



UNIVERSITI TUNKU ABDUL RAHMAN

Faculty of Information and Communications Technology

Bachelor of Computer Science (Honours)

UCCD2063 Artificial Intelligence Techniques

Group 96

June 2024




Student Name	Leow Yi kang	Karlson Tay	Ho Yu Hao
Student ID	2301048	2300353	2301757
Contribution	30%	30%	40%
Signature			

Table of Contents

Chapter1: Introduction

1.1 Background	3
1.2 Objective	3

Chapter2: Methods

2.1 Dataset description	4
2.2 Data exploration	4-6
2.3 Data visualization	7-10
2.4 Data pre-processing	11
2.5, 2.6, 2.7 Model selection, training, validation, tuning, testing	12-39

Chapter3: Result and Discussion 40-44

Chapter4: Conclusion 45

Chapter 1: Introduction

1.1 Background

Cardiovascular diseases (CVDs) are one of the main causes of death around the world, which highlights the importance of assessing the risk of these diseases early and accurately. Being able to predict the risk of cardiovascular problems before they happen can help doctors create better prevention plans and treatments, potentially saving lives and lowering healthcare costs. The goal is to predict a person's risk level (low, medium, or high) of developing cardiovascular disease based on their health data.

The dataset provided in "dataset.csv" includes various health indicators such as age, alcohol, junk food, exercise habits, and other relevant features. These indicators can be used to predict an individual's cardiovascular risk. The challenge is to choose the right machine learning models, handle the data correctly, and adjust the models to get the most accurate predictions possible.

1.2 Objective

In this assignment, we implement a model that performs classification prediction for the given dataset "dataset.csv". The project aims to:

- Pick the best machine learning models, work with the data properly, and fine-tune the models to make the predictions as accurate as possible.
- Using different machine learning algorithms to determine which model performs best in predicting cardiovascular risk.
- Train the selected models using the dataset and fine-tune the models' hyperparameters to improve their accuracy and robustness.
- Assess the models' performance using appropriate metrics to compare the different machine learning models and choose the most effective one.

Chapter 2: Methods

2.1 Dataset description

The given dataset “dataset.csv” has contains 2100 individual across 18 columns which are 17 input matrix gender, age, height, weight, family history, alcohol, junk food, vege day, meals day, snack, smoking, water intake, transportation, exercise, tv, income, discipline and 1 output vector cardiovascular.

2.2 Data exploration

```
<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
```

Original dataset the figure above shows that the assigned dataset has 2100 samples with 17 attributes. The memory usage of this dataset is 295.4KB. There is not any missing value in this dataset. There are 10 categorical attributes and 8 numeric attributes.

	Gender	Age	Height(cm)	Weight(kg)	Family_history	Alcohol
0	Female	42	172.2	82.9	no	low
1	Female	19	175.3	80.0	yes	none
2	Female	43	158.3	81.9	yes	none
3	Female	23	165.0	70.0	yes	low
4	Male	23	169.0	75.0	yes	low

We can observe the figure above that the 5 data from the dataset that there are 2 categorical attributes which data type nominal which are Gender and Family history. The other 3 age, Height and Weight categorical attributes which data type numerical. For the Alcohol it looks like nominal because it seems like it can be separate to ordinal.

Junk_food	Vege_day	Meals_day	Snack	Smoking	Water_intake(L)
yes	3	3	Sometimes	no	2.72
yes	2	1	Sometimes	no	2.65
yes	3	1	Sometimes	no	1.89
no	2	1	Sometimes	no	2.00
yes	3	3	Sometimes	no	2.82

Next, we can observe the figure above that the 5 data from the dataset there are 2 categorical attributes which data type nominal which are Junk food and Smoking. The other 3 Vege day, Meals Day and Water intake categorical attributes which data type numerical. For the Snack it looks like it can be nominal or also ordinal but not sure yet will analyze in further detail.

Transportation	Exercise	TV	Income	Discipline	Cardiovascular_risk(y)
car	3	rare	2081	no	medium
bus	3	moderate	5551	no	medium
car	1	rare	14046	no	high
bus	0	rare	9451	no	medium
bus	1	often	17857	no	medium

We can observe from the figure above that the 5 data from the dataset that there are 3 categorical attribute Transportation, TV, Discipline which data type nominal. 2 categorical attributes Exercise and Income which data type numerical. Also, there just only one dependent variable which is Cardiovascular risk(y).

```

Gender                                0
Age                                  0
Height(cm)                          0
Weight(kg)                          0
Family_history                      0
Alcohol                             0
Junk_food                           0
Vege_day                            0
Meals_day                           0
Snack                               0
Smoking                             0
Water_intake(L)                     0
Transportation                       0
Exercise                             0
TV                                   0
Income                              0
Discipline                           0
Cardiovascular_risk(y)              0
dtype: int64

```

From the above analysis, it is clear that there are no missing values in the dataset.

2.3 Data Visualization

Histograms of Numeric Attributes

Based on the histograms provided, here are the detailed observations:

Age

- The age histogram shows that the highest class of age is below 20 years old. The data distribution is skewed to the right, indicating that most data points are concentrated in the lower age range.

Height (cm)

- The height histogram shows a relatively normal distribution with a slight skew to the right. Most individuals fall within the 150-180 cm range.

Weight (kg)

- The weight histogram is skewed to the right, with most individuals weighing between 50-80 kg.

Veg_day (Vegetables per day)

- The veg_day histogram shows that most people consume fewer vegetables per day, with the data distribution skewed to the right.

Meals_day (Meals per day)

- The meals_day histogram indicates that most people have around 3 meals per day, with a slight skew to the right.

Water intake (L)

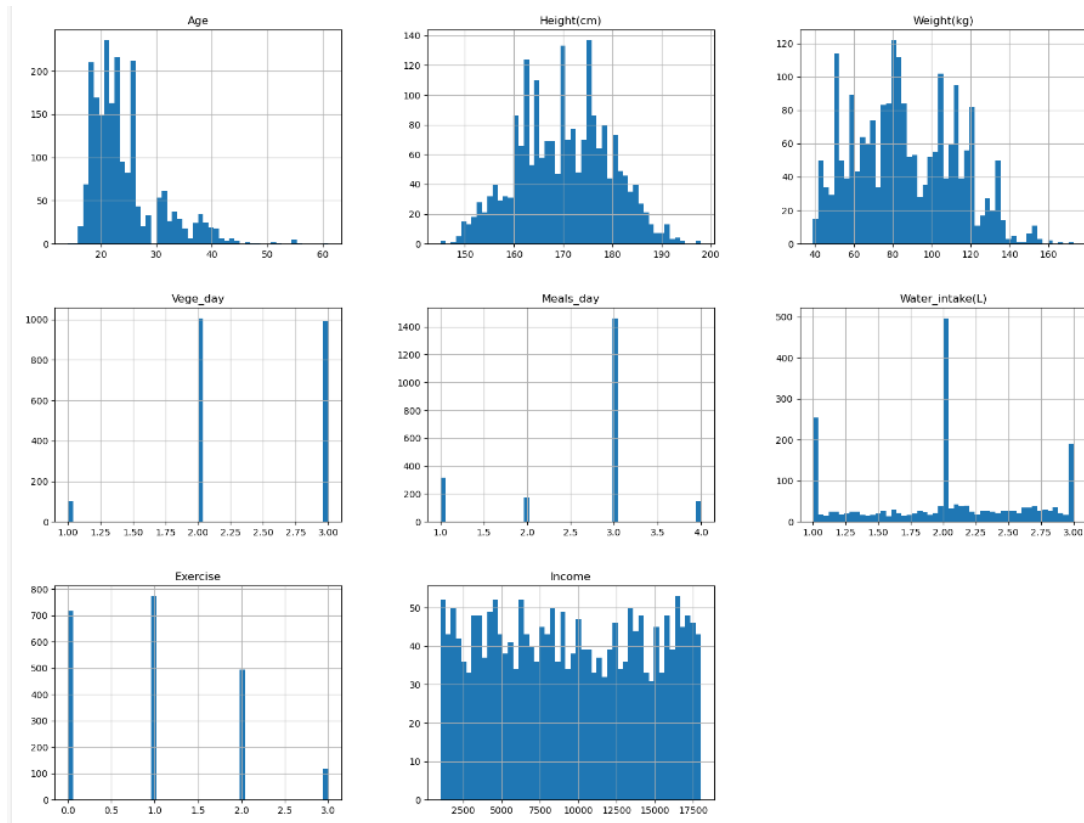
- The water_intake histogram shows that most people consume between 1 to 2 liters of water per day, with the data distribution skewed to the right.

Exercise (hours per day)

- The exercise histogram shows that most people exercise less than 1 hour per day, with the data distribution skewed to the right.

Income

- The income histogram is skewed to the right, indicating that most individuals have lower income levels, with a few high-income outliers.



Bar Charts of Categorical Attributes

Based on the bar charts provided, here are the detailed observations:

Gender

- The bar chart shows that there are more females than males in the dataset.

Family History

- The bar chart indicates that a significant number of individuals have a family history of the condition being studied.

Alcohol

- The bar chart shows that most individuals do not consume alcohol frequently.

Junk Food

- The bar chart indicates that the majority of individuals consume junk food occasionally.

Snack

- The bar chart shows that most individuals snack sometimes, with fewer individuals snacking frequently.

Smoking

- The bar chart indicates that smoking is not a common habit among the individuals in the dataset.

Transportation

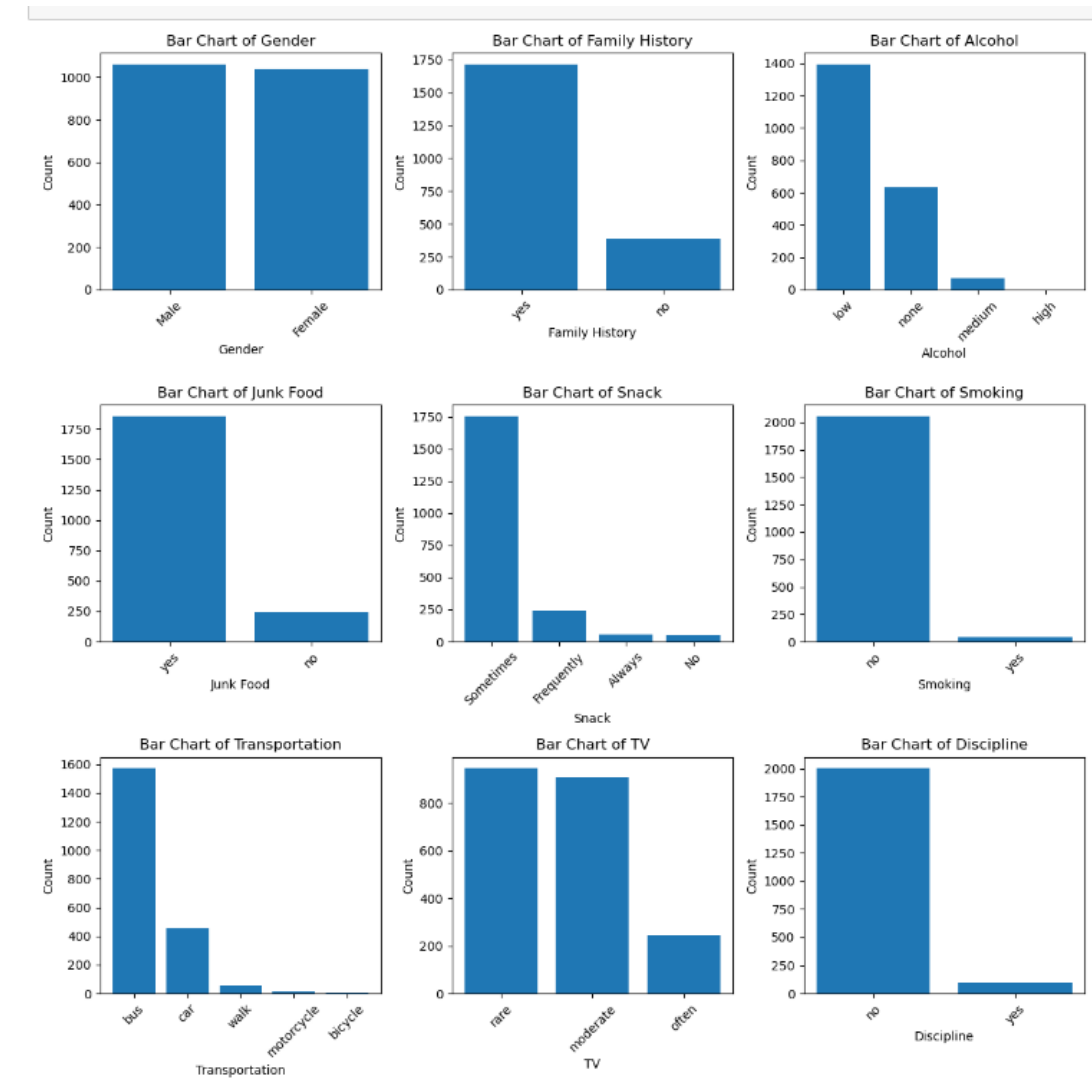
- The bar chart shows that the most common mode of transportation is by car, followed by motorcycle and bicycle.

