PREDICTIVE ANALYTICS

SEC 01

ALY 6020, FALL 2022

WEEK 1: MODULE 1 MIDWEEK PROJECT 1

SUBMITTED BY: Akash Raj, Ananya Sharma, Vaibhav Arora, Vaibhav Jain

CRN: 70916

SUBMITTED TO: Dr. Mary Donhoffner

DATE: September 26, 2022

INTRODUCTION

This report explores the iris dataset and creates kNN predictive data model to predict the species of the flower. This report also includes importing the dataset and exploratory data analysis using python.

The data set has three iris species Iris Setosa, Iris Virginica, and Iris Versicolor. It contains 150 rows and 5 columns. This project aims to understand the nuances of the Nearest Neighbor algorithm and classify the models accurately. (Real Python, 2022)

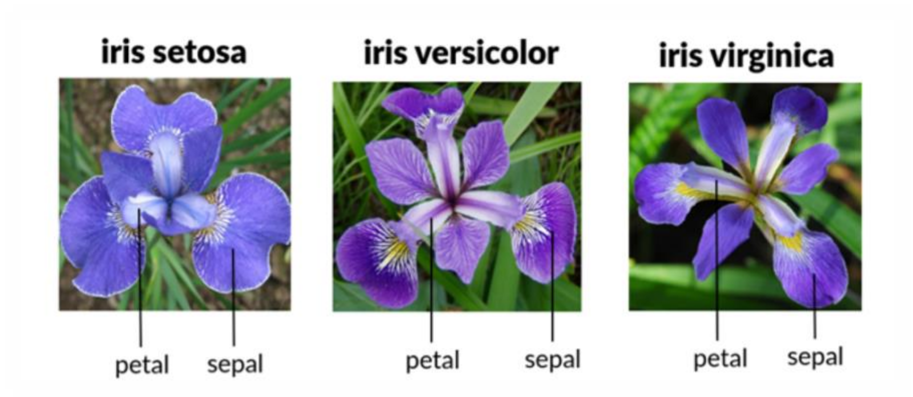
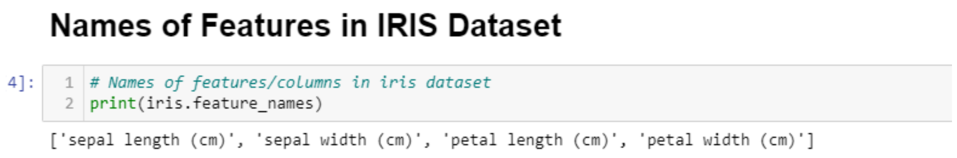


Fig 1: Iris Data Set (Naeem, 2022)

The iris data set is available in the SkLearn library and we have loaded the same using the **load_iris()** function.



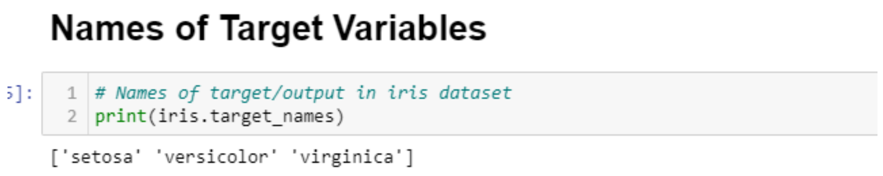
Names of Features in IRIS Dataset

```
4]:  # Names of features/columns in iris dataset
     print(iris.feature_names)

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Fig 2: Features

The data set has four distinct features, namely Sepal Length, Petal Length, Petal Width, and Petal Length.



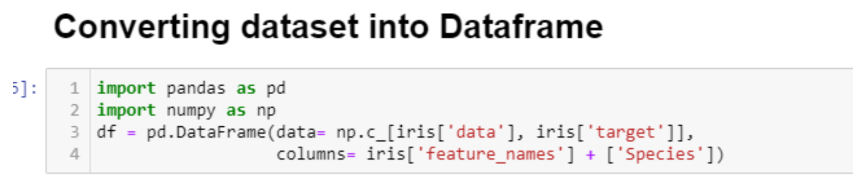
Names of Target Variables

```
5]:  # Names of target/output in iris dataset
     print(iris.target_names)

['setosa' 'versicolor' 'virginica']
```

Fig 3: Target Variables

Additionally, the data set has three targets that we are trying to identify:
0: Setosa
1: Versicolor
2: Virginia

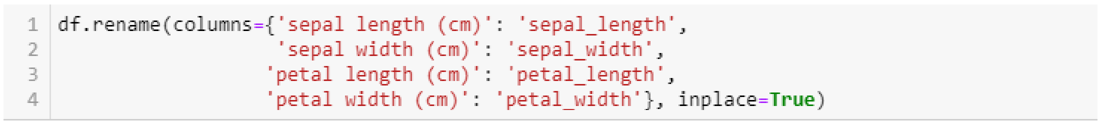


Converting dataset into Dataframe

5]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                  columns= iris['feature_names'] + ['Species'])
```

