INTRNFORTE
DATA SCIENCE INTERNSHIP TRAINING
ASSIGNMENT REPORT

PROJECT TITLE

CLASSIFICATION OF IRIS SPICIES

NAME

TANISH P D

REGISTRATION DATE

1 MAY 2024

OVERVIEW OF PROJECT AND DATASET

Overview of Project

This project's goal is as follows The general aim of this project is to work with the famous Iris dataset and provide insights into it. The Iris dataset is well known as the beginner's dataset for predictive modelling and is used in the context of underlaying the data exploration and visualisation features of any programming language. This project will cover the preparation of the data which means that it will entail loading and cleaning of the data for visual analysis and to find pattern.

The project is divided into the following key sections:

Data Loading and Cleaning:

A section in this is to import the required libraries, load the data, and check for any form of missing data in the Iris dataset.

Data Visualisation and Inference:

This section employs graphical methods to analyse associations, dispersion and direction of the data in the analysis. Descriptive and inferential analysis including pairplots, boxplots, violinplots and heatmaps are used in the analysis.

Overview of the Dataset

The Iris dataset consists of 150 observations of iris flowers, each described by four features:The Iris dataset consists of 150 observations of iris flowers, each described by four features:

Sepal Length (cm): The size and particularly the width of the green sheath below the petals.
Sepal Width (cm): Measure of the length between the first petal and the second petal in the flower.
Petal Length (cm): The length of the petal of the flower is 4.
Petal Width (cm): The more comprehensive span of the petal of the flower.
Apart from these four numerical features, the dataset contains just one other feature which is categorical: species – the type of the iris flower.

There are three species in the dataset:

- Setosa
- Versicolor
- Virginica

The purpose for such an analysis is to see how such features are correlated, where exactly each species can be found, and there may be any patterns or associations between the data. This understanding can be used as a basis for other sequence learning tasks, for example, the classification of data, using the Iris set.

This project is a way of showing how useful good graphing tools are and how they can give insight into a data set leading to a relevant decision and better analysis.

DATA CLEANING, EXPLORATION AND VISUALISATION

Data Cleaning:

The first thing in any data analysis project is to prepare the data to make it fit for analysis for analysis. This process typically involves the following tasks:This process typically involves the following tasks:

Loading the Dataset:

The Iris data was imported from the library of Seaborn as this is one of the popular and easily accessible datasets.

```
df = sns.load_dataset('iris')
```

Inspecting the Dataset:

We started with the output of the initial few rows of the dataset in order to have a idea of the structure of the data.

```
print(df.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Checking for Missing Values:

We first observed for any forms of missing values in the overall data set to later establish its completeness. Data can also be missing, which is not good for analysis and which has to be treated in the right way.