TV

- The bar chart indicates that most individuals watch TV occasionally, with fewer individuals watching frequently.

Discipline

- The bar chart shows that most individuals have a moderate level of discipline, with fewer individuals having high or low levels of discipline.



2.4 Data pre-processing

Problem	Action
Missing Value (x_train, y_train)	None missing value and perform fillna(0) in x_train and y_train
Categorical – total 8 attributes 'Height(cm)', 'Weight(kg)', 'Vege_day', 'Meals_day', 'Water_intake(L)', 'Exercise', 'Income', 'Age',	Preprocessing numerical data: Standardization
Categorical – total 9 attributes -(nominal)- 'Gender', 'Family_history', 'Junk_food', 'Smoking', 'Discipline' -(ordinal)- 'Alcohol', 'Snack', 'TV', 'Transportation'	Preprocessing categorical data: Ordinal Encoder and one-hot encoding
Output vector (y) - Cardiovascular_risk (y)	Preprocessing data with Label Encoder (high, medium, low)

2.5 Model selection (Three Model Selection)

1) Logistic Regression

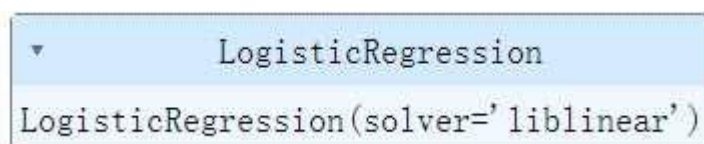
Logistic Regression is a suitable model for your dataset because it is used for binary classification problems where the outcome variable is categorical. It works well when you want to predict probabilities for two classes.

Justification For Logistic Regression:

1. **Binary Outcome:** Logistic regression is designed for predicting binary outcomes. If your target variable is categorical with two outcomes, logistic regression will be appropriate.
2. **Interpretability:** Logistic regression provides coefficients that can be interpreted as the log-odds of the dependent variable being in one class versus the other, which is useful for understanding the influence of each feature.
3. **Feature Scaling:** Logistic regression performs well even with features of varying scales, but it's advisable to standardize your features for better performance.

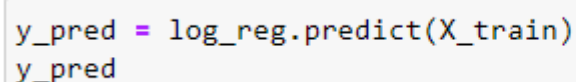
2.6 Model training and validation (Logistic Regression)

1. Import Logistic Regression from `sklearn.linear_model` to perform Logistic Regression



```
▼ LogisticRegression
LogisticRegression(solver='liblinear')
```

Perform prediction using `X_train` data



```
y_pred = log_reg.predict(X_train)
y_pred
```

2. Create a function (show10results(y_train, y_pred, num=10)) to show 10 random samples for actual and predicted value.

```
Result for 10 random samples:
Actual = 0 Predict = 0
Actual = 2 Predict = 1
Actual = 0 Predict = 0
Actual = 2 Predict = 2
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 2 Predict = 2
Actual = 0 Predict = 0
Actual = 1 Predict = 1
Actual = 1 Predict = 1
```

3. Import accuracy_score, confusion_matrix, classification_report from sklearn.metrics

Print out the output of train_accuracy, confusion_matrix, classification_report for training data.

```
Training Accuracy: 0.955952380952381
Confusion Matrix:
[[776  0  5]
 [  0 419 16]
 [ 22 31 411]]
Classification Report (Training):
```

	precision	recall	f1-score	support
0	0.97	0.99	0.98	781
1	0.93	0.96	0.95	435
2	0.95	0.89	0.92	464
accuracy			0.96	1680
macro avg	0.95	0.95	0.95	1680
weighted avg	0.96	0.96	0.96	1680

```
Unique classes in Y_train: [0 1 2]
```

The training model achieved an accuracy of up to 95%, which is the highest compared to other models. The confusion matrix further highlights this model's performance, showing that class 0 was correctly predicted 776 times, class 1 was correctly predicted 419 times, and class 2 was correctly predicted 411 times. Additionally, all three classifications have a precision rate of over 90%. The Y_train data contains three unique categories: 0, 1, and 2.

4. Import cross_val_predict from sklearn.model_selection

Apply show10results () function to print out 10 random samples for y_train and y_pred_cv

```
from sklearn.model_selection import cross_val_predict
y_pred_cv = cross_val_predict(log_reg, X_train, Y_train, cv = 5)
```

```
Result for 10 random samples:
Actual = 2 Predict = 2
Actual = 1 Predict = 1
Actual = 1 Predict = 1
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 2 Predict = 2
Actual = 2 Predict = 2
Actual = 0 Predict = 0
Actual = 2 Predict = 2
Actual = 1 Predict = 1
y_pred_cv.shape : (1680,)
Y_train.shape : (1680,)
```

5. Import precision_score, recall_score, f1_score from sklearn.metrics

Print out the output of precision_scr, recall_scr, f1_scr

```
Precision Score: 0.9442876380592372
Recall Score: 0.9446428571428571
F1 Score: 0.9439440432393683
y_pred_cv.shape : (1680,)
Y_train.shape : (1680,)
```

The model achieved precision, recall, and F1 scores of 94%, reflecting its strong ability to correctly identify and classify the data. These consistent scores highlight the model's effectiveness in handling the three unique categories—0, 1, and 2—in the Y_train dataset.

```

In [25]: #perform decision function scores
y_scores = log_reg.decision_function(X_train)
y_scores

Out[25]: array([[ -5.05344815,  -1.01885886,  -0.75122461],
 [ 11.74419286, -16.06709471, -2.07049253],
 [ -5.30674788,  -2.41084706, -0.92664201],
 ...,
 [  1.67861598,  -8.04765669, -1.18913539],
 [  2.59743172, -9.80455437, -0.19876263],
 [ -2.09385854,  -6.0891795 , -0.14452814]])

In [32]: y_scores_cv = cross_val_predict(log_reg, X_train, Y_train, cv = 3, method = "predict_proba")
y_scores_cv

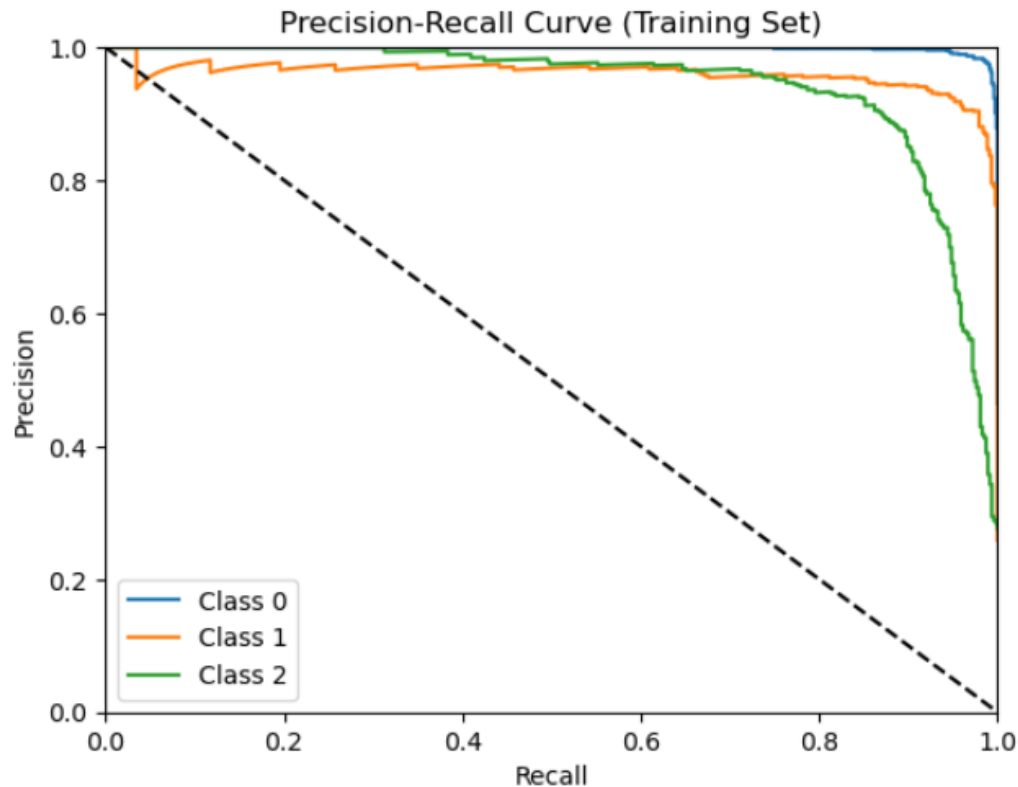
Out[32]: array([[1.94579143e-02, 4.44028971e-01, 5.36513115e-01],
 [8.98991956e-01, 5.40934321e-07, 1.01007503e-01],
 [2.88319571e-02, 2.56270049e-01, 7.14897994e-01],
 ...,
 [7.98677592e-01, 1.05734051e-03, 2.00265067e-01],
 [6.97323570e-01, 1.32704961e-04, 3.02543725e-01],
 [2.65934220e-01, 8.46518239e-03, 7.25600598e-01]])

```

I used the decision function to compute the scores for the training data, allowing me to evaluate the model's confidence in its predictions. Additionally, I performed cross-validation with `predict_proba` to obtain probability estimates for the classes, further validating the model's performance across different folds.

