# CSC3206 Artificial Intelligence

# Assignment 2

Grouped Submission: Group 1

Semester: September 2024

Lecturer: Ms Amanda Song Cheen

1. Toh Kar Ming – 21091137
2. Tung Qi Yong – 22022750
3. Syehrran A/L Arulsamy – 22000608
4. Ang Yen Ping – 22023170

# Table of Contents

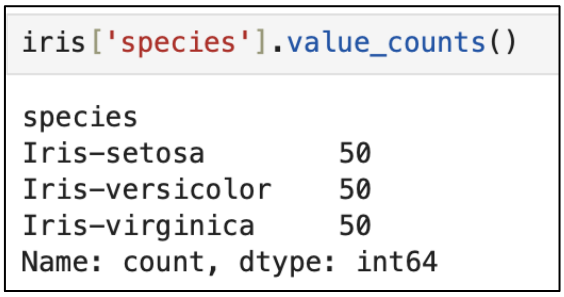
# 1.0 Introduction

## 1.1 Source of The Dataset

The chosen dataset, Iris Flower Dataset, is a multivariate dataset that collected by Ronald Fisher for his paper published in 1936. The dataset used for this assignment is taken from Kaggle and is uploaded by a user, MathNerd (2018).

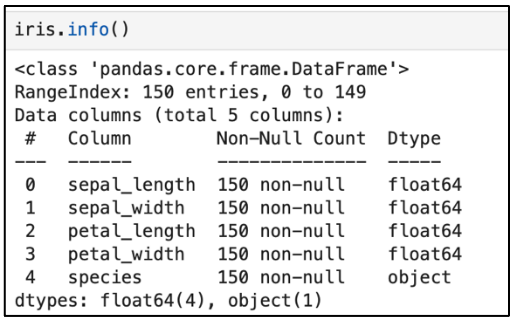
## 1.2 Characteristics of the Dataset

This dataset consists of 5 features, the sepal_length, sepal_width, petal_length, petal_width and species. In figure 1, there are only 3 species of data are included in this dataset and 50 records for each, which are the Iris Setosa, Iris Virginica, and Iris Versicolor. As shown in figure 2, the datatype of species is object and the rest are floating numbers, this indicates the need of encoding species value during Exploratory Data Analysis.



```
iris['species'].value_counts()

species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

*Figure 1: Dataset Information – Amount of records*



```
iris.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
```

*Figure 2: Dataset Information – Data Type*

### 1.3 Purpose of Machine Learning Application on The Dataset

As Iris dataset is a labelled dataset, it could be seen as classification and regression problem which provide higher flexibility for algorithms selection. Total of 3 machine learning techniques are chosen for this dataset, including Decision Tree, Logistic Regression and K-Nearest Neighbours.

The application of Machine Learning techniques is expected to provide classification models that could classify species of the Iris based on input value of the rest of 4 features.

## 2.0 Justification of the Machine Learning Techniques Chosen

### 2.1 Logistic Regression

Logistic Regression is an unsupervised machine learning algorithm used for classification tasks to predict a categorical outcome. It was chosen for the Iris dataset because of its capability to handle multiclass classification problems. Though usually used in binary classification problems, Logistic Regression can be used in multiclass classification problems such as seen in the Iris dataset by extending its binary form. It usually uses the One-vs-Rest (OvR) approach to separate binary classifiers for each class, treating one class as positive and others as negative, then combining their results to make predictions. To make this clearer, the iris dataset will go through three evaluations as follows Case 1: Iris-Setosa or Iris-Virginica/Versicolor, Case 2: Iris-Virginica or Iris-Setosa/Versicolor or Case 3: Iris-Versicolor or Iris-Setosa/Virginica. After evaluating the three cases, the results will be combined and evaluated to see which class it belongs in.

Additionally, Logistic Regression is also suitable for Iris dataset because the Iris dataset only has 150 samples, with 50 samples for each class. Logistic Regression performs well on small datasets because it is computationally efficient and does not require extensive hyperparameter tuning unlike other complex models. This makes it an excellent choice for small-scale problems like this.

### 2.3 Decision Tree

Decision Tree is an unsupervised machine learning algorithm used for classification and regression tasks. The basic concept is to split the data into branches based on decision, the node representing data’s attribute and the prediction shown by the leaf node. The Classification Decision Tree will be implemented for the Iris dataset as the target variable for the dataset is categorical data. On the other hand, Decision Tree are performing well and efficient when applying to the small datasets like Irish. Which will be able to train and produce the result in a relatively short time. Then, the result of the Decision Tree is easy to be visualized and easy to understand as the straightforward visualization structure allow people to observe every single decision made by the Decision Tree.

In short, Decision Tree will be a good choice to be applied to the Iris dataset based on the strength of efficiency, visualization, and able to handle the categorical dataset like Iris. However, there are some issues and limitations need to be solved when implementing Decision Tree algorithm such as overfitting. Overfitting occurs when the structure of the tree become more complex and the trained model capturing the noise and minor details. This might cause the accuracy of the model is high in train dataset but low when the new data come in.

## 2.3 K-Nearest Neighbours

K-Nearest Neighbours (KNN) was chosen for its simplicity and effectiveness in handling small datasets like the Iris dataset. As a non-parametric algorithm, KNN makes no assumptions about data distribution, making it flexible and suitable for datasets with less-defined patterns. It classifies data points based on the majority class of their nearest neighbours, determined by a distance metric such as Euclidean distance by default. This intuitive approach is particularly effective for multiclass problems like the Iris dataset, which contains three species (Iris-Setosa, Iris-Versicolor, Iris-Virginica).

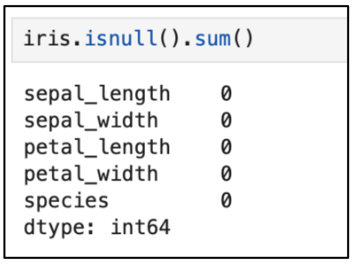
KNN is ideal for small datasets like the iris dataset since it does not involve traditional training, minimizing the risk of overfitting. The Iris dataset also has classes that are relatively well-separated in the feature space which is beneficial for KNN as it relies on distance metrics to classify data points. For instance, Iris-Setosa is said to be linearly separable from the other two species in terms of the feature values, making it easy for KNN to classify Iris-Setosa accurately. However, its performance can be sensitive to the choice of K-value and feature scaling. To address these challenges, parameter tuning is conducted to find the optimal K-value, and data normalization ensures that all features contribute equally to distance calculations. The balanced nature of the Iris dataset, combined with its numerical features, aligns well with KNN's distance-based decisions, making it a robust choice for this classification task.

## 3.0 Exploratory Data Analysis (EDA)

The EDA is a crucial preliminary work that a data scientist must conduct before modelling, it is said to occupy 70% of data scientist effort in a data science project. The objective of EDA is to produce cleaned dataset and provide insight that can help identify appropriate modelling technique. To achieve the objective, techniques such as completeness checking, data visualisation, correlation analysis, outliers' detection and descriptive summary calculation are used in the process. Pandas, Matplotlib, Seaborn and Scikit-learn are few of the python libraries that provide assistance to EDA process.

### 3.1 Data Cleaning

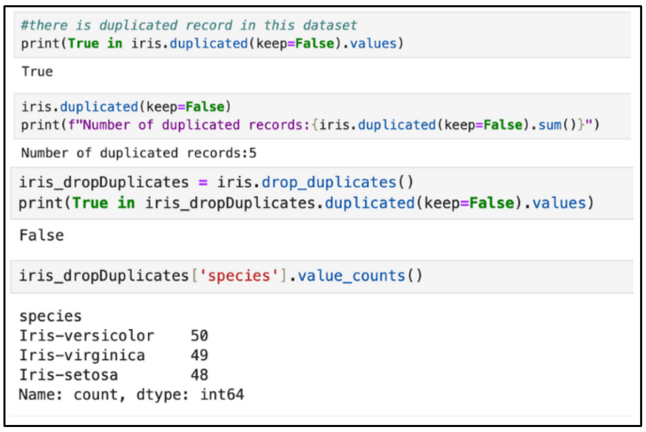
Data cleaning is part of EDA that aims to check completeness, duplication and formatting of data, corresponded action will be taken if dataset is found to be problematic.
Iris dataset is relatively clean, there is no null value as shown in figure 3, and all records are formatted other than few duplicated records.



```
iris.isnull().sum()

sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

*Figure 3: Null Value Checking*

Total of five duplication records are found, 3 are dropped and 2 kept as original copies. The dataset is now with 147 records, categorised records are shown in figure 5.



```
#there is duplicated record in this dataset
print(True in iris.duplicated(keep=False).values)

True

iris.duplicated(keep=False)
print(f"Number of duplicated records:{iris.duplicated(keep=False).sum()}")

Number of duplicated records:5

iris_dropDuplicates = iris.drop_duplicates()
print(True in iris_dropDuplicates.duplicated(keep=False).values)

False

iris_dropDuplicates['species'].value_counts()

species
Iris-versicolor    50
Iris-virginica     49
Iris-setosa        48
Name: count, dtype: int64
```

*Figure 4: Duplicated Records*

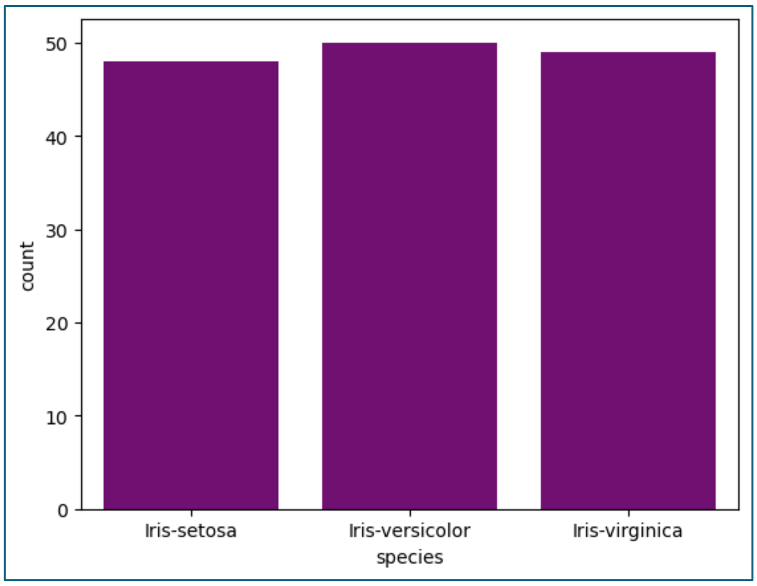
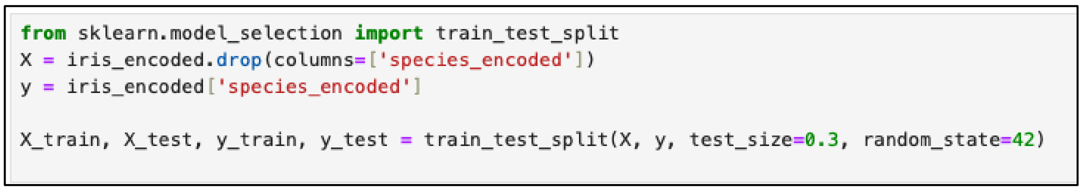


