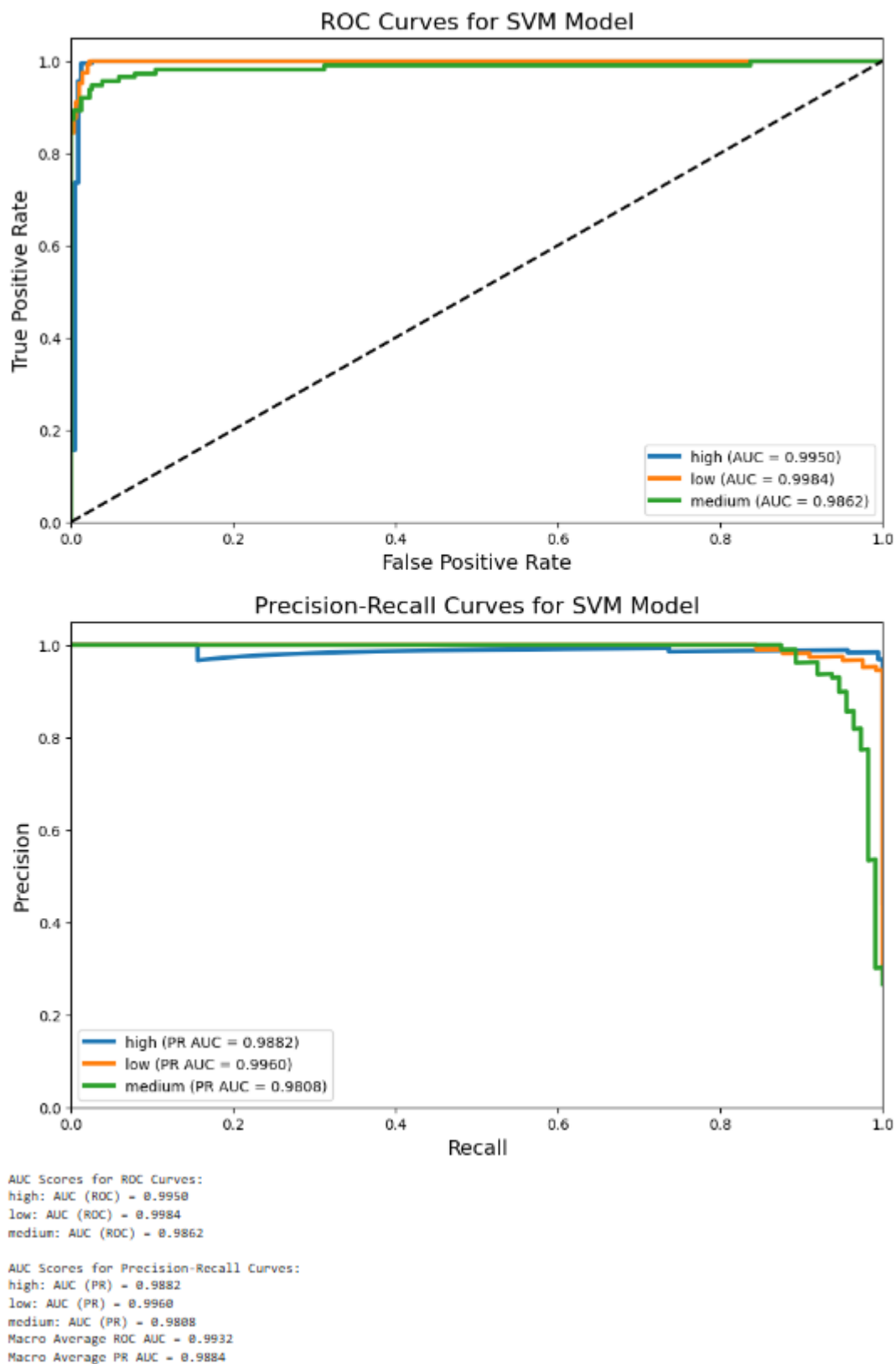


Figure 3.2.1.3 ROC and Precision-Recall curves for SVM Model After Fine-Tuning

Figure 3.2.3 shows the ROC and Precision-Recall curves for the SVM Model after fine-tuning using GridSearchCV. It defines a grid of hyperparameter values, including

regularization parameter (C), kernel type, and gamma for the RBF kernel. GridSearchCV then iterates through all combinations of these hyperparameters, trains an SVM model for each combination using cross-validation, and evaluates its performance based on accuracy. The best hyperparameter combination and corresponding cross-validation score are printed, guiding the selection of the optimal SVM model for the given dataset.