Fig 4: Data frame

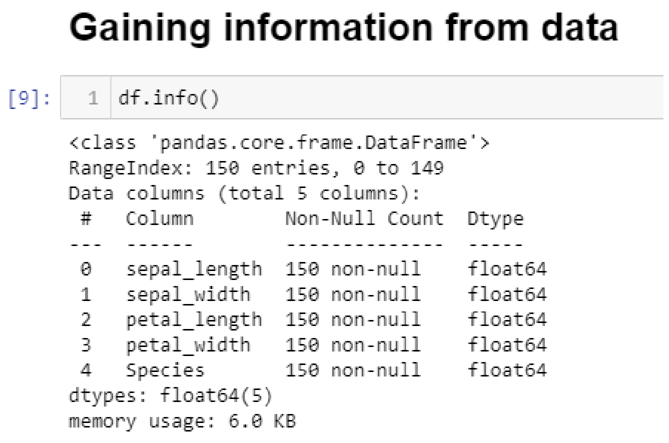
The dataset must be converted to data frame format for further modifications and performing any algorithm fitting henceforth.



```
df.rename(columns={'sepal length (cm)': 'sepal_length',
                   'sepal width (cm)': 'sepal_width',
                  'petal length (cm)': 'petal_length',
                  'petal width (cm)': 'petal_width'}, inplace=True)
```

Fig 5: Rename columns

After conversion into a data frame, we have renamed a few columns for ease of name convention.



Gaining information from data

[9]:
```
df.info()
```

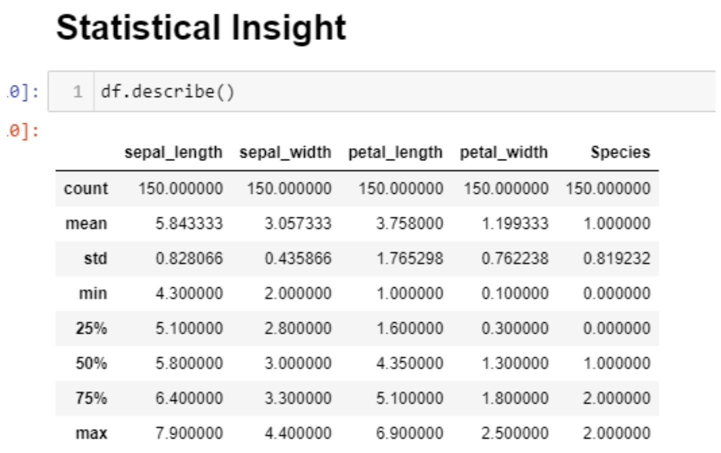
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   Species       150 non-null    float64
dtypes: float64(5)
memory usage: 6.0 KB
```

Fig 6: Info

The **info ()** function details the data type, columns, and non-Null values.

EXPLORATORY DATA ANALYSIS

Now we understand the dataset including distribution, discover a pattern, and if there are any outliers or anomalies (Sonal, 2021). First, we use the **describe ()** function to check descriptive statistics, mean, and dispersion. The **describe ()** function (Fig 7) performs statistical analysis of the data set and gives the mean, maximum value, minimum value, standard deviation, and the various percentiles.



Statistical Insight

