INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

2022

LAB PRACTICAL FILE

Submitted by:

Anushkka Dhamija

16001032020

B. TECH IT 2

5^{th} SEMESTER

Submitted to:

Ms. Niyati

IGDTUW

Department of INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the experiments entered here have been satisfactorily performed and successfully completed

By: Ms. **ANUSHKKA DHAMIJA**

Studying at: INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

Pursuing: B-TECH program

Under BRANCH: INFORMATION TECHNOLOGY

Enrollment no.: 16001032020

For Subject: ARTIFICIAL INTELLIGENCE

During: Third Academic Year (5 semester)

Under the guidance of: MS. NIYATI BALYAN

TEACHER'S SIGN **DATE**

INDEX

DRIVE LINK FOR COLLAB NOTEBOOKS:
https://drive.google.com/drive/folders/1TcVg9jAVbf4zm6v-Xu2QrjljZUvMJ1sW?usp=share_link

Experiment 1- Exploratory data analysis

AIM- Exploratory data analysis via statistical descriptions and visualization of iris dataset.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sbn
```

In [ ]:
```
df = pd.read_csv('iris.csv')
print(df.head())
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
0   1            5.1           3.5            1.4           0.2  Iris-setosa
1   2            4.9           3.0            1.4           0.2  Iris-setosa
2   3            4.7           3.2            1.3           0.2  Iris-setosa
3   4            4.6           3.1            1.5           0.2  Iris-setosa
4   5            5.0           3.6            1.4           0.2  Iris-setosa
```

In [ ]:
```
df.Species
```

Out[ ]:
```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

In [ ]:
```
df.head()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [ ]:
```
df.tail()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [ ]:
```
counts = df.value_counts('Species')
print(counts)
```

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [ ]:
```
species = df.Species.unique()
print(species)
```

```
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

In [ ]:
```
df.describe()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [ ]:
```
print(df.std)
```

```
<bound method NDFrame._add_numeric_operations.<locals>.std of       Id  Sepal
LengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8

            Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]>
```

In [ ]:
```
print(df.std(axis=1))
```

```
0       2.010721
1       1.772005
2       1.754138
3       1.813009
4       2.165179
         ...
145    63.394345
146    64.010335
147    64.344914
148    64.719224
149    65.335924
Length: 150, dtype: float64

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

In [ ]:
```
print(df.std(axis=0))
```

```
Id               43.445368
SepalLengthCm     0.828066
SepalWidthCm      0.433594
PetalLengthCm     1.764420
PetalWidthCm      0.763161
dtype: float64

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

In [ ]:
```
print(df.skew(axis=1))
```

```
0      0.738325
1      0.591741
2     -0.157518
3     -0.520475
4     -0.495982
         ...
145    2.231813
146    2.231665
147    2.231830
148    2.232854
149    2.232579
Length: 150, dtype: float64

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

In [ ]:
```
print(df.skew(axis=0))
```

```
Id               0.000000
SepalLengthCm    0.314911
SepalWidthCm     0.334053
PetalLengthCm   -0.274464
PetalWidthCm    -0.104997
dtype: float64

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

In [ ]:
```
df.var(axis=0)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarnin
g: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=N
one') is deprecated; in a future version this will raise TypeError.  Select o
nly valid columns before calling the reduction.
  """Entry point for launching an IPython kernel.
```

Out[ ]:
```
Id                1887.500000
SepalLengthCm        0.685694
SepalWidthCm         0.188004
PetalLengthCm        3.113179
PetalWidthCm         0.582414
dtype: float64
```

In [ ]:
```
print(df.max(axis=0))
```

```
Id                           150
SepalLengthCm                7.9
SepalWidthCm                 4.4
PetalLengthCm                6.9
PetalWidthCm                 2.5
Species           Iris-virginica
dtype: object
```

In [ ]:
```
print(df.min(axis=0))
```

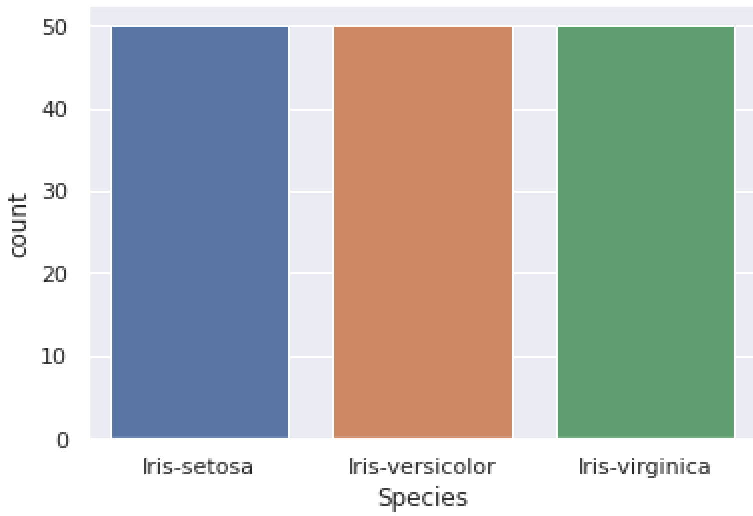
```
Id                          1
SepalLengthCm             4.3
SepalWidthCm              2.0
PetalLengthCm             1.0
PetalWidthCm              0.1
Species           Iris-setosa
dtype: object
```

In [ ]:
```
ansArr = [ ]
for i in range(4):
  ansArr.append(df.max(axis=0)[i] - df.min(axis=0)[i])
print(ansArr)
```

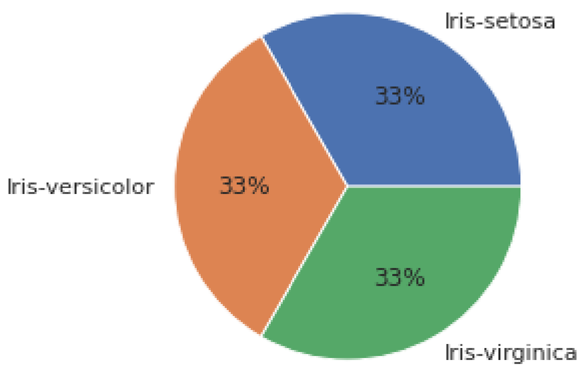
```
[149, 3.6000000000000005, 2.4000000000000004, 5.9]
```