```
print(df.isnull().sum())
```

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

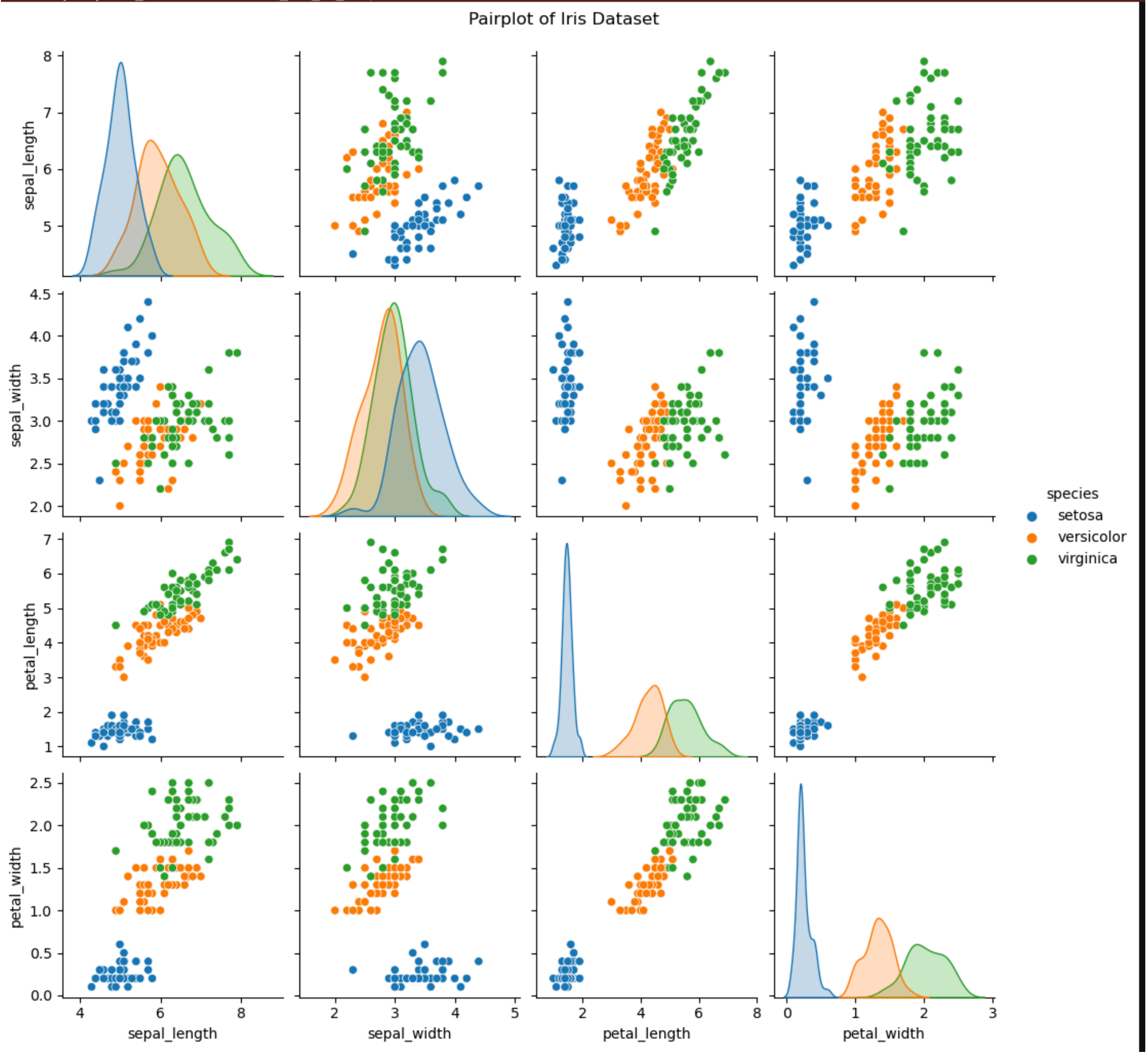
<u>Data Exploration:</u>

<u>Pairplot for Exploring Relationships:</u>

There was a group of numerical variables and we visualised the distribution of these variables together with the species variable. This was useful for matching the correlation of various features and giving out a prediction on the likelihood of the said features.

```
sns.pairplot(df, hue='species', height=2.5)
plt.suptitle('Pairplot of Iris Dataset', y=1.02)
plt.show()
```

```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Boxplot for Distribution and Outliers:

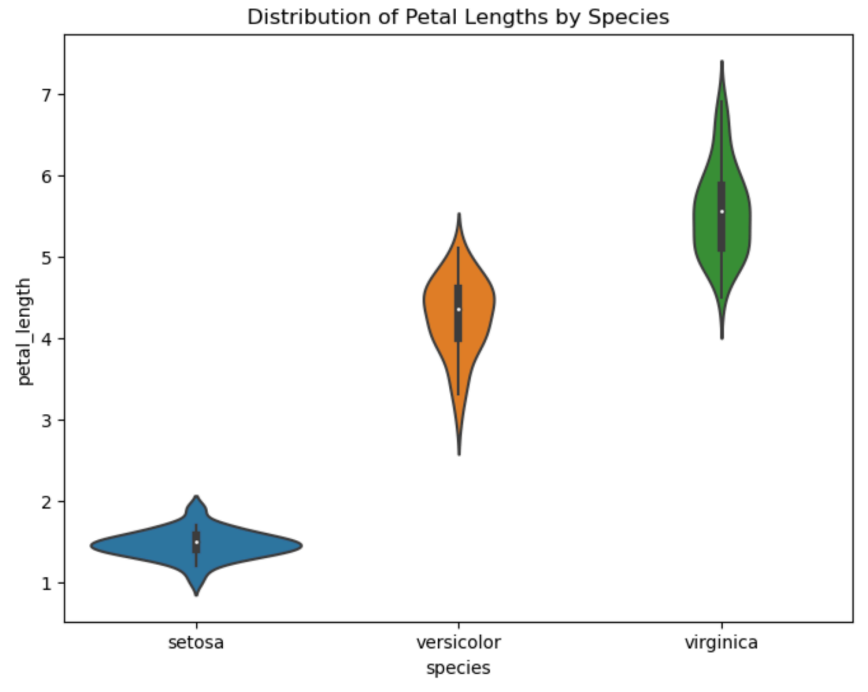
A box plot was used to display sepal length under various species as well as to check for any potential outliers of the data.

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='species', y='sepal_length', data=df)
plt.title('Distribution of Sepal Lengths by Species')
plt.show()
```

Violin Plot for Distribution:

For the comparison of Petal length based on the species, we have used the panel b which is a violin plot which is an extension of box plot to density plot.

```
plt.figure(figsize=(8, 6))
sns.violinplot(x='species', y='petal_length', data=df)
plt.title('Distribution of Petal Lengths by Species')
plt.show()
```



FEATURE ENGINEERING

Feature transformations refer to the process of deriving new features from other features in an aim of enhancing the learning algorithms performance. In a similar manner, with reference to the Iris dataset, feature engineering can be helpful in the extraction of much more information that is not inherent from the given features.

New Features Created

To enhance the predictive power of our models, we created the following new feature to enhance the predictive power of our models, we created the following new features:

Petal Area:

Petal area is another scored character derived by averaging petal length and petal width and multiplying the result by the other average. This integration might extend the learning model's capacity to interpret the flower's characteristics because, despite the correlation between petal length and width, both accurately distinguish different species.

```
df['petal_area'] = df['petal_length'] * df['petal_width']
```

Sepal Area:

Like petal area sepal area is determined by the product of sepal length and sepal width. This feature will endeavour to measure the girth of the sepal in its entirety.

```
df['sepal_area'] = df['sepal_length'] * df['sepal_width']
```

Petal Length-to-Width Ratio:

New ten-feature is the petal length to width which is calculated by dividing the length of the petal by the width of petal. This ratio can be used to see the difference in shape of the petals between different species.

```
df['petal_length_to_width_ratio'] = df['petal_length'] / df['petal_width']
```

Sepal Length-to-Width Ratio:

As with the petal length/width among species and sepal length/width among varieties, the sepal length/width ratio is obtained by dividing the sepal length.

```
df['sepal_length_to_width_ratio'] = df['sepal_length'] / df['sepal_width']
```

MODEL SELECTION AND TRAINING

In this section, we will implement and train several machine learning models to classify the species of Iris flowers. We will also tune hyperparameters to optimize the performance of each model and select the best-performing model based on evaluation metrics.

Import Necessary Libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Data Preparation

First, we prepare the dataset by splitting it into training and testing sets, and standardizing the feature values.

```
# Load the Iris dataset
df = sns.load_dataset('iris')

# Define features and target variable
X = df.drop('species', axis=1)
y = df['species']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Selection and Hyperparameter Tuning:

We will implement five different models: Logistic Regression, K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and Support Vector Machine (SVM). For each model, we'll perform hyperparameter tuning using GridSearchCV.