6. Import `precision_recall_curve` from `sklearn.metrics`

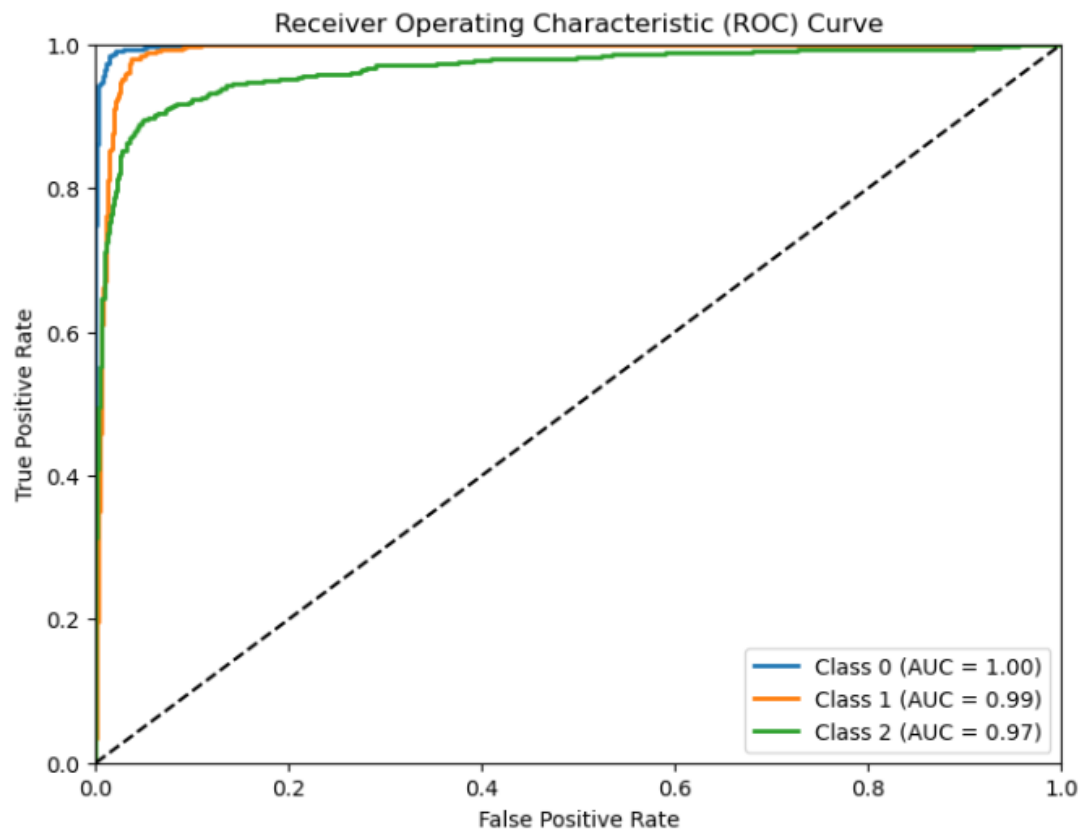
Create a function (`plot_precision_vs_recall(precisions, recalls)`) to plot a Precision-Recall Graph for Training Set.



The Precision-Recall curve output shows that the curves are concentrated towards the upper right corner, which indicates strong model performance. Specifically, it suggests the model can maintain high precision even as recall increases, meaning it consistently identifies true positives with minimal false positives across the classes. This is a positive sign of a well-performing classifier, particularly in scenarios where both high precision and high recall are crucial.

7. Import `roc_curve` from `sklearn.metrics`

Create a function (`plot_roc_curve(fpr, tpr)`) to plot TPR vs FPR Graph for Training Set



8. Import `roc_auc_score` from `sklearn.metrics` to print out Area Under Curve (AUC) in 4 decimal places. The macro- average AUC is 0.9851.

```
Class 0 AUC = 0.9985
Class 1 AUC = 0.9909
Class 2 AUC = 0.9659
Macro-average AUC = 0.9851
```

2.7 Model tuning and testing

Model tuning

1. Import RandomizedSearchCV from sklearn.model_selection and import loguniform from scipy.stats

Print out the Final Score and the Final Hyperparameters

```
LogisticRegression
LogisticRegression(C=26.779972328542662, penalty='l1', solver='liblinear')
```

```
Best Score: 0.9928571428571429
Best Hyperparameters: {'C': 26.779972328542662, 'penalty': 'l1', 'solver': 'liblinear'}
```

We can output that the best hyperparameters of this training model.

Model Testing

2. Use cross_val_predict to training the test data

Test Set (Logistic Regression)

Cross_val_predict

```
In [128]: y_pred_test = log_reg.predict(x_test)
y_pred_test_cv = cross_val_predict(log_reg, x_test, y_test, cv=3)
y_pred_test_cv
```

```
Out[128]: array([1, 0, 1, 0, 0, 1, 0, 1, 0, 2, 0, 2, 0, 0, 1, 1, 1, 0, 1, 2, 2, 2,
2, 1, 1, 1, 2, 1, 1, 0, 1, 2, 2, 1, 1, 1, 0, 2, 1, 1, 1, 2, 1, 1,
0, 0, 0, 1, 2, 1, 0, 2, 0, 0, 1, 1, 2, 2, 0, 2, 0, 1, 2, 0, 2, 0,
0, 1, 2, 0, 2, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 2, 2, 0,
0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 0, 0, 2, 0, 1, 2, 2,
0, 1, 0, 2, 0, 1, 1, 2, 2, 1, 0, 0, 1, 0, 0, 2, 1, 0, 2, 1, 1, 0,
1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 2, 0, 1, 0, 0, 0, 0, 0, 2,
0, 1, 0, 1, 2, 1, 0, 1, 0, 1, 2, 1, 0, 2, 1, 1, 0, 0, 2, 2, 2, 0,
0, 1, 2, 0, 0, 2, 1, 0, 2, 0, 2, 2, 0, 2, 2, 1, 1, 0, 1, 1, 0, 1,
1, 0, 2, 0, 2, 1, 0, 1, 1, 0, 2, 0, 0, 2, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 2, 2, 0, 1, 2, 0, 2, 2, 0, 1, 2, 1, 0, 0, 1, 2, 0, 1, 0,
1, 2, 0, 0, 1, 1, 2, 1, 1, 2, 0, 0, 2, 2, 0, 2, 0, 1, 2, 0, 2, 0,
2, 0, 0, 0, 2, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 0, 1, 2, 2, 0, 2, 0,
2, 0, 0, 0, 2, 0, 2, 2, 1, 1, 2, 0, 1, 1, 1, 2, 2, 2, 0, 0, 0, 0,
1, 2, 0, 0, 1, 1, 2, 0, 1, 0, 1, 2, 0, 0, 2, 0, 2, 2, 1, 0, 0, 2,
1, 0, 0, 0, 1, 0, 2, 0, 2, 1, 0, 2, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
0, 2, 1, 0, 1, 0, 0, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 2, 1, 1, 1,
0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 0, 0, 2, 1, 1, 1,
2, 0, 0, 1, 0, 1, 0, 2, 0, 0, 2, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0,
2, 0])
```

3. Print out Testing Accuracy for testing set

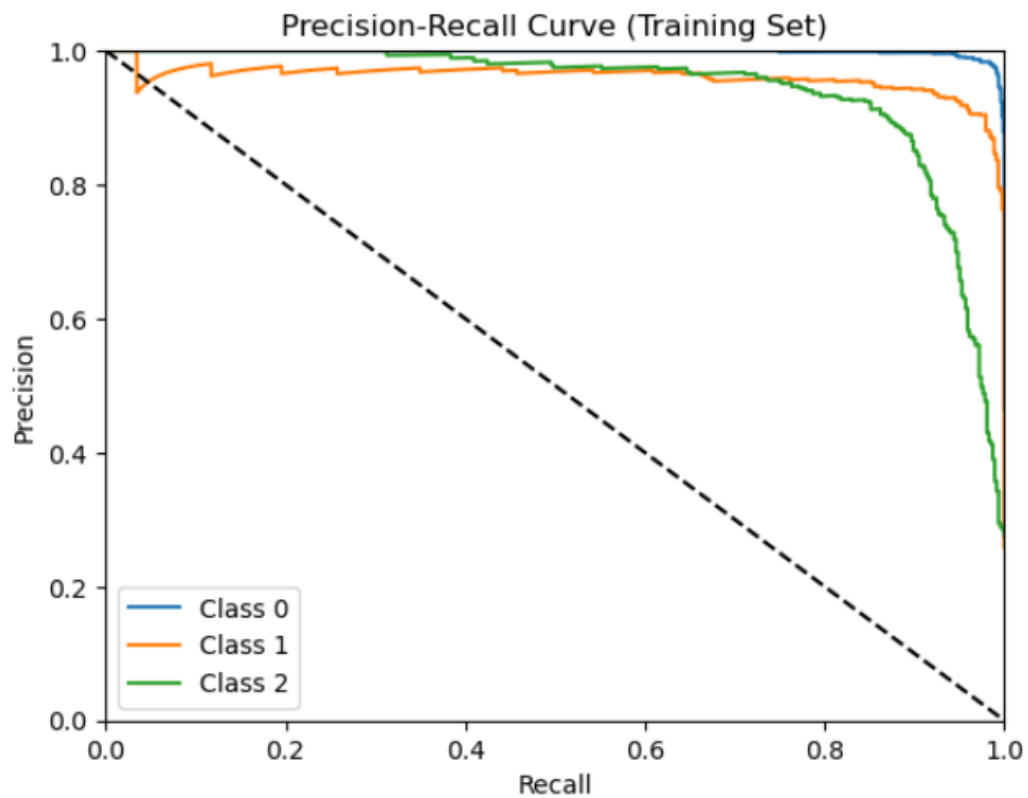
```
Testing Accuracy: 0.9595238095238096
Confusion Matrix:
[[185  0  1]
 [ 0 115  7]
 [ 3  6 103]]
Classification Report (Testing):
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	186
1	0.95	0.94	0.95	122
2	0.93	0.92	0.92	112
accuracy			0.96	420
macro avg	0.95	0.95	0.95	420
weighted avg	0.96	0.96	0.96	420

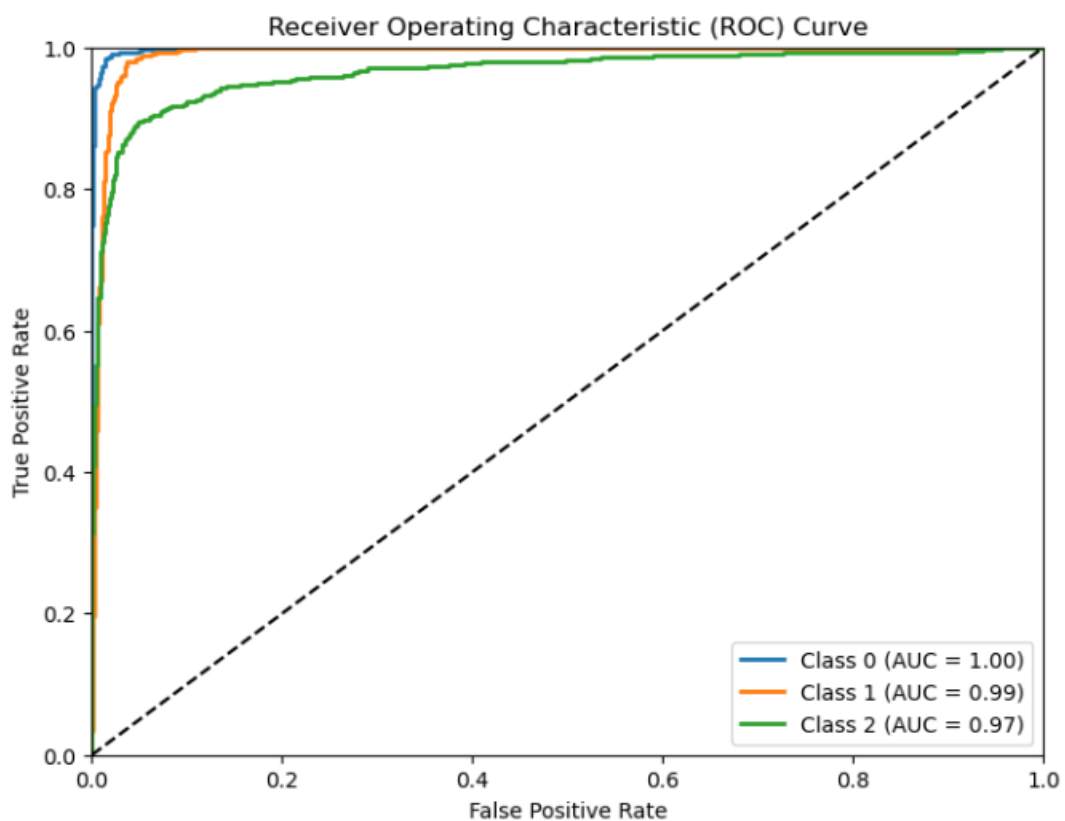
4. Print out the precision score, recall score and f1 score for testing set