In [ ]:
```
sbn.countplot(x='Species', data=df)
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3156df40d0>

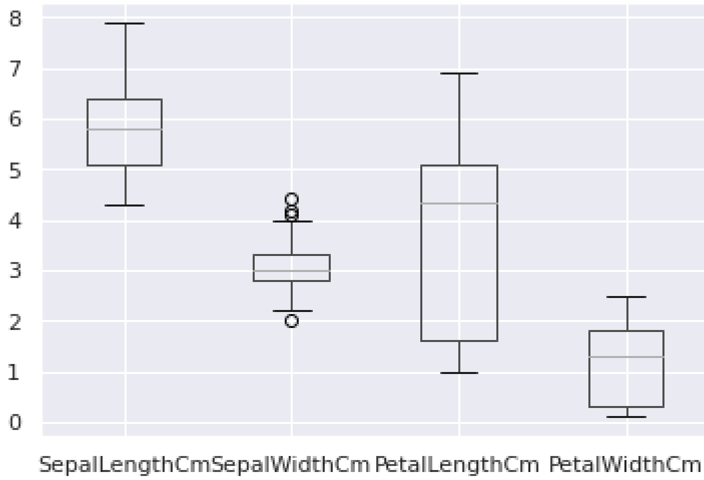


In [ ]:
```
plt.pie(counts, labels=species, autopct='%1.0f%%')
plt.show()
```

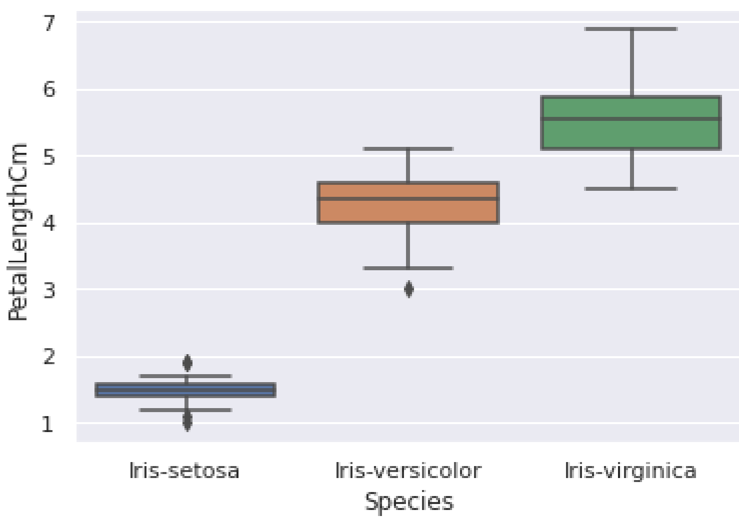


In [ ]:
```
new_df = df.drop(['Id'], axis=1)
new_df.boxplot()
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3156b051d0>

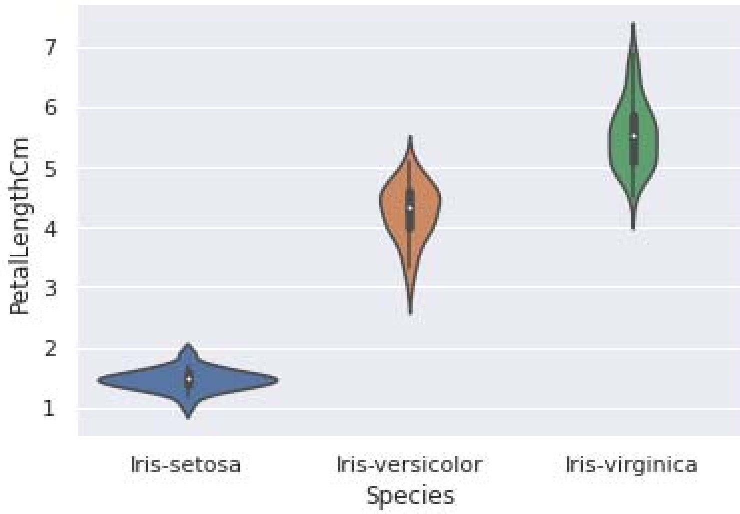


In [ ]:
```
sbn.boxplot(x="Species", y="PetalLengthCm", data=df)
plt.show()
```



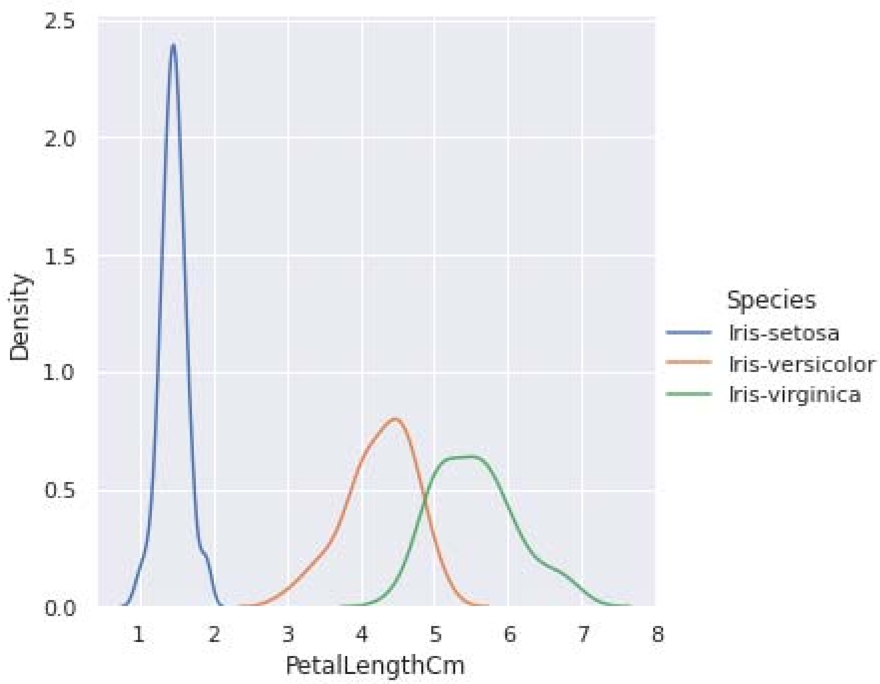
In [ ]:

```
sbn.violinplot(x="Species", y="PetalLengthCm", data=df)
plt.show()
```