*Figure 5: Number of records after duplication has dropped*

### 3.2 Data Splitting

The dataset was split following standard 70/30 ratio, with 70% allocated for training and 30% allocated for testing. The splitting process contain randomness while choosing the cutoff point, which means the data is not divided sequentially but instead it is randomly selected as long as the result has reached the said 70/30 ratio. To ensure the splitting output remain consistent for later usage of the training and testing dataset, random_state is set as 42 to ensure the consistency of dataset. The 70/30 ratio is determined for Iris dataset as it is a small dataset and the allocate 30% data for testing would provide good model evaluation.

As shown in Figure 6, ‘species’ as target which assigned to y, the rest of features as training features which assigned to X.



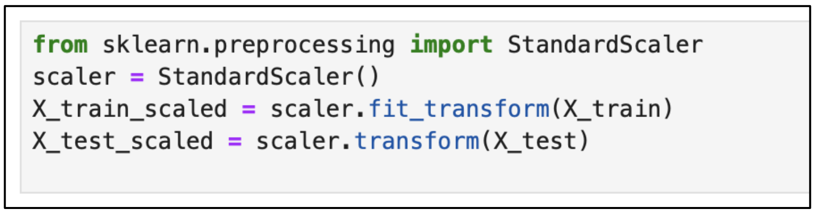
```
from sklearn.model_selection import train_test_split
X = iris_encoded.drop(columns=['species_encoded'])
y = iris_encoded['species_encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

*Figure 6: Data Splitting Process*

### 3.3 Data Normalization

Normalisation, also known as Data Scaling, is a process of standardising data value into specific range of 0 to 1. This eliminates the dominant feature that could lead to bias prediction and can improve model performance, especially the distance-based algorithm. Normalisation methods including Min-Max Normalisation, Standardisation, Scaling to Unit Length and Mean-Centred. To prevent data leakage, data normalisation is suggested to conduct after data splitting, this assignment follow the suggested sequence.

The Standardisation is chosen to normalise training and testing features data in Iris dataset, which are X_train and X_test data, Standardisation method is provided by preprocessing module in scikit-learn library. The normalisation process is shown in Figure 7.

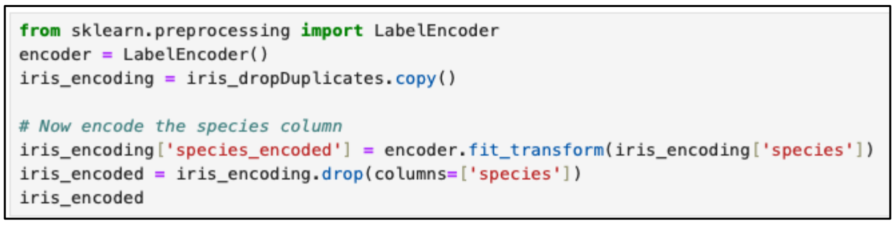


```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

*Figure 7: Normalisation Process*

### 3.4 Data Encoding

Encoding is a process of converting non-numerical feature value into numeric value so that machine learning algorithm can understand. In Iris dataset, the species feature must be transformed to numerical value for the ease of later work. The Iris-Setosa, Iris-Versicolor, and Iris-Virginica are transformed into 0, 1, 2 respectively. Figure 8 show the encoding process, LabelEncoder from preprocessing module is imported to assist the encoding process.



```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
iris_encoding = iris_dropDuplicates.copy()

# Now encode the species column
iris_encoding['species_encoded'] = encoder.fit_transform(iris_encoding['species'])
iris_encoded = iris_encoding.drop(columns=['species'])
iris_encoded
```

*Figure 8: Encoding Process*

## 3.5 Resolving Outliers

To discover outlier in the dataset, the box plot is used for visualisation, as shown in Figure 9. Each boxplot has feature value as x-axis and species as y-axis.

Boxplots visualise the distribution of data records by presenting them as box and edges. Bottom box is data records of 25% to 50%, upper box is data records of 50% to 75%, edges are the data records that outside of these 2 ranges.

The boxplot effectively identifies outliers in the dataset by presenting value that out of range as hollow circles. From boxplots in Figure 9, there are total of 7 outliers in selected dataset. With outlier in sepal width record of Iris-Sentosa has the greatest distance from the minimum value, and 2 outliers detected in petal width record of Iris-Sentosa.

For this dataset, the outliers are decided to not be dropped as data records are limited. In addition, the detected outliers cannot be said as an error because there is a possibility to measured object to have such width and length, also the outliers occupied only 4.7% of entire dataset, which means existence of outliers will not impact model performance heavily.

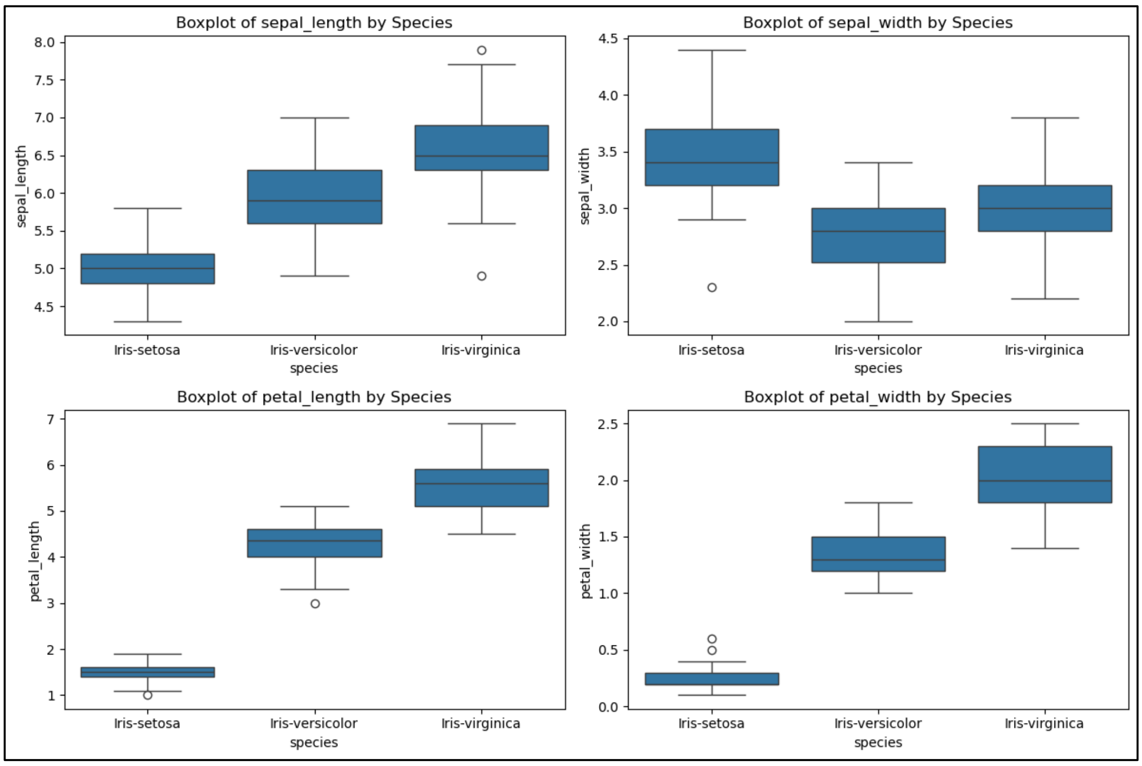


*Figure 9: Boxplot of Iris Dataset*

## 3.5 Data Pattern and Correlation Analysis

To understand dataset better, scatter plot and heatmap are used. As shown in figure 10, the Iris-Setosa can be easily differentiated from the other two species, but the Iris-Versicolor and Iris-Virginica overlap at the border. From the overlapping of nodes, it has said that there is high similarity between Iris-Versicolor and Iris-Virginica. However, Iris-Virginia has the longest Petals, Sepal Length, and Petal Width, whereas Iris-Sentosa has the widest sepal width.

