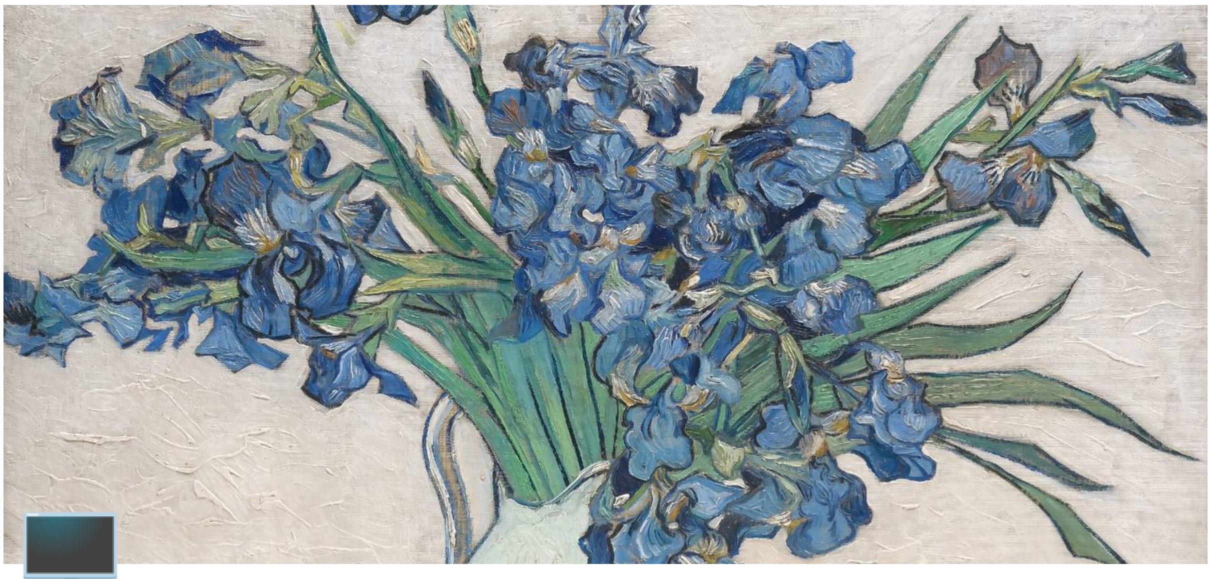


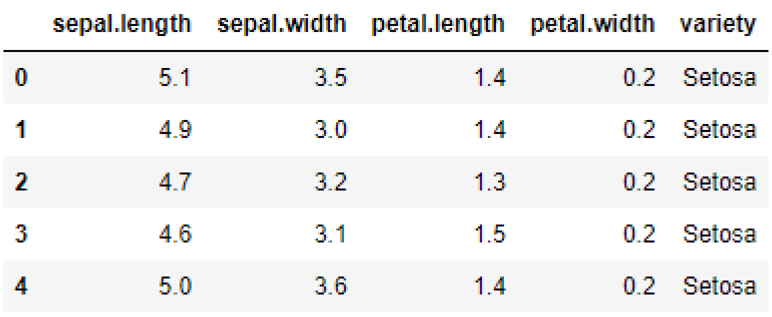
# Iris Dataset Analysis

Import the python libraries we will be needing for our analysis namely *Pandas and Matplotlib.*

```
#Importing the libraries
import pandas as pd
import matplotlib.pyplot as plt
```