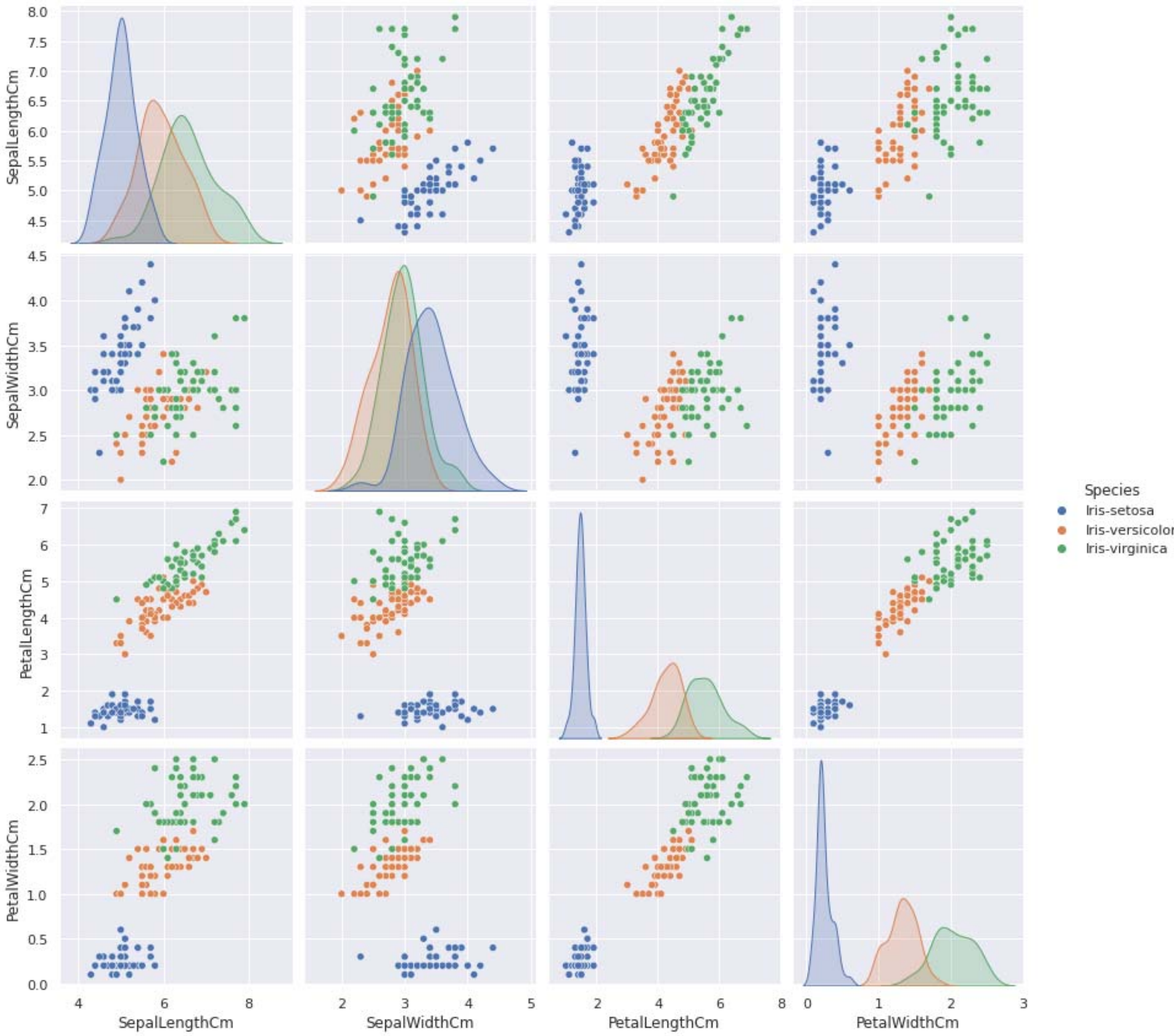


In [ ]:

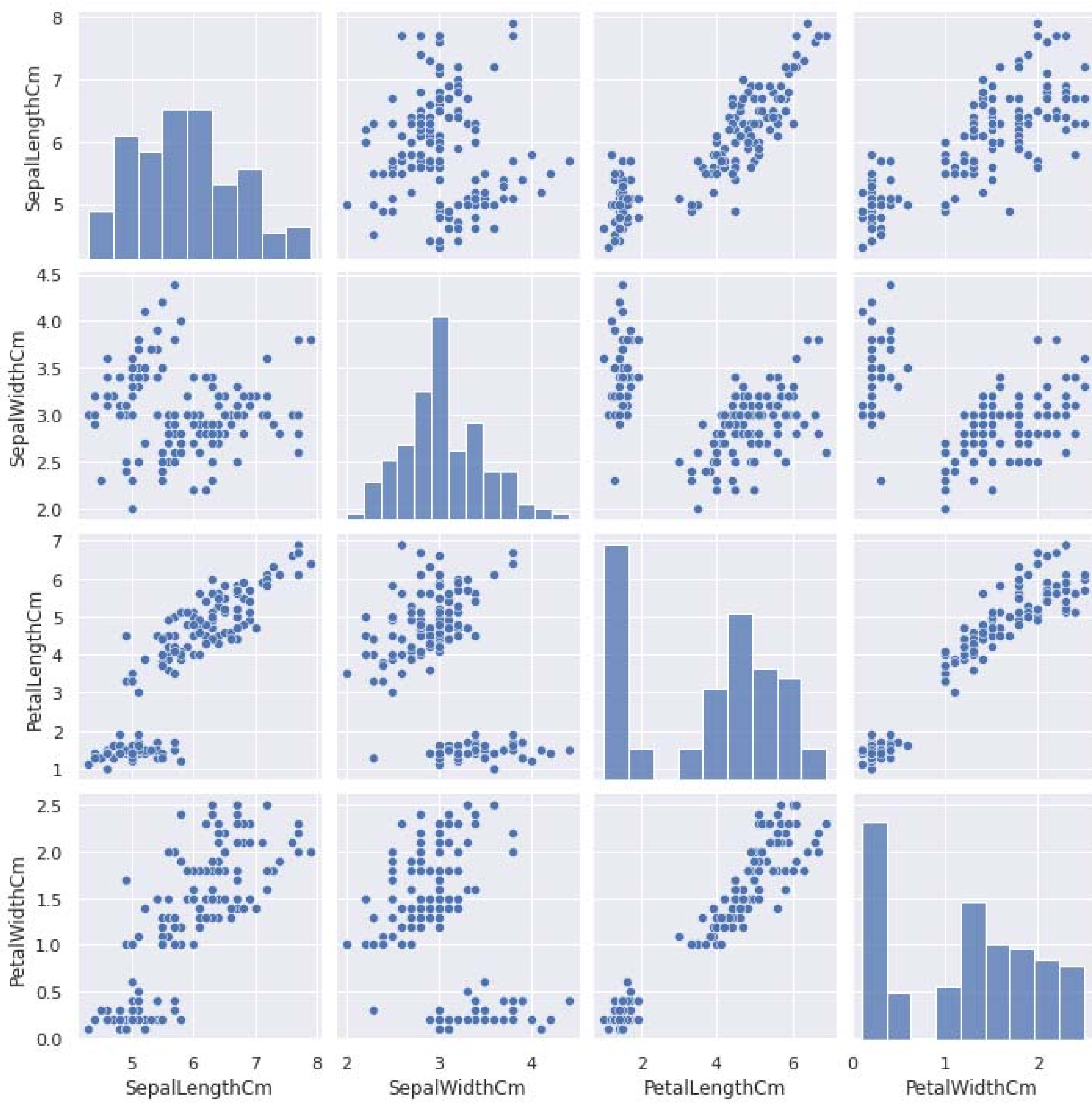
```
sbn.FacetGrid(df, hue="Species", height=5) \
   .map(sbn.kdeplot, "PetalLengthCm") \
   .add_legend()
plt.show()
```



In [ ]:
```
sbn.pairplot(new_df, hue="Species", height=3)
plt.show()
```



In [ ]:
```
sbn.pairplot(new_df)
```
Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f315872ff10>
```



Experiment 2- Naive Bayes

AIM- Naive Bayes on Iris Dataset

In [3]:
```
import pandas as pd
iris = pd.read_csv('iris.csv')
```

In [4]:
```
X = iris.iloc[:,:4].values
y = iris['Species'].values
```

In [5]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
```

In [6]:
```
from sklearn.naive_bayes import GaussianNB
nvclassifier = GaussianNB()
nvclassifier.fit(X_train, y_train) #training
```

Out[6]:
```
GaussianNB()
```

In [7]:
```
y_pred = nvclassifier.predict(X_test) # predicting test set
print(y_pred)
```

```
['Iris-versicolor' 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor'
 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-setosa' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa']
```

In [9]:
```
nvclassifier.score(X_test, y_test)
```

Out[9]:
```
1.0
```

In [10]:
```python
from sklearn.model_selection import train_test_split
shuffled = iris.sample(frac=1)
X = iris.iloc[:,:4].values
y = iris['Species'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
# Fitting Naive Bayes Classification to the Training set with linear kernel
from sklearn.naive_bayes import GaussianNB
nvclassifier = GaussianNB()
nvclassifier.fit(X_train, y_train)   #Training
# Predicting the Test set results
y_pred = nvclassifier.predict(X_test)
print(y_pred)
```

```
['Iris-versicolor' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-virginica' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-virginica'
 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor'
 'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor']
```