3.2.2 Random Forest Model

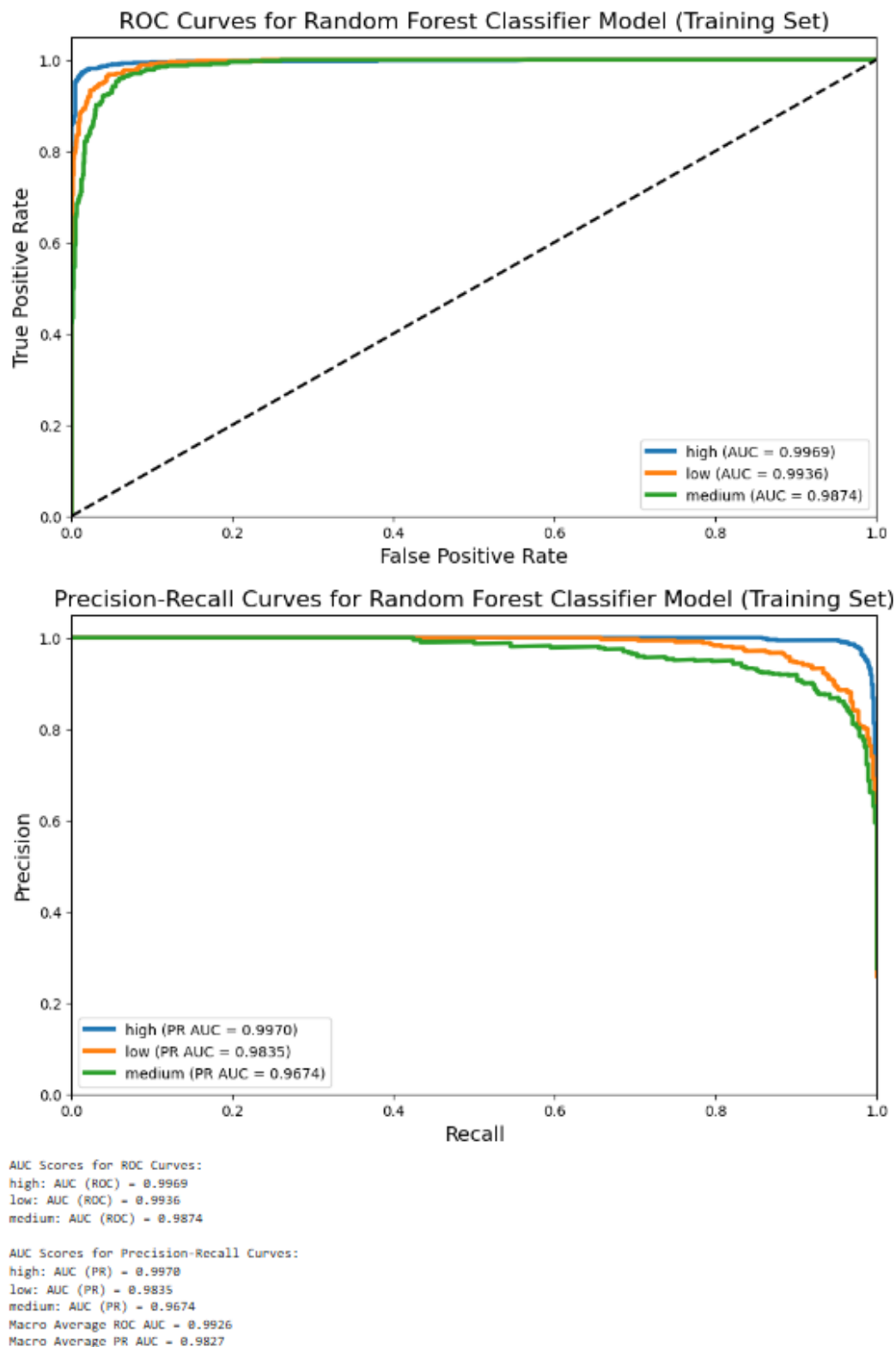


Figure 3.2.2.1 ROC and Precision-Recall curves for Random Forest Model (Training Set)

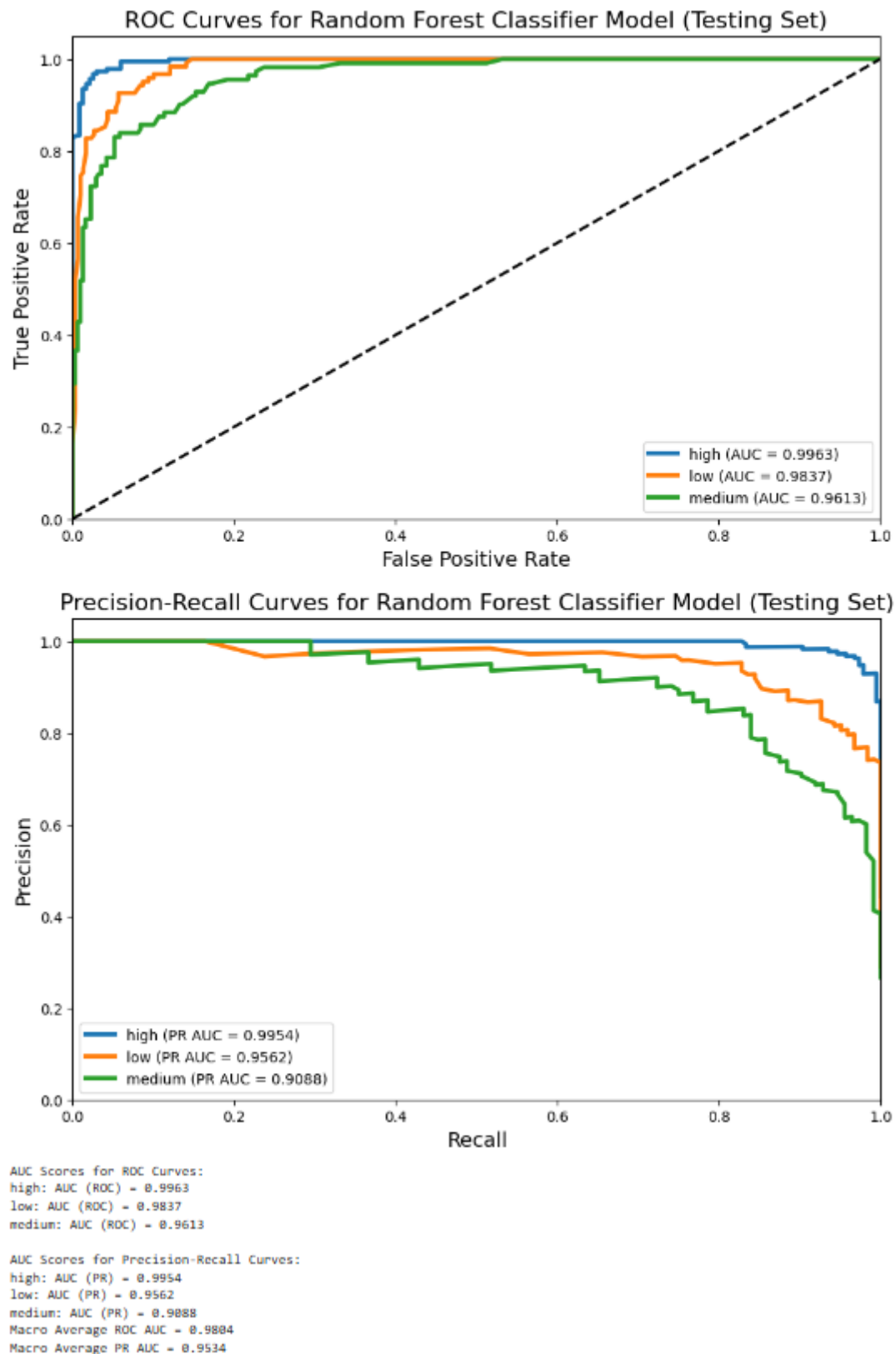


Figure 3.2.2.2 ROC and Precision-Recall curves for Random Forest Model (Testing Set)

