# IrisSpeciesPredictor

**By: Tiffany Chu, Gaurang Ahuja, Nguyen Nguyen, Vienne Lee**

## Summary

This project investigates whether iris species can be predicted using sepal and petal measurements. After loading and validating the dataset, we explore some basic patterns, do some EDA and then train a model. The overall results show that petal measurements provide strong separation between species, allowing the models to achieve high accuracy.

## Introduction

The Iris dataset is a well-known benchmark in machine learning, containing measurements of iris flowers collected to study how physical characteristics differ across species.

### Feature Summary

There are 4 numerical features in this dataset:

1. `sepal_length` : The length of the sepal (outer part of the flower)
2. `sepal_width` : The width of the sepal
3. `petal_length` : The length of the petal
4. `petal_width` : The width of the petal

There is 1 categorical feature in the dataset:

1. `species` : The target variable (label)

We are using the Iris dataset to answer the question: **"Can we predict the Iris species using petal and sepal measurements?"**

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are not linearly separable from each other. (From: https://archive.ics.uci.edu/dataset/53/iris)

## Methods & Results

### Loading the Data + Checking Correct file format

We use the `data_loader` script to get the data set from the URL and load it as a data frame. Within the script, we check if the data is indeed in CSV format, and if it is, we save the data locally within the `/data` folder.

In [1]:
```
from src.data_loader import load_data

url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"

data = load_data(url)
data.head()
```

```
File saved to: ./data/iris.csv
```

Out[1]:

| | 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 |

## Validation Checks

Using the `validate_iris` script, we validate the data frame for the following:

- Correct column names
- Correct data types in each column
- Empty observations
- Duplicate observations
- Correct category levels (i.e., no string mismatches or single values)
- Outlier or anomalous values
- Missingness against expected threshold
- Target/response variable follows expected distribution
- Whether there are anomalous correlations between target/response variable and features/explanatory variables
- Whether there are anomalous correlations between features/explanatory variables

In [2]:
```
from src.validate_iris import validate_data

data = validate_data(data)

data.head()
```

```
Schema validation passed (columns + types + basic ranges)
No empty rows
Validation FAILED: Found 1 duplicate rows. They will be dropped.
The dataset now have 149 rows and 5 columns.
Species categories OK
Target variable distribution looks reasonable
Validation FAILED: Outliers detected! Might want to consider using StandardScaler tr
ansformation.
Missingness is within allowed limits.
Warning: Feature petal_width has way to high correlation with the target column (spe
cies) and could lead to ovefitting or data leakage: 0.9565135482715577
Warning: Features 'petal_length' and 'petal_width' are too correlated and could lead
to multicollinearity or repeat feature: 0.9627722945997859
```

Out[2]:

| | 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 |

# Train/test split before any EDA

In [3]:
```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(
    data, test_size=0.3, random_state=123
)
```

## Basic Data Stats

In the code below, we check some basic stats for the dataframe such as the number of rows, columns etc. We convert the columns names to a more standard format using underscores and lowercase. We then check how many unique species there are.

In [4]:
```
train_df_shape = train_df.shape
print(f"Dataframe has {train_df_shape[0]} rows")
print(f"Dataframe has {train_df_shape[1]} cols\n")
```

```
Dataframe has 104 rows
Dataframe has 5 cols
```

In [5]:
```
print(f"Dataframe info:\n")
train_df.info()
```

```
Dataframe info:

<class 'pandas.core.frame.DataFrame'>
Index: 104 entries, 114 to 109
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  104 non-null    float64
 1   sepal_width   104 non-null    float64
 2   petal_length  104 non-null    float64
 3   petal_width   104 non-null    float64
 4   species       104 non-null    object 
dtypes: float64(4), object(1)
memory usage: 4.9+ KB
```

In [6]:
```
#Clean up column names
train_df.columns = train_df.columns.str.strip().str.lower().str.replace(" ", "_")
```

In [7]:
```
#See how many uniques species there are
print(f"\nUnique species of Iris: {train_df['species'].nunique()}\n")
```

```
Unique species of Iris: 3
```

In [8]:
```
#See the first few rows after cleanup
train_df.head()
```

Out[8]:

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **114** | 5.8 | 2.8 | 5.1 | 2.4 | virginica |
| **135** | 7.7 | 3.0 | 6.1 | 2.3 | virginica |
| **53** | 5.5 | 2.3 | 4.0 | 1.3 | versicolor |
| **19** | 5.1 | 3.8 | 1.5 | 0.3 | setosa |
| **38** | 4.4 | 3.0 | 1.3 | 0.2 | setosa |

## Data Wrangling

Since our goal is classification, this section will look at statistics which will help distinguish between the species. Therefore, we will look at things like mean, median, min, max etc for each feature. This will be followed by graphical analysis where we will explore bar plot, pairwise plot, and boxplot.