In [12]:
```
y_pred = nvclassifier.predict(X_test)
pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

Out[12]:

	Actual	Predicted
0	Iris-versicolor	Iris-versicolor
1	Iris-setosa	Iris-setosa
2	Iris-versicolor	Iris-versicolor
3	Iris-versicolor	Iris-versicolor
4	Iris-versicolor	Iris-versicolor
5	Iris-versicolor	Iris-versicolor
6	Iris-virginica	Iris-virginica
7	Iris-virginica	Iris-virginica
8	Iris-versicolor	Iris-versicolor
9	Iris-versicolor	Iris-versicolor
10	Iris-setosa	Iris-setosa
11	Iris-virginica	Iris-virginica
12	Iris-virginica	Iris-virginica
13	Iris-setosa	Iris-setosa
14	Iris-virginica	Iris-virginica
15	Iris-versicolor	Iris-versicolor
16	Iris-versicolor	Iris-versicolor
17	Iris-versicolor	Iris-versicolor
18	Iris-versicolor	Iris-versicolor
19	Iris-setosa	Iris-setosa
20	Iris-virginica	Iris-virginica
21	Iris-versicolor	Iris-versicolor
22	Iris-setosa	Iris-setosa
23	Iris-virginica	Iris-virginica
24	Iris-setosa	Iris-setosa
25	Iris-versicolor	Iris-versicolor
26	Iris-virginica	Iris-virginica
27	Iris-versicolor	Iris-versicolor
28	Iris-setosa	Iris-setosa
29	Iris-setosa	Iris-setosa
30	Iris-setosa	Iris-setosa
31	Iris-virginica	Iris-virginica
32	Iris-virginica	Iris-virginica
33	Iris-versicolor	Iris-versicolor
34	Iris-versicolor	Iris-versicolor

	Actual	Predicted
35	Iris-setosa	Iris-setosa
36	Iris-virginica	Iris-virginica
37	Iris-versicolor	Iris-versicolor
38	Iris-virginica	Iris-virginica
39	Iris-versicolor	Iris-versicolor
40	Iris-virginica	Iris-virginica
41	Iris-virginica	Iris-virginica
42	Iris-setosa	Iris-setosa
43	Iris-setosa	Iris-setosa
44	Iris-versicolor	Iris-versicolor

Experiment 3- Linear Regression

AIM- Predict housing price using univariate linear regression on kc_housing dataset

We make predictions using models in TensorFlow

In [ ]:
```
import tensorflow as tf
import pandas as pd
import numpy as np

tf.__version__
```

Out[ ]:
```
'2.9.2'
```

In [ ]:
```
# Load the dataset as a dataframe named housing
housing = pd.read_csv('kc_house_data.csv')

# View price columns
print(housing['price'])
```

```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           ...
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21613, dtype: float64
```

In [ ]:
```
size_log = np.log(np.array(housing['sqft_lot'], np.float32))
price_log = np.log(np.array(housing['price'], np.float32))
bedrooms = np.array(housing['bedrooms'], np.float32)
```

In [ ]:
```
# Define a linear regression model
def linear_regression(intercept, slope, features=size_log):
    return intercept + slope * features

# Set loss_function() to take the variables as arguments
def loss_function(intercept, slope, features=size_log, targets=price_log):
    # Set the predicted values
    predictions = linear_regression(intercept, slope, features)

    # Return the mean squared error loss
    return tf.keras.losses.mse(targets, predictions)

# Compute the loss function for different slope and intercept values
print(loss_function(0.1, 0.1).numpy())
print(loss_function(0.1, 0.5).numpy())
```

```
145.44653
71.866
```

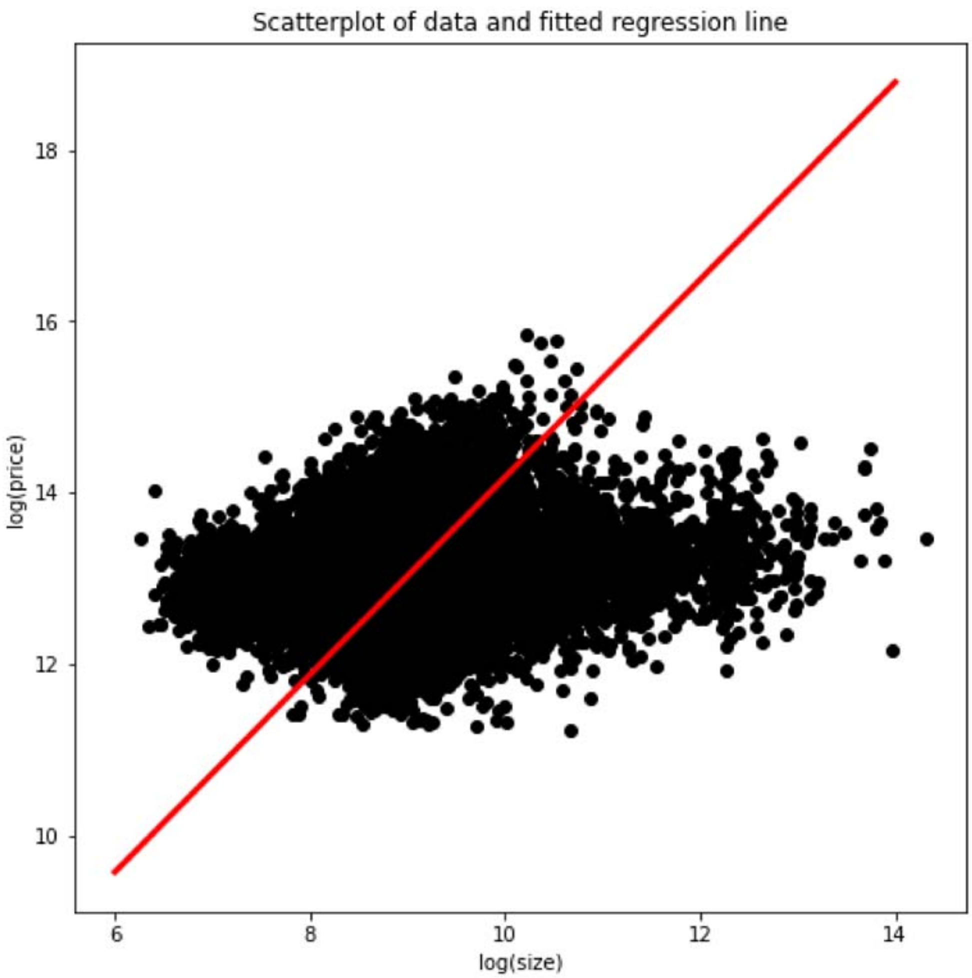
Training Linear Model

In [ ]:
```
import matplotlib.pyplot as plt

def plot_results(intercept, slope):
    size_range = np.linspace(6,14,100)
    price_pred = [intercept + slope * s for s in size_range]
    plt.figure(figsize=(8, 8))
    plt.scatter(size_log, price_log, color = 'black');
    plt.plot(size_range, price_pred, linewidth=3.0, color='red');
    plt.xlabel('log(size)');
    plt.ylabel('log(price)');
    plt.title('Scatterplot of data and fitted regression line');
```

In [ ]:

```
intercept = tf.Variable(0.0, tf.float32)
slope = tf.Variable(0.0, tf.float32)

# Initialize an adam optimizer
opt = tf.keras.optimizers.Adam(learning_rate=0.5)

for j in range(100):
    # Apply minimize, pass the loss function, and supply the variables
    opt.minimize(lambda: loss_function(intercept, slope), var_list=[intercept,
slope])

    # Print every 10th value of the loss
    if j % 10 == 0:
        print(loss_function(intercept, slope).numpy())

# Plot data and regressoin line
plot_results(intercept, slope)
```

```
65.263725
1.4908673
2.3815567
2.9084811
2.611049
1.760517
1.3468053
1.3559624
1.288412
1.2425313
```



MUltiple Linear Regression

In [ ]:
```
def print_results(params):
    return print('loss: {:0.3f}, intercept: {:0.3f}, slope_1: {:0.3f}, slope_
2: {:0.3f}'
                 .format(loss_function(params).numpy(),
                         params[0].numpy(),
                         params[1].numpy(),
                         params[2].numpy()))
```

In [ ]:
```
params = tf.Variable([0.1, 0.05, 0.02], tf.float32)