The Random Forest model exhibited exceptional performance on the training set, achieving perfect scores across all metrics, including accuracy, precision, recall, and F1 score,

all at 1.0000. The confusion matrix for the training set shows no misclassifications, with all instances correctly classified into their respective classes. This indicates that the model has effectively learned the patterns in the training data. However, such perfect performance on the training set suggests a possible risk of overfitting, where the model might have learned the training data too well and may struggle to generalize as effectively to new, unseen data.

When applied to the test set, the model still performed very well, with an accuracy of 0.9667 and macro-average precision, recall, and F1 scores all around 0.9616. However, the confusion matrix for the test set reveals some misclassifications, particularly within classes 2 and 3. Class 2 had 8 instances incorrectly classified as class 3, and class 3 had 3 instances misclassified as class 2. These errors suggest that the model may struggle to differentiate between these classes, likely due to overlapping feature spaces or similarities in the data points. While the performance remains strong, further fine-tuning or additional feature engineering might be necessary to improve the model's ability to distinguish between these specific classes.

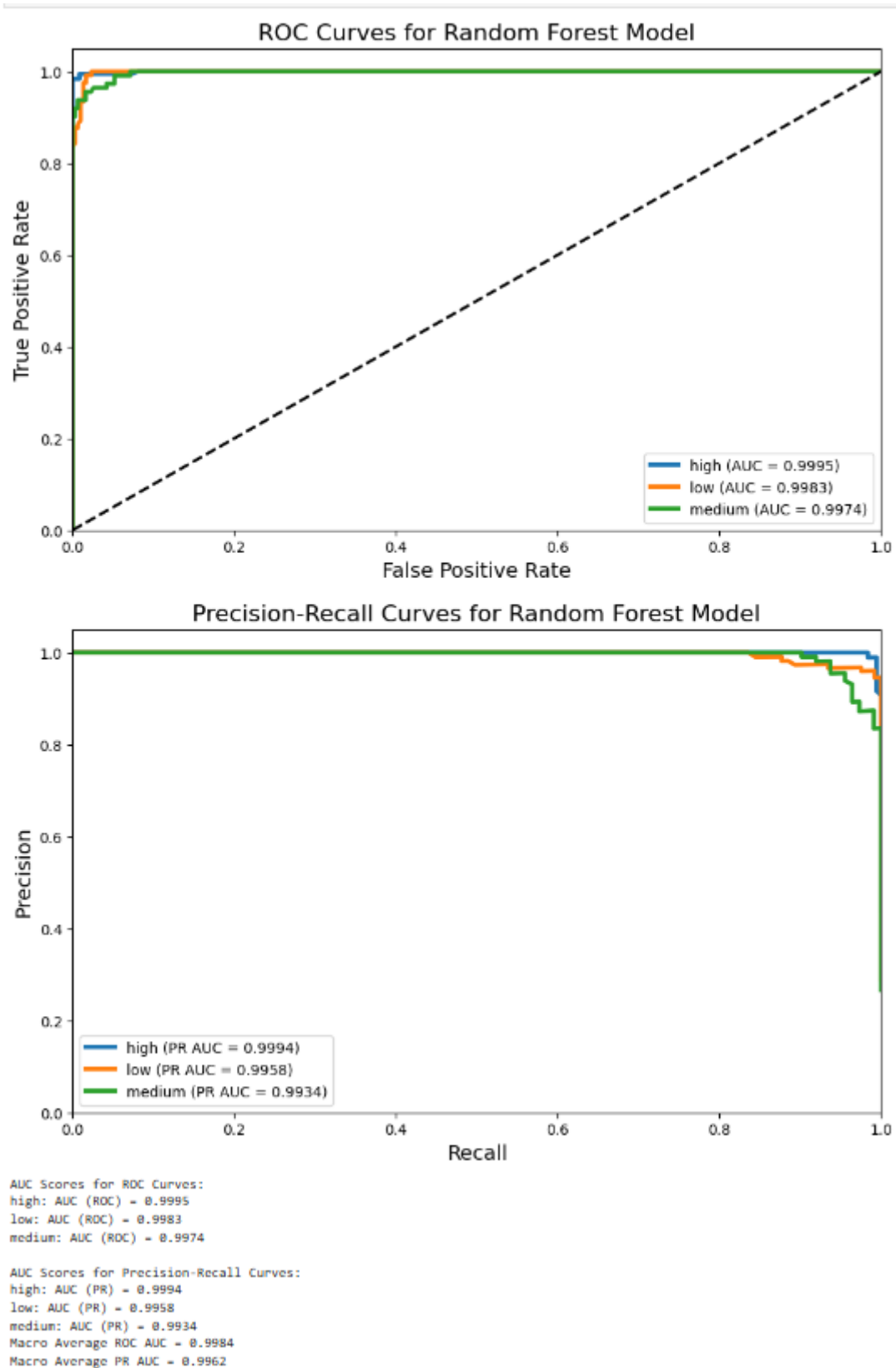


Figure 3.2.2.3 ROC and Precision-Recall curves for Random Forest Model After Fine-Tuning

Figure 3.2.2.3 shows the ROC and Precision-Recall curves for the SVM Model after fine-tuning using GridSearchCV. It defines a grid of hyperparameter values, including the number of trees, maximum tree depth, minimum samples required for splitting a node, and minimum samples required at leaf nodes. GridSearchCV then iterates through all possible combinations of these hyperparameters, trains a Random Forest model for each combination using cross-validation, and evaluates its performance based on accuracy. The best hyperparameter combination and corresponding cross-validation score are printed, guiding the selection of the optimal Random Forest model for the given dataset.