In [9]:
```
#Summary of numeric columns
train_df.describe()
```

Out[9]:

| | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| **count** | 104.000000 | 104.000000 | 104.000000 | 104.000000 |
| **mean** | 5.833654 | 3.048077 | 3.777885 | 1.228846 |
| **std** | 0.772917 | 0.408877 | 1.670937 | 0.740726 |
| **min** | 4.400000 | 2.200000 | 1.000000 | 0.100000 |
| **25%** | 5.100000 | 2.800000 | 1.600000 | 0.375000 |
| **50%** | 5.800000 | 3.000000 | 4.400000 | 1.300000 |
| **75%** | 6.400000 | 3.300000 | 5.000000 | 1.800000 |
| **max** | 7.700000 | 4.400000 | 6.900000 | 2.500000 |

In [10]:
```
#Summary per specie
train_df.groupby("species").agg(["mean", "median", "std", "min", "max"])
```

Out[10]:

| | sepal_length | | | | | sepal_width | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **mean** | **median** | **std** | **min** | **max** | **mean** | **median** | **std** | **min** | **max** |
| **species** | | | | | | | | | | |
| **setosa** | 5.000000 | 5.00 | 0.336011 | 4.4 | 5.8 | 3.415625 | 3.4 | 0.400894 | 2.3 | 4.4 |
| **versicolor** | 6.000000 | 5.95 | 0.505863 | 4.9 | 7.0 | 2.832500 | 2.9 | 0.283194 | 2.2 | 3.4 |
| **virginica** | 6.459375 | 6.35 | 0.626748 | 4.9 | 7.7 | 2.950000 | 3.0 | 0.290717 | 2.5 | 3.6 |

In [11]:
```
#Count per specie
print(train_df['species'].value_counts())
```

```
species
versicolor    40
virginica     32
setosa        32
Name: count, dtype: int64
```

### Insights from Data Statistics

From the above code cells, we can see:

- Most species have approximately the same number of observations
- Petal measurements display the strongest separation
- Setosa is distinct while versicolor and virginica show some overlap
- Petal dimensions have meaningful differences

# EDA Plots

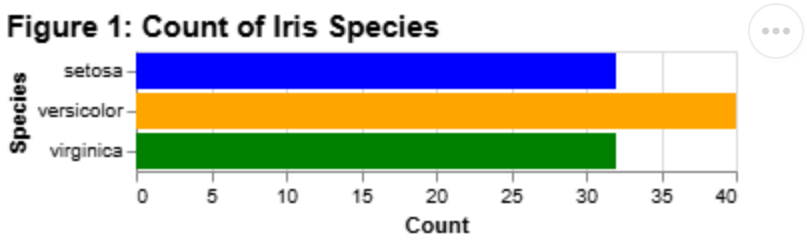
Below are the plots visualize the underlying patterns, distributions, and relationships within the three features (sepal_length, sepal_width, petal_length, petal_width) in the Iris dataset.

## Bar Plot

Bar plot provides an overview of the count of each species.

In [12]:
```
import altair as alt

alt.Chart(train_df).mark_bar().encode(
    x = alt.X("count()").title("Count"),
    y = alt.Y("species").title("Species"),
    color = alt.Color(
        "species",
        scale=alt.Scale(
            domain=['setosa', 'versicolor', 'virginica'],
            range=['blue', 'orange', 'green'])
    ).legend(None)
).properties(
    title = alt.TitleParams(
        text = "Figure 1: Count of Iris Species",
        anchor = 'start',
        fontSize = 15
    )
)
```

Out[12]:



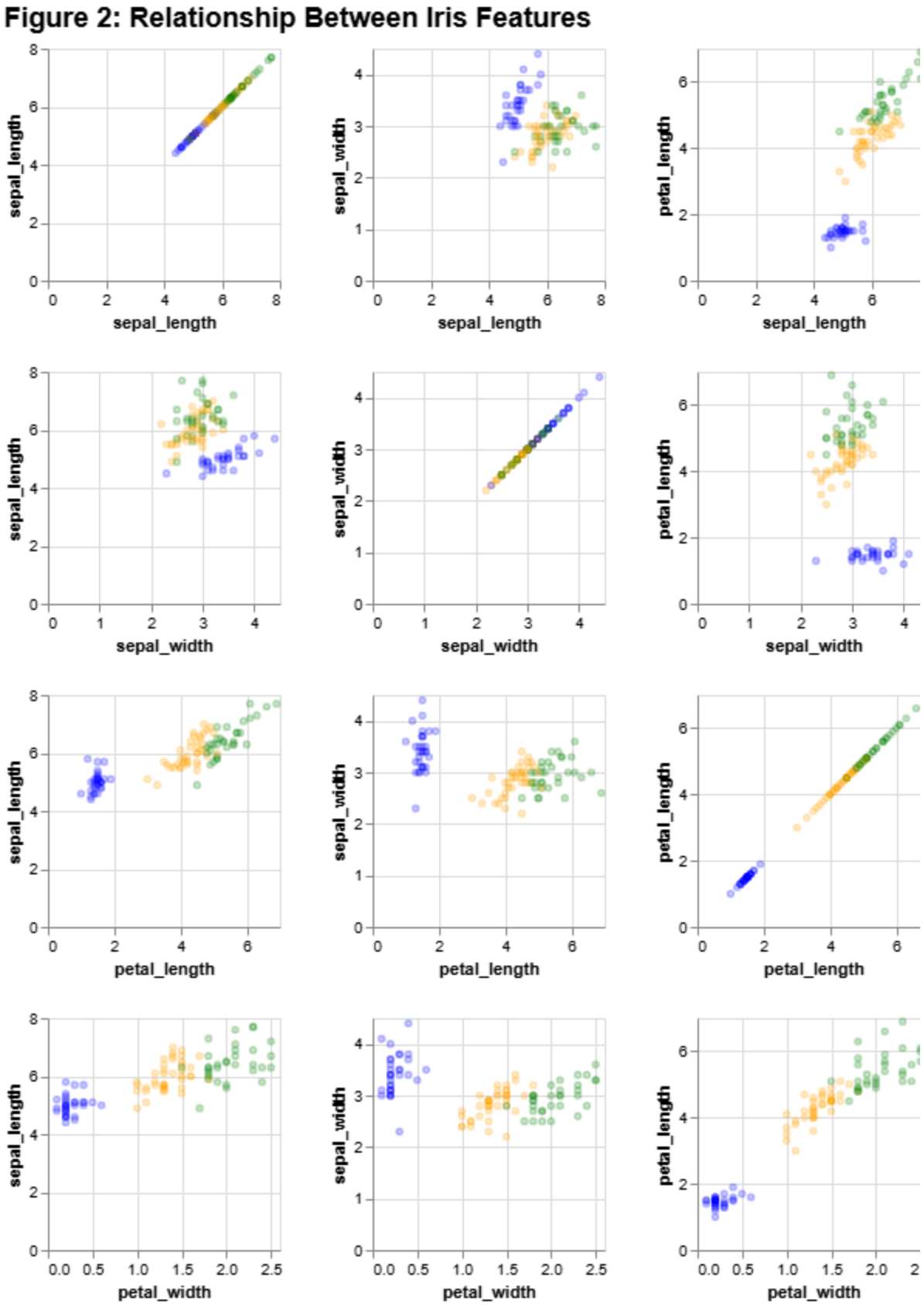

## Pairwise Plot

Pairwise plots show the relationships between feature pair and show any clusters. The plots help identify which pair of features show the clearest separation and allows us to identify which features are the most informative for prediction.

In [13]:
```
alt.Chart(train_df).mark_point(opacity=0.3, size=10).encode(
     alt.X(alt.repeat('row')).type('quantitative'),
     alt.Y(alt.repeat('column')).type('quantitative'),
     color = alt.Color(
        "species",
        scale=alt.Scale(
            domain=['setosa', 'versicolor', 'virginica'],
            range=['blue', 'orange', 'green'])
        ).title("Species")
).properties(
    width=150,
    height=150
```

```
).repeat(
    column=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
    row=['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
).properties(
    title = alt.TitleParams(
        text = "Figure 2: Relationship Between Iris Features",
        fontSize = 18
    )
)
```

Out[13]:



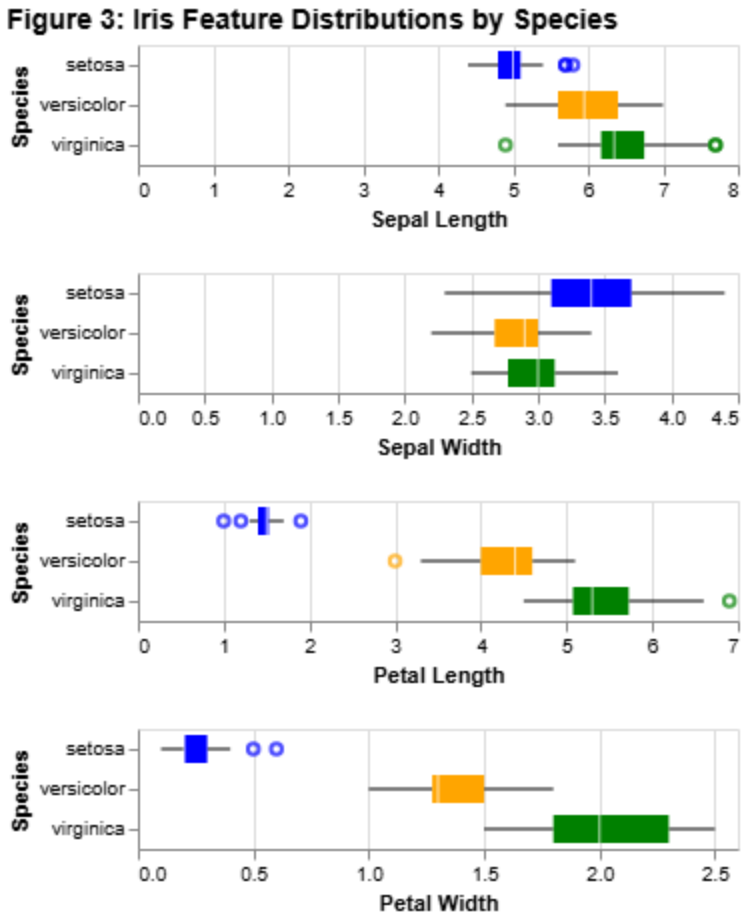

## Boxplot

Boxplots show how each feature varies across species. The median, spread, outliers, and overlaps helps identify features that are most predictive for classifying the Iris species.

In [14]:

```
sepal_length_BP = alt.Chart(train_df).mark_boxplot().encode(
     x = alt.X("sepal_length").title("Sepal Length"),
     y = alt.Y("species").title("Species"),
    color = alt.Color("species",
    scale=alt.Scale(
            domain=['setosa', 'versicolor', 'virginica'],
            range=['blue', 'orange', 'green'])
    ).legend(None)
)

sepal_width_BP = alt.Chart(train_df).mark_boxplot().encode(
     x = alt.X("sepal_width").title("Sepal Width"),
     y = alt.Y("species").title("Species"),
    color = alt.Color("species",
    scale=alt.Scale(
            domain=['setosa', 'versicolor', 'virginica'],
            range=['blue', 'orange', 'green'])
    ).legend(None)
)

petal_length_BP = alt.Chart(train_df).mark_boxplot().encode(
     x = alt.X("petal_length").title("Petal Length"),
     y = alt.Y("species").title("Species"),
    color = alt.Color("species",
    scale=alt.Scale(
            domain=['setosa', 'versicolor', 'virginica'],
            range=['blue', 'orange', 'green'])
    ).legend(None)
)

petal_width_BP = alt.Chart(train_df).mark_boxplot().encode(
     x = alt.X("petal_width").title("Petal Width"),
     y = alt.Y("species").title("Species"),
    color = alt.Color("species",
    scale=alt.Scale(
            domain=['setosa', 'versicolor', 'virginica'],
            range=['blue', 'orange', 'green'])
    ).legend(None)
)

(sepal_length_BP & sepal_width_BP & petal_length_BP & petal_width_BP).properties(
    title = alt.TitleParams(
        text = "Figure 3: Iris Feature Distributions by Species",
        fontSize = 14
    )
)
```

Out[14]:




# Model Training

Performs classification or regression analysis and then plot.