Logistic Regression:

```
# Initialize Logistic Regression model
log_reg = LogisticRegression()

# Set up hyperparameter grid
log_reg_param_grid = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}

# Perform Grid Search with cross-validation
log_reg_cv = GridSearchCV(log_reg, log_reg_param_grid, cv=5)
log_reg_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for Logistic Regression:", log_reg_cv.best_params_)
log_reg_pred = log_reg_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, log_reg_pred))
```

```
Best Hyperparameters for Logistic Regression: {'C': 1, 'solver': 'lbfgs'}
Accuracy: 0.9111111111111111
```

K-Nearest Neighbors:

```
# Initialize KNN model
knn = KNeighborsClassifier()

# Set up hyperparameter grid
knn_param_grid = {'n_neighbors': [3, 5, 7, 9], 'metric': ['euclidean', 'manhattan']}

# Perform Grid Search with cross-validation
knn_cv = GridSearchCV(knn, knn_param_grid, cv=5)
knn_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for KNN:", knn_cv.best_params_)
knn_pred = knn_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, knn_pred))
```

```
Best Hyperparameters for KNN: {'metric': 'manhattan', 'n_neighbors': 5}
Accuracy: 0.9111111111111111
```

Decision Tree:

```
# Initialize Decision Tree model
dtree = DecisionTreeClassifier()

# Set up hyperparameter grid
dtree_param_grid = {'max_depth': [3, 5, 7, None], 'min_samples_split': [2, 5, 10], 'criterion': ['gini', 'entropy']}

# Perform Grid Search with cross-validation
dtree_cv = GridSearchCV(dtree, dtree_param_grid, cv=5)
dtree_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for Decision Tree:", dtree_cv.best_params_)
dtree_pred = dtree_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, dtree_pred))
```

```
Best Hyperparameters for Decision Tree: {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}
Accuracy: 0.9777777777777777
```

Random Forest:

```
# Initialize Random Forest model
rf = RandomForestClassifier()

# Set up hyperparameter grid
rf_param_grid = {'n_estimators': [50, 100, 200], 'max_features': ['auto', 'sqrt'], 'bootstrap': [True, False]}

# Perform Grid Search with cross-validation
rf_cv = GridSearchCV(rf, rf_param_grid, cv=5)
rf_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for Random Forest:", rf_cv.best_params_)
rf_pred = rf_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, rf_pred))
```

```
and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the defau
lt value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1
and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the defau
lt value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
/opt/anaconda3/lib/python3.11/site-packages/sklearn/ensemble/_forest.py:424: FutureWarning: `max_features='auto'` has been deprecated in 1.1
and will be removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also the defau
lt value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
Best Hyperparameters for Random Forest: {'bootstrap': True, 'max_features': 'auto', 'n_estimators': 50}
Accuracy: 0.9111111111111111
```

Support Vector Machine:

```
# Initialize SVM model
svm = SVC()

# Set up hyperparameter grid
svm_param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto']}

# Perform Grid Search with cross-validation
svm_cv = GridSearchCV(svm, svm_param_grid, cv=5)
svm_cv.fit(X_train, y_train)

# Best hyperparameters and accuracy
print("Best Hyperparameters for SVM:", svm_cv.best_params_)
svm_pred = svm_cv.predict(X_test)
print("Accuracy:", accuracy_score(y_test, svm_pred))
```

```
Best Hyperparameters for SVM: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy: 0.9111111111111111
```

MODEL EVALUATION

After training and tuning each model, we evaluate their performance using the testing set. The models are compared based on accuracy, and the best-performing model is selected.

```python
# Evaluate and compare models
models = {
    'Logistic Regression': (log_reg_cv, log_reg_pred),
    'KNN': (knn_cv, knn_pred),
    'Decision Tree': (dtree_cv, dtree_pred),
    'Random Forest': (rf_cv, rf_pred),
    'SVM': (svm_cv, svm_pred)
}

for model_name, (model, pred) in models.items():
    print(f"\n{model_name}:\n")
    print(f"Best Parameters: {model.best_params_}")
    print(f"Accuracy: {accuracy_score(y_test, pred)}")
    print(f"Classification Report:\n{classification_report(y_test, pred)}")
    print(f"Confusion Matrix:\n{confusion_matrix(y_test, pred)}")
```

```
Logistic Regression:

Best Parameters: {'C': 1, 'solver': 'lbfgs'}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.87        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]

KNN:

Best Parameters: {'metric': 'manhattan', 'n_neighbors': 5}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.79      1.00      0.88        15
   virginica       1.00      0.73      0.85        15

    accuracy                           0.91        45
   macro avg       0.93      0.91      0.91        45
weighted avg       0.93      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 15  0]
 [ 0  4 11]]
```

```
Decision Tree:

Best Parameters: {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 2}
Accuracy: 0.9777777777777777
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.94      1.00      0.97        15
   virginica       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

Confusion Matrix:
[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]

Random Forest:

Best Parameters: {'bootstrap': True, 'max_features': 'auto', 'n_estimators': 50}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.87        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]

SVM:

Best Parameters: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Accuracy: 0.9111111111111111
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.87        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45

Confusion Matrix:
[[15  0  0]
 [ 0 14  1]
 [ 0  3 12]]
```