3.2.3 K-Nearest Neighbors Model

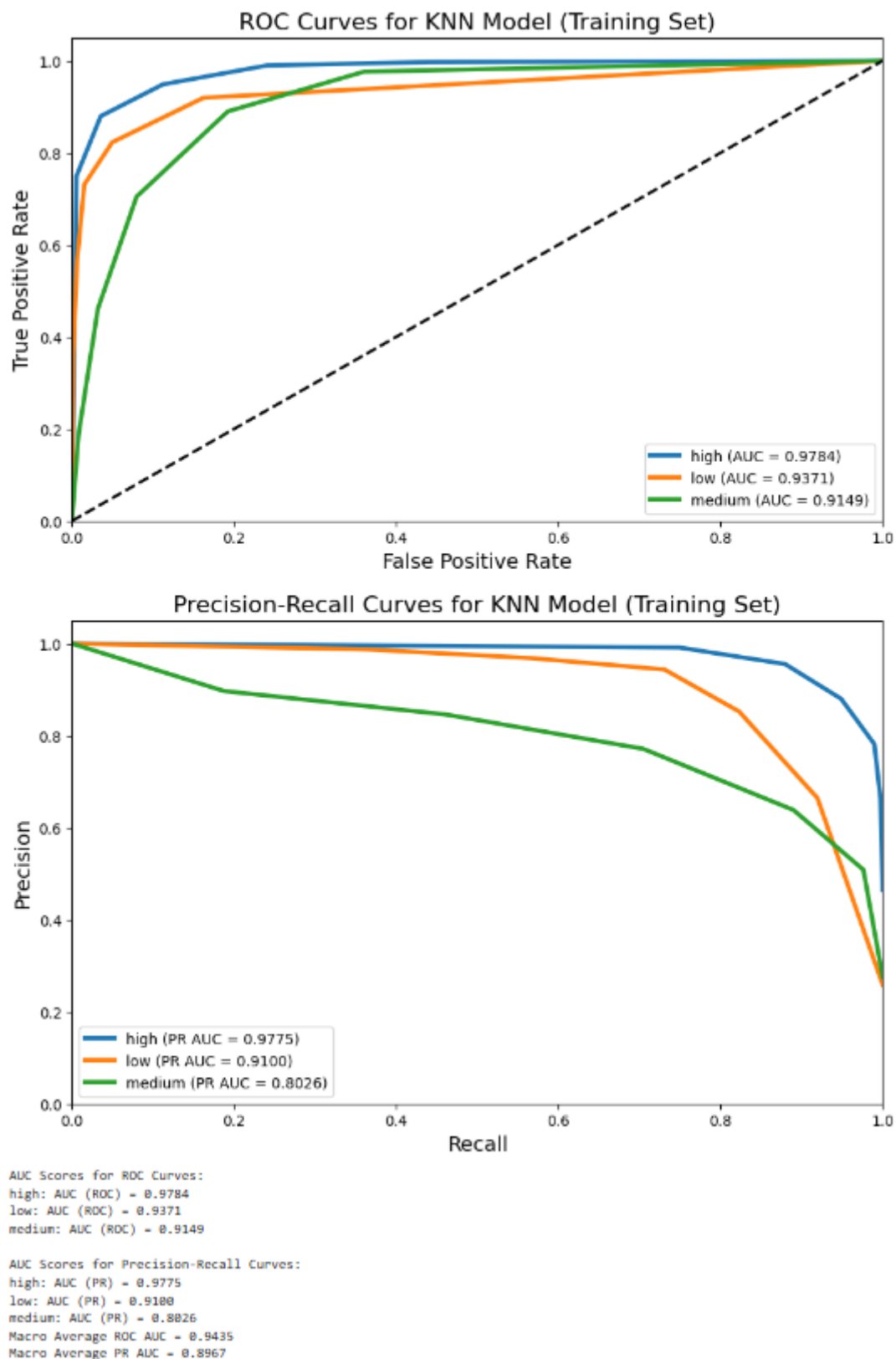


Figure 3.2.3.1 ROC and Precision-Recall curves for K-NN Model (Training Set)