In [15]:
```
X_train = train_df.drop('species', axis=1)
y_train = train_df['species']

X_test = test_df.drop('species', axis=1)
y_test = test_df['species']
```

In [16]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import RandomizedSearchCV
import pandas as pd

scaling = StandardScaler()

#Decision Tree

param_grid = {
    "decisiontreeclassifier__max_depth": range(1,20)
    }
```

```
pipe = make_pipeline(scaling, DecisionTreeClassifier())
ds_random_search = RandomizedSearchCV(
            pipe, param_distributions=param_grid, n_jobs=-1, n_iter=50, cv=5, retur
    )

ds_random_search.fit(X_train, y_train)
pd.DataFrame(ds_random_search.cv_results_)[['mean_fit_time', 'mean_score_time', 'pa
                                            'mean_test_score', 'mean_t
```

```
/Users/gaurang/miniforge3/envs/522_project/lib/python3.11/site-packages/sklearn/mode
l_selection/_search.py:317: UserWarning: The total space of parameters 19 is smaller
than n_iter=50. Running 19 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

Out[16]:

| | mean_fit_time | mean_score_time | param_decisiontreeclassifier__max_depth | mean_test_scc |
|---|---|---|---|---|
| **1** | 0.002980 | 0.001481 | 2 | 0.9614 |
| **9** | 0.002352 | 0.000791 | 10 | 0.9514 |
| **10** | 0.002199 | 0.001115 | 11 | 0.9514 |
| **17** | 0.003499 | 0.000559 | 18 | 0.9514 |
| **16** | 0.004050 | 0.001161 | 17 | 0.9514 |

In [17]:

```
#K-NN classifer

param_grid = {
    "n_neighbors": range(1,20),
    }

param_grid = {
    "kneighborsclassifier__n_neighbors": range(1,20)
    }

pipe = make_pipeline(scaling, KNeighborsClassifier())
knn_random_search = RandomizedSearchCV(
            pipe, param_distributions=param_grid, n_jobs=-1, n_iter=50, cv=5, retur
    )

knn_random_search.fit(X_train, y_train)
pd.DataFrame(knn_random_search.cv_results_)[['mean_fit_time', 'mean_score_time', 'p
                                            'mean_test_score', 'mean_t
```

```
/Users/gaurang/miniforge3/envs/522_project/lib/python3.11/site-packages/sklearn/mode
l_selection/_search.py:317: UserWarning: The total space of parameters 19 is smaller
than n_iter=50. Running 19 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(
```

Out[17]:

| | mean_fit_time | mean_score_time | param_kneighborsclassifier__n_neighbors | mean_test_sc |
|---|---|---|---|---|
| **2** | 0.002227 | 0.007484 | 3 | 0.9704 |
| **10** | 0.003909 | 0.001557 | 11 | 0.9704 |
| **3** | 0.001929 | 0.001235 | 4 | 0.9609 |
| **4** | 0.001993 | 0.003000 | 5 | 0.9609 |
| **9** | 0.002391 | 0.001084 | 10 | 0.9514 |

In [18]:

```
decision_tree = ds_random_search.best_estimator_
knn = knn_random_search.best_estimator_

#Test on test set for both Classifcation mode
from sklearn.metrics import confusion_matrix

print("Decision tree model accuracy on test set: ", decision_tree.score(X_test, y_t

cm_ds = pd.DataFrame(
    confusion_matrix(y_test, decision_tree.predict(X_test)),
    index=decision_tree.classes_,
    columns=decision_tree.classes_
)
print("Decision tree model confusion matrix predict on test set:")
print(cm_ds)

print('\n')

print("KNN model accuracy on test set: ", knn.score(X_test, y_test))
cm_knn = pd.DataFrame(
    confusion_matrix(y_test, knn.predict(X_test)),
    index=knn.classes_,
    columns=knn.classes_
)
print("K-NN confusion matrix predict on test set:")
print(cm_knn)
```

```
Decision tree model accuracy on test set:  0.9333333333333333
Decision tree model confusion matrix predict on test set:
            setosa  versicolor  virginica
setosa          18           0          0
versicolor       0          10          0
virginica        0           3         14


KNN model accuracy on test set:  0.9555555555555556
K-NN confusion matrix predict on test set:
            setosa  versicolor  virginica
setosa          18           0          0
versicolor       0           9          1
virginica        0           1         16
```

# Discussion and Analysis

In our investigation of the Iris dataset, we establish whether these features are strong predictors for Iris species classification and rank their predictive power. This discussion begins with the insights derived from our in-depth exploratory data analysis, then the outcomes of our machine learning models, and finally exploring the implications for biological classification and machine learning practices for future research.
To summarize the data, the Iris dataset consists of 150 samples with measurements for four features: sepal length, sepal width, petal length, and petal width, across three evenly distributed species: Setosa, Versicolor, and Virginica.