```
0]:  df.describe()
0]:
```

	sepal_length	sepal_width	petal_length	petal_width	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Fig 7: Describe()

The highest means and the maximum value is the sepal length, whereas its the lowest for petal width. However, the standard deviation for the two is relatively close. Similarly, the average value of sepal width and petal length is similar whereas there is a significant difference in max values with 4.4 and 6.9 respectively.

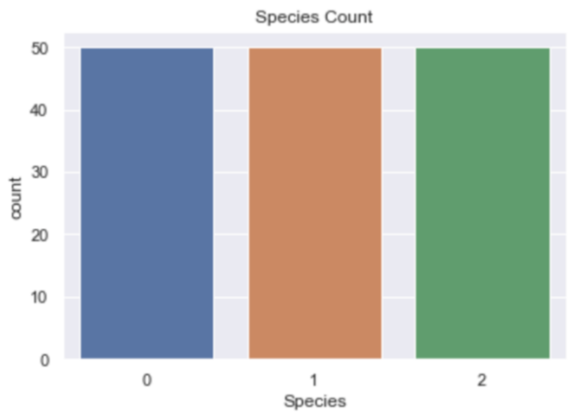


Fig 8: Count Plot

First, we want to see the count of species, and to do so we created a count plot. With the graph, we can see all the species have the same count which is 50.

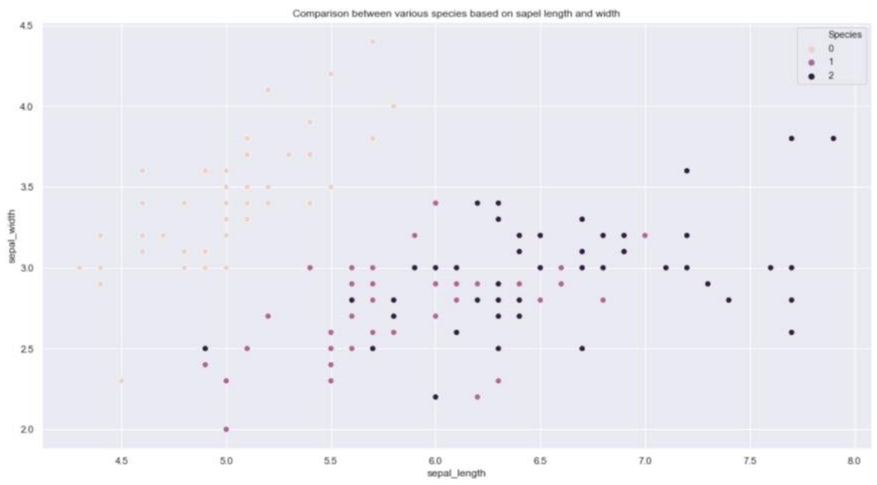


Fig 9: Scatter Plot Univariate Sepal

Now we want to understand the comparison of various species based on sepal and petal. First, we see the comparison of simple length and width grouped by species. The setosa species has a higher weight but the length is smaller for sepal. Versicolor lies in the middle region for both length and sepal width. Where is Virginica Species having a larger sepal length, but the width is smaller.

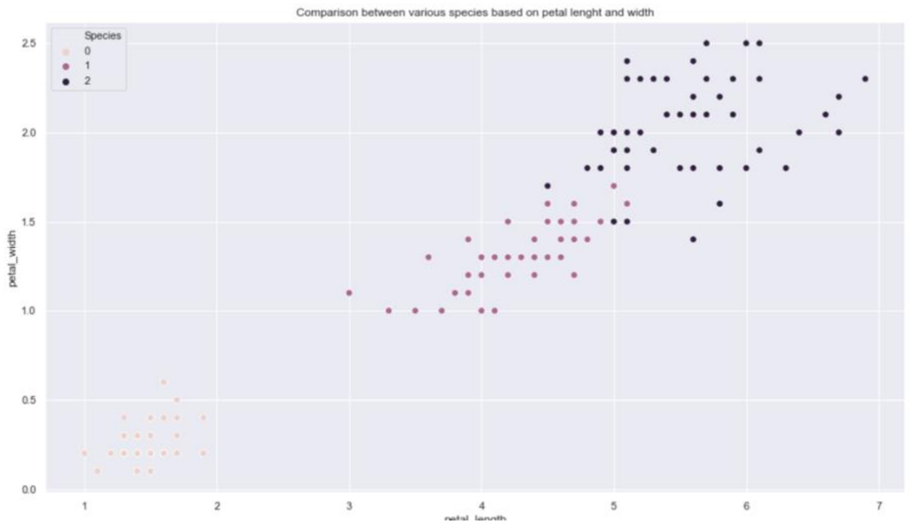


Fig 10: Scatter Plot Univariate Petal

For the comparison of various species based on the length and width of the petal, we find that Setosa pieces have a smaller Length and width of the petal. Versicolor species' length and width are light in the middle range whereas Virginica species has the highest length and width of the petal.

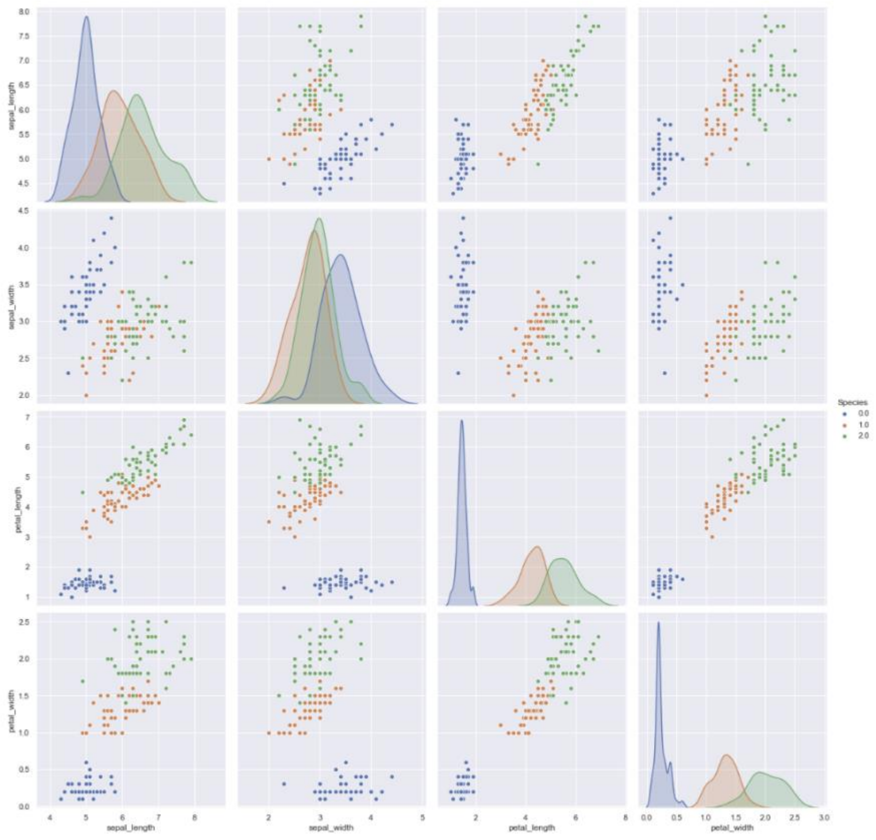


Fig 11: Pair Plot

Pair plots can be used to scatter plot multiple features instead of plotting them one by one. There are the following observations from the pair plot.

- The most useful feature that can be used to identify the species of the flower is petal length and petal width as there is a clear distinction in the species.
- Species 0 which is Setosa Can be identified distinctly whereas Virginica and Versicolor are overlapped.

Now we plot the correlation heat map to check if there is any relationship between the three variables and if one variable affects the other. We observe that simple length and width are slightly correlated with each other. However, this does not affect the data.

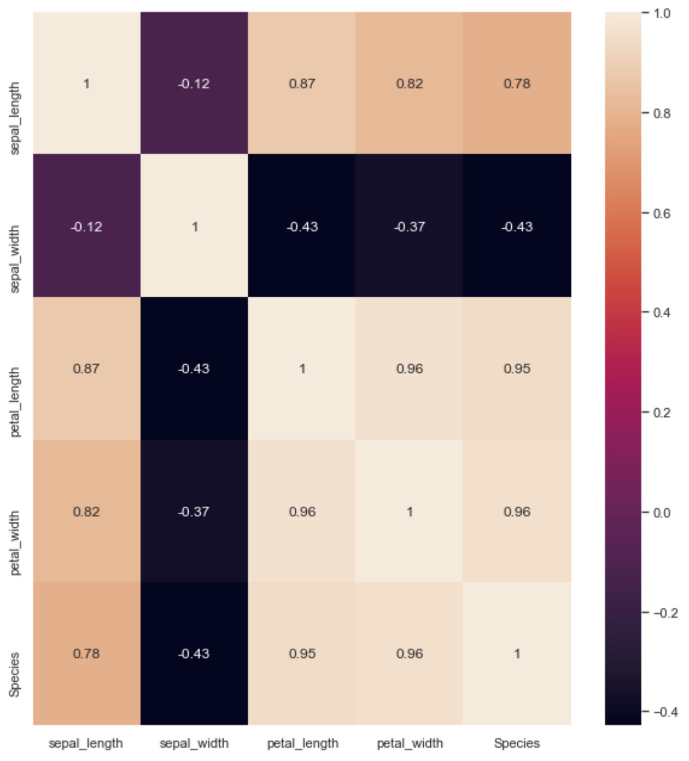


Fig 12: Correlation Heat Map