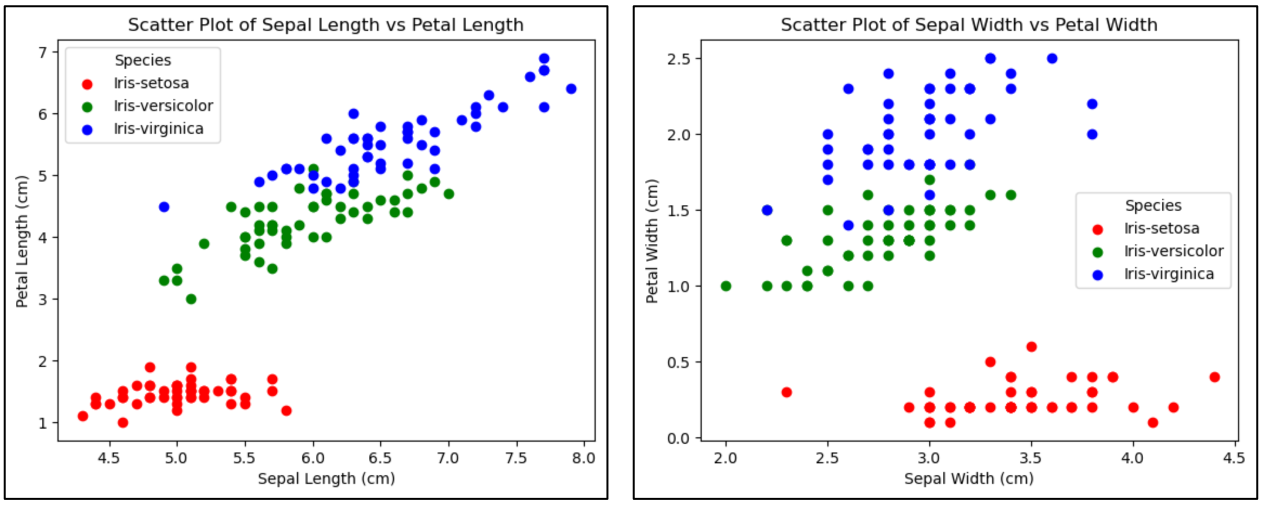
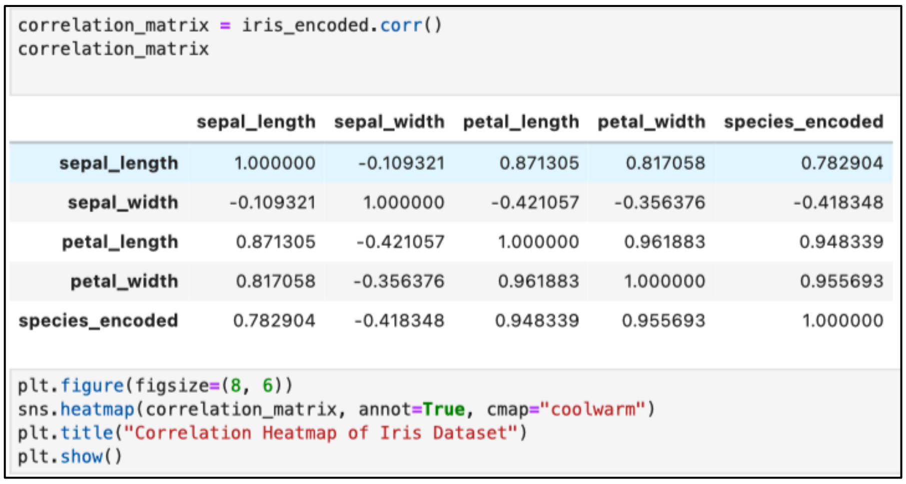


*Figure 10: Scatter plots of Iris Dataset*

To understand the correlation among features, correlation matrix is used as calculation method and heatmap is used to plot out the result, process shown in figure 11. From the heatmap which shown in figure 12, in the species_encoded column, there is negative correlation between sepal_width and species, and petal_length and petal_width are high correlated with the species. This also indirectly implies the result we got from the scatter plots.



```
correlation_matrix = iris_encoded.corr()
correlation_matrix
```

| | sepal_length | sepal_width | petal_length | petal_width | species_encoded |
|---|---|---|---|---|---|
| **sepal_length** | 1.000000 | -0.109321 | 0.871305 | 0.817058 | 0.782904 |
| **sepal_width** | -0.109321 | 1.000000 | -0.421057 | -0.356376 | -0.418348 |
| **petal_length** | 0.871305 | -0.421057 | 1.000000 | 0.961883 | 0.948339 |
| **petal_width** | 0.817058 | -0.356376 | 0.961883 | 1.000000 | 0.955693 |
| **species_encoded** | 0.782904 | -0.418348 | 0.948339 | 0.955693 | 1.000000 |

```
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap of Iris Dataset")
plt.show()
```

*Figure 11: Correlation calculation and plotting process*

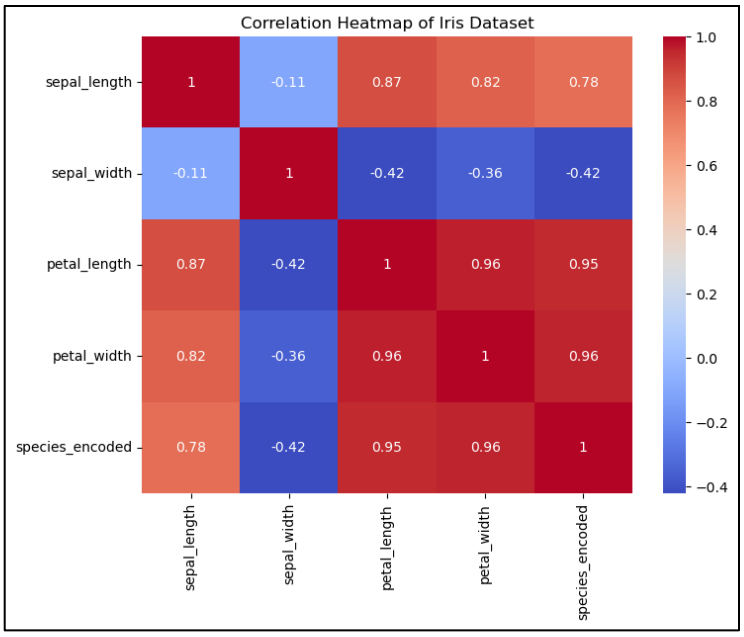


*Figure 12: Heatmap that shows correlation*

## 4.0 Application of Machine Learning Techniques

### 4.1 Workflow

Generally, the applications of several machine learning techniques followed the workflow as seen below. The data set acquiring and data pre-processing were carried out in section 3.0. In this however, the training of ML models and parameter tuning will be carried out.

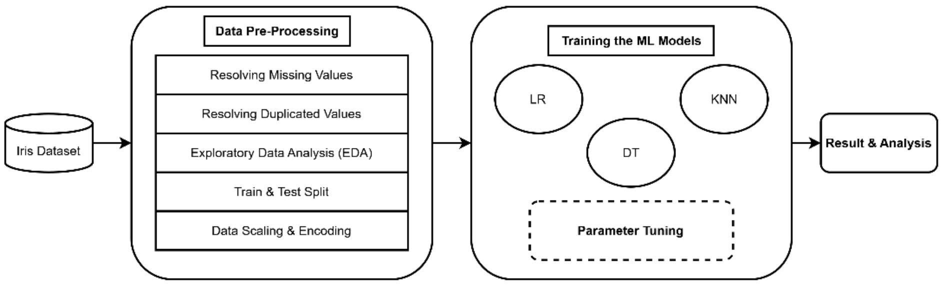
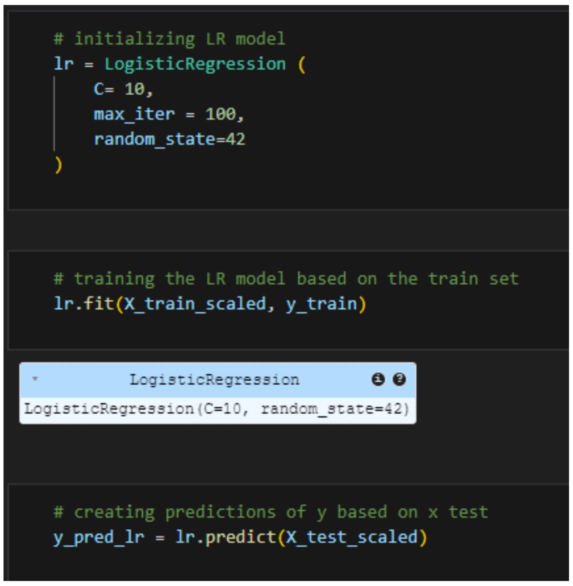


*Figure 13: Workflow of The Study*

The dataset was split before this into a set of inputs (X) and outputs (y). Then each of these sets were further split into training sets (X_train & y_train) and testing sets (X_test & y_test). Initially, the model initialized then, it'll be trained based on the training sets where it will learn based on input of X_train, the output will be y_train. Then, the trained model will be used to predict y values based on X_train, obtaining an output of y_pred. This predicted y results based on X_test, will then be compared with the actual y values from y_test to evaluate the respective machine learning technique's performance. Additionally, parameter tuning will also be carried out in order to improve each model's performance. The only thing that differs from each model implemented; would be the way parameter tuning was conducted.



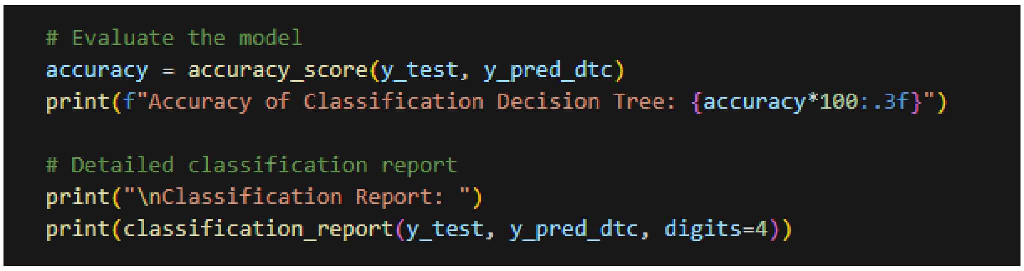
```
# initializing LR model
lr = LogisticRegression (
    C= 10,
    max_iter = 100,
    random_state=42
)
```

```
# training the LR model based on the train set
lr.fit(X_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(C=10, random_state=42)
```

```
# creating predictions of y based on x test
y_pred_lr = lr.predict(X_test_scaled)
```

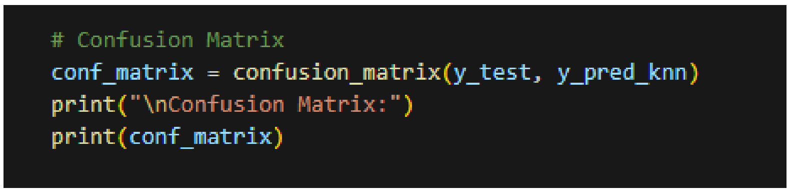
*Figure 13: Model Initialization, Training and Prediction for Logistic Regression*