```
Precision Score: 0.9593101334397272
Recall Score: 0.919047619047619
F1 Score: 0.918380790366884
```

5. Plot Precision-Recall Graph for Test Set



6. Plot TPR vs FPR Graph for Test Set



The ROC curves are near the upper left corner, indicating that the model performs well with high True Positive Rates and low False Positive Rates. This suggests effective class differentiation and reliable performance.

7. Print out the value of Area Under Curve (AUC) for Test Set

```
Class 0 AUC = 0.9985
Class 1 AUC = 0.9909
Class 2 AUC = 0.9659
Macro-average AUC = 0.9851
```

The high AUC scores for all classes, with a macro-average AUC of 0.9851, indicate excellent overall model performance.

2) KNeighborsClassifier

KNeighborsClassifier is simple to implement, doesn't require any assumptions about the data distribution, and works well with smaller datasets. However, it can be computationally expensive and less effective with large, high-dimensional datasets.

Justification For KNeighborsClassifier

1. **Simplicity and Intuitiveness:** The K-Nearest Neighbors (KNN) algorithm is easy to understand and use. It classifies a data point by looking at how its nearby points are classified, making it simple to learn and explain.
2. **Adaptability:** The algorithms can be easily adapted to different types of distance metrics and handle multi-class classification problems.
3. **Versatility:** The algorithm can be used for both two-class and multi-class classification problems. It can also work with both numbers and categories, making it flexible for different types of tasks.

2.6 Model training and validation (KNeighborsClassifier)

1. Import KNeighborsClassifier from sklearn.neighbors to perform KNeighborsClassifier

```
▼ KNeighborsClassifier  
KNeighborsClassifier()
```

Perform prediction using X_train data

```
y_pred_knn = knn_classifier.predict(X_train)  
y_pred_knn
```

2. Create a function (show10results (y_train, y_pred, num=10)) to show 10 random samples for actual and predicted value.

Result for 10 random samples:

```
Actual = 0 Predict = 0
Actual = 1 Predict = 1
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 1 Predict = 1
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 2 Predict = 2
```

3. Import accuracy_score, confusion_matrix, classification_report from sklearn.metrics

Print out the output of train_accuracy, confusion_matrix, classification_report for training data.

Training Accuracy: 0.8958333333333334

Confusion Matrix:

```
[[757  4 20]
 [ 25 363 47]
 [ 55  24 385]]
```

Classification Report (Training):

	precision	recall	f1-score	support
0	0.90	0.97	0.94	781
1	0.93	0.83	0.88	435
2	0.85	0.83	0.84	464
accuracy			0.90	1680
macro avg	0.89	0.88	0.89	1680
weighted avg	0.90	0.90	0.89	1680

Unique classes in Y_train: [0 1 2]

The training model achieved an accuracy of up to 89%. The confusion matrix further highlights this model's performance, showing that class 0 was correctly predicted 757 times, class 1 was correctly predicted 363 times, and class 2 was correctly predicted 385 times. Additionally, all three classifications have a precision rate of over 80%. The Y_train data contains three unique categories: 0, 1, and 2.

4. Import cross_val_predict from sklearn.model_selection

Apply show10results () function to print out 10 random samples

```
from sklearn.model_selection import cross_val_predict
y_pred_knn_cv = cross_val_predict(knn_classifier, X_train, Y_train, cv = 5)
show10results(Y_train, y_pred_knn_cv)
```

Result for 10 random samples:

Actual = 1 Predict = 1

Actual = 2 Predict = 0

Actual = 2 Predict = 2

Actual = 2 Predict = 2

Actual = 0 Predict = 0

Actual = 1 Predict = 1

Actual = 0 Predict = 0

Actual = 1 Predict = 1

Actual = 0 Predict = 0

Actual = 0 Predict = 0

```
y_scores_knn = knn_classifier.predict_proba(X_train)
y_scores_knn
y_scores_knn_cv = cross_val_predict(knn_classifier, X_train, Y_train, cv = 3, method='predict_proba')
y_scores_knn_cv
```

I used the decision function to compute the scores for the training data, allowing me to evaluate the model's confidence in its predictions. Additionally, I performed cross-validation with predict_proba to obtain probability estimates for the classes, further validating the model's performance across different folds.

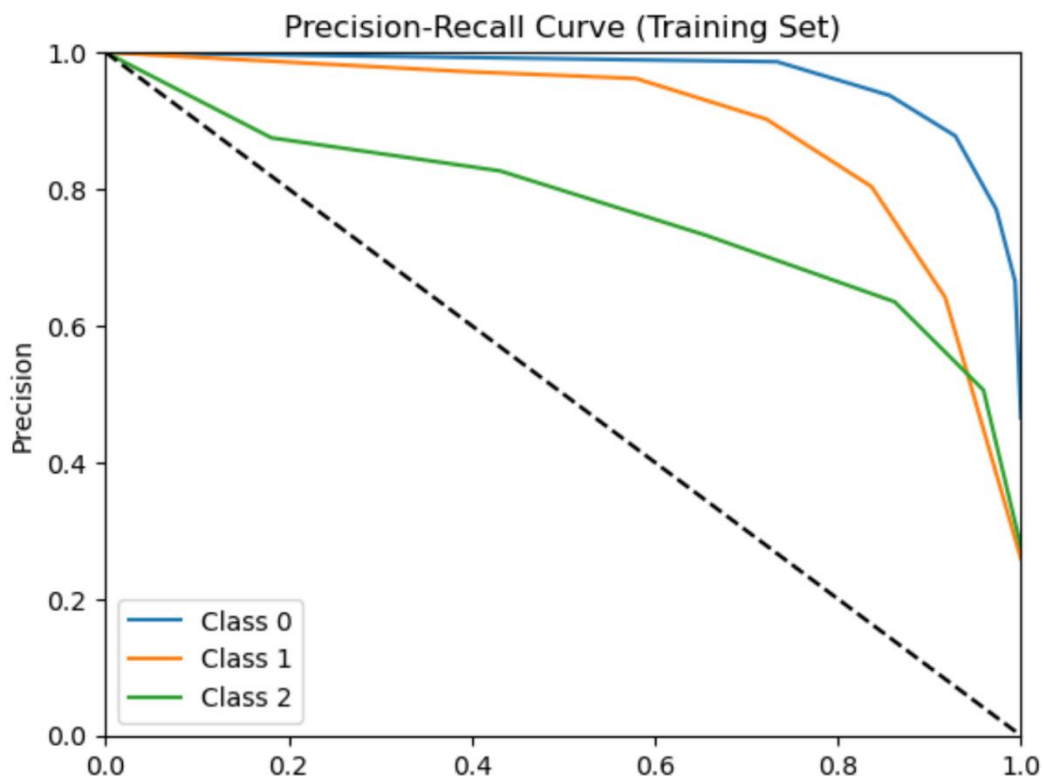
5. Import `precision_score`, `recall_score`, `f1_score` from `sklearn.metrics`

Print out the output of `precision_scr`, `recall_scr`, `f1_scr`

```
Precision Score: 0.8257905376858308
Recall Score: 0.825
F1 Score: 0.8223628201070288
y_pred_cv.shape : (1680,)
Y_train.shape : (1680,)
```

6. Import `precision_recall_curve` from `sklearn.metrics`

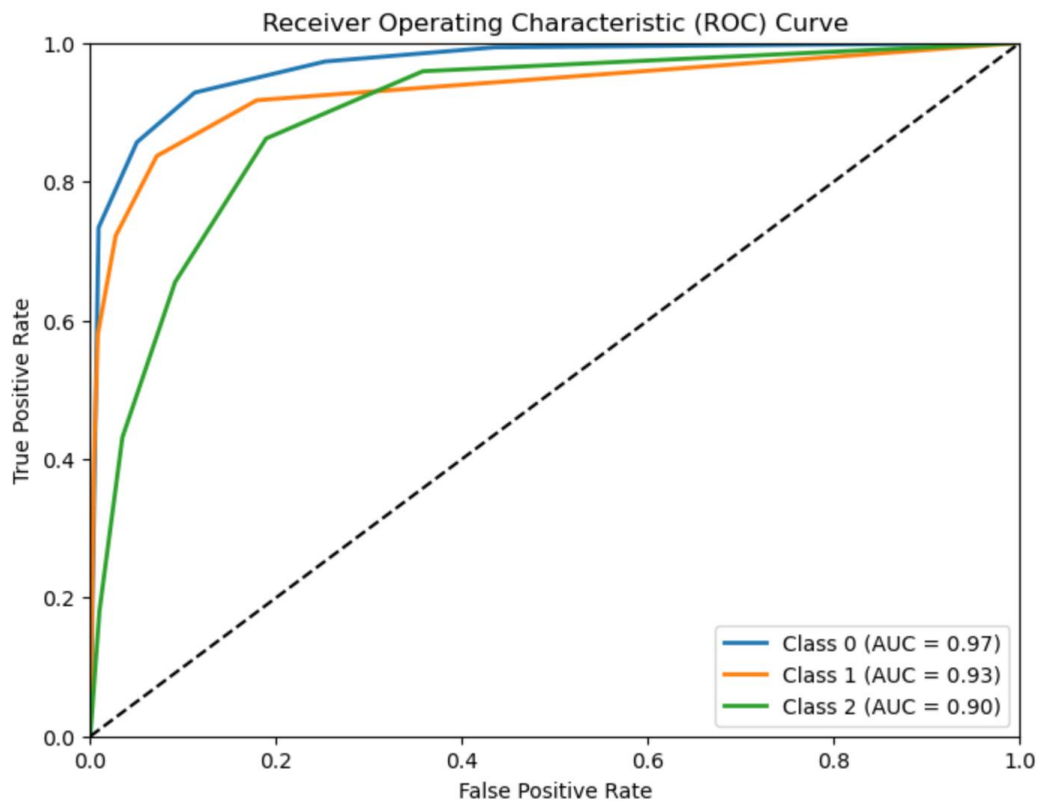
Create a function (`plot_precision_vs_recall(precisions, recalls)`) to plot a Precision-Recall Graph for Training Set.



The precision-recall curves for all classes are above the diagonal dashed line, which represents random guessing. This indicates that the model is performing better than random guessing for all classes. The model performs best in Class 0, well in Class 1, and struggles the most with Class 2. All classes perform better than random guessing.

7. Import `roc_curve` from `sklearn.metrics`

Create a function (`plot_roc_curve(fpr, tpr)`) to plot TPR vs FPR Graph for Training Set



8. Import `roc_auc_score` from `sklearn.metrics` to print out Area Under Curve (AUC) in 4 decimal places. The macro-average AUC is 0.9315.

Class 0 AUC = 0.9675

Class 1 AUC = 0.9310

Class 2 AUC = 0.8961

Macro-average AUC = 0.9315

2.7 Model tuning and testing

Model tuning

1.Import GridSearchCV from sklearn.model_selection and import KNeighborsClassifier from sklearn.neighbors

Print out the Final Score and the Final Hyperparameters

```
Best Hyperparameters: {'algorithm': 'auto', 'leaf_size': 20, 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}
Best Accuracy: 89.11%
Best Scores: 0.8910714285714286
```

Model Testing

2.Use cross_val_predict to training the test data

```
y_pred_test_knn = knn_classifier.predict(x_test)
y_pred_test_knn_cv = cross_val_predict(knn_classifier, x_test, y_test, cv=3)
y_pred_test_knn_cv
```