Box plots are useful for visualizing describe() function which shows minimum, maximum, medium, and upper and lower quartiles.

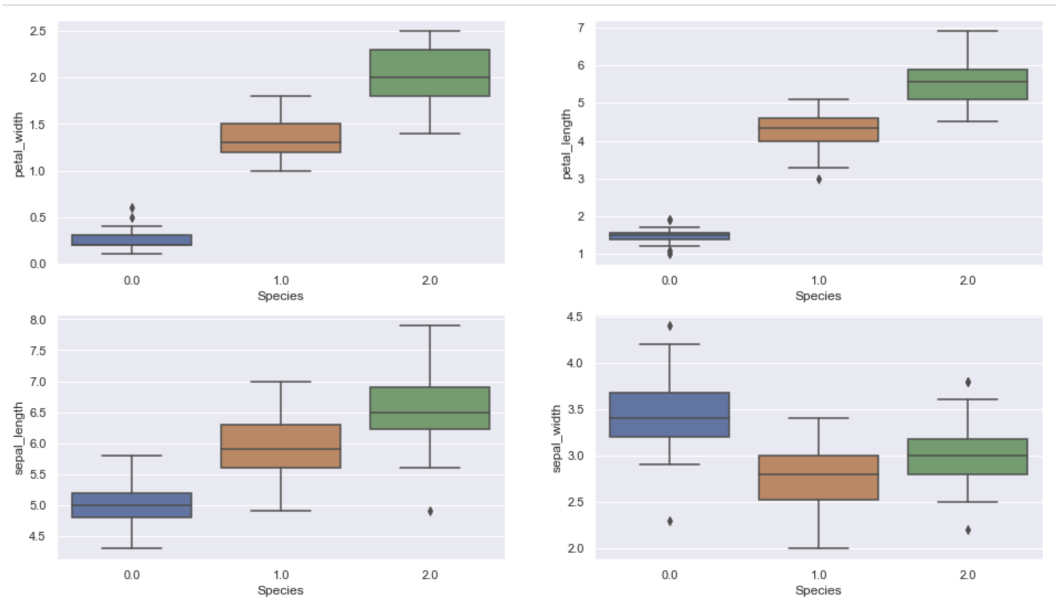


Fig 13: Box Plot

From the box plot of the four features grouped by species, we observe that Setosa has less distribution and smaller features. Versicolor lies in the middle range and has average features. Virginica has high distribution and higher range for petal width, petal length, and sepal length.

Now we create a violin plot to Check the distribution of data. It helps us understand the density of the length and width of each species.

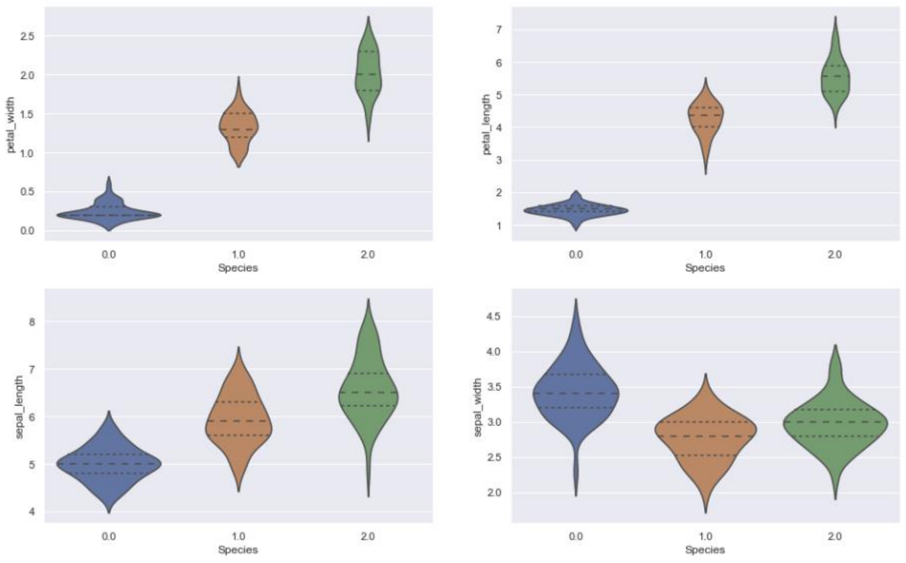


Fig 14: Violin Plot

- Setosa species have less density in petal length and petal width.
- Versicolor species have medium Distribution for petal length and width whereas there is a high distribution for sepal width.
- Virginica species have high distribution for sepal length and width whereas the distribution is relatively low for petal length and width.

PREDICTIVE MODELLING

The objective of the analysis is to predict the species using kNN, and the attributes of the flower as the independent variable. The following steps are required to build the and evaluate the kNN model:

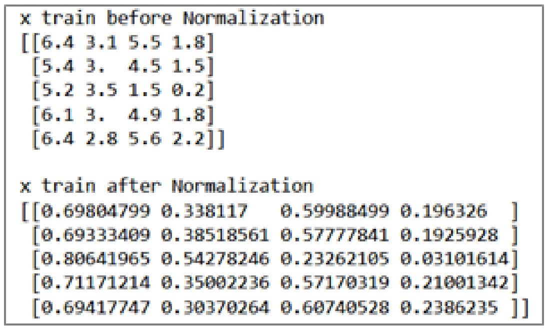
- Splitting the data into test and train,
- Normalizing the independent features,
- Building a kNN model, and identifying the optimal value of k,
- Predicting the test results, and
- Evaluating the model

Part 1: Training and Testing Split

To ensure that the model is robust, the kNN model should be built on the training set, and evaluated on the testing set. The function ‘train_test_split’ from the module ‘model_selection’ of ‘Sklearn’ is used to split the data into the training and testing set. There are total of 150 observations in the data, which will be randomly split into 80% training data and 20% testing data. After the split, the training data has 120 records, and the testing data has 30 records.

Part 2: Feature normalization

In general, machine learning models converge faster when the features are normalized. Normalizer transforms all the feature value between -1 and 1. The function ‘Normalizer’ from the module ‘preprocessing’ is used to perform the feature normalization. Different values of the feature are transformed between -1 and 1.