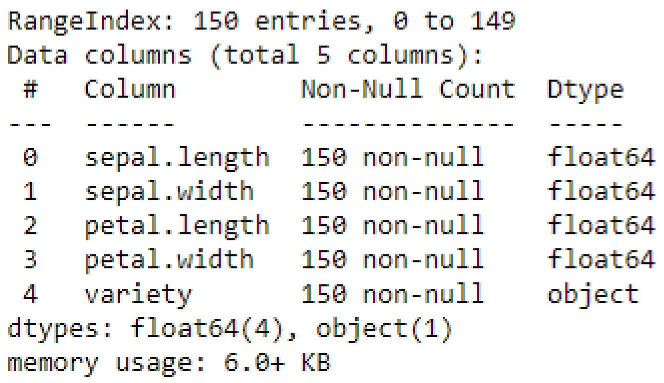
## we will be using the Iris Dataset which can be downloaded from ***here***

```
# Loading the dataset
df=pd.read_csv("iris.csv")
# Display the first five observations
df.head()
```



| | sepal.length | sepal.width | petal.length | petal.width | variety |
|---|---|---|---|---|---|
| **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 |

```
# Display the Variable Names,non-null values and their Data Types
df.info()
```



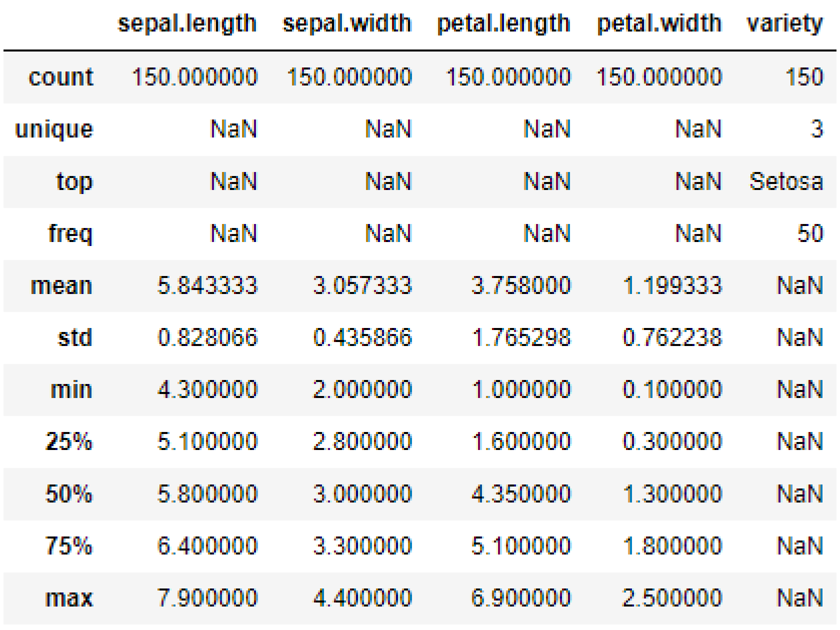
```
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
#Display the number of variables &  number of observations
print("Number of observations: ",df.shape[0])
print("Number of variables:  ",df.shape[1])
```

```
#summary of all the numerical variables
df.describe()
```

```
#summary of all the numerical variables
df.describe()
#summary of all numerical and categorical variables
df.describe(include='all')
```

and the *include* argument and assigning it the value *'all'*
 we get the summary of the categorical variables too.



| | sepal.length | sepal.width | petal.length | petal.width | variety |
|---|---|---|---|---|---|
| **count** | 150.000000 | 150.000000 | 150.000000 | 150.000000 | 150 |
| **unique** | NaN | NaN | NaN | NaN | 3 |
| **top** | NaN | NaN | NaN | NaN | Setosa |
| **freq** | NaN | NaN | NaN | NaN | 50 |
| **mean** | 5.843333 | 3.057333 | 3.758000 | 1.199333 | NaN |
| **std** | 0.828066 | 0.435866 | 1.765298 | 0.762238 | NaN |
| **min** | 4.300000 | 2.000000 | 1.000000 | 0.100000 | NaN |
| **25%** | 5.100000 | 2.800000 | 1.600000 | 0.300000 | NaN |
| **50%** | 5.800000 | 3.000000 | 4.350000 | 1.300000 | NaN |
| **75%** | 6.400000 | 3.300000 | 5.100000 | 1.800000 | NaN |
| **max** | 7.900000 | 4.400000 | 6.900000 | 2.500000 | NaN |

```
#Types of variety in dataset
df["variety"].unique()
```

```
# change categorical to numurical
df.replace({"variety":{"Setosa":0,"Versicolor":1,"Virginica":2}},inplace=True)
df["variety"].unique()
```

```
# Detecting the Missing Values
df.isnull().sum()
```

```
# To check for the duplicates in our data
df.duplicated().sum()
```

```
# To remove the duplicates values
df.drop_duplicates(inplace=True)
```

```
# Detecting Outliers
p0=df.min()
p100=df.max()
q1=df.quantile(0.25)
q2=df.quantile(0.5)
q3=df.quantile(0.75)
```

```
iqr=q3-q1
print(p0)
print(p100)
```