```
array([1, 0, 1, 0, 0, 1, 0, 1, 1, 2, 0, 0, 0, 0, 1, 1, 2, 1, 1, 2, 2, 2,
       2, 1, 2, 2, 0, 1, 0, 0, 1, 0, 2, 2, 1, 1, 0, 0, 1, 1, 1, 1, 1, 2,
       0, 0, 0, 1, 0, 1, 0, 2, 0, 0, 2, 1, 0, 2, 0, 2, 0, 1, 2, 0, 2, 0,
       0, 1, 2, 0, 0, 2, 2, 2, 0, 0, 0, 0, 2, 0, 0, 1, 0, 2, 1, 2, 0, 0,
       2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 1, 2, 1, 1, 1, 2, 0, 2, 0, 2, 2, 0,
       0, 1, 0, 2, 0, 1, 2, 0, 2, 1, 0, 0, 1, 0, 0, 2, 1, 0, 2, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 2, 1, 0, 2, 0, 1, 0, 0, 0, 0, 0, 2,
       0, 1, 0, 1, 2, 1, 0, 2, 0, 1, 0, 2, 0, 0, 2, 1, 0, 0, 1, 0, 2, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 2, 2, 0, 2, 2, 1, 2, 0, 2, 1, 0, 1,
       1, 0, 0, 0, 2, 1, 0, 1, 1, 0, 2, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       2, 0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 2, 1, 0, 0, 1, 2, 2, 2, 0,
       2, 2, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 2, 0, 0, 0, 0, 1, 2, 0, 2, 0,
       2, 0, 0, 0, 1, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 0, 1, 0, 2, 0, 2, 0,
       0, 0, 2, 0, 0, 0, 0, 0, 2, 1, 2, 0, 1, 1, 1, 2, 0, 2, 0, 0, 0, 0,
       1, 2, 0, 0, 1, 2, 0, 0, 0, 0, 1, 2, 2, 0, 2, 0, 2, 2, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 2, 2, 1, 1, 0, 2, 0, 0, 1, 2, 0, 1, 0, 1, 0, 1,
       0, 2, 1, 0, 1, 0, 0, 1, 0, 2, 0, 0, 0, 2, 0, 2, 2, 0, 1, 1, 1, 1,
       0, 1, 0, 2, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 0, 1, 2, 1, 1, 1,
       2, 0, 0, 2, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 2,
       2, 0])
```

3. Print out Testing Accuracy for testing set

Testing Accuracy: 0.7785714285714286

Confusion Matrix:

```
[[171  3 12]
```

```
[  3 91 28]
```

```
[ 34 13 65]]
```

Classification Report (Testing):

	precision	recall	f1-score	support
0	0.82	0.92	0.87	186
1	0.85	0.75	0.79	122
2	0.62	0.58	0.60	112
accuracy			0.78	420
macro avg	0.76	0.75	0.75	420
weighted avg	0.78	0.78	0.78	420

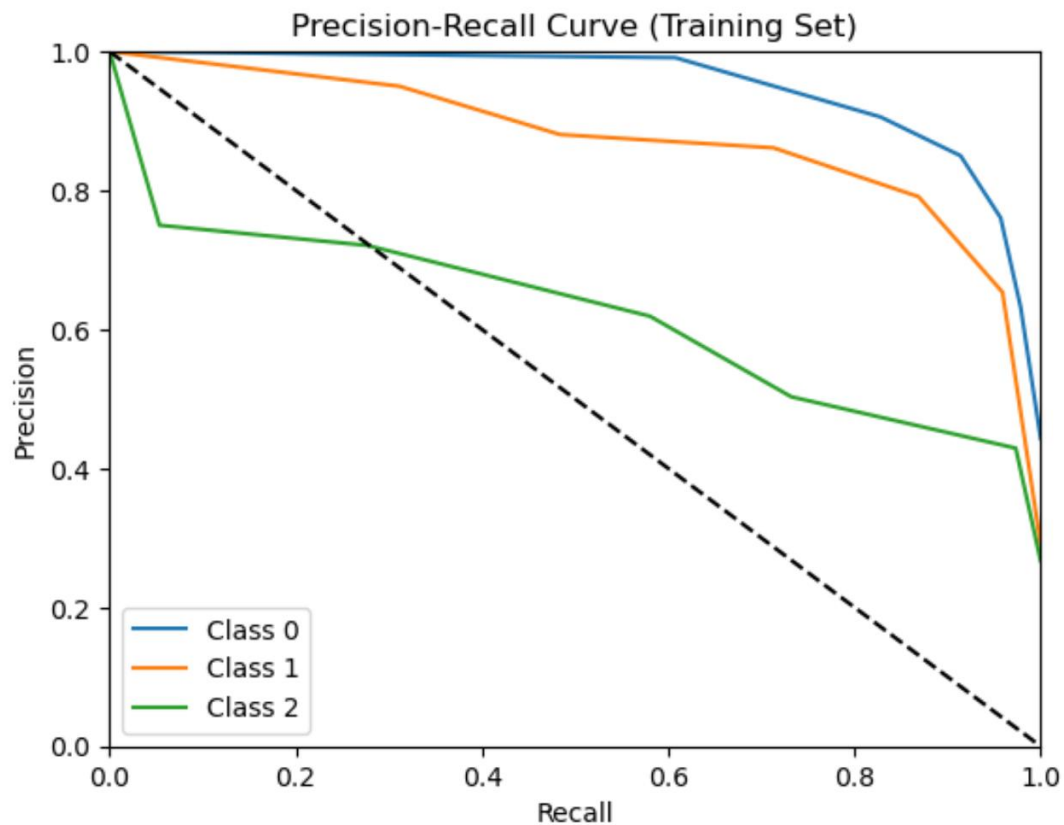
4. Print out the precision score, recall score and f1 score for testing set

Precision Score: 0.8752744958666191

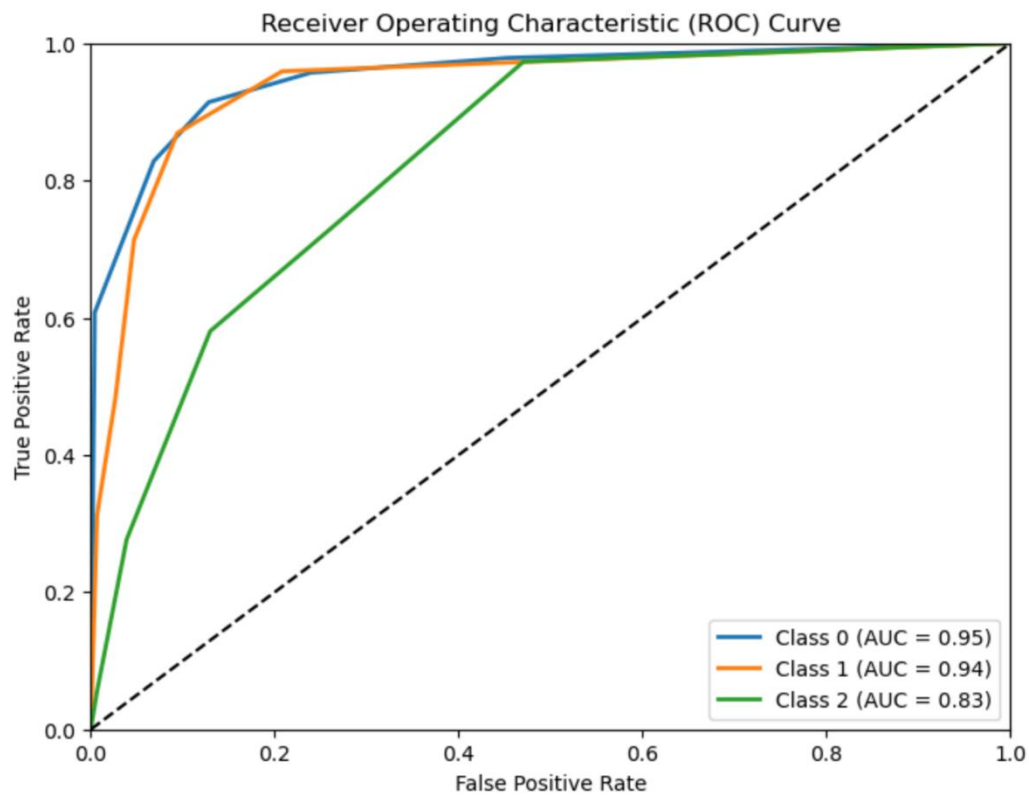
Recall Score: 0.7785714285714286

F1 Score: 0.7750220226982166

5. Plot Precision-Recall Graph for Test Set



6. Plot TPR vs FPR Graph for Test Set



The ROC curves are slightly near the upper left corner, class0 and 1 is most near to upper left indicating that the model achieves high True Positive Rates and low False Positive Rates. This reflects effective class slightly differentiation and strong overall performance.

7. Print out the value of Area Under Curve (AUC) for Test Set

```
Class 0 AUC = 0.9513
Class 1 AUC = 0.9382
Class 2 AUC = 0.8312
Macro-average AUC = 0.9069
```

The high AUC scores for all classes, with a macro-average AUC of 0.9069, indicate excellent overall model performance.

3) Random Forest Classifier

Random Forest Classification can work with smaller datasets and is able to handle both numerical and categorical data. With large datasets, Random Forest Classification can take up a lot of memory.

Justification For Random Forest Classifier

1. **High accuracy:** Random Forest combines the results of multiple decision trees trained on distinct subsets of the data, and the individual trees will vote for the predicted classification.
2. **Estimating feature importance:** The contributions of each feature towards classification is evaluated and calculated for their importance. Features that result in higher reduction of impurities are considered more significant. Knowledge of which features are important can aid with feature selection or dimensionality reduction to make prediction less computationally expensive.

2.6 Model training and validation (Random Forest Classifier)

1. Import RandomForestClassifier from sklearn.ensemble

```
▼ RandomForestClassifier ⓘ ⓘ  
RandomForestClassifier(random_state=30)
```

2. Predict using X_train data

```
# Perform predictions  
y_pred_rf = rf.predict(X_train)
```

Show 10 predictions on the training set

```
# Show 10 results of the prediction label and actual label
show10results(Y_train, y_pred_rf)
```

Result for 10 random samples:

```
Actual = 0 Predict = 0
Actual = 1 Predict = 1
Actual = 2 Predict = 2
Actual = 1 Predict = 1
Actual = 0 Predict = 0
Actual = 1 Predict = 1
Actual = 1 Predict = 1
Actual = 0 Predict = 0
Actual = 0 Predict = 0
Actual = 1 Predict = 1
```

3. Import `accuracy_score`, `confusion_matrix`, `classification_report` from `sklearn.metrics`

Print the training accuracy, confusion matrix and classification report.