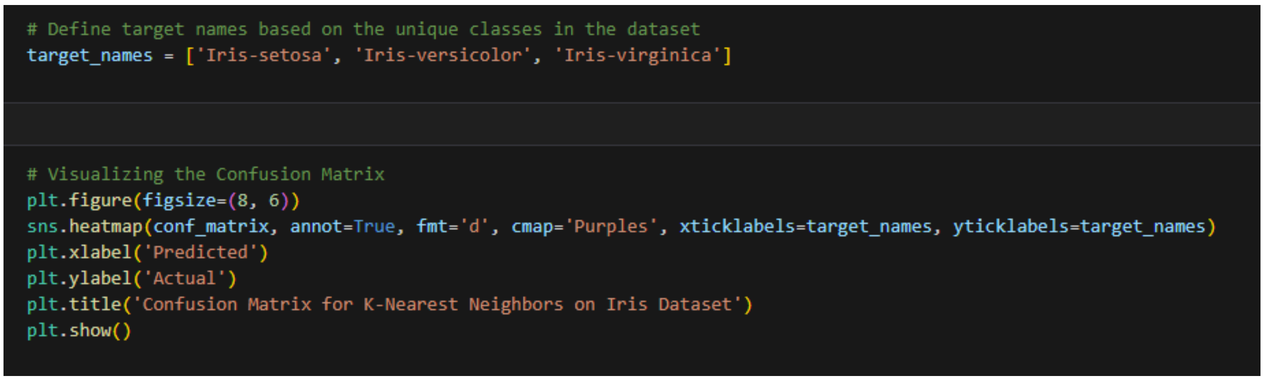
```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_dtc)
print(f"Accuracy of Classification Decision Tree: {accuracy*100:.3f}")

# Detailed classification report
print("\nClassification Report: ")
print(classification_report(y_test, y_pred_dtc, digits=4))
```

*Figure 14: Obtaining Accuracy, Precision, Recall and F1-Score of Decision Tree*

Additionally, the confusion matrix and Receiver Operating Characteristic - Area Under the Curve (ROC-AUC) Curve was obtained also for each model to further evaluate their performance. A confusion matrix summarized the performance of a classification model by showing the number of correct and incorrect predictions for each class. Specifically, it tells us the number of true positive, true negative, false positive and false negative predictions for the model. This is useful for spotting imbalances of specific weaknesses in classification. The ROC-AUC curve on the other hand, shows the trade-off between True Positive Rate and False Positive at varying threshold values. This helps to assess the model's ability to distinguish between classes. The area under the curve (AUC) value quantifies this performance where AUC of 1 would mean perfect classification and AUC of 0.5 means random guessing ended up occurring instead.



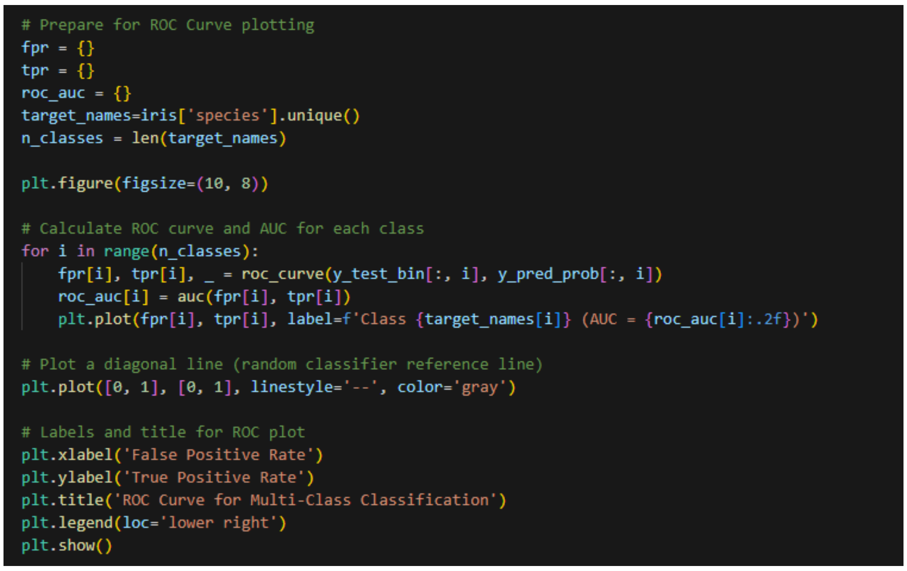
```python
# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_knn)
print("\nConfusion Matrix:")
print(conf_matrix)
```



```python
# Define target names based on the unique classes in the dataset
target_names = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

```python
# Visualizing the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Purples', xticklabels=target_names, yticklabels=target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for K-Nearest Neighbors on Iris Dataset')
plt.show()
```

*Figure 15: Confusion Matrix Carried Out for K-Nearest Neighbours*



```python
# Prepare for ROC Curve plotting
fpr = {}
tpr = {}
roc_auc = {}
target_names=iris['species'].unique()
n_classes = len(target_names)

plt.figure(figsize=(10, 8))

# Calculate ROC curve and AUC for each class
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    plt.plot(fpr[i], tpr[i], label=f'Class {target_names[i]} (AUC = {roc_auc[i]:.2f})')

# Plot a diagonal line (random classifier reference line)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

# Labels and title for ROC plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class Classification')
plt.legend(loc='lower right')
plt.show()
```

*Figure 16: ROC-AUC Curve Plotting for K-Nearest Neighbours*

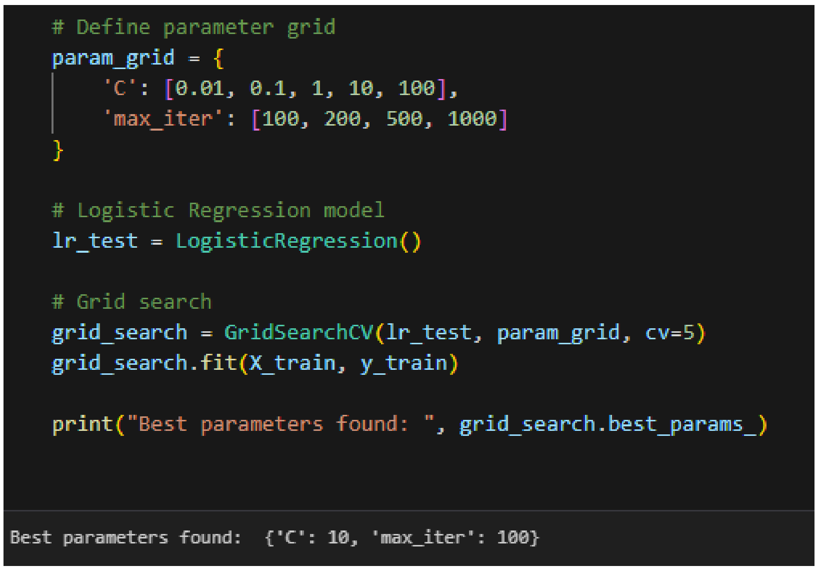
## 4.2 Parameter Tuning of Models

### 4.2.1 Logistic Regression Parameter Tuning

Parameter tuning on C and Max Iterations were carried out for Logistic Regression in order to improve the results of the performance evaluation as these parameters can influence how well the model learns from the data and make predictions. The C parameter control how much the model simplifies its decision-making process to avoid overfitting. A smaller C value forces the model to generalize more, reducing the chances of overfitting but possibly missing important patterns in the data. A larger C value on the other hand will allow the model to focus more on fitting the training data closely, which can improve the accuracy on the training set but may not work well on new data. Finding the right balance for C helps the model perform well on both training and unseen data.

Max iterations determine how many steps the model is allowed to take while it learns. If this value is too low, the model might not have enough time to fully learn the patterns in the data, resulting in warnings or poor performance. Increasing max iterations gives the model more time to train, which can lead to better results, especially with complex or larger datasets. However, setting it too high might increase the training time unnecessarily.

By adjusting the C and Max Iterations carefully, we can make the model better at understanding the data and predicting accurately without overcomplicating or oversimplifying its decisions. As seen below, for Logistic Regression, parameter tuning using grid search was done to determine optimal C and Max Iterations values. Grid search is a systematic way to find the best combination of parameter for a machine learning model by testing all possible combinations of specified parameter values to identify the one that gives it the best performance. The following figure depicts how grid search was used for Logistic Regression.



```
# Define parameter grid
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'max_iter': [100, 200, 500, 1000]
}

# Logistic Regression model
lr_test = LogisticRegression()

# Grid search
grid_search = GridSearchCV(lr_test, param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Best parameters found: ", grid_search.best_params_)
```

```
Best parameters found:  {'C': 10, 'max_iter': 100}
```

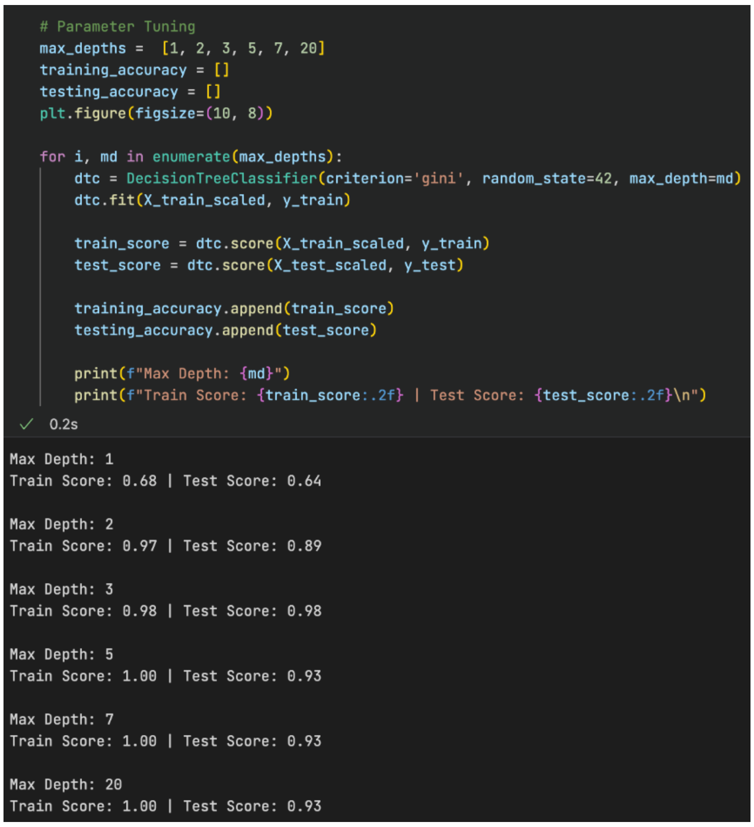
*Figure 15: Finding Optimal C and Max Iteration with Grid Search*

It was found after running the code block that the optimal C and Max Iteration value for Logistic Regression was 10 and 100 respectively. The model training, predicting and performance evaluation was rerun again for Logistic Regression as seen in section 4.1 but this time with the new parameters initialized in the model.

### 4.2.2 Decision Tree Parameter Tuning

The parameter tuning for Decision Tree will mainly focus on the max depth value of the tree, which aims to balance the performance and the accuracy of the model. The max depth value will directly affect the complexity of a decision tree as it will limit the depth of the prediction model when training the model. If the max depth value is low, the model might be led to a underfitting situation, which may not capture enough pattern in the training dataset and the accuracy of the data will be poor. On the other hands, if the max depth value is too high it might also lead to overfitting, which make the model fit too well to the pattern in the training dataset and affecting the performance too.

Thus, the max depth value plays an important role in optimizing performance, improving the accuracy of the model, and avoiding underfitting and overfitting issues of the Decision Tree. To determine the best max depth value for the Decision Tree in Iris dataset the parameter tuning code shown in below figure has been implemented.