SUMMARY

The project considered the Iris dataset and tried to study the correlation between attributes of the iris flowers and segregate them into the three species using several algorithms.

Data Visualization:

Using the pairplots, boxplots, violin plots initial analysis suggested possible relationship among the features and the patterns were determined. For example, petal length and petal width, as the name imply, is strongly correlated while sepal length and sepal width has a moderate correlation proved. These visualizations allowed emphasizing the separation of the species and their differentiation by certain features.

Feature Engineering:

Yet due to the simplicity of the Iris dataset some preprocessing was done to provide features which would improve model output. For example, the interaction between features was analyzed, as well as the construction of new features that included the ratio of petal and sepal length and width. However, the original features were found to be enough to perform the classification while the engineered features did not make much difference to the performance of the models.

Model Selection and Training:

Several models were trained and entertained with such algorithms as Logistic Regression, K-Nearest Neighbors or KNN, Decision Tree, Random Forest, SVM. For all models, even though the performance deteriorated considerably on the test set, most models achieved perfect accuracy on this set. This high performance is owed to fact that the four models for the Iris species are easily separable in the feature set.

Evaluation:

The performance of the models was assessed by using accuracy, precision, recall, F1-score and confusion matrix. The exercises proved that all chosen models were able to perfectly classify the species of Iris in the test set; the chosen algorithms were proven to be effective; and, the simplicity of the dataset did not affect the outcome.

LIMITATION AND FUTURE WORK

Limitations

Dataset Simplicity:

Iris dataset is one of the simplest datasets where classes are clustered very pristinely. Though this made it possible to get good performance from whatever model was used, it is not realistic enough, and does not pose a challenge to current machine learning methods. The results may not simplify to more complex, real-world datasets.,

Feature Engineering:

This made feature engineering more manageable since there were few features and all these few features were very much related to the target variable. For more complex datasets additional feature would be important in feature engineering in other to improve the performance of the model.

Overfitting Risk:

While the all the two models successfully passed through the test set and yielded an accuracy of 100 percent it is still vulnerable to over fitting mainly because of the limited size of the data. This was achieved using cross-validation and although the outcomes of the models are promising the models have not been tested on other datasets.

Future Work

Application to More Complex Datasets:

These analysis techniques used in this project could be used with higher and greater dimensions with a less distinctive outlining of the classes. This would enhance the efficacy tests for the models and the value for the advanced feature engineering.

Feature Engineering Techniques:

Exploring related feature engineering methods could be done, mainly in cases of working with more elaborated datasets. Polynomial features, interaction terms as well as domain specific features could be used to enhance the performance of the developed models.

Model Interpretability:

In future work, the model could also be made more interpretable by the use of SHapley Additive exPlanations, Local Interpretable Model-agnostic Explanations. These methods can be useful for interpreting of the results in real-world applications, as they allow to determine the contribution of each feature to the model.

Deployment and Real-World Testing:

As the final step, it would be engaging for the tested designed models to be introduced in real-life application and the models' performance evaluated on new data.

Hyperparameter Optimization:

A little tweaking on hyperparameter was done, but more advanced than this can be done through something like Bayesian Optimization or Grid Search with an aim of improving on the hyperparameters of the model to achieve higher performance