```
x train before Normalization
[[6.4 3.1 5.5 1.8]
 [5.4 3.  4.5 1.5]
 [5.2 3.5 1.5 0.2]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.2]]

x train after Normalization
[[0.69804799 0.338117   0.59988499 0.196326  ]
 [0.69333409 0.38518561 0.57777841 0.1925928 ]
 [0.80641965 0.54278246 0.23262105 0.03101614]
 [0.71171214 0.35002236 0.57170319 0.21001342]
 [0.69417747 0.30370264 0.60740528 0.2386235 ]]
```

Fig 15: Training Data before and after Normalization

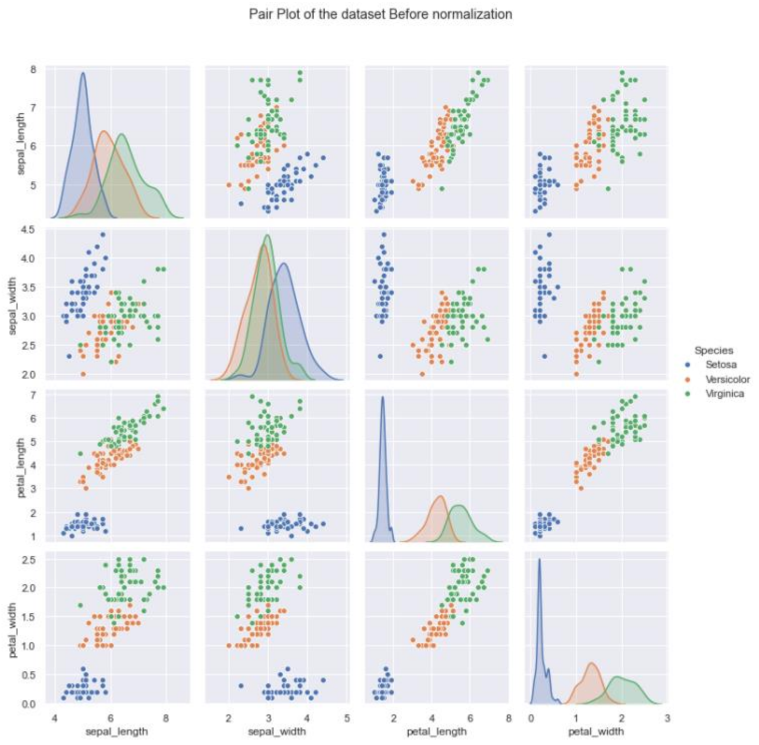


Fig 16: Pair Plot before Normalization

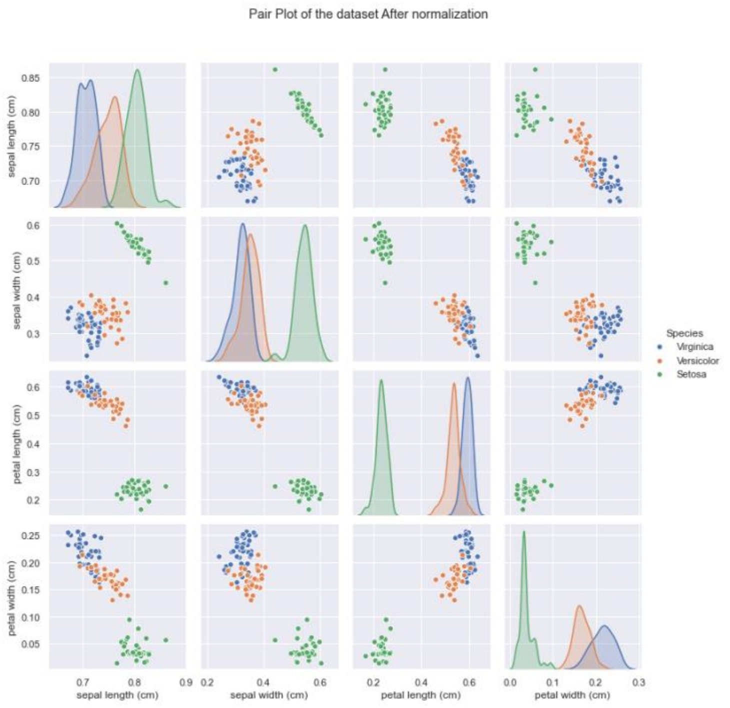
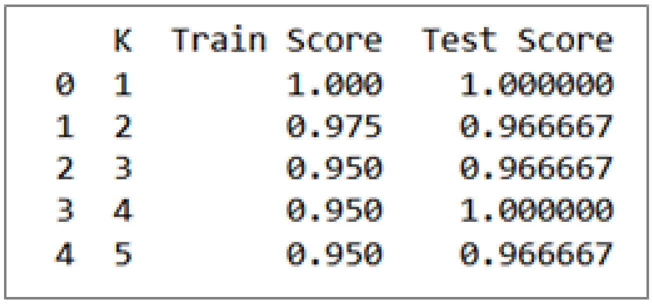


Fig 17: Pair Plot After Normalization

The distribution of the features remail similar even after normalization, only the values of the axis changes as the scale changes.

Part 3: Building a kNN model

The k-nearest neighbors (KNN) algorithm is a classification technique for classifying a data point into different techniques based on what category the data points nearest to it belong to. In this case, the different categories are 'Virginica', 'Versicolor', and 'Setosa'. The function 'KNeighborsClassifier' from 'Sklearn' is used to fit a kNN model. It is also important to identify the optimal value of 'k', such that it not very large or small. The kNN model is run for different values of k on train data and validated on the test data.