*What are Outliers?* Outliers are the extreme values on the low and the high side of the data. Handling Outliers involves 2 steps: Detecting outliers and Treatment of outliers.

```
sepal.length    4.3
sepal.width     2.0
petal.length    1.0
petal.width     0.1
variety         0.0
dtype: float64
sepal.length    7.9
sepal.width     4.4
petal.length    6.9
petal.width     2.5
variety         2.0
dtype: float64
```

```
# Detecting Outliers
p0=df.min()
p100=df.max()
q1=df.quantile(0.25)
q2=df.quantile(0.5)
q3=df.quantile(0.75)
iqr=q3-q1
print(p0)
print(p100)
```

**Detecting Outliers**

For this we consider any variable from our data frame and determine the *upper cut off* and the *lower cutoff* with the help of any of the 3 methods namely :

- Percentile Method
- IQR Method
- Standard Deviation Method

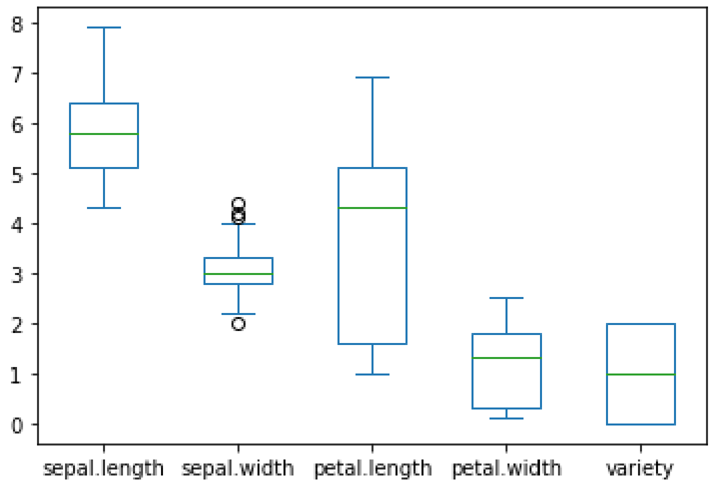
Let’s consider the *Purchase* variable. Now we will be determining if there are any outliers in our data set using the **IQR(Interquartile range) Method**. What is this method about? You will get to know about it as we go along the process so let’s start. Finding the minimum(p0), maximum(p100), first quartile(q1), second quartile(q2), the third quartile(q3), and the iqr(interquartile range) of the values in the Purchase variable.

```
sepal.length    3.15
sepal.width     2.05
petal.length   -3.65
petal.width    -1.95
variety        -3.00
dtype: float64
sepal.length     8.35
sepal.width      4.05
petal.length    10.35
petal.width      4.05
variety          5.00
dtype: float64
```