```
# Parameter Tuning
max_depths =  [1, 2, 3, 5, 7, 20]
training_accuracy = []
testing_accuracy = []
plt.figure(figsize=(10, 8))

for i, md in enumerate(max_depths):
    dtc = DecisionTreeClassifier(criterion='gini', random_state=42, max_depth=md)
    dtc.fit(X_train_scaled, y_train)

    train_score = dtc.score(X_train_scaled, y_train)
    test_score = dtc.score(X_test_scaled, y_test)

    training_accuracy.append(train_score)
    testing_accuracy.append(test_score)

    print(f"Max Depth: {md}")
    print(f"Train Score: {train_score:.2f} | Test Score: {test_score:.2f}\n")
```

```
✓ 0.2s
```

```
Max Depth: 1
Train Score: 0.68 | Test Score: 0.64

Max Depth: 2
Train Score: 0.97 | Test Score: 0.89

Max Depth: 3
Train Score: 0.98 | Test Score: 0.98

Max Depth: 5
Train Score: 1.00 | Test Score: 0.93

Max Depth: 7
Train Score: 1.00 | Test Score: 0.93

Max Depth: 20
Train Score: 1.00 | Test Score: 0.93
```

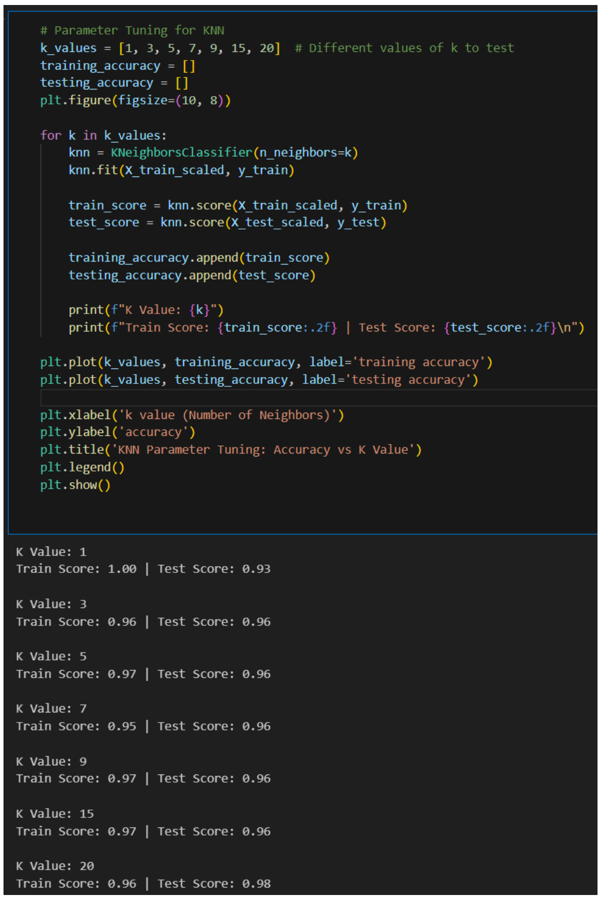
*Figure 16: Finding the Optimal Max Depth Value for Decision Tree*

As the result of this code, we found that the optimal max depth value for the Iris dataset is 3. Where the max depth of 3 had achieved the highest accuracy without overfitting to the model. Then, the model will be rerun again by passing the optimal max depth value of 3 into the Decision Tree model by following the step specified in section 4.1 such as train mode, predict, and evaluate performance.

### 4.2.3 K-Nearest Neighbours Parameter Tuning

The parameter tuning process for K-Nearest Neighbours (KNN) focused on optimizing the k-value, which determines the number of nearest neighbours considered during classification. This parameter plays a critical role in balancing the model's performance. A smaller k-value can lead to overfitting, where the model becomes overly sensitive to noise in the training data and struggles to generalize to unseen data. Conversely, a larger k-value may result in underfitting, as the model over smooths the decision boundaries, ignoring important patterns in the data.

To identify the optimal k-value, a range of values (1, 3, 5, 7, 9, 15, and 20) was tested systematically. For each k-value, the model was trained on the training dataset (X_train_scaled, y_train) and evaluated on the testing dataset (X_test_scaled, y_test). Both training and testing accuracies were recorded to analyse the performance trends. A Python script was implemented to automate this process, as shown below.



```
# Parameter Tuning for KNN
k_values = [1, 3, 5, 7, 9, 15, 20]  # Different values of k to test
training_accuracy = []
testing_accuracy = []
plt.figure(figsize=(10, 8))

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)

    train_score = knn.score(X_train_scaled, y_train)
    test_score = knn.score(X_test_scaled, y_test)

    training_accuracy.append(train_score)
    testing_accuracy.append(test_score)

    print(f"K Value: {k}")
    print(f"Train Score: {train_score:.2f} | Test Score: {test_score:.2f}\n")

plt.plot(k_values, training_accuracy, label='training accuracy')
plt.plot(k_values, testing_accuracy, label='testing accuracy')

plt.xlabel('k value (Number of Neighbors)')
plt.ylabel('accuracy')
plt.title('KNN Parameter Tuning: Accuracy vs K Value')
plt.legend()
plt.show()
```

```
K Value: 1
Train Score: 1.00 | Test Score: 0.93

K Value: 3
Train Score: 0.96 | Test Score: 0.96

K Value: 5
Train Score: 0.97 | Test Score: 0.96

K Value: 7
Train Score: 0.95 | Test Score: 0.96

K Value: 9
Train Score: 0.97 | Test Score: 0.96

K Value: 15
Train Score: 0.97 | Test Score: 0.96

K Value: 20
Train Score: 0.96 | Test Score: 0.98
```

*Figure 17: Code for Tuning K-Value in KNN*

The results of this tuning process are visualized in the "Accuracy vs. K-Value Plot" in Figure 18, which illustrates the relationship between k-values and model accuracy. The plot demonstrates that the training accuracy slightly decreases as k increases, as expected, because larger k-values smooth the decision boundaries. Conversely, the testing accuracy improves and peaks at k = 20, achieving a testing accuracy of 98%. This indicates that k = 20 provides the best trade-off between overfitting and underfitting for this dataset. Then, the model will be rerun again by passing the optimal k-value of 20 into the KNN model by following the steps specified in section 4.1, such as train mode, predict, and evaluate performance.

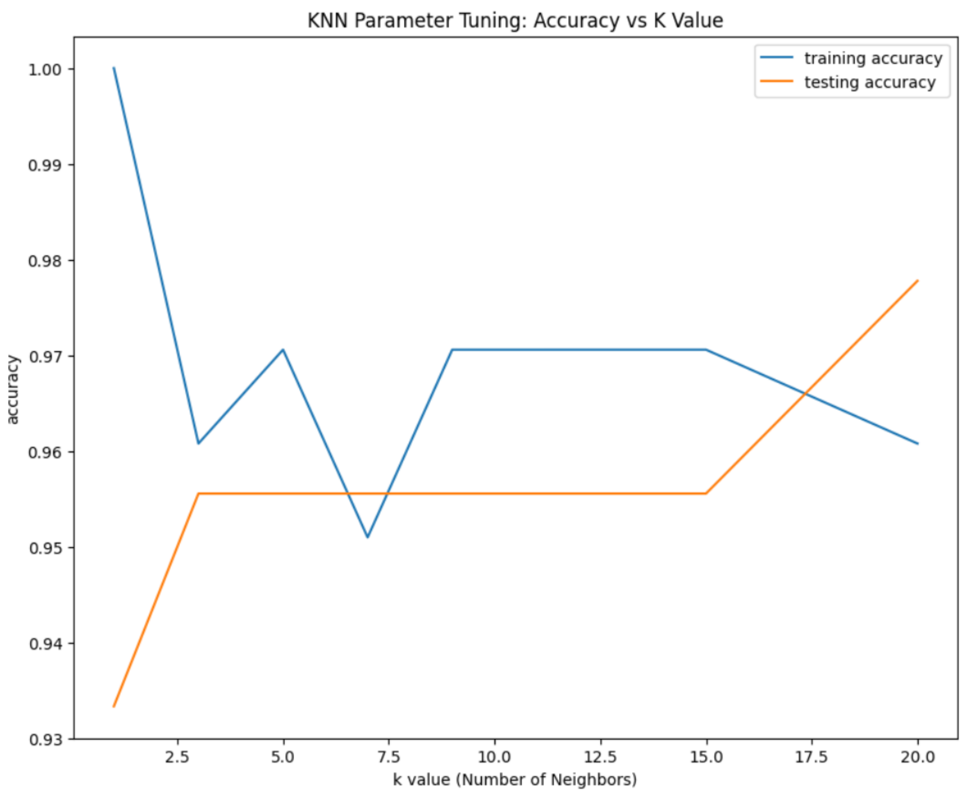


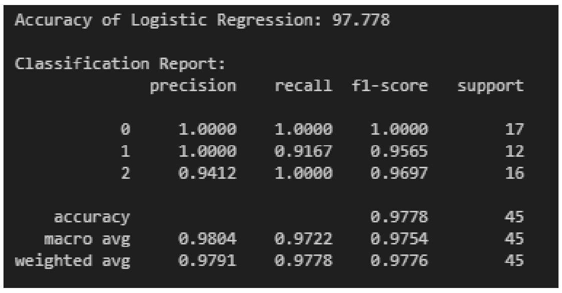
*Figure 18: Accuracy vs K value Plot*

## 5.0 Results

The result output of each model was compiled in this section. For the classification report, macro average was chosen over weighted average because macro average gives equal importance to all classes, ensuring that the evaluation will reflect balanced performance across the classes in the iris dataset (Iris-Setosa, Iris-Versicolor and Iris-Virginica). Since the dataset is balanced as well, macro averaging avoids bias, unlike weighted average which can overly emphasize larger class sized. This makes macro averaging more suitable for understanding overall model performance for the application towards iris dataset.

### 5.1 Results of Logistic Regression

By running the performance code block of Logistic Regression, we got the output as seen below.