# Define the linear regression model
def linear_regression(params, feature1=size_log, feature2=bedrooms):
    return params[0] + feature1 * params[1] + feature2 * params[2]

# Define the loss function
def loss_function(params, targets=price_log, feature1=size_log, feature2=bedro
oms):
    # Set the predicted values
    predictions = linear_regression(params, feature1, feature2)

    # Use the mean absolute error loss
    return tf.keras.losses.mae(targets, predictions)

# Define the optimize operation
opt = tf.keras.optimizers.Adam()

# Perform minimization and print trainable variables
for j in range(10):
    opt.minimize(lambda: loss_function(params), var_list=[params])
    print_results(params)
```

```
loss: 12.418, intercept: 0.101, slope_1: 0.051, slope_2: 0.021
loss: 12.404, intercept: 0.102, slope_1: 0.052, slope_2: 0.022
loss: 12.391, intercept: 0.103, slope_1: 0.053, slope_2: 0.023
loss: 12.377, intercept: 0.104, slope_1: 0.054, slope_2: 0.024
loss: 12.364, intercept: 0.105, slope_1: 0.055, slope_2: 0.025
loss: 12.351, intercept: 0.106, slope_1: 0.056, slope_2: 0.026
loss: 12.337, intercept: 0.107, slope_1: 0.057, slope_2: 0.027
loss: 12.324, intercept: 0.108, slope_1: 0.058, slope_2: 0.028
loss: 12.311, intercept: 0.109, slope_1: 0.059, slope_2: 0.029
loss: 12.297, intercept: 0.110, slope_1: 0.060, slope_2: 0.030
```

Batch Training

In [ ]:
```
# preparing to batch train
# Define the intercept and slope
intercept = tf.Variable(10.0, tf.float32)
slope = tf.Variable(0.5, tf.float32)

# Define the model
def linear_regression(intercept, slope, features):
    # Define the predicted values
    return intercept + slope * features

# Define the loss function
def loss_function(intercept, slope, targets, features):
    # Define the predicted values
    predictions = linear_regression(intercept, slope, features)

    # Define the MSE loss
    return tf.keras.losses.mse(targets, predictions)
```

In [ ]:

```
# training a linear model in batches
intercept = tf.Variable(10.0, tf.float32)
slope = tf.Variable(0.5, tf.float32)

# Initialize adam optimizer
opt = tf.keras.optimizers.Adam()

# Load data in batches
for batch in pd.read_csv('kc_house_data.csv', chunksize=100):
    size_batch = np.array(batch['sqft_lot'], np.float32)

    # Extract the price values for the current batch
    price_batch = np.array(batch['price'], np.float32)

    # Complete the loss, fill in the variable list, and minimize
    opt.minimize(lambda: loss_function(intercept, slope, price_batch, size_bat
ch),
                 var_list=[intercept, slope])

# Print trained parameters
print(intercept.numpy(), slope.numpy())
```

```
10.217888 0.7016001
```

Experiment 4- Gradient Descent and Refression Metrics

AIM- Evaluate regressor model on housing price dataset

In [1]:
```
import keras
keras.__version__
```

Out[1]:
```
'2.9.0'
```

In [2]:
```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) =  boston_housing.load_
data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-data
sets/boston_housing.npz
57026/57026 [==============================] - 0s 0us/step
```

In [3]:
```
train_data[1],train_data.shape
```

Out[3]:
```
(array([2.1770e-02, 8.2500e+01, 2.0300e+00, 0.0000e+00, 4.1500e-01,
        7.6100e+00, 1.5700e+01, 6.2700e+00, 2.0000e+00, 3.4800e+02,
        1.4700e+01, 3.9538e+02, 3.1100e+00]), (404, 13))
```

In [4]:
```
test_data.shape
```

Out[4]:
```
(102, 13)
```

In [5]:
```
train_targets
```

Out[5]:
```
array([15.2, 42.3, 50. , 21.1, 17.7, 18.5, 11.3, 15.6, 15.6, 14.4, 12.1,
       17.9, 23.1, 19.9, 15.7,  8.8, 50. , 22.5, 24.1, 27.5, 10.9, 30.8,
       32.9, 24. , 18.5, 13.3, 22.9, 34.7, 16.6, 17.5, 22.3, 16.1, 14.9,
       23.1, 34.9, 25. , 13.9, 13.1, 20.4, 20. , 15.2, 24.7, 22.2, 16.7,
       12.7, 15.6, 18.4, 21. , 30.1, 15.1, 18.7,  9.6, 31.5, 24.8, 19.1,
       22. , 14.5, 11. , 32. , 29.4, 20.3, 24.4, 14.6, 19.5, 14.1, 14.3,
       15.6, 10.5,  6.3, 19.3, 19.3, 13.4, 36.4, 17.8, 13.5, 16.5,  8.3,
       14.3, 16. , 13.4, 28.6, 43.5, 20.2, 22. , 23. , 20.7, 12.5, 48.5,
       14.6, 13.4, 23.7, 50. , 21.7, 39.8, 38.7, 22.2, 34.9, 22.5, 31.1,
       28.7, 46. , 41.7, 21. , 26.6, 15. , 24.4, 13.3, 21.2, 11.7, 21.7,
       19.4, 50. , 22.8, 19.7, 24.7, 36.2, 14.2, 18.9, 18.3, 20.6, 24.6,
       18.2,  8.7, 44. , 10.4, 13.2, 21.2, 37. , 30.7, 22.9, 20. , 19.3,
       31.7, 32. , 23.1, 18.8, 10.9, 50. , 19.6,  5. , 14.4, 19.8, 13.8,
       19.6, 23.9, 24.5, 25. , 19.9, 17.2, 24.6, 13.5, 26.6, 21.4, 11.9,
       22.6, 19.6,  8.5, 23.7, 23.1, 22.4, 20.5, 23.6, 18.4, 35.2, 23.1,
       27.9, 20.6, 23.7, 28. , 13.6, 27.1, 23.6, 20.6, 18.2, 21.7, 17.1,
        8.4, 25.3, 13.8, 22.2, 18.4, 20.7, 31.6, 30.5, 20.3,  8.8, 19.2,
       19.4, 23.1, 23. , 14.8, 48.8, 22.6, 33.4, 21.1, 13.6, 32.2, 13.1,
       23.4, 18.9, 23.9, 11.8, 23.3, 22.8, 19.6, 16.7, 13.4, 22.2, 20.4,
       21.8, 26.4, 14.9, 24.1, 23.8, 12.3, 29.1, 21. , 19.5, 23.3, 23.8,
       17.8, 11.5, 21.7, 19.9, 25. , 33.4, 28.5, 21.4, 24.3, 27.5, 33.1,
       16.2, 23.3, 48.3, 22.9, 22.8, 13.1, 12.7, 22.6, 15. , 15.3, 10.5,
       24. , 18.5, 21.7, 19.5, 33.2, 23.2,  5. , 19.1, 12.7, 22.3, 10.2,
       13.9, 16.3, 17. , 20.1, 29.9, 17.2, 37.3, 45.4, 17.8, 23.2, 29. ,
       22. , 18. , 17.4, 34.6, 20.1, 25. , 15.6, 24.8, 28.2, 21.2, 21.4,
       23.8, 31. , 26.2, 17.4, 37.9, 17.5, 20. ,  8.3, 23.9,  8.4, 13.8,
        7.2, 11.7, 17.1, 21.6, 50. , 16.1, 20.4, 20.6, 21.4, 20.6, 36.5,
        8.5, 24.8, 10.8, 21.9, 17.3, 18.9, 36.2, 14.9, 18.2, 33.3, 21.8,
       19.7, 31.6, 24.8, 19.4, 22.8,  7.5, 44.8, 16.8, 18.7, 50. , 50. ,
       19.5, 20.1, 50. , 17.2, 20.8, 19.3, 41.3, 20.4, 20.5, 13.8, 16.5,
       23.9, 20.6, 31.5, 23.3, 16.8, 14. , 33.8, 36.1, 12.8, 18.3, 18.7,
       19.1, 29. , 30.1, 50. , 50. , 22. , 11.9, 37.6, 50. , 22.7, 20.8,
       23.5, 27.9, 50. , 19.3, 23.9, 22.6, 15.2, 21.7, 19.2, 43.8, 20.3,
       33.2, 19.9, 22.5, 32.7, 22. , 17.1, 19. , 15. , 16.1, 25.1, 23.7,
       28.7, 37.2, 22.6, 16.4, 25. , 29.8, 22.1, 17.4, 18.1, 30.3, 17.5,
       24.7, 12.6, 26.5, 28.7, 13.3, 10.4, 24.4, 23. , 20. , 17.8,  7. ,
       11.8, 24.4, 13.8, 19.4, 25.2, 19.4, 19.4, 29.1])
```

Preparing the data

In [6]:
```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