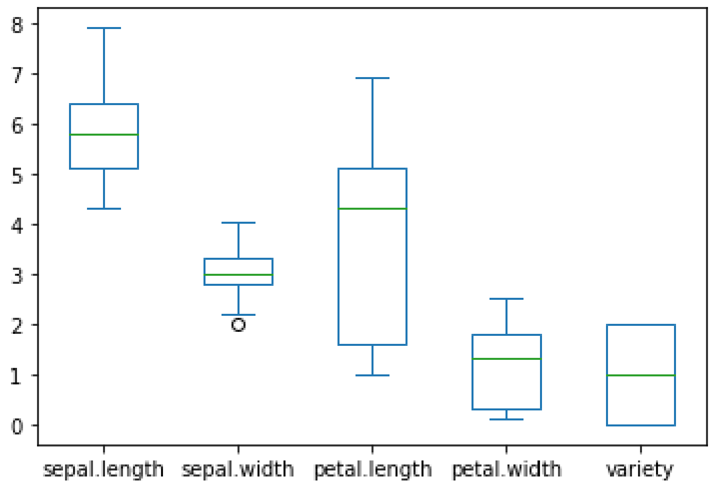
```
#  If lc < p0 → There are NO Outliers on the lower side
#  If uc > p100 → There are NO Outliers on the higher side
lc = q1 - 1.5*iqr
uc = q3 + 1.5*iqr
print(lc)
print(uc)
```

```
sepal.length    3.15
sepal.width     2.05
petal.length   -3.65
petal.width    -1.95
variety        -3.00
dtype: float64
sepal.length     8.35
sepal.width      4.05
petal.length    10.35
petal.width      4.05
variety          5.00
dtype: float64
```

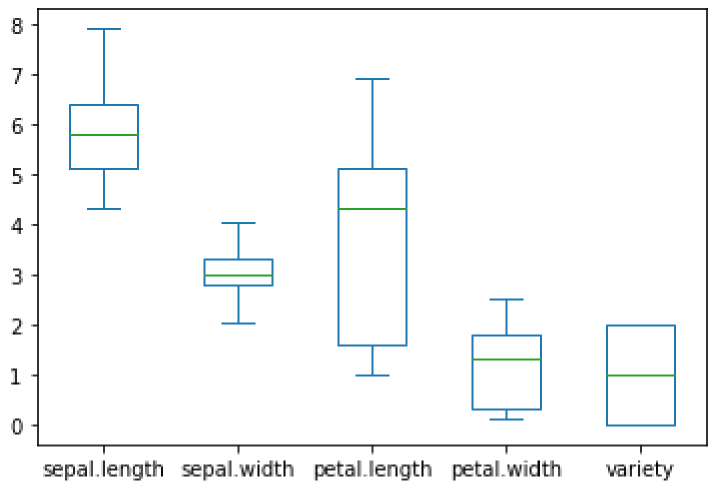
```
df.plot(kind='box')
```



```
df.clip(upper=uc,inplace=True, axis=1)
df.plot(kind='box')
```



```
df.clip(lower=lc,inplace=True, axis=1)
df.plot(kind="box")
```



```
#Import train_test_split function
from sklearn.model_selection import train_test_split
```

```
# Split the data into training and testing dataset
x=df[["sepal.length","sepal.width","petal.length","petal.width"]]
y=df["variety"]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=20)
```

```
#import Random Forest Model
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=100)
rfc.fit(x_train,y_train)
y_pred=rfc.predict(x_test)
```

```
# check accuracy
from sklearn.metrics import accuracy_score, confusion_matrix
print(confusion_matrix(y_test,y_pred))
```

```
[[13  0  0]
 [ 0 18  0]
 [ 0  4 10]]
```