```
   K  Train Score  Test Score
0  1        1.000    1.000000
1  2        0.975    0.966667
2  3        0.950    0.966667
3  4        0.950    1.000000
4  5        0.950    0.966667
```

Fig 18: K value, train and test accuracy

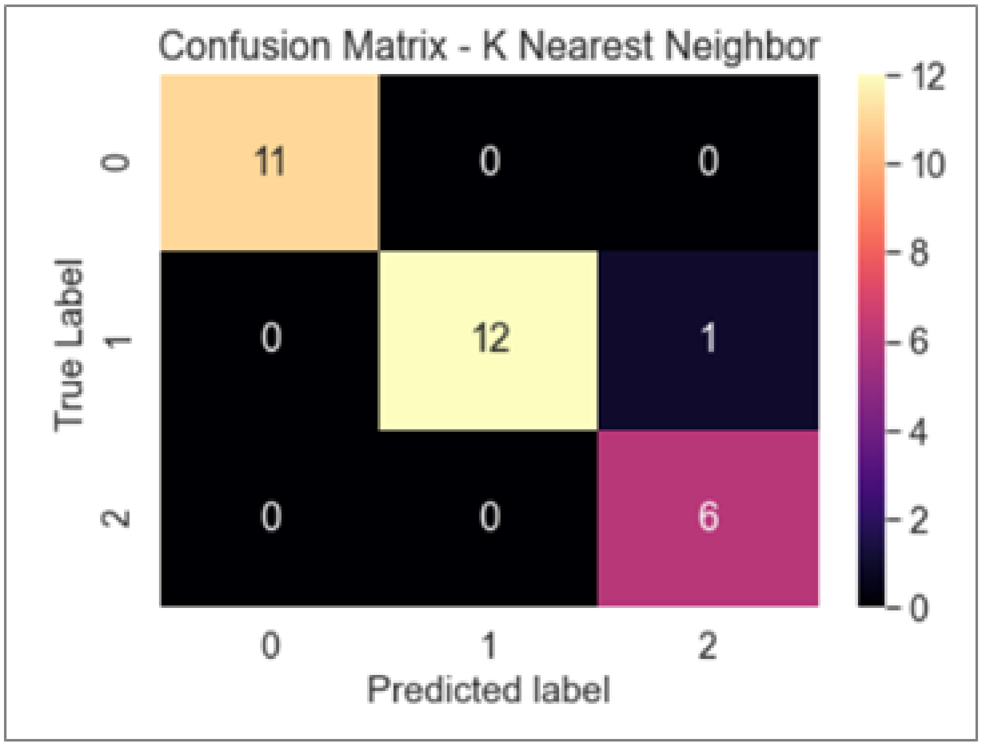
The value chosen for 'k' in the final kNN model is 5, for which the training score is 95% and the test score is 96.6%.

Prediction And Model Evaluation

Model evaluation involves investigating a machine learning model's performance and strengths and weaknesses using various evaluation metrics. Model evaluation is necessary to assess the efficacy of a model during the early stages of research, and it also plays a role in model monitoring.

<u>**Confusion matrix**</u>

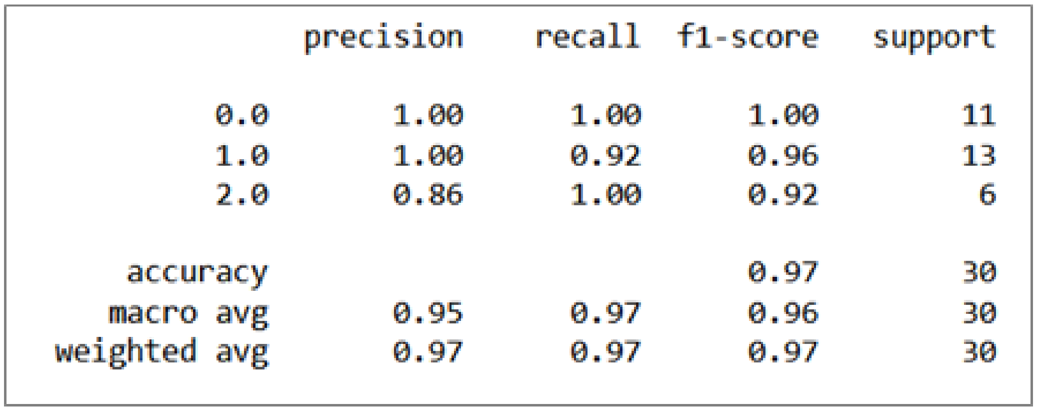
The final model is used to predict the classes of the data points in the test set. The predicted and the actual values of the target categories are used to create a confusion matrix and a classification report. The confusion matrix can be created using the ‘confusion_matrix’ from the ‘sklearn metrics’ module.



The categories for 29 out of 30 observations have been correctly classified by the kNN model. Only one data point that belonged to category 1 was incorrectly classified as category 2.

Classification report

The confusion matrix is a very good tool to get a general sense of correct and incorrect predictions on the test set. However, it's important to look at all the classification metrics to thoroughly evaluate the model. The classification report can be created using the function 'classification_report' from the 'sklearn metrics' module.