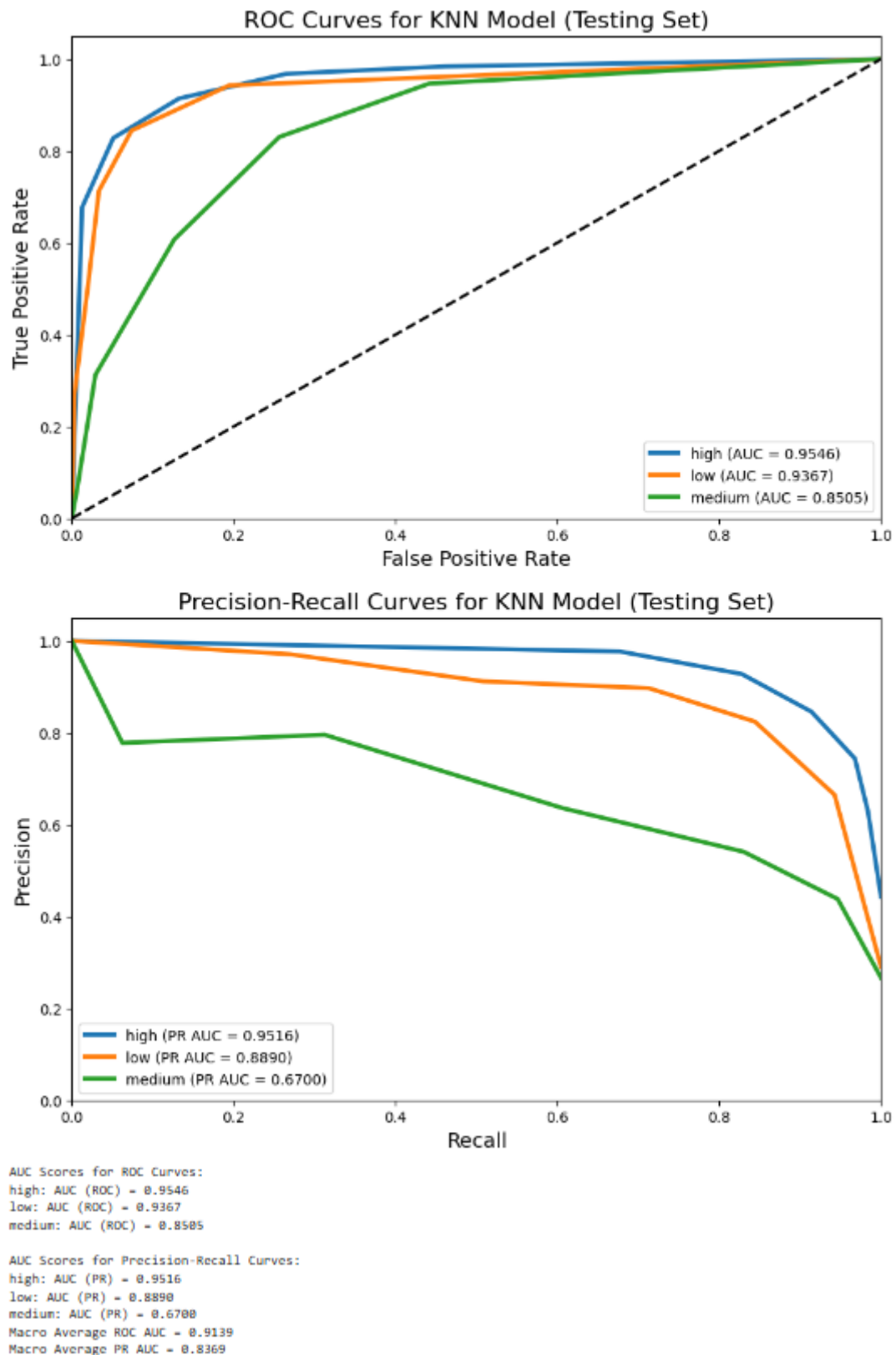


Figure 3.2.3.2 ROC and Precision-Recall curves for K-NN Model (Testing Set)

Based on Figure 3.2.3.1 and Figure 3.2.3.2, The K-Nearest Neighbors (KNN) model demonstrated solid performance on the training set with an accuracy of 0.9262 and macro-

average precision, recall, and F1 scores all above 0.91. The confusion matrix for the training set shows that most instances were classified correctly, but there are some misclassifications, particularly in classes 2 and 3. For example, 48 instances of class 2 were incorrectly classified as class 3, and 27 instances of class 1 were misclassified as class 3. These errors indicate that the model struggles with distinguishing between certain classes, which may be due to overlapping features or similarities between data points in these classes.

On the test set, the model's accuracy dropped to 0.8905, with macro-average precision, recall, and F1 scores also decreasing slightly compared to the training set. The confusion matrix for the test set reveals similar patterns of misclassification, especially between classes 2 and 3. Class 2 had 22 instances incorrectly classified as class 3, and class 3 had 10 instances misclassified as class 1. These errors suggest that the model's ability to generalize to new data is somewhat limited, and further improvements could be made. Possible approaches include adjusting the value of k in the KNN algorithm, incorporating feature scaling, or using more sophisticated techniques like feature selection to enhance the model's ability to differentiate between similar classes.