Building network

In [7]:
```python
from keras import models
from keras import layers

def build_model():
    # Because we will need to instantiate
    # the same model multiple times,
    # we use a function to construct it.
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

Validating approach using K-Fold validation

In [8]:
```python
import numpy as np

k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []
for i in range(k):
    print('processing fold #', i)
    # Prepare the validation data: data from partition # k
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples
]

    # Prepare the training data: data from all other partitions
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    # Build the Keras model (already compiled)
    model = build_model()
    # Train the model (in silent mode, verbose=0)
    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0)
    # Evaluate the model on the validation data
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

```
processing fold # 0
processing fold # 1
processing fold # 2
processing fold # 3
```

In [9]:
```
all_scores
```
Out[9]:
```
[1.994046688079834, 2.3331761360168457, 2.501741886138916, 2.462741613388061
5]
```

In [10]:
```
np.mean(all_scores)
```
Out[10]:
```
2.3229265809059143
```

Training model for higher epochs: 500

In [11]:
```
from keras import backend as K

# Some memory clean-up
K.clear_session()
```

In [15]:
```
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    # Prepare the validation data: data from partition # k
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1) * num_val_samples
]

    # Prepare the training data: data from all other partitions
    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    # Build the Keras model (already compiled)
    model = build_model()
    # Train the model (in silent mode, verbose=0)
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    all_mae_histories.append(mae_history)
```

```
processing fold # 0

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-15-7476cfe66b87> in <module>
     23                         validation_data=(val_data, val_targets),
     24                         epochs=num_epochs, batch_size=1, verbose=0)
---> 25     mae_history = history.history['val_mean_absolute_error']
     26     all_mae_histories.append(mae_history)

KeyError: 'val_mean_absolute_error'
```

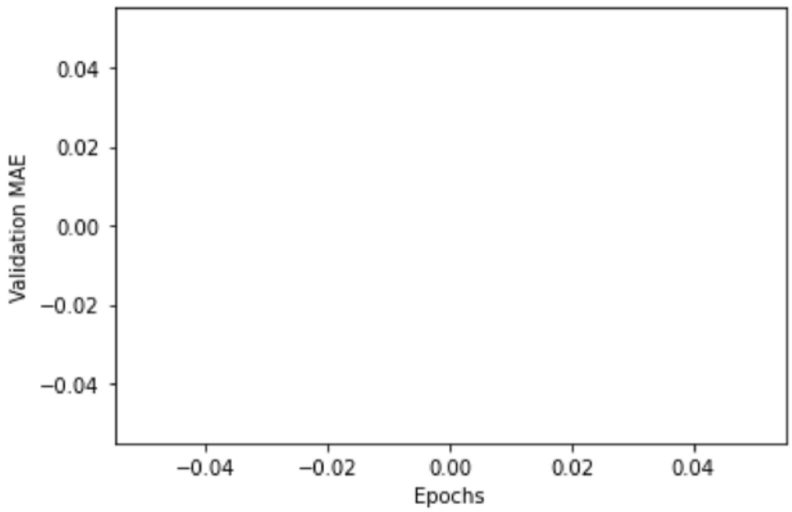
Average per-epoch MAE scores for all folds

In [13]:
```
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3441: Runtim
eWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.7/dist-packages/numpy/core/_methods.py:189: RuntimeWar
ning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

Plotting

In [14]:
```
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



Improving Scaling and viewing plot better

In [ ]:
```
def smooth_curve(points, factor=0.9):
  smoothed_points = []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point * (1 - factor))
    else:
      smoothed_points.append(point)
  return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```

Fitting freshly compiled model

In [ ]:
```
# Get a fresh, compiled model.
model = build_model()
# Train it on the entirety of the data.
model.fit(train_data, train_targets,
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

In [ ]:
```
test_mae_score
```

Experiment 5- Logistic Regression

AIM- Classify iris dataset using Logistic regression

In [2]:
```
import pandas as pd
import numpy as np
import random
import warnings
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn import datasets
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
d = sns.load_dataset("iris")
```

In [4]:
```
d
```

Out[4]:

	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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