```
Training Accuracy: 1.0
Confusion Matrix:
[[781  0  0]
 [ 0 435  0]
 [ 0  0 464]]
Classification Report (Training):
```

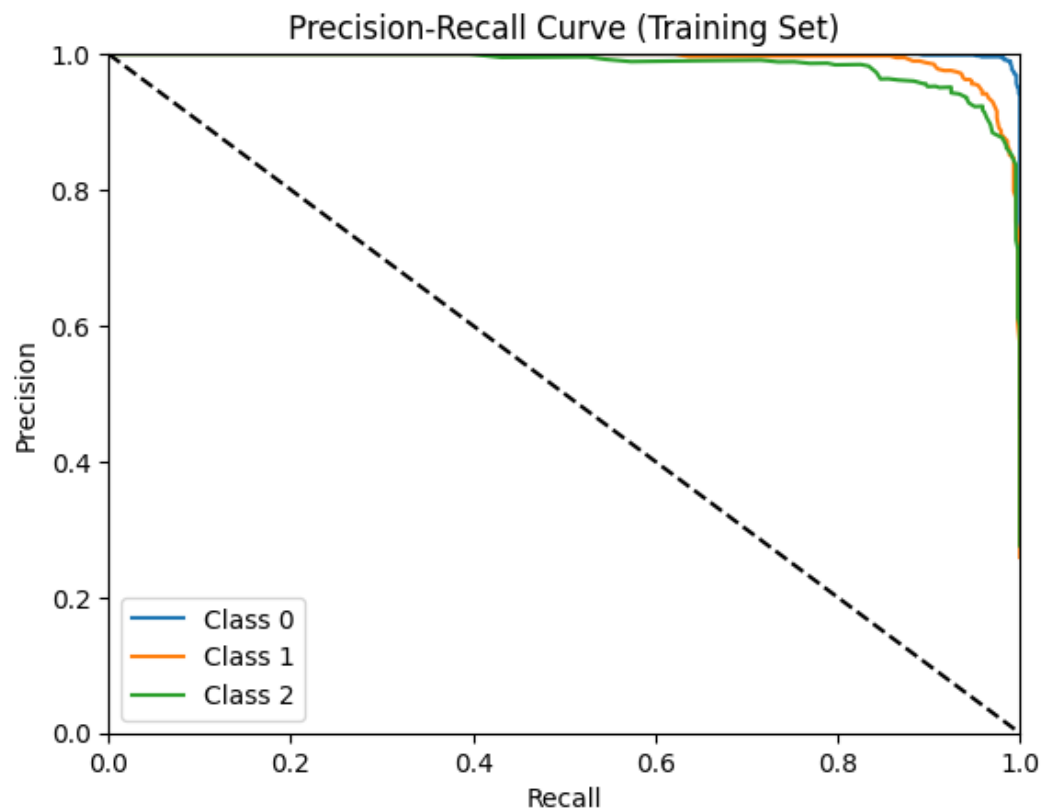
	precision	recall	f1-score	support
0	1.00	1.00	1.00	781
1	1.00	1.00	1.00	435
2	1.00	1.00	1.00	464
accuracy			1.00	1680
macro avg	1.00	1.00	1.00	1680
weighted avg	1.00	1.00	1.00	1680

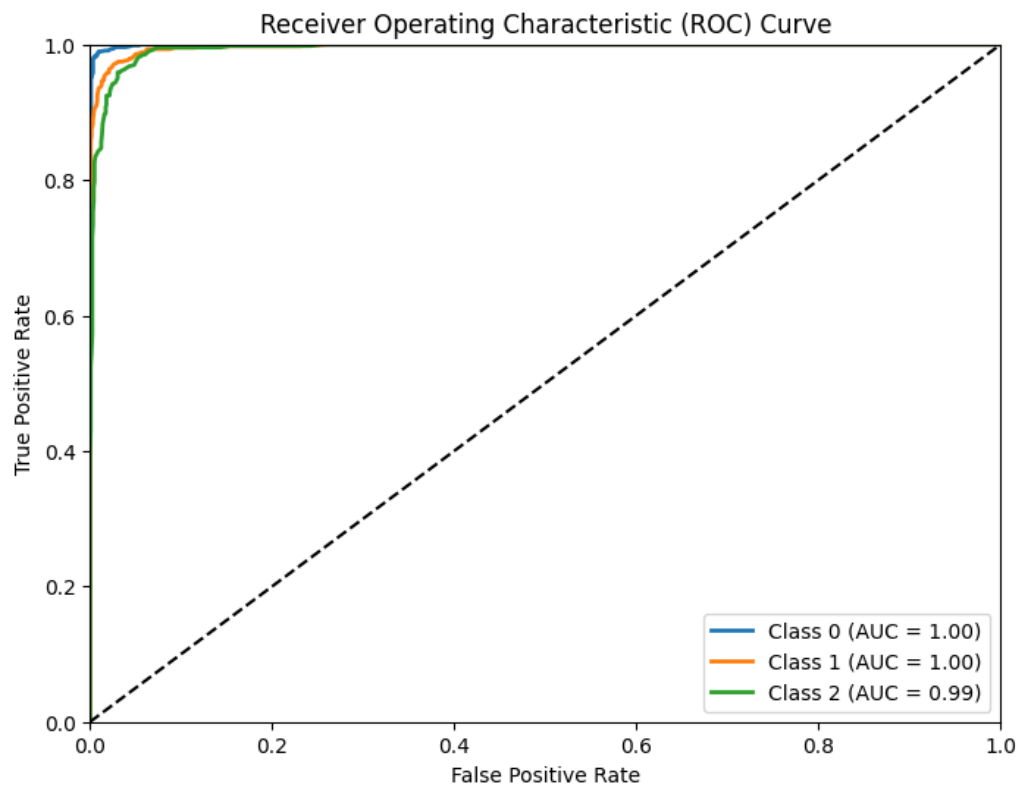
The accuracy of the training model is 100%. All three classes were predicted correctly, with no false positives or false negatives for either of them.

4. Next, the model is put through cross validation. From `sklearn.model_selection`, `cross_val_predict` is imported. I set it to cross validate 5 folds, and the following are the precision score, recall score, and F1 score of the cross validated model.

```
Precision Score: 0.9649525125912913  
Recall Score: 0.9648809523809524  
F1 Score: 0.9649010801899581
```

5. Plot the precision-recall curve and the ROC curve for the training set





6. Import `roc_auc_score` from `sklearn.metrics` and `label_binarize` from `sklearn.preprocessing`. The following are the area under the curve for all three classes. Class 0 has the highest area among the three, but all are above 99%.

```
Class 0 AUC = 0.9994
Class 1 AUC = 0.9967
Class 2 AUC = 0.9940
Macro-average AUC = 0.9967
```

2.7 Model tuning and testing

1. Import GridSearchCV from sklearn.model_selection.

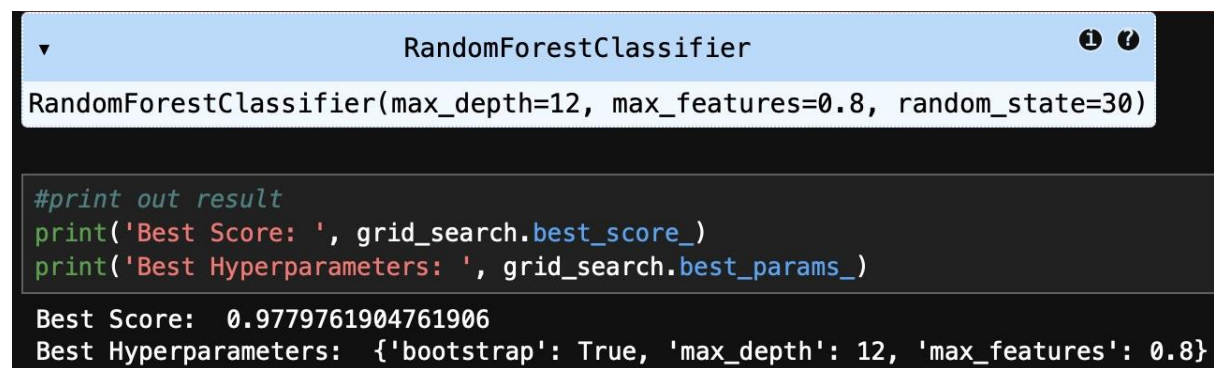
I set the following parameter grid:

Bootstrap: True, False

Max_depth: 3, 6, 9, 12, 15

Max_features: 0.2, 0.5, 0.8, 1.0

Using floats in max_features, it chooses a fraction of the total number of features.



```
▼ RandomForestClassifier ⓘ ⓘ
RandomForestClassifier(max_depth=12, max_features=0.8, random_state=30)

#print out result
print('Best Score: ', grid_search.best_score_)
print('Best Hyperparameters: ', grid_search.best_params_)

Best Score: 0.9779761904761906
Best Hyperparameters: {'bootstrap': True, 'max_depth': 12, 'max_features': 0.8}
```

After the grid search, the random forest classifier settles with the above parameters, and has a best score of 97.8%.

2. I then printed the feature importance list and found that Weight(kg) is the biggest contributing feature, followed by Height(cm). The other features contribute less than 3%.

```
[(np.float64(0.7373736582752954), 'Weight(kg)'),
 (np.float64(0.14884436767780854), 'Height(cm)'),
 (np.float64(0.02662244938324064), 'Family_history'),
 (np.float64(0.02623151356403432), 'Snack'),
 (np.float64(0.014124005359595515), 'Age'),
 (np.float64(0.008610468433203969), 'Gender'),
 (np.float64(0.005532334660635027), 'Transportation'),
 (np.float64(0.005506762105655892), 'Alcohol'),
 (np.float64(0.005231440811863496), 'Income'),
 (np.float64(0.004867131799471183), 'Water_intake(L)'),
 (np.float64(0.0036145584485092444), 'Exercise'),
 (np.float64(0.003249263785950691), 'TV'),
 (np.float64(0.0024133145306123667), 'Junk_food'),
 (np.float64(0.00238733164277512), 'Meals_day'),
 (np.float64(0.0022590173954021237), 'Smoking'),
 (np.float64(0.0017732811466118087), 'Vege_day'),
 (np.float64(0.0013591009793347656), 'Discipline')]
```

3. After getting our model ready for the test set, we predict y using x_test. The following is the result of the predictions.

```
y_pred_test = best_model_rf.predict(x_test)
y_pred_test_cv = cross_val_predict(best_model_rf, x_test, y_test, cv=5)
y_pred_test_cv
```