```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        11
         1.0       1.00      0.92      0.96        13
         2.0       0.86      1.00      0.92         6

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

The overall accuracy of the model is ~97%, whereas the precision of the individual categories is 100%, 100%, and 86%. Similarly, the recall of the individual categories is 100%, 92%, and 100%.

QUESTIONS

1. **What was the overall accuracy of the model?**

   The overall accuracy of the model is 97%.

2. **What was the accuracy of each type of iris?**

   The accuracy of each type of iris is 100%, 92%, and 100%. Precision of each type of iris is 100%, 100%, and 85% respectively. Recall of each type of iris is 100%, 92%, and 100% respectively.

3. Would you classify the model as a good model or not?

The final kNN model with k value as 5 has an overall accuracy of 95% on the training set, and 97.5% on the test set. This means that the model is neither under fitting not over-fitting. This model is good and can be generalized.

REFERENCES

1. Real Python. (2022, September 1). The k-Nearest Neighbors (kNN) Algorithm in Python. Retrieved September 26, 2022, from https://realpython.com/knn-python/#knn-is-a-supervised-learner-for-both-classification-and-regression

2. Sonal, S. (2021, September 20). Importance of exploratory data analysis before ML Modelling. Eduonix Blog. Retrieved September 26, 2022, from https://blog.eduonix.com/bigdata-and-hadoop/importance-exploratory-data-analysis-ml-modelling/

3. *Iris_Knn[Keshav Singh]. (2017, July 16). k-Nearest Neighbor kNN with IRIS dataset* [Video]. YouTube. Retrieved September 26, 2022, from https://www.youtube.com/watch?v=h5dda-gRpVk

4. *Kapur, I. (2021, December 14). k-NN on Iris Dataset - Towards Data Science*. Medium. Retrieved September 25, 2022, from https://towardsdatascience.com/k-nn-on-iris-dataset-3b827f2591e

5. *Naeem, A. (2022, March 13). Iris Dataset Classification Using 3 Machine Learning Algos*.Embedded Robotics. Retrieved September 26, 2022, from https://www.embeddedrobotics.com/iris-dataset-classification/

6. *Intro to Machine Learning in R (K Nearest Neighbours Algorithm)*. (*2020, January 15*). Retrieved September 24, 2022, from https://ourcodingclub.github.io/tutorials/machine-learning/

7. *Fit k-nearest neighbor classifier - MATLAB fitcknn. (2022, May 14*). Retrieved September 26, 2022, from https://www.mathworks.com/help/stats/fitcknn.html

APPENDIX

Importing Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
sns.set()
from sklearn.datasets import load_iris
iris= load_iris()
```

Seperating Features and target Variables

```
# Store features matrix in X
X= iris.data
#Store target vector in
y= iris.target
```

Names of Features in IRIS Dataset

```
# Names of features/columns in iris dataset
print(iris.feature_names)
```

Names of Target Variables

```
# Names of target/output in iris dataset
print(iris.target_names)
```

Converting dataset into Dataframe