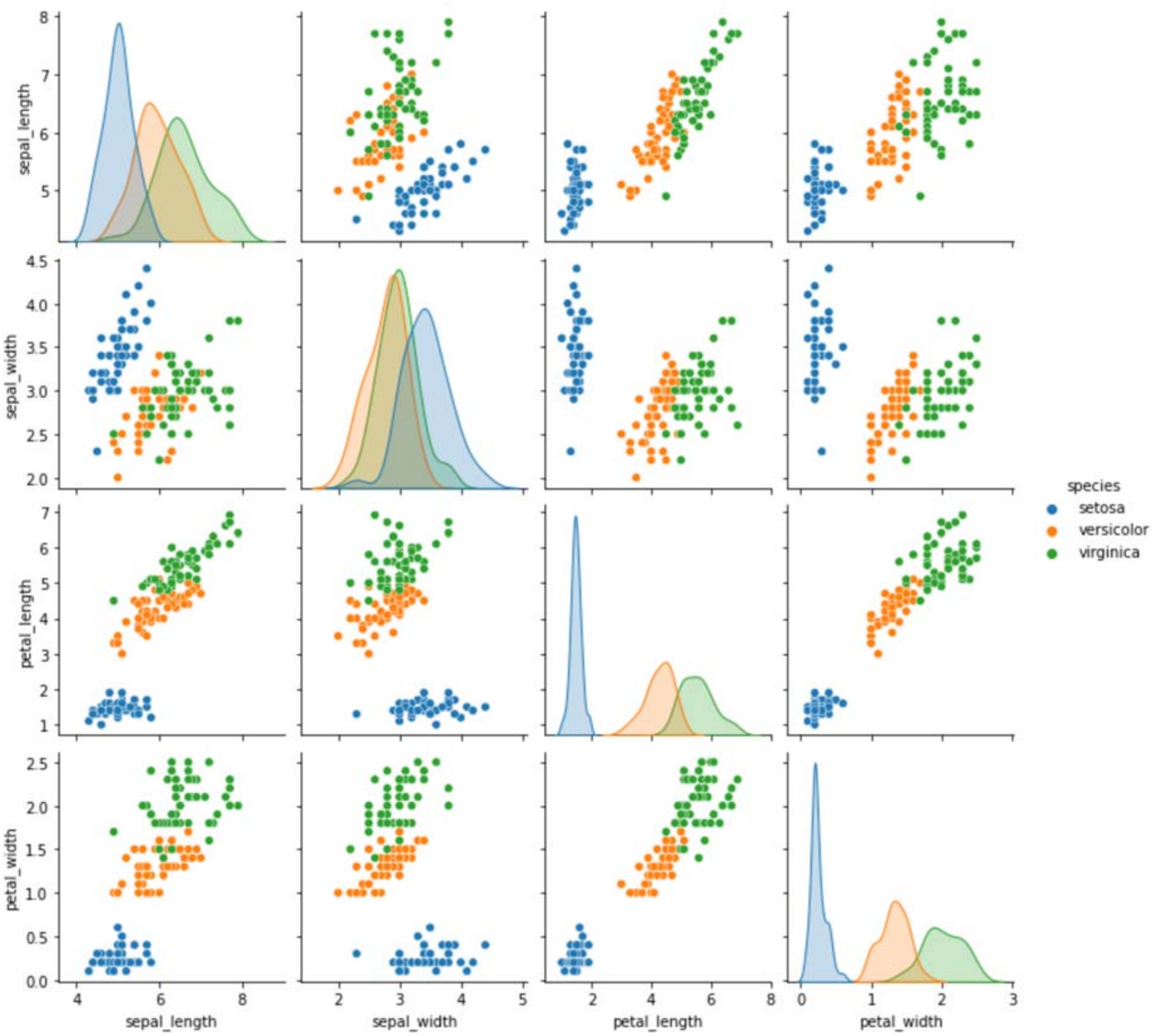
150 rows × 5 columns

In [5]:
```
d.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    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [6]:
```
# visualize data
import seaborn as sns
sns.pairplot(d,hue = 'species')
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x7f64ffb88250>
```



In [7]:
```
# select features and labels
X = d.drop(['species'], axis = 1)
Y = d['species']

# 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 = 0.2)
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(120, 4) (120,)
(30, 4) (30,)
```

In [8]:
```
# training with logistic regression
from sklearn.linear_model import LogisticRegression
m = LogisticRegression()
m.fit(x_train, y_train)
```

Out[8]:
```
LogisticRegression()
```

In [9]:
```
# accuracy of prediction
m.score(x_test, y_test)
```

Out[9]:
```
0.9333333333333333
```

In [11]:
```
y_pred = m.predict(x_test)
pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
```

Out[11]:

	Actual	Predicted
110	virginica	virginica
39	setosa	setosa
27	setosa	setosa
114	virginica	virginica
119	virginica	versicolor
84	versicolor	versicolor
70	versicolor	virginica
68	versicolor	versicolor
139	virginica	virginica
4	setosa	setosa
48	setosa	setosa
24	setosa	setosa
138	virginica	virginica
124	virginica	virginica
14	setosa	setosa
34	setosa	setosa
140	virginica	virginica
103	virginica	virginica
75	versicolor	versicolor
25	setosa	setosa
98	versicolor	versicolor
149	virginica	virginica
69	versicolor	versicolor
52	versicolor	versicolor
47	setosa	setosa
81	versicolor	versicolor
28	setosa	setosa
120	virginica	virginica
56	versicolor	versicolor
1	setosa	setosa

Experiment 6- Classification metrics

AIM- Find accuracy and draw heatmap/confusion matrix for iris classification

In [1]:
```
import pandas as pd
import numpy as np
import random
import warnings
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn import datasets
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
d = sns.load_dataset("iris")
```

In [3]:
```
# select features and labels
X = d.drop(['species'], axis = 1)
Y = d['species']