```
array([1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 2, 0, 0, 1, 1, 2, 0, 1, 2, 2, 0,
       2, 1, 2, 1, 2, 1, 2, 0, 1, 2, 2, 1, 1, 1, 0, 2, 1, 1, 1, 2, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 2, 0, 0, 1, 1, 2, 2, 0, 2, 0, 1, 2, 0, 2, 0,
       0, 1, 2, 0, 2, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 1, 2, 2, 0,
       2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 1, 1, 1, 1, 0, 0, 2, 0, 1, 2, 2,
       0, 1, 0, 2, 0, 1, 1, 2, 2, 1, 0, 0, 1, 0, 0, 2, 1, 0, 2, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 2, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 2, 1, 0, 2, 0, 1, 2, 1, 0, 2, 1, 1, 0, 0, 2, 2, 2, 0,
       0, 1, 0, 0, 0, 1, 1, 0, 2, 0, 2, 2, 0, 2, 2, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 2, 0, 2, 1, 0, 1, 2, 0, 2, 0, 0, 2, 1, 0, 0, 1, 0, 0, 0, 0,
       2, 0, 0, 2, 0, 2, 2, 2, 0, 2, 2, 0, 1, 2, 1, 0, 0, 1, 2, 0, 1, 0,
       2, 2, 0, 0, 2, 1, 2, 1, 1, 0, 0, 0, 2, 2, 0, 2, 0, 1, 1, 2, 2, 0,
       2, 0, 0, 0, 2, 1, 0, 1, 0, 1, 1, 0, 2, 0, 2, 0, 1, 1, 2, 0, 2, 0,
       0, 0, 0, 0, 2, 0, 2, 2, 1, 1, 2, 0, 1, 1, 1, 2, 2, 2, 0, 0, 0, 0,
       1, 2, 2, 0, 1, 1, 2, 0, 1, 2, 1, 2, 0, 0, 2, 0, 2, 1, 1, 0, 0, 2,
       1, 0, 0, 0, 1, 0, 2, 2, 2, 1, 0, 2, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
       0, 2, 1, 0, 1, 0, 0, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 2, 1, 1, 1,
       0, 2, 0, 2, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 2, 0, 0, 2, 1, 1, 1,
       2, 0, 0, 1, 0, 1, 0, 2, 0, 0, 2, 0, 0, 0, 1, 0, 2, 0, 0, 2, 0, 0,
       2, 0])
```

4. I printed the training accuracy, confusion matrix and classification report for the test set.

The accuracy is 96.4%

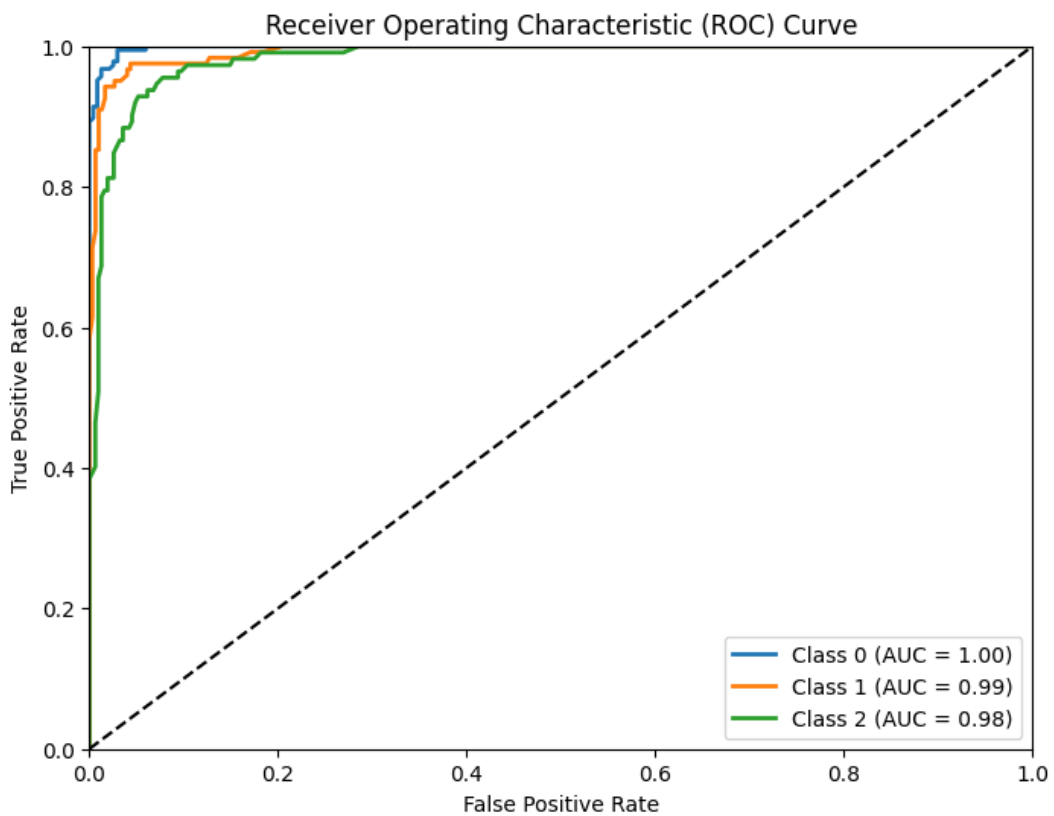
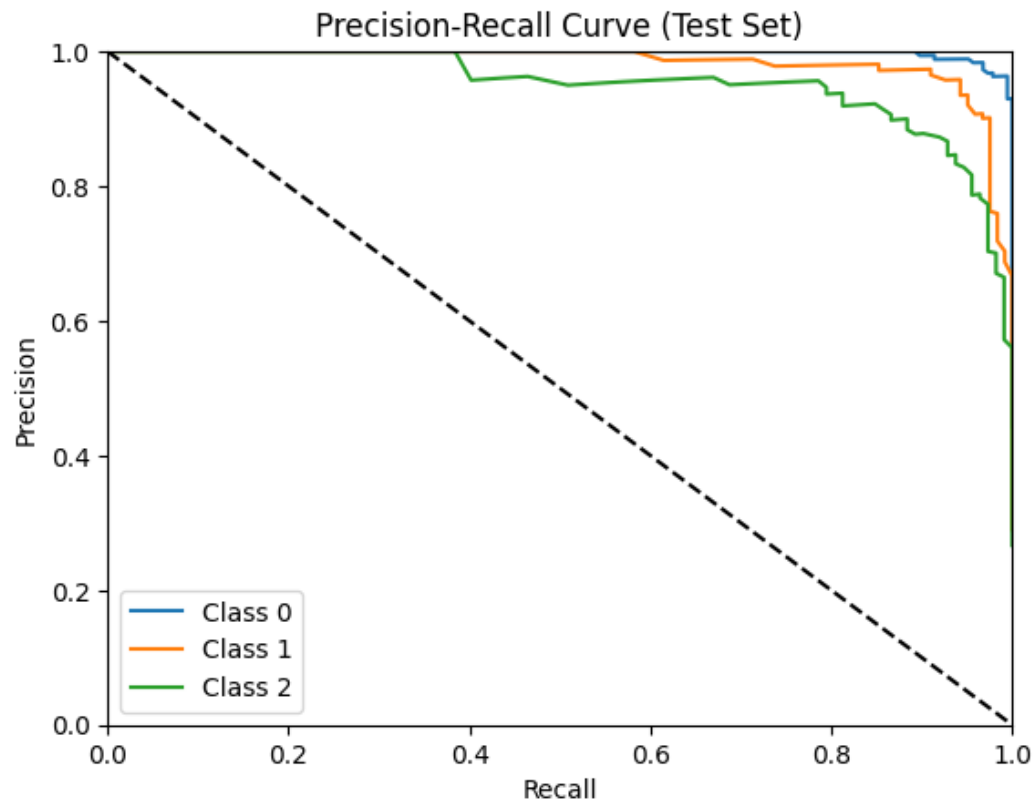
```
Training Accuracy: 0.9642857142857143
Confusion Matrix:
[[184  0  2]
 [ 0 113  9]
 [ 3  1 108]]
Classification Report (Test):
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	186
1	0.99	0.93	0.96	122
2	0.91	0.96	0.94	112
accuracy			0.96	420
macro avg	0.96	0.96	0.96	420
weighted avg	0.97	0.96	0.96	420

5. Print the precision score, recall score, and F1 score.

```
Precision Score: 0.9656974432516228
Recall Score: 0.9404761904761905
F1 Score: 0.9403329046636295
```

6. Plot the precision-recall curve and the ROC curve for the test set.



7. Print the area under the curve for all three classes. The areas for the test set are less than the areas for the training set. The prediction accuracy drops off more for Class 1 and even more for Class 2.

```
Class 0 AUC = 0.9983  
Class 1 AUC = 0.9924  
Class 2 AUC = 0.9825  
Macro-average AUC = 0.9911
```

Chapter 3: Result and Discussion

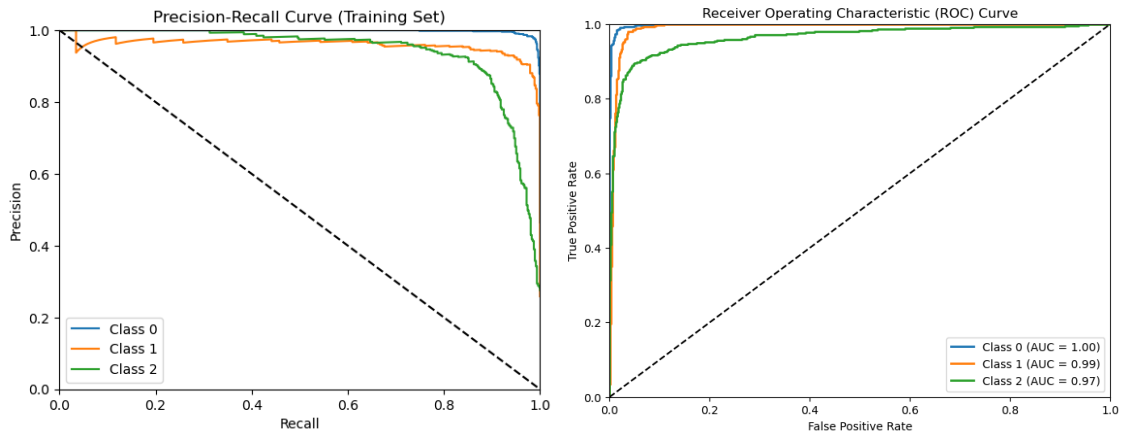
The following table shows summarizes the various scores and accuracy of the Logistic Regression, K Neighbors Classifier and Random Forest Classifier models. The cell with the highest value in each row is highlighted yellow.

		LR	KNN	RFC
Training	Accuracy	0.9560	0.8958	1.0000
	Precision Score	0.9443	0.8258	0.9650
	Recall Score	0.9446	0.8250	0.9649
	F1 Score	0.9439	0.8224	0.9649
	Find-tuned Accuracy	0.9929	0.8911	0.9780
Testing	Accuracy	0.9595	0.8738	0.9643
	Precision Score	0.9182	0.7762	0.9402
	Recall Score	0.9190	0.7786	0.9405
	F1 Score	0.9184	0.7750	0.9403
	Class 0 AUC	0.9995	0.9513	0.9983
	Class 1 AUC	0.9931	0.9382	0.9924
	Class 2 AUC	0.7836	0.8312	0.9825
	Macro-average AUC	0.9254	0.9069	0.9911

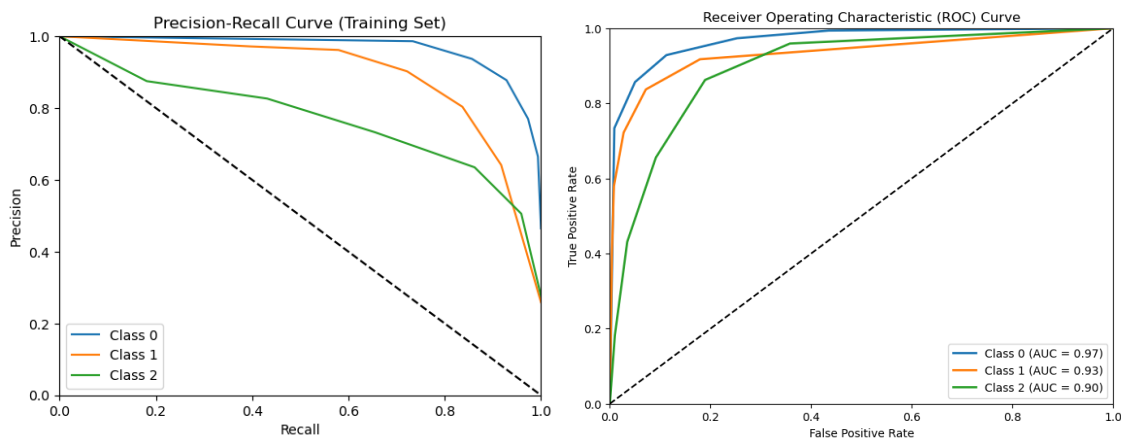
We can see that the RFC model has perfect accuracy for the training model. It is very overfitted. After cross validation of the 3 models, RFC still emerges with the highest precision score, recall score and F1 score, with LR being very close behind. KNN also does quite well, with an accuracy close to 90% and the three scores being over 80%.

Before using the models to predict the test set, all three models went through fine-tuning. The LR model used randomized search while the KNN and RFC models used grid search. The results of the fine-tuning showed a great increase in accuracy for the LR model, that resulted in a 99.3% accuracy.

