Name: Gore Aniket Machhindra

Email: aniket.m.gore.1901@gmail.com

Role: Data Science Intern

Task 1 Beginner: Iris Flowers Classification ML Project

Content -

- Importing Libraries
- Reading Dataset
- Exploratory Data Analysis
- Visualization
- Build the Model
- Conclusion

Importing Libraries

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
#%matplotlib inline

import warnings; warnings.filterwarnings('ignore')
```

Reading Dataset

```
data = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.dat
```

Exploratory Data Analysis

```
data.head()
```

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

```
data.columns
```

```
Index(['5.1', '3.5', '1.4', '0.2', 'Iris-setosa'], dtype='object')
```

```
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
```

```
data.columns = columns
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

```
data.shape
```

```
(149, 5)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149 entries, 0 to 148
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  149 non-null    float64
 1   sepal_width   149 non-null    float64
 2   petal_length  149 non-null    float64
 3   petal_width   149 non-null    float64
 4   species       149 non-null    object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

```
data.isnull().sum()
```

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

```
data.describe()
```

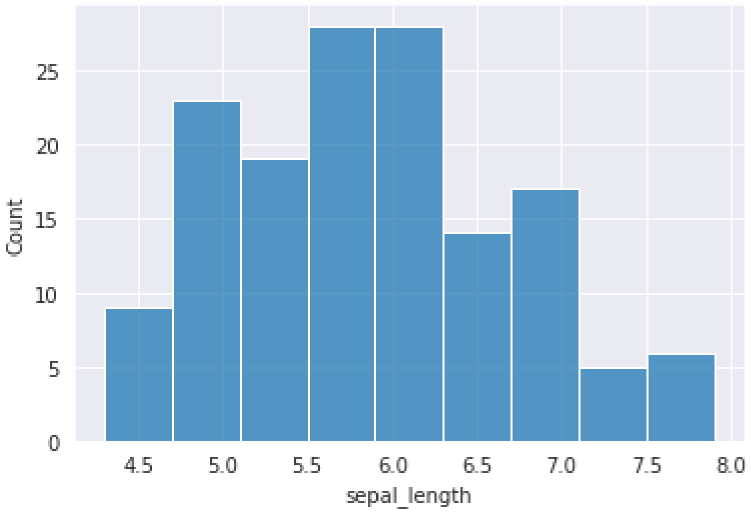
	sepal_length	sepal_width	petal_length	petal_width
count	149.000000	149.000000	149.000000	149.000000
mean	5.848322	3.051007	3.774497	1.205369
std	0.828594	0.433499	1.759651	0.761292
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Visualization

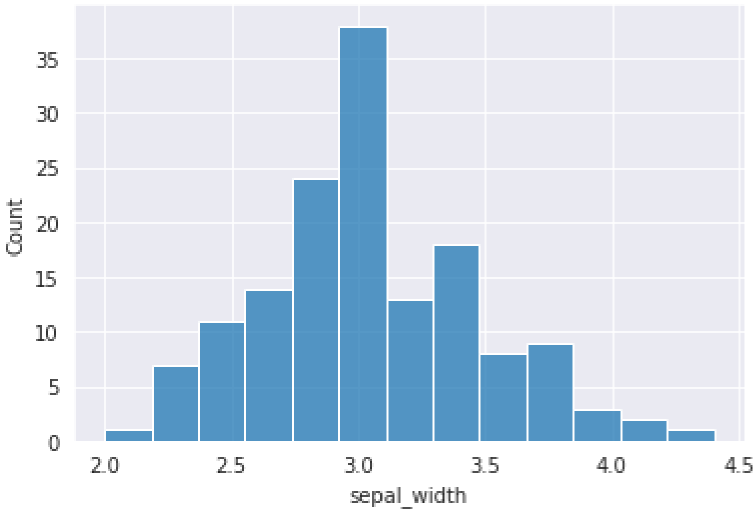
```
sns.set_style('darkgrid')
sns.heatmap(data.corr(), annot=True)
plt.title('Correlation Matrix');
```

Correlation Matrix
sepal_length
1
-0.1
0.87
0.82
1.0
0.8

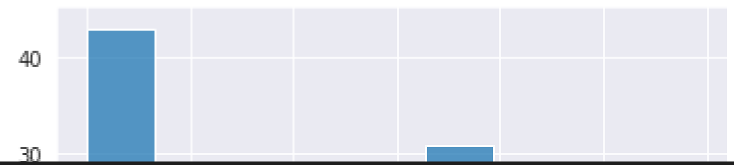
```
sns.histplot(data['sepal_length']);
```



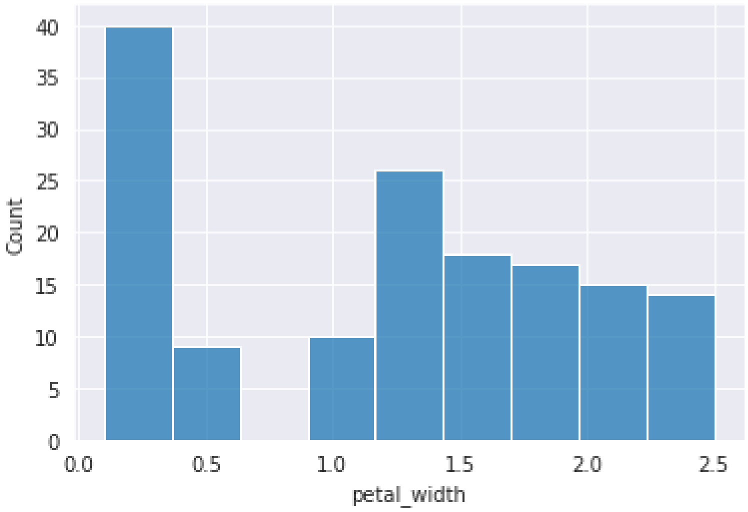
```
sns.histplot(data['sepal_width']);
```



```
sns.histplot(data['petal_length']);
```



```
sns.histplot(data['petal_width']);
```



Build the Model

```
#Preeprocessing data  -> drop species to get X, only extract species to get Y
X = data.drop('species', axis=1)
y = data['species']


# Train Test Split  -> use train_test_split()
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)


# Feature Scaling

from sklearn.preprocessing import MinMaxScaler
# make MinMaxScaler object
mms = MinMaxScaler()

# fit scalar-object over the X_train dataset
X_train = mms.fit_transform(X_train)


# use scalar-object to transform the X_train and X_test data set
X_test = mms.transform(X_test)


```

```python
# Training and Predictions
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=9)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

# Evaluating the Algorithm
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[13  0  0]
 [ 0 14  0]
 [ 0  1 10]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        13
Iris-versicolor       0.93      1.00      0.97        14
 Iris-virginica       1.00      0.91      0.95        11

       accuracy                           0.97        38
      macro avg       0.98      0.97      0.97        38
   weighted avg       0.98      0.97      0.97        38
```

Conclusion

The results show that our KNN algorithm was able to classify all the 37 out of 38 records in the test set with 97% accuracy