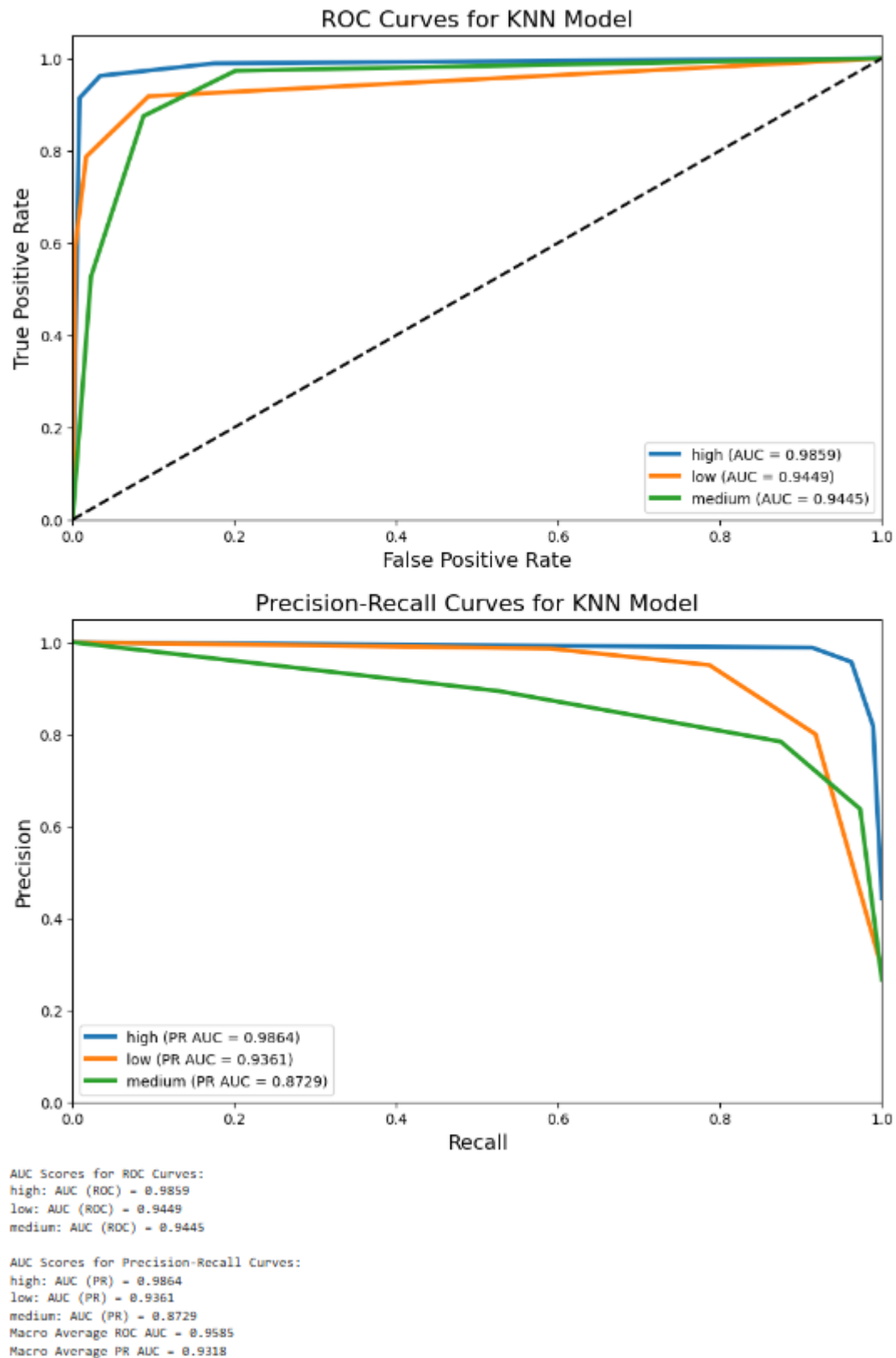


Figure 3.2.3.3 ROC and Precision-Recall curves for K-NN Model After Fine-Tuning

The fine-tuning for K-NN calculates prediction probabilities for the test data using the best KNN model obtained from GridSearchCV and then evaluates the model's performance using ROC AUC and PR AUC metrics.

The `predict_proba()` method is used to generate probability estimates for each class for the test data. These probabilities represent the model's confidence in each class prediction. Next, the `plot_multiclass_curves()` function is called, which is assumed to be defined elsewhere in your code. This function should return a dictionary containing the ROC AUC scores and PR AUC scores for each class. The `roc_auc_scores` and `pr_auc_scores` are then extracted from this dictionary. Finally, the macro average for both ROC AUC and PR AUC is calculated by taking the mean of the scores for all classes. This provides an overall measure of the model's performance across all classes.

3.3 Strengths and Weaknesses

	Strengths	Weaknesses
SVM	<ul style="list-style-type: none"> • Effective in high-dimensional datasets. • Works well for Non-Linear Data. • Robust to Overfitting. • Margin Maximization leading to good generalization on unseen data. • Effective for binary classification 	<ul style="list-style-type: none"> • Not ideal for large datasets • Requires careful parameter tuning. • Memory-Intensive • Less interpretable compared to models like decision trees. • Struggles with imbalanced data.
Random Forest	<ul style="list-style-type: none"> • Handles both Classification and Regression Tasks • Able to handles missing data • Reduces Overfitting by averaging the predictions of multiple trees. • Feature Importance • Works Well with Non-Linear Data 	<ul style="list-style-type: none"> • Requires more computational resources. • Can become slow with large datasets. • Harder to interpret compared to decision trees. • Less effective with sparse data.

	<ul style="list-style-type: none"> • Resistant to outliers and noisy data 	<ul style="list-style-type: none"> • Potential overfitting if number of trees and depth not chosen properly
K-NN	<ul style="list-style-type: none"> • Easy to understand and implement. • No assumptions about data • Effective for small datasets • Handles multi-class classification • Able to capture complex patterns 	<ul style="list-style-type: none"> • Sensitive to noisy data and outliers. • Inefficient for large datasets • Performance depends on scaling (Need normalization or standardization in preprocessing) • Ineffective in high-dimensional data.

Table 3.3 Strengths and Weaknesses of the 3 models

3.4 Investigate features importance for the prediction

Random Forest Features

Feature importance in a Random Forest model is crucial for understanding which variables significantly influence the model's predictions. Random Forests are powerful ensemble learning methods, but they can be complex and less interpretable due to the multitude of decision trees used. By calculating feature importance, we can gain insights into which features the model relies on most, making the model more transparent and easier to interpret. This can also aid in identifying the most critical features, which helps improve the model's accuracy and reduce overfitting.

Another important use of feature importance is in **feature selection**. Not all features contribute equally to a model's predictive power. By identifying and focusing on the most influential features, you can remove irrelevant or redundant ones. This process can simplify the model, improve computation efficiency, and even enhance predictive performance. In situations where data collection is costly or difficult, knowing the most important features of the dataset can help prioritize which data to collect.

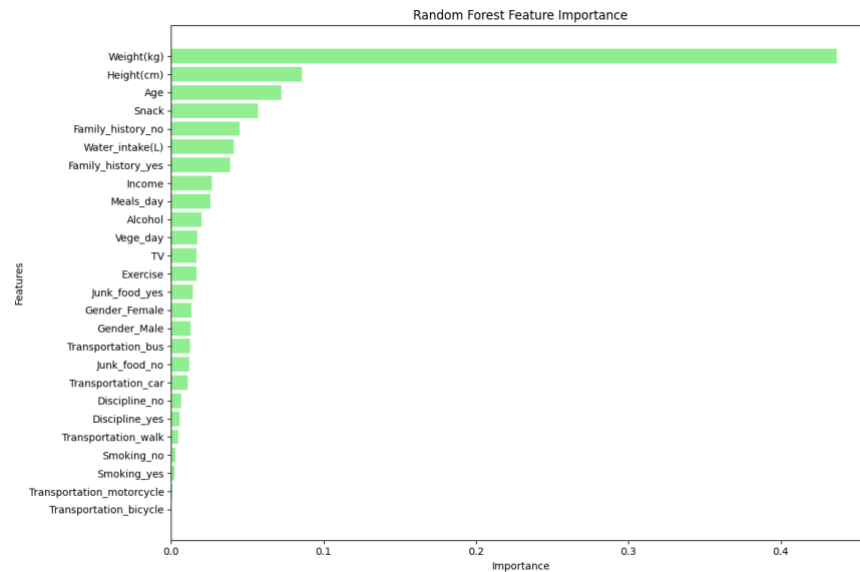


Figure 3.4.1 RandomForest Classifier Feature Chart

In the chart provided, Weight(kg) and Height(cm) stand out as the most important features, indicating that the model heavily relies on these attributes for its predictions. Features like Age and Snack also have a significant influence. On the other hand, features such as Transportation_motorcycle and Transportation_bicycle are less relevant, as shown by their shorter bars, implying that they contribute minimally to the model's performance. Understanding this ranking allows for better decisions on data refinement and helps streamline the overall analysis process.