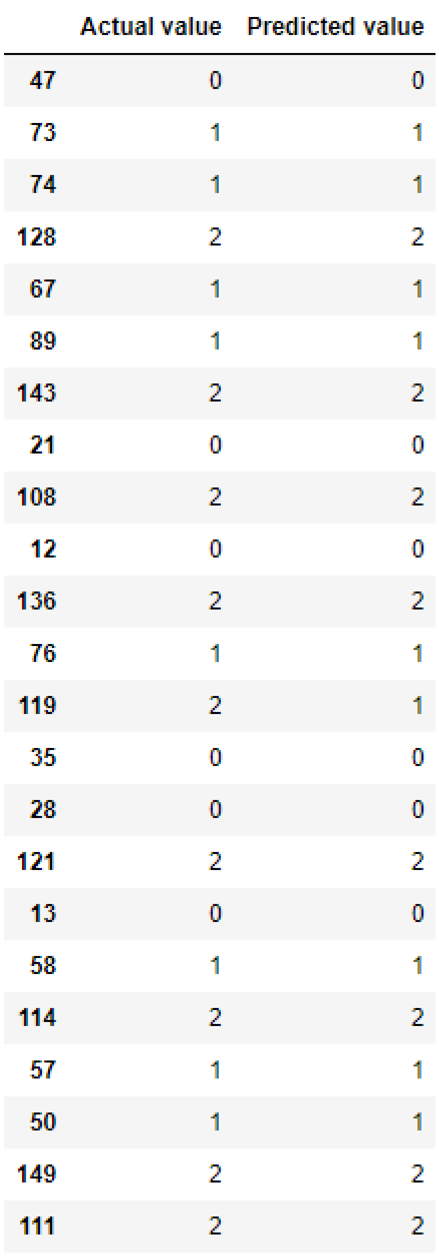
That matrix means is that I had 13 times 0 and 13 times it predicted them to be 0.

```
#To check acuuracy
print("Accuracy: ",accuracy_score(y_test,y_pred))
```

**Accuracy**:  **0.9111111111111111**

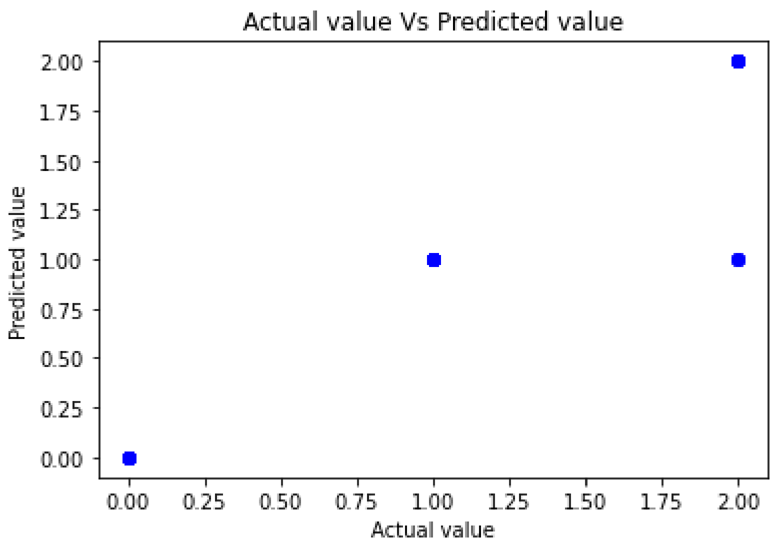
Accuracy is 91% when n_estimators=100

```
# Compare the actual and predicted values
data=pd.DataFrame({"Actual value": y_test,"Predicted value": y_pred})
data
```



| | Actual value | Predicted value |
|---|---|---|
| **47** | 0 | 0 |
| **73** | 1 | 1 |
| **74** | 1 | 1 |
| **128** | 2 | 2 |
| **67** | 1 | 1 |
| **89** | 1 | 1 |
| **143** | 2 | 2 |
| **21** | 0 | 0 |
| **108** | 2 | 2 |
| **12** | 0 | 0 |
| **136** | 2 | 2 |
| **76** | 1 | 1 |
| **119** | 2 | 1 |
| **35** | 0 | 0 |
| **28** | 0 | 0 |
| **121** | 2 | 2 |
| **13** | 0 | 0 |
| **58** | 1 | 1 |
| **114** | 2 | 2 |
| **57** | 1 | 1 |
| **50** | 1 | 1 |
| **149** | 2 | 2 |
| **111** | 2 | 2 |

```
#Draw the scatter plot based on actual and predicted value
plt.scatter(y_test,y_pred,color="b")
plt.title("Actual value Vs Predicted value")
plt.xlabel("Actual value")
plt.ylabel("Predicted value")
```


Actual value Vs Predicted value
Predicted value
Actual value