In our exploratory data analysis (EDA), the summary statistics show that petal measurements (length and width) exhibit more obvious differences across species compared to sepal feature measurements. Sepal measurements are not as varied across species and thus provide weaker separation when distinguishing species. Our visualizations of bar plots, pairwise feature plots, and boxplots confirm these patterns, they each reveal the power of petal measurement for distinguishing species during classification. Specifically, the boxplot shows how the species Setosa is easily distinguishable from just its petal measurements.

Both Decision Tree and KNN models perform strongly on this Iris test set (95% vs 93% accuracy, respectively). Petal and sepal feature differences allow decent separation between species, especially between the species Versicolor and Virginica. The misclassifications in confusion matrices show this overlap. The cross-validation results suggest that Decision Trees with max_depth values from 4 to 18 fit the training data perfectly while achieving similarly high validation accuracy (94.3%) on unseen folds. For the KNN hyperparameter tuning, findings show that choosing n_neighbors between 4 and 12 outputs a classification accuracy around 97% on cross-validation test scores.

Examining our confusion matrix for model accuracies:

- In the decision tree test performance, Setosa was classified perfectly (as it is the most distinguishable, while 3 virginica got misclassified as versicolor.
- In our KNN model, we find that setosa is perfect again, and KNN misclassifies: 1 versicolor as virginica, and 2 virginica as versicolor.
- This is expected as the two classes (virginica and versicolor) overlap in features.

Our decision tree model is slightly better than KNN, with better accuracy, fewer total mistakes, perfect versicolor classification, though same performance on setosa and virginica. This suggests high but not perfect predictability of iris species from its measurements, with room for improving classification on overlapping species using more advanced methods or additional features. These results demonstrate a typical well-performing classification pipeline for the Iris dataset, validating the exploratory and modeling insights

The findings that petal measurements strongly separate iris species and have high model accuracy are expected and align with prior knowledge and research on the Iris dataset. The petal length and width consistently show clearer distinctions among the species, particularly

separating Setosa from Versicolor and Virginica, however, Sepal measurements tend to show more overlap and offer weaker discriminatory power.

However, there are some limitations associated with our study, some key limitations of the dataset and the related findings include: The dataset contains only 150 samples with 4 numeric features and 3 species classes, which is considered quite small and simple compared to data on other plant species. This limits the complexity of patterns that can be learned and makes it less applicable to more complex classification tasks. Its size also prevents it from being useful for complex machine learning techniques like deep learning, which require larger datasets. Only sepal and petal measurements are considered, and other potentially informative features such as genetic data, environmental variables, or flower color are not recorded as features in the data. This constrains prediction to specific traits and may limit generalizability to other flowers or datasets
It appears that the species Setosa is clearly separable, while Versicolor and Virginica show overlap in some measurements, leading to some classification errors and ambiguity that models must handle or risk making errors in The dataset is clean with no missing values or measurement noise, which is not typical/ normal in many real datasets. This means models trained here may be optimistic compared to actual real world performance So while appropriate for demonstrating classification approaches and feature importance, the Iris dataset's limitations prevent our model from being broadly applicable and generalizable, which must be considered when interpreting and generalizing findings

The impact of these findings is significant in practical feature selection for machine learning. They show the importance of choosing the most informative features for classification tasks, improving model performance and simplifying models by focusing on fewer but more relevant measurements. This can aid in resource efficiency and interpretability in biological studies, botany, and related scientific work.
This brings up future questions that could use further research:

- How do more advanced machine learning algorithms (e.g., SVM) compare in classification accuracy using petal vs sepal features?
- Can combining petal and sepal features with additional biological or environmental data improve model robustness or reveal deeper insights?
- How well do these findings generalize to other flower species or datasets—can similar feature importance patterns be found?
- Could unsupervised learning reveal new subgroups or variations in iris species beyond the three classical ones using these or other features?

In summary, these findings are expected and show the predictive importance of petal dimensions in iris classification. They have implications for feature selection and model design, and point to further possible research on flower classification and its related biological questions

# References


- Iris dataset reference: https://archive.ics.uci.edu/dataset/53/iris
- DSCI573 notes: https://pages.github.ubc.ca/mds-2025-26/DSCI_573_feat-model-select_students/
- DSCI571 notes: https://pages.github.ubc.ca/mds-2025-26/DSCI_571_sup-learn-1_students/
- DSCI531 notes: https://pages.github.ubc.ca/mds-2025-26/DSCI_531_viz-1_students/
- scikit-learn documentation: https://scikit-learn.org/stable/