Support Vector Machine Feature

The feature importance in a linear SVM is determined by the magnitude of the coefficients. Larger absolute values of the coefficients indicate that the corresponding feature has a stronger impact on the classification decision. Positive coefficients push the decision towards the positive class, while negative coefficients push it away. In this case, the charts help visualise which features are the most influential in determining cardiovascular risk levels, providing a clearer understanding of which factors should be prioritised in health assessments and interventions.

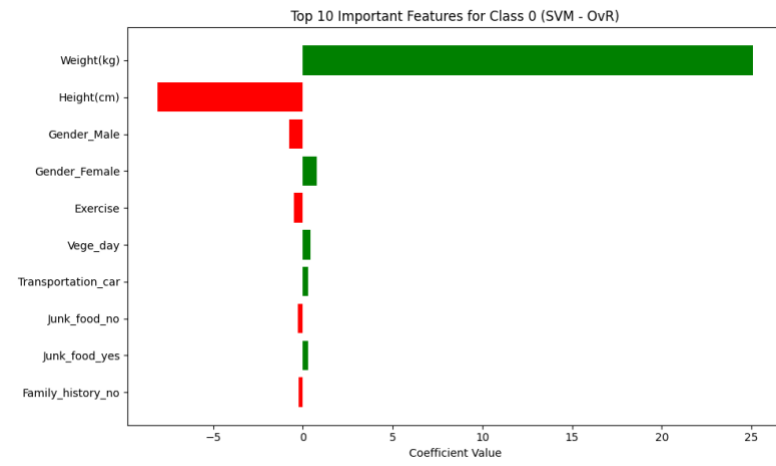


Figure 3.4.2 SVM-Class 0 (high) Chart

Class 0: High-Risk Group

Weight (kg) appears to be the most significant positive feature, indicating that higher weight strongly correlates with being classified as high-risk. This suggests that individuals with a higher weight are more likely to belong to this class. On the other hand, **Height (cm)** has a strong negative coefficient, indicating that taller individuals are less likely to be classified in this high-risk group. Other features like **Gender (Male)**, **Exercise**, and **Transportation (car)** also contribute in a lesser degree. The negative coefficient for **Gender (Male)** suggests that being male decreases the likelihood of falling into the high-risk group, while Exercise has a small positive influence, meaning those who exercise may still fall into this category depending on other factors like weight.

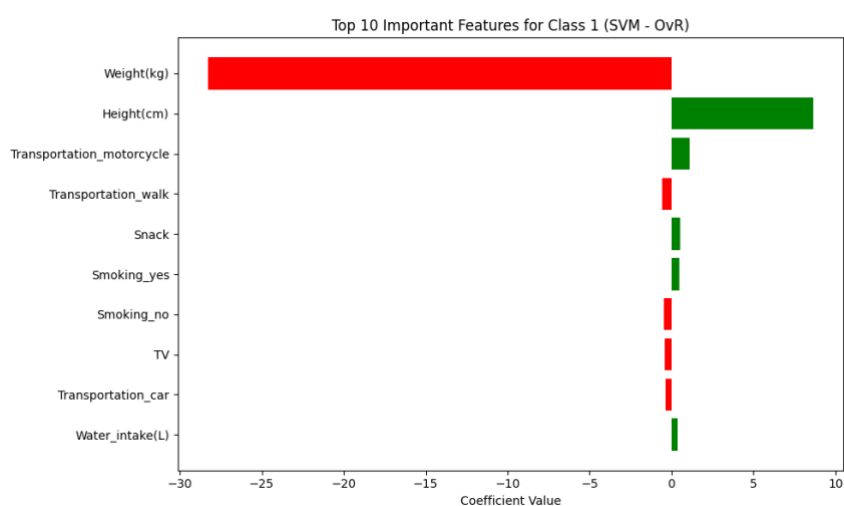


Figure 3.4.3 SVM-Class 1 (low) Chart

Class 1: Low-Risk Group

Weight (kg) again plays a major role but with a negative coefficient, indicating that lower weight significantly correlates with a low-risk classification. This suggests that maintaining a lower weight is a strong indicator of reduced cardiovascular risk. Similarly, **Height (cm)** shows a strong positive coefficient, indicating that taller individuals are more likely to be classified as low-risk. Lifestyle factors such as **Snack consumption** and **Smoking** show up as relevant features with smaller coefficients, implying that individuals who smoke or consume unhealthy snacks may still be classified as low-risk based on other favorable factors.