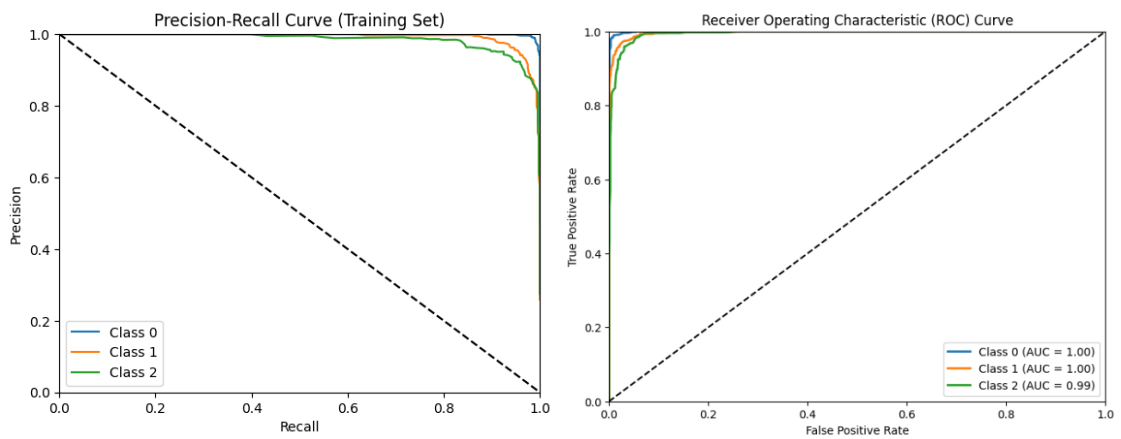
When predicting the test set, the accuracy of LR dropped back down to being slightly higher than the initial training accuracy. The accuracy of the other two models also dropped a little from their fine-tuned accuracy. After all, they are predicting new data. The precision, recall and F1 scores of RFC are once again the highest, slightly better than LR.



LR Precision-Recall Curve and ROC Curve (Training)



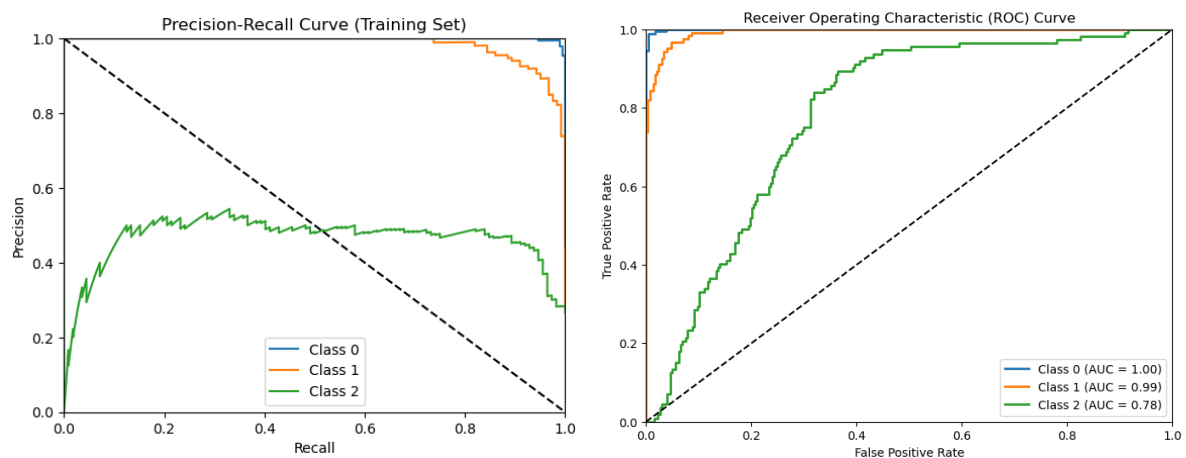
KNN Precision-Recall Curve and ROC Curve (Training)



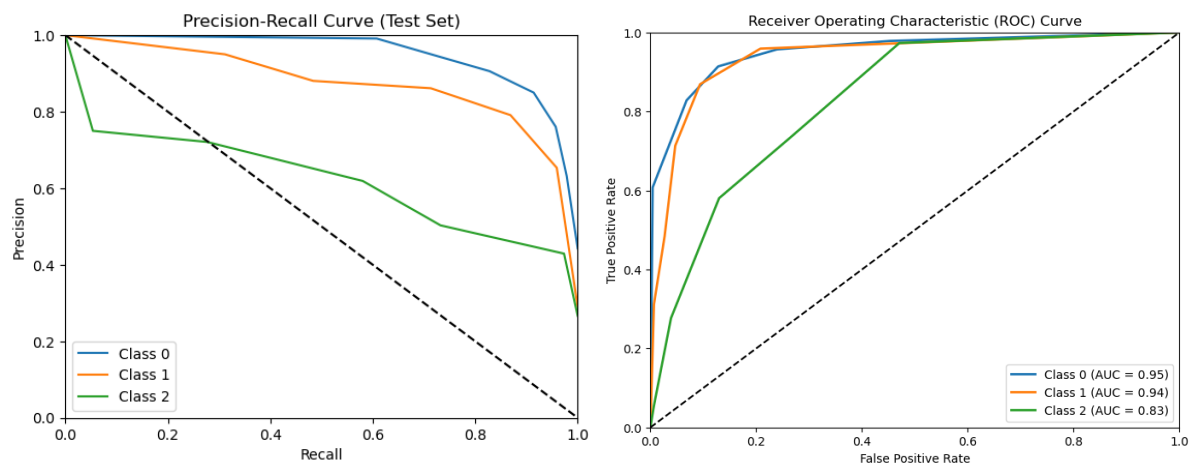
RF Precision-Recall Curve and ROC Curve (Training)

Looking at the precision-recall curves of the three models for the training set, we can see that it is generally harder to accurately predict Class 2 without false positives and false negatives, compared to Class 1. And it's the same for Class 1 and Class 0. For the three models, recall can generally increase without sacrificing much precision. However, precision is unable to increase as much before rapidly losing recall. The ROC curves are quite close to the upper left corner, showing a high accuracy of the models.

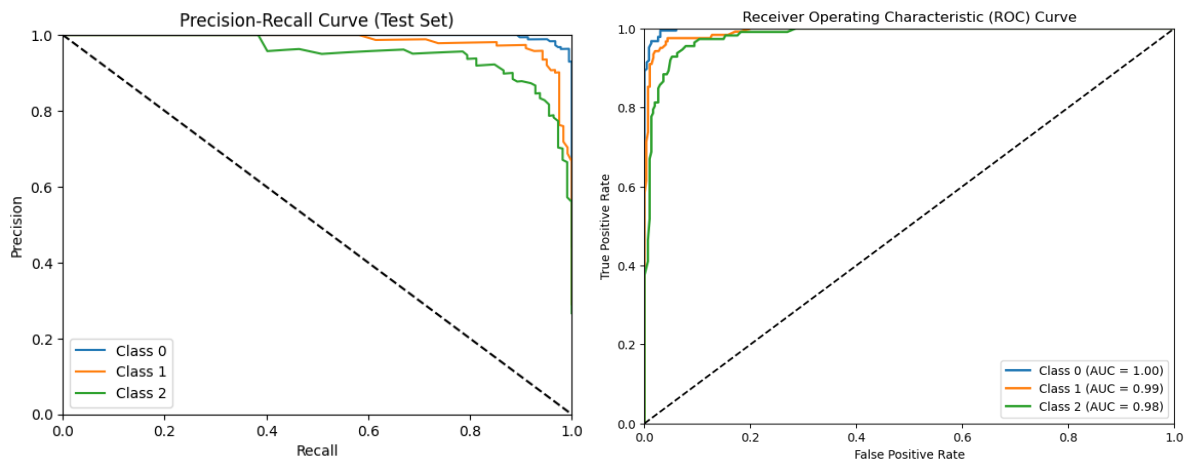
Next, we will look at the precision-recall curves and roc curves for the test set.



LR Precision-Recall Curve and ROC Curve (Testing)



KNN Precision-Recall Curve and ROC Curve (Testing)



RF Precision-Recall Curve and ROC Curve (Testing)

It is much clearer that the models struggle to accurately predict Class 2. It is difficult to have moderately high precision without sacrificing a lot of recall. As listed in the summary table at the start of this section, the area under the curve of Class 0 and Class 1 are the highest for LR, while the AUC of Class 2 is highest for RFC.

KNN is simple to understand and implement, and has a consistent accuracy throughout training, fine-tuning and testing. LR is also easy to understand and implement, while also being computationally efficient. On the other hand, RFC and KNN are more computationally expensive and require more memory. The dataset we worked on is a rather small dataset with 2100 entries and 18 features. It is still reasonable to use either of the three models, but if the dataset is a lot larger, LR will be faster to train and predict, due to its linear nature. This linear nature may also be a disadvantage of the model depending on the situation, as it can handle limited complexity, underperforming when the decision boundary is complex. RFC can handle this complexity due to having many decision trees. LR predicts Class 0 and Class 1 well, suggesting that the data for the two classes is not complex.

```
[(np.float64(0.7373736582752954), 'Weight(kg)'),  
(np.float64(0.14884436767780854), 'Height(cm)'),  
(np.float64(0.02662244938324064), 'Family_history'),  
(np.float64(0.02623151356403432), 'Snack'),  
(np.float64(0.014124005359595515), 'Age'),  
(np.float64(0.008610468433203969), 'Gender'),  
(np.float64(0.005532334660635027), 'Transportation'),  
(np.float64(0.005506762105655892), 'Alcohol'),  
(np.float64(0.005231440811863496), 'Income'),  
(np.float64(0.004867131799471183), 'Water_intake(L)'),  
(np.float64(0.0036145584485092444), 'Exercise'),  
(np.float64(0.003249263785950691), 'TV'),  
(np.float64(0.0024133145306123667), 'Junk_food'),  
(np.float64(0.00238733164277512), 'Meals_day'),  
(np.float64(0.0022590173954021237), 'Smoking'),  
(np.float64(0.0017732811466118087), 'Vege_day'),  
(np.float64(0.0013591009793347656), 'Discipline')]
```

Looking at the feature importance above, we can conclude that Weight(kg) is the most important feature for prediction, followed by Height(cm). The rest of the features have a lot less impact in the predictions.

To sum up the analysis, all three models do a good job predicting the data. LR does the best job predicting Class 0 and Class 1, and RFC is the best for predicting Class 2.

Chapter 4: Conclusion

After analysis, we can conclude that the Random Forest Model is the best algorithm by comparing with other two algorithms due to well performance analysis in Training Set.

Random Forest Classifier has the highest validation accuracy score and compared with Logistic Regression, 95.60% and KNeighborsClassifier, 89.58%. Random Forest Classifier has the highest precision score, 96.50% by comparing with Logistic Regression, 94.43% KNeighborsClassifier, 82.58%. Random Forest Classifier has the highest recall score, 96.49% by comparing with Logistic Regression, 94.46% and KNeighborsClassifier, 82.50%. Random Forest Classifier has the highest F1 score, 96.49% by comparing with Logistic Regression, 94.39% and KNeighborsClassifier, 82.24%.

Furthermore, we also can conclude that the Random Forest Model is the best algorithm by comparing with other two algorithms due to well performance analysis in Test Set.

Random Forest Classifier has the highest validation accuracy score, 96.43% compared with Logistic Regression, 95.95% and KNeighborsClassifier, 87.38%. Random Forest Classifier has the highest precision score, 94.02% by comparing with Logistic Regression, 91.82% KNeighborsClassifier, 77.62%. Random Forest Classifier has the highest recall score, 94.05% by comparing with Logistic Regression, 91.90% and KNeighborsClassifier, 77.86%. Random Forest Classifier has the highest F1 score, 94.03% by comparing with Logistic Regression, 91.84% and KNeighborsClassifier, 77.50%.

Based on our analysis, we found that the Random Forest Classifier is the most suitable model for the dataset. Additionally, we observed that different datasets paired with different training algorithms result in varying levels of accuracy.