```
Accuracy of Logistic Regression: 97.778

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000        17
           1     1.0000    0.9167    0.9565        12
           2     0.9412    1.0000    0.9697        16

    accuracy                         0.9778        45
   macro avg     0.9804    0.9722    0.9754        45
weighted avg     0.9791    0.9778    0.9776        45
```

*Figure 19: Accuracy, Precision, Recall and F1-Score of Logistic Regression*

As seen above, the accuracy and macro averaged precision, recall and F1-Score are 97.78%, 98.04%, 97.22% and 97.54% respectively. The confusion matrix for Logistic Regression on the other hand is as seen below.

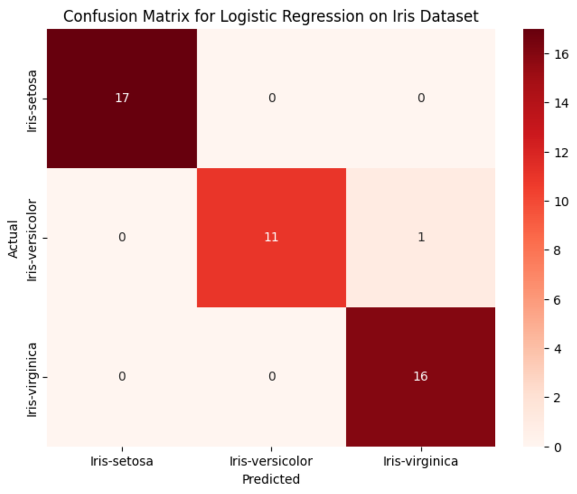
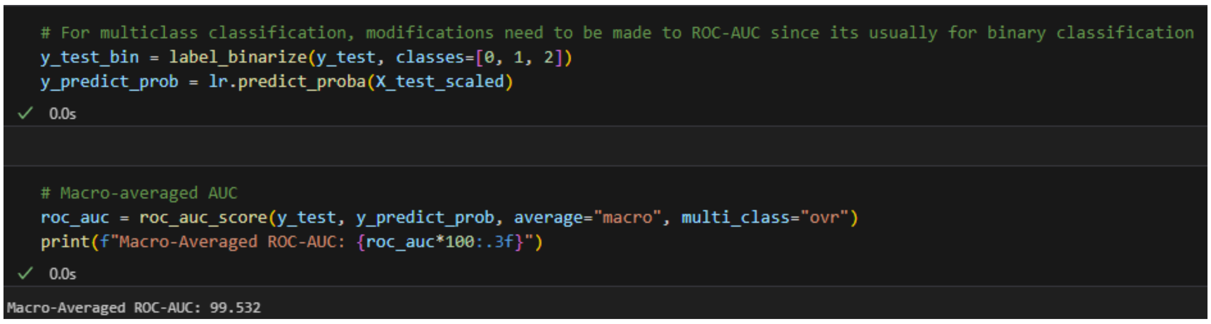


*Figure 20: Confusion Matrix for Logistic Regression on Iris Dataset*

Based on the confusion matrix, it shows us that for Iris-Setosa, all 17 instances of it were predicted correctly (top-left cell) and there were zero misclassifications for Iris-Setosa. For Iris-Versicolor, it showed that 11 instances of Iris-Versicolor were correctly classified (middle cell). However, 1 Iris-Versicolor was misclassified as Iris-Virginica (shown in middle-right cell). Lastly, for Iris-Virginica, all 16 instances of it were predicted correctly (bottom-right cell) and none of it was misclassified. This shows that the model performed perfectly for Iris-Setosa and Iris-Virginica with no misclassifications for both of those classes. There was only one single misclassification for Iris-Versicolor, indicating that the model struggles slightly to distinguish between Iris-Versicolor from Iris-Virginica. Despite the minimal error shown though, the Logistic Regression model shows a strong performance on the iris dataset.

The following code was run to determine the macro-averaged ROC-AUC. This calculates the ROC-AUC for each class individually and then averages these values, treating all classes equally, regardless of their size or frequency in the dataset. This will tell us the overall ability of the model to distinguish between classes without being affected by class imbalance.



```
# For multiclass classification, modifications need to be made to ROC-AUC since its usually for binary classification
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
y_predict_prob = lr.predict_proba(X_test_scaled)
```
```
✓ 0.0s
```
```
# Macro-averaged AUC
roc_auc = roc_auc_score(y_test, y_predict_prob, average="macro", multi_class="ovr")
print(f"Macro-Averaged ROC-AUC: {roc_auc*100:.3f}")
```
```
✓ 0.0s
Macro-Averaged ROC-AUC: 99.532
```

*Figure 21: Macro-Averaged ROC-AUC of Logistic Regression*

As shown in Figure 21, the Macro-Averaged ROC-AUC of Logistic Regression is 99.53%. This means that Logistic Regression model on the iris dataset is very effective at distinguishing between the three species, with a very high performance across all classes, suggesting that the classifier is operating at near-optimality.

The following depicts the ROC Curve for the Iris Dataset based on its true positive rate and false positive rate between classes before it was macro-averaged. For Iris-Setosa, it showed an accuracy of 1.00 indicating perfect classification. For Iris-Virginica and Iris-Versicolor, it showed an accuracy of 0.99 depicting that it had near-perfect separability. The macro-average of all this results in a result of 99.53% as shown before this.

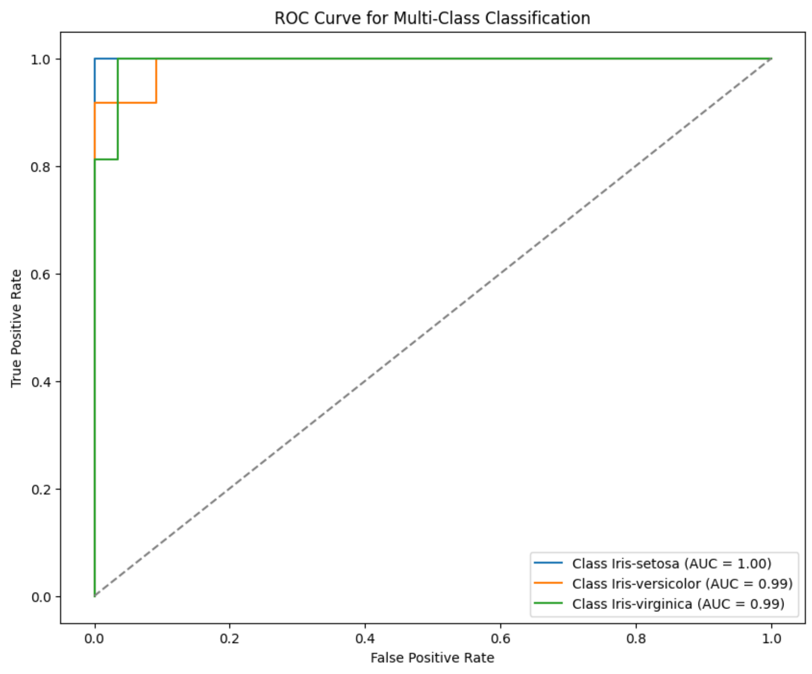
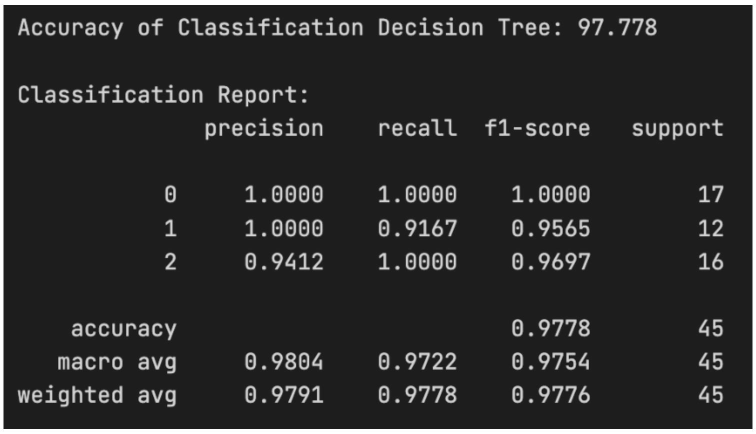


*Figure 22: ROC-AUC Curve for Multiple Classes of Iris Dataset*

## 5.2 Results of Decision Tree