```
import pandas as pd
import numpy as np
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                  columns= iris['feature_names'] + ['Species'])
df['Species'] = df['Species'].astype(int)
df.head(5)
df.rename(columns={'sepal length (cm)': 'sepal_length',
                   'sepal width (cm)': 'sepal_width',
                  'petal length (cm)': 'petal_length',
                  'petal width (cm)': 'petal_width'}, inplace=True)
```

Gaining information from data

```
df.info()
```

Statistical Insight

```
df.describe()
```

Checking Duplicated data

```
df[df.duplicated()]
```

Checking the balance

```
df["Species"].value_counts()
```

Exploratory Data Analysis

```
plt.title("Species Count")
sns.countplot(df["Species"])
```

Univariate Analysis

```
# >> Comparison between various species based on sepal length and width
plt.figure(figsize=(17,9))
plt.title("Comparison between various species based on Sepal length and width")
sns.scatterplot(df["sepal_length"],df["sepal_width"],hue = df["Species"],palette=['green','brown','
magenta'],s=50)
```

```
# >> Comparison between various species based on petal length and width
plt.figure(figsize=(16,9))
plt.title("Comparison between various species based on petal lenght and width")
sns.scatterplot(df["petal_length"], df["petal_width"], hue = df["Species"], s= 50)
```

Bi-Variate Analysis

```
sns.pairplot(df,hue="Species",height=4)
```

Checking Correlation

```
plt.figure(figsize=(10,11))
sns.heatmap(df.corr(),annot=True)
plt.plot()
```

Box plots to know about distribution

```
ig, axes = plt.subplots(2, 2, figsize=(16,9))
sns.boxplot( y="petal_width", x= "Species", data=df, orient="v" , ax=axes[0, 0])
sns.boxplot( y="petal_length", x= "Species", data=df, orient="v" , ax=axes[0, 1])
sns.boxplot( y="sepal_length", x= "Species", data=df, orient="v" , ax=axes[1, 0])
sns.boxplot( y="sepal_width", x= "Species", data=df, orient="v" , ax=axes[1, 1])
plt.show()
```

Violin Plot for checking distribution

```
# The violin plot shows density of the length and width in the species. The thinner part denotes
# that there is less density whereas the fatter part conveys higher density
fig, axes = plt.subplots(2, 2, figsize=(16,10))
sns.violinplot( y="petal_width", x= "Species", data=df, orient="v" , ax=axes[0, 0],
inner='quartile')
sns.violinplot( y="petal_length", x= "Species", data=df, orient="v" , ax=axes[0, 1],
inner='quartile')
sns.violinplot( y="sepal_length", x= "Species", data=df, orient="v" , ax=axes[1, 0],
inner='quartile')
sns.violinplot( y="sepal_width", x= "Species", data=df, orient="v" , ax=axes[1, 1],
inner='quartile')
plt.show()
```

Modeling

```
x= df.iloc[:, :-1]
y= df.iloc[:, -1]
```

Split the data into train and test sets

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y,
                                                   test_size= 0.2,
                                                   shuffle= True, #shuffle the data to avoid bias
                                                   random_state= 0)
x_train= np.asarray(x_train)
y_train= np.asarray(y_train)

x_test= np.asarray(x_test)
y_test= np.asarray(y_test)
print(f'training set size: {x_train.shape[0]} samples \ntest set size: {x_test.shape[0]} samples')
from sklearn.preprocessing import Normalizer
scaler= Normalizer().fit(x_train) # the scaler is fitted to the training set
normalized_x_train= scaler.transform(x_train) # the scaler is applied to the training set
normalized_x_test= scaler.transform(x_test) # the scaler is applied to the test set
print('x train before Normalization')
print(x_train[0:5])
print('\nx train after Normalization')
print(normalized_x_train[0:5])
## Before
# View the relationships between variables; color code by species type
di= {0.0: 'Setosa', 1.0: 'Versicolor', 2.0:'Virginica'} # dictionary

before= sns.pairplot(df.replace({'Species': di}), hue= 'Species')
before.fig.suptitle('Pair Plot of the dataset Before normalization', y=1.08)

## After
iris_df_2= pd.DataFrame(data= np.c_[normalized_x_train, y_train],
                        columns= iris['feature_names'] + ['Species'])
di= {0.0: 'Setosa', 1.0: 'Versicolor', 2.0: 'Virginica'}
after= sns.pairplot(iris_df_2.replace({'Species':di}), hue= 'Species')
after.fig.suptitle('Pair Plot of the dataset After normalization', y=1.08)
```

Running KNN for various values of n_neighbors and storing results

```
from sklearn.neighbors import KNeighborsClassifier
knn_r_acc = []
for i in range(1,6,1):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    test_score = knn.score(x_test,y_test)
    train_score = knn.score(x_test,y_test)
    knn_r_acc.append((i, test_score))
df = pd.DataFrame(knn_r_acc, columns=['K','Test Score'])
print(df)
```

Predicting the Test set results

```
y_pred = knn.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

print(cm)

# accuracy_score(y_test, y_pred)
matrix_df = pd.DataFrame(cm)
```

```
#plot the result
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10,7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")


#set axis titles
ax.set_title('Confusion Matrix - K Nearest Neighbor')
ax.set_xlabel("Predicted label", fontsize =15)
ax.set_ylabel("True Label", fontsize=15)



plt.show()
```

Accuracy of each type of iris

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```