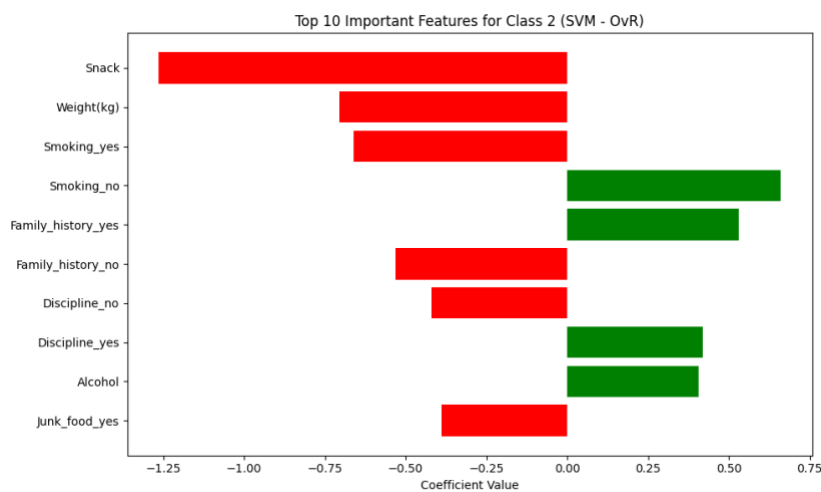


Figure 3.4.4 SVM-Class 2 (medium) Chart

Class 2: Medium-Risk Group

Snack consumption stands out as the most significant negative feature, indicating that consuming snacks correlates strongly with being in the medium-risk group. **Weight (kg)** also plays a role but with a negative coefficient, suggesting that individuals with slightly higher weight may still belong to this medium-risk category, albeit with caution. Other lifestyle and behavioral factors, such as **Smoking (yes)**, **Family History (yes)**, and **Discipline (no)**, appear as important factors. The mixed positive and negative coefficients for these features indicate that individuals in this class may have moderate risk factors influenced by a combination of genetic predisposition, lifestyle choices, and behavior.

Permutation Features for KNN

K-Nearest Neighbors (KNN) is a non-parametric algorithm. It does not have an inherent mechanism to provide feature importance like models such as Random Forest or Logistic Regression. KNN works by calculating distances between data points to find the

nearest neighbours and make predictions, but it does not learn explicit weights or coefficients for each feature during training. As a result, KNN doesn't naturally offer insight into how important each feature is for the model's decisions.

This is where the permutation feature importance comes in. Since KNN lacks built-in feature importance, permutation importance is used to evaluate how much each feature contributes to the model's predictive accuracy. The idea behind permutation importance is that it measures the change in model performance when the values of a specific feature are randomly shuffled. If the model's performance drops significantly after shuffling a feature, it suggests that the feature is important for making accurate predictions. If shuffling has little effect, the feature is likely less relevant.

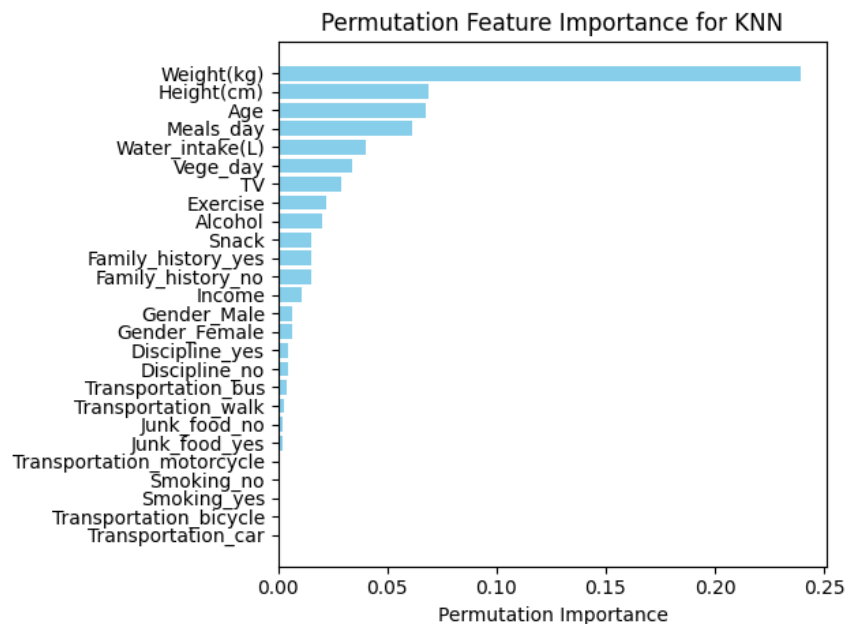


Figure 3.4.5 Permutation Feature Importance of KNN

In the chart above, **Weight(kg)**, **Height(cm)**, and **Age** are identified as the most important features for the KNN model because shuffling these features caused the greatest reduction in accuracy. On the other hand, features like **Transportation_car** and **Transportation_bicycle** had very little impact, indicating that the model doesn't rely on them as much for predictions. This process allows us to gauge which features the KNN model indirectly depends on, even though KNN itself doesn't provide feature importance.

4.0 Conclusion

The project's goal was to create machine learning models that would forecast cardiovascular risk in relation to a range of lifestyle and health-related variables. Preprocessing was done on the dataset to standardise numerical features, encode categorical data, and manage missing values. Random Forest, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) were the machine learning models that were used; these models were selected based on their abilities to handle various data kinds and classification tasks. To guarantee peak performance, these models were trained, verified, and adjusted using methods like GridSearchCV and cross-validation.

The Random Forest model emerged as the top performer, demonstrating exceptional accuracy and robustness both before and after tuning. It effectively managed the dataset's complexity and offered insights into the significance of each feature, emphasising the most important cardiovascular risk indicators. The model did, however, exhibit a slight overfitting risk, particularly on the training set, where it produced flawless results. In spite of this, it continued to perform well on the test set, demonstrating its capacity to generalise to new data successfully.

Additionally, the SVM model performed well, especially when it came to managing high-dimensional data and striking a solid balance between recall and precision. It performed better at balancing the trade-offs between sensitivity and specificity than the Random Forest, although it did not match it in terms of interpretability or feature relevance. For this reason, it is a dependable option for cardiovascular risk prediction. Its performance was further improved by fine-tuning, especially in situations when class disparity was an issue.

Despite being useful in certain situations, the KNN model performed worse than the other models, especially when it came to the test set. Its inability to generalise was hampered by its sensitivity to noisy data and dependence on feature scaling. While adjusting helped it perform a little better, KNN was still not as good as Random Forest and SVM at managing the dataset's complexity, suggesting that it might not be the ideal option for this specific task.

In short, the Random Forest and SVM models demonstrated good performance in predicting cardiovascular risk, but Random Forest's overall resilience and interpretability gave it a little advantage. While helpful in some circumstances, KNN performed less well on this job. The experiment illustrated how crucial feature analysis, model tweaking, and selection are

to creating trustworthy prediction models, particularly in vital domains like healthcare. Therefore, Random Forest is the best model in this project.