```
Accuracy of Classification Decision Tree: 97.778

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000        17
           1     1.0000    0.9167    0.9565        12
           2     0.9412    1.0000    0.9697        16

    accuracy                         0.9778        45
   macro avg     0.9804    0.9722    0.9754        45
weighted avg     0.9791    0.9778    0.9776        45
```

*Figure 23: Accuracy, Precision, Recall and F1-Score of Decision Tree*

The figure above shows the accuracy and the classification report of the Decision Tree model in Iris dataset with the optimal max depth value of 3 after parameter tuning process. The model’s accuracy is 97.78% and the macro averaged precision, recall, and F1-Score are 98.04%, 97.22%, and 97.54% respectively.

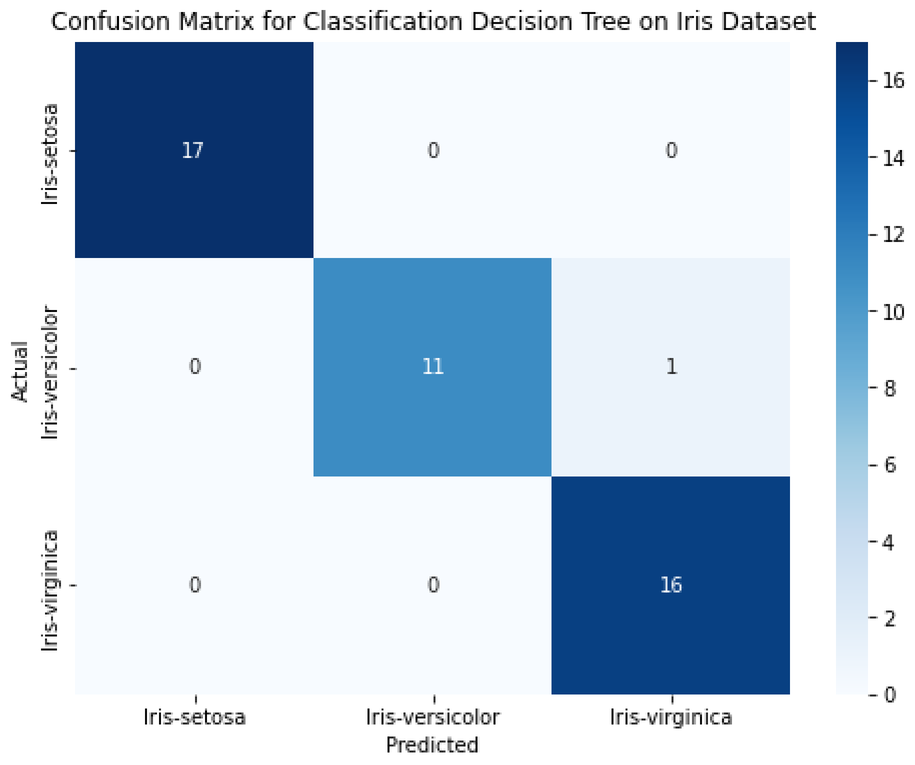
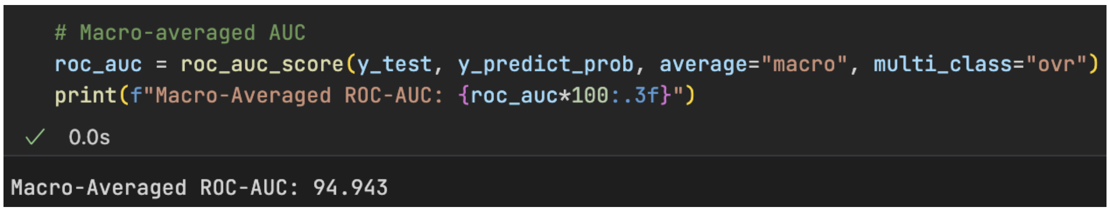


*Figure 24: Confusion Matrix for Classification Decision Tree on Iris Dataset*

The figure above showing the visualization of the confusion matrix for Decision Tree model, which provides a clear and detailed view of the classification performance on the Iris dataset. The confusion matrix showing that that all the Irish-Setosa has been predicted all correctly (top-left cell) with zero misclassification. On the other hand, the prediction results for Iris-Versicolor shows that 11 instances were classified correctly (middle cell) but there is only one instances was misclassified as Iris-Virginica (middle-right cell). Then, the Irish-Virginica also get fully 16 instances of correct classification without any misclassified (bottom-right cell). In short, the model works perfectly without any misclassification on Iris-Sentosa and Iris-Virginica, but for the Iris-Versicolor there is only one misclassified. Which prove that the Decision Tree model having a strong performance and high accuracy on Iris dataset.



```
# Macro-averaged AUC
roc_auc = roc_auc_score(y_test, y_predict_prob, average="macro", multi_class="ovr")
print(f"Macro-Averaged ROC-AUC: {roc_auc*100:.3f}")
```

✓ 0.0s

```
Macro-Averaged ROC-AUC: 94.943
```

*Figure 25: Macro-Averaged ROC-AUC of Decision Tree*

The figure above shows that, the Macro-Averaged ROC-AUC of Decision Tree is 94.94%. Which means that the model having high distinguishing power on the Iris dataset, the model able to separate the class of iris flower accurately.

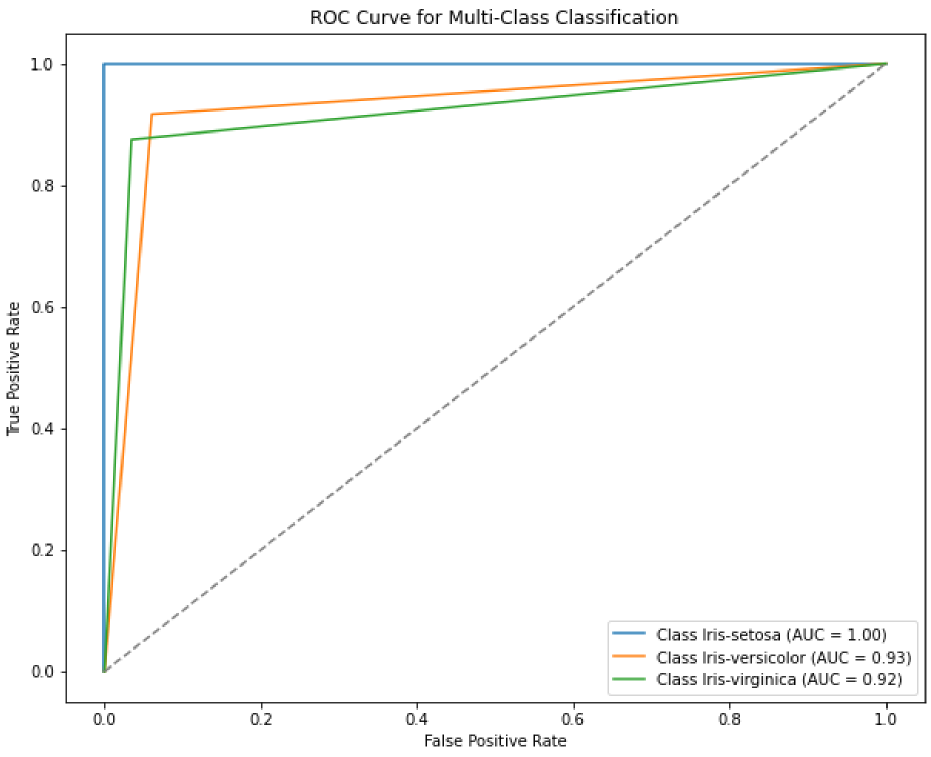
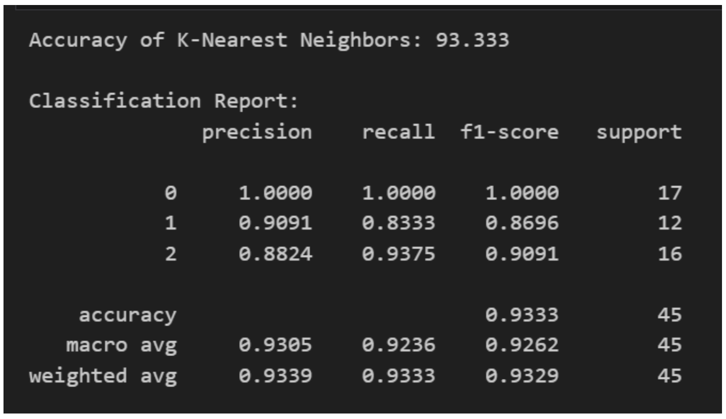


*Figure 26: ROC-AUC Curve for Multiple Classes of Iris Dataset*

The figure above showing the ROC curve for the Iris dataset based on the true positive rate and the false positive rate between classes. The Iris-Sentosa demonstrates perfect classification with 1.00 value for AUC, which shows a strong ability to distinguish Iris-Sentosa from other classes without misclassification. The Iris-Versicolor have a 0.93 value for AUC, which showing high distinguish power with only few misclassifications with other classes. The Iris-Virginica showing a similar result with Iris-Versicolor which are 0.92 of the AUC value. The macro-average of all this result is 94.94% as shown in Figure 25.

### 5.3 Results of K-Nearest Neighbours



```
Accuracy of K-Nearest Neighbors: 93.333

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000        17
           1     0.9091    0.8333    0.8696        12
           2     0.8824    0.9375    0.9091        16

    accuracy                         0.9333        45
   macro avg     0.9305    0.9236    0.9262        45
weighted avg     0.9339    0.9333    0.9329        45
```

*Figure 27: Accuracy, Precision, Recall and F1-Score of K-Nearest Neighbours*

As seen in Figure 27, the overall accuracy of the K-Nearest Neighbours model is 93.33%. The macro-averaged precision, recall, and F1-score are 93.05%, 92.36%, and 92.62%, respectively. These metrics indicate strong overall performance, although there are some variations in the scores for specific classes. The confusion matrix for K-Nearest Neighbours is as seen below.

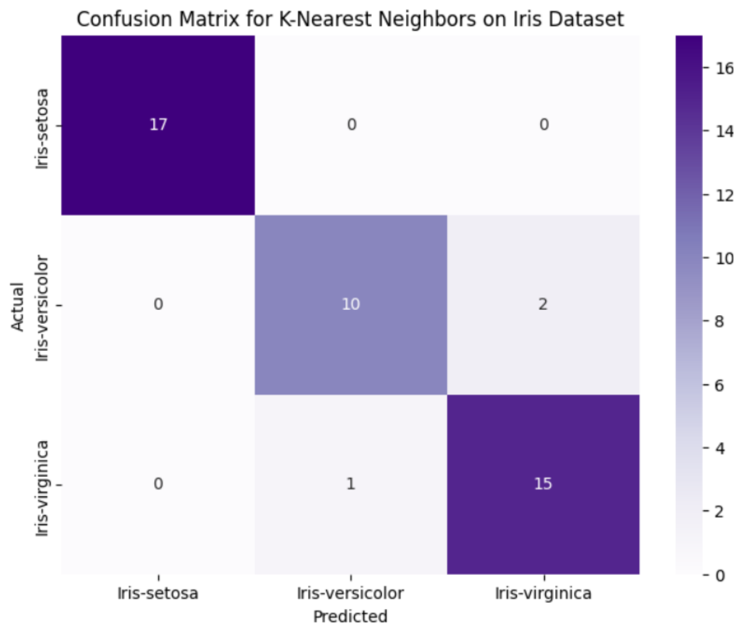
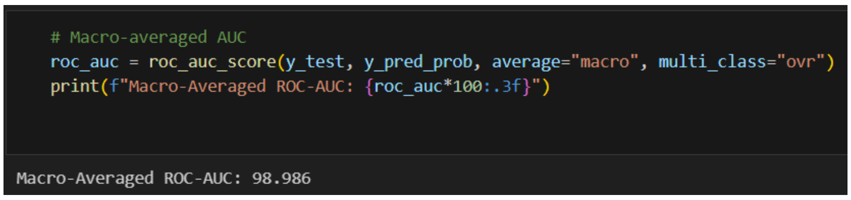


*Figure 28: Confusion Matrix for Classification K-Nearest Neighbours on Iris Dataset*

Based on the confusion matrix, the results show that for Iris-Setosa, all 17 instances were predicted correctly (top-left cell), with zero misclassifications. For Iris-Versicolor, 10 instances were correctly classified (middle cell), but 2 instances were misclassified as Iris-Virginica (middle-right cell). Lastly, for Iris-Virginica, 15 instances were correctly classified (bottom-right cell), and 1 instance was misclassified as Iris-Versicolor (bottom-middle cell). These results indicate that the KNN model performed perfectly for Iris-Setosa, with no misclassifications for this class. However, there were minor misclassifications for Iris-Versicolor and Iris-Virginica, showing that the model occasionally struggles to distinguish between these two classes. Despite these few misclassifications, the KNN model demonstrates strong overall performance on the Iris dataset, correctly classifying the vast majority of instances across all three classes.



```
# Macro-averaged AUC
roc_auc = roc_auc_score(y_test, y_pred_prob, average="macro", multi_class="ovr")
print(f"Macro-Averaged ROC-AUC: {roc_auc*100:.3f}")
```

```
Macro-Averaged ROC-AUC: 98.986
```

*Figure 29: Macro-Averaged ROC-AUC of K-Nearest Neighbours*

As shown in Figure 29, the macro averaged ROC-AUC of K-Nearest Neighbours is 98.98%. This indicates that the KNN model is highly effective at distinguishing between the three Iris species. The near-perfect score demonstrates that the classifier performs exceptionally well across all classes, suggesting that it operates at near optimality in terms of its ability to separate the species in the datasets.

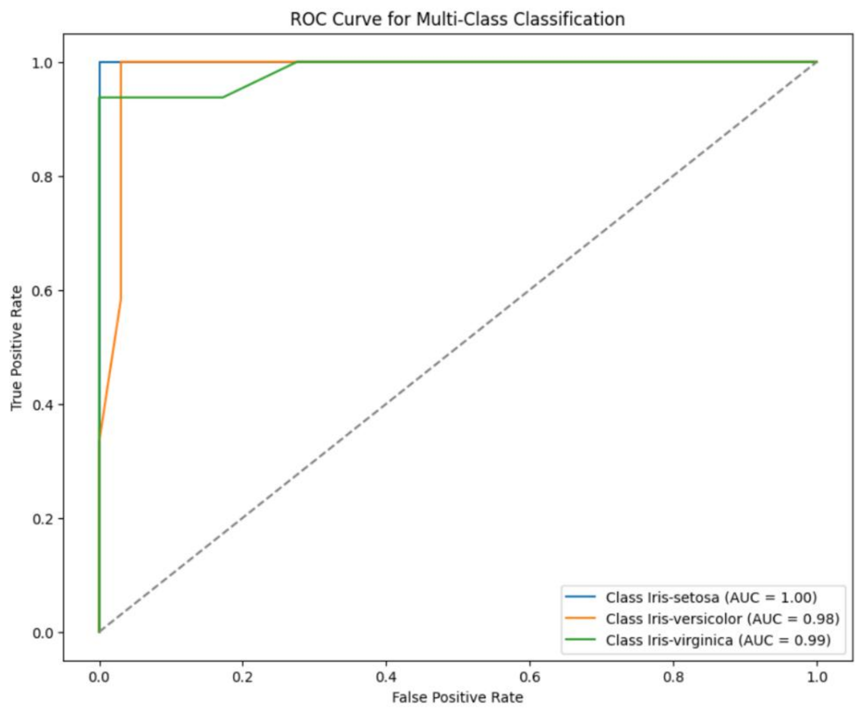


*Figure 30: ROC-AUC Curve for Multiple Classes of Iris Dataset*

The above depicts the ROC curve for the Iris Dataset based on its true positive rate and false positive rate between classes before it was macro averaged. For Iris-Setosa, the ROC curve shows an AUC of 1.00, indicating perfect classification for this class. For Iris-Versicolor and Iris-Virginica, the AUC values are 0.98 and 0.99 respectively, reflecting near perfect separability for these two classes. These results highlight the KNN model's ability to distinguish effectively between the three Iris species. The macro averaged ROC-AUC of 98.98% further confirms the model's strong overall performance, demonstrating its capability to classify the dataset with high accuracy across all classes.

## 6.0 Discussion

### 6.1 Compilation of Performance Matrices

The table below depicts the results of the study. Logistic Regression and Decision Tree were shown to have the same and highest results for accuracy (97.78%), precision (98.04%), recall (97.22%) and f1-score (97.54%). K-Nearest Neighbours on the other hand are seen to have the poorer performance compared to the other two techniques used with an accuracy of 93.33%, precision of 93.05%, recall of 92.36% and f1-score of 92.62%.

*Table 1: Compilation of Performance Results of the Models*

| Model | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 97.78 | 98.04 | 97.22 | 97.54 |
| Decision Tree | 97.78 | 98.04 | 97.22 | 97.54 |
| K-Nearest Neighbours | 93.33 | 93.05 | 92.36 | 92.62 |

### 6.2 Discussion of Results

The reason why Logistic Regression and Decision Tree produced the identical results for accuracy, precision, recall and f1-score is probably because the iris dataset is simple, balanced and nearly linearly separable as well if you refer to the scatter plots in section 3.5, Figure 10 where Iris-Setosa can be easily differentiated from the other two species, but the Iris-Versicolor and Iris-Virginica overlap slightly at the border. Because the classes are relatively well separable as seen in the scatter plots, this makes Logistic Regression’s linear boundaries and the Decision Tree’s piecewise boundaries both effectively able to classify the dataset. To summarize, the linear boundaries are probably emulated by simple Decision Trees which caused the result output to be similar.

K-Nearest Neighbours on the other hand had a lower accuracy which indicates that the model made more incorrect predictions, a lower precision meaning that the model had a higher false positive rate, a lower recall which shows that the model was missing more true positives and had more false negatives predictions and a lower f1-score when compared to the other two models which indicates a poor balance between precision and recall. Overall, K-Nearest Neighbours still performed decently since its performance was in the 90’s range however when its performance matrix was compared to the other models, it was seen to be the lowest performing model. This means that Decision Tree and Logistic Regression are the more suitable models for evaluating the iris dataset.

### 6.3 Challenges Encountered

During the process, we encountered several challenges that tested our problem-solving skills and adaptability. These challenges ranged from technical complexities in model optimization to issue arising during data preprocessing. Despite these obstacles, we were able to address them effectively through systematic approaches and careful planning, ensuring this assignment's success.

One of the key challenges was the complexity of parameter tuning for the machine learning models. Parameters such as **C** in Logistic Regression, max_depth in Decision Tree, and **k** in K-Nearest Neighbours significantly influenced model performance. Selecting inappropriate values could lead to underfitting or overfitting, making it crucial to identify the optimal parameters. To tackle this, we employed grid search and systematically tested different parameters values. This approach allowed us to pinpoint configurations that maximized accuracy while maintaining model generalization.

Another notable challenge was the overlapping features between classes in the Iris dataset, particularly between Iris-Versicolor and Iris-Virginica. This overlap made it difficult for the models, especially K-Nearest Neighbours, to achieve perfect classification. To overcome this, we normalized the data to ensure all features contributed equally to distance calculations in KNN. Visualisation such as scatter plots were also utilized to understand the extent of class overlap. These insights informed our parameter tuning and analysis, leading to improved model performance.

Lastly, the data preprocessing phase posed challenges due to the time and effort required to prepare the dataset. The task like normalizing features, encoding categorical variables, and resolving outliers demanded significant manual effort in terms of understanding what steps were necessary depending on the model's characteristics. We addressed this by leveraging Python libraries like Pandas and Scikit-learn as well as supporting documentations to automate and understand these processes which reduces the workload and ensuring data consistency. Overall, these challenges reinforced the importance of systematic approaches and the use of efficient tools in successfully completing a machine learning assignment.

## 6.4 Future Improvements

There are few future improvements can be implemented to enhance the performance and the robustness of the models such as applications of ensemble methods and exploring a larger dataset.

First, ensemble methods are a technique that combines predictions of multiple models to achieve better performance. There are few methods can be involved in this project such as bagging, boosting, and stacking. The bagging method involves training multiple models on different subsets of data, which helps to avoid overfitting issues to obtain a more stable and accurate result. By implementing the boosting method, the predictive errors on initial iterations of training will be corrected in each other subsequent iterations incrementally, which can improve the overall accuracy. Then, in stacking, strengths of diverse algorithms are utilized by combining their predictions using a meta-model, which leads to achieve a more superior performance compared to just using a single base model. In short, the applications of ensemble method can help to improve the accuracy and the performance of the machine learning algorithms.

Additionally, another improvement would be to explore a larger Iris dataset as currently the Iris dataset used is considered to be a small dataset by machine learning standards. There are some limitations of using a small dataset like Iris. A model trained with a small dataset might end up failing to capture all possible underlying relationships between features. Not only that, but the accuracy of the model will also be unstable with the use of a smaller dataset, and it is easier to encounter overfitting issues. Thus, by implementing the larger dataset, more complex patterns can be introduced and learnt by the model, which allow the model to be trained more effectively with higher performance and accuracy.

In conclusion, the future improvements stated above will help in enhance the performance and accuracy of the machine learning models applied towards the Iris dataset in this study. Ensemble methods including bagging, boosting, and stacking can improve the accuracy and reduce the overfitting issue and at the same time, the larger dataset will help to ensure the reliability of the models.

## References

MathNerd (2018, March 22). *Kaggle.* https://www.kaggle.com/datasets/arshid/iris-flower-dataset