# 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 = 0.2)
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
(120, 4) (120,)
(30, 4) (30,)
```

In [4]:
```
# training with logistic regression
from sklearn.linear_model import LogisticRegression
m = LogisticRegression()
m.fit(x_train, y_train)
```

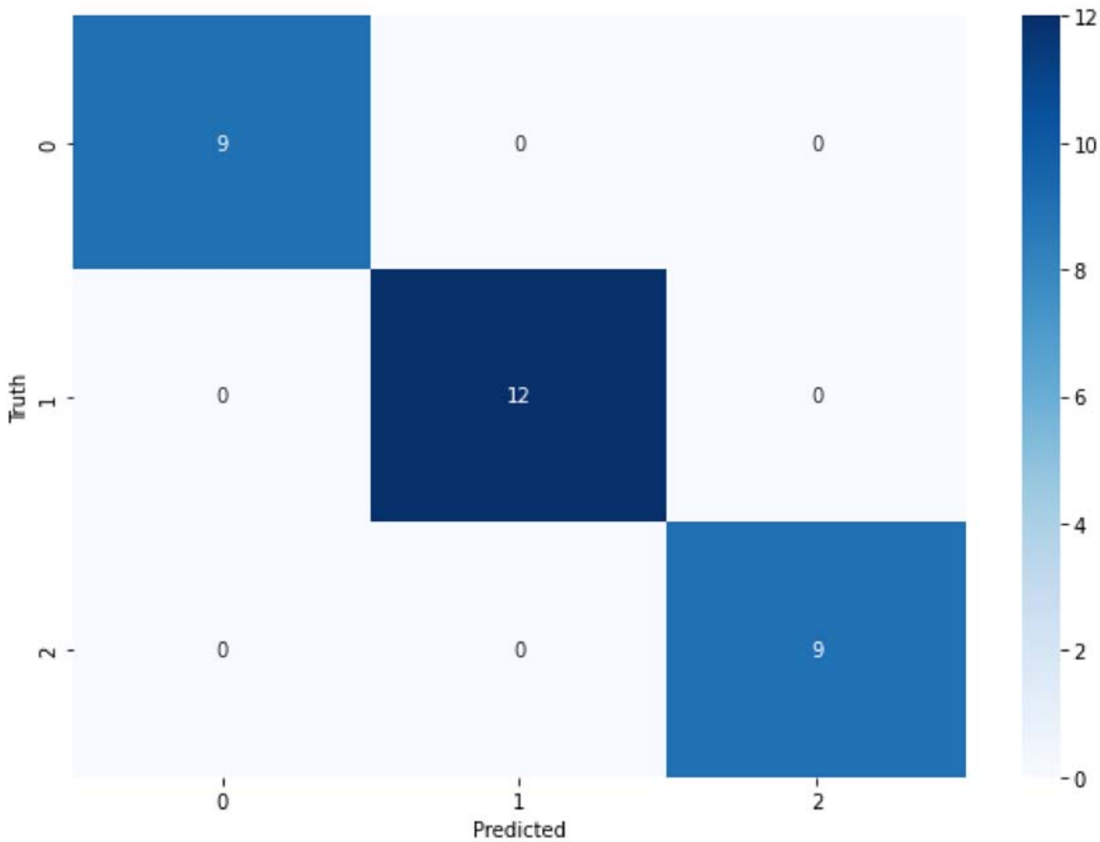
Out[4]:
```
LogisticRegression()
```

In [5]:
```
# confusion matrix visualization
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, m.predict(x_test))
cm
```

Out[5]:
```
array([[ 9,  0,  0],
       [ 0, 12,  0],
       [ 0,  0,  9]])
```

In [6]:
```python
# heatmap visualization of confusion  matrix
import seaborn as sns
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot = True, cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Truth")
```

Out[6]:
```
Text(69.0, 0.5, 'Truth')
```



Experiment 7- Naive Bayes Classifier

AIM- Apply Gaussian Naive Bayes on make_blobs dataset which generates points with Gaussian Distribution and Multinomial Naive Bayes on fetch_20newsgroups dataset.

Bayesian Classification

Naive Bayes classifiers are built on Bayesian classification methods. These rely on Bayes's theorem, which is an equation describing the relationship of conditional probabilities of statistical quantities. In Bayesian classification, we're interested in finding the probability of a label given some observed features, which we can write as $P(L \mid \text{features})$. Bayes's theorem tells us how to express this in terms of quantities we can compute more directly:

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L) P(L)}{P(\text{features})}$$

If we are trying to decide between two labels—let's call them L_1 and L_2—then one way to make this decision is to compute the ratio of the posterior probabilities for each label:

$$\frac{P(L_1 \mid \text{features})}{P(L_2 \mid \text{features})} = \frac{P(\text{features} \mid L_1)}{P(\text{features} \mid L_2)} \frac{P(L_1)}{P(L_2)}$$

All we need now is some model by which we can compute $P(\text{features} \mid L_i)$ for each label. Such a model is called a *generative model* because it specifies the hypothetical random process that generates the data. Specifying this generative model for each label is the main piece of the training of such a Bayesian classifier. The general version of such a training step is a very difficult task, but we can make it simpler through the use of some simplifying assumptions about the form of this model.

This is where the "naive" in "naive Bayes" comes in: if we make very naive assumptions about the generative model for each label, we can find a rough approximation of the generative model for each class, and then proceed with the Bayesian classification. Different types of naive Bayes classifiers rest on different naive assumptions about the data, and we will examine a few of these in the following sections.

We begin with the standard imports:

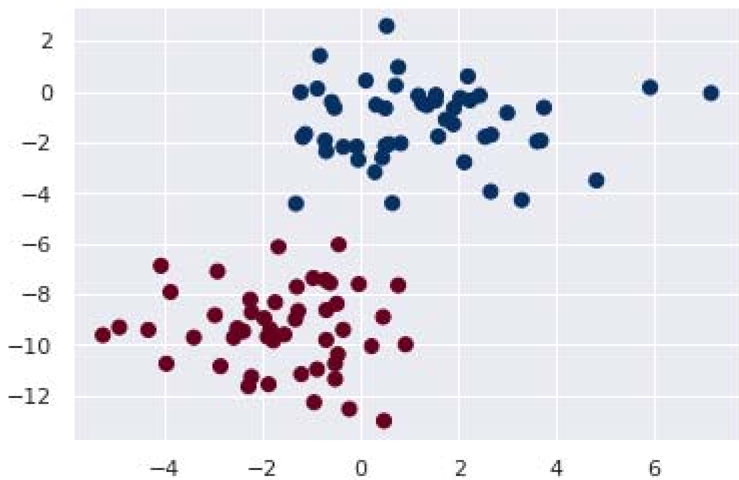
Import Standard Packages

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

Gaussian Naive Bayes

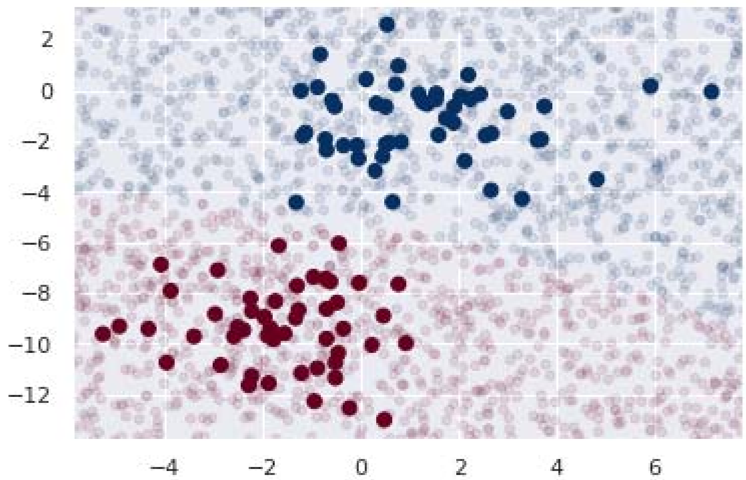
In [ ]:
```
# assumption: data from each label is drawn from a simple gaussian distributio
n
from sklearn.datasets import make_blobs # make_blobs generates blobs of points
with a Gaussian Distribution
X, y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu') # scatter plot visualiza
tion for data
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f3bdbb245d0>
```



In [ ]:
```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(X, y) # data fitted to model (training)

rng = np.random.RandomState(0)
Xnew = [-6, -14] + [14, 18] * rng.rand(2000, 2) # generate new data randomly
ynew = model.predict(Xnew) # predicted labels for new data
```

In [ ]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu') # default alpha=1
lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='RdBu', alpha=0.1) # new data plotted which gives us the decision boundary for the labels. alpha value is reduced to show transpaerncy of new data points
plt.axis(lim);
```



In [ ]:
```
yprob = model.predict_proba(Xnew) # predict_proba is used to predict class probabilities
yprob[-8:].round(2)

# in the output, columns give the posterior probabilities of the first and second label, respectively
```

Out[ ]:
```
array([[0.89, 0.11],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [1.  , 0.  ],
       [0.  , 1.  ],
       [0.15, 0.85]])
```

Multinomial Naive Bayes

In [ ]:
```
from sklearn.datasets import fetch_20newsgroups

data = fetch_20newsgroups() # download data
data.target_names # view target names
```

Out[ ]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [ ]:
```
categories = ['talk.religion.misc', 'soc.religion.christian', 'sci.space', 'co
mp.graphics'] # only few categories selected for analysis from dataset
train = fetch_20newsgroups(subset='train', categories=categories) # download t
raining dataset
test = fetch_20newsgroups(subset='test', categories=categories) #download test
ing dataset
```

In [ ]:
```
print(train.data[3]) # view entry in data
```

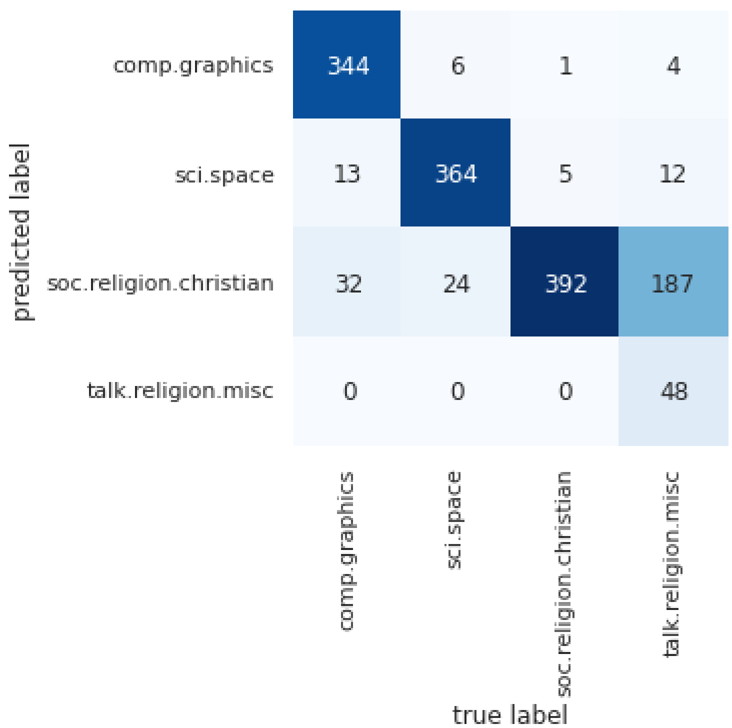
```
From: revdak@netcom.com (D. Andrew Kille)
Subject: Re: Serbian genocide Work of God?
Organization: NETCOM On-line Communication Services (408 241-9760 guest)
Lines: 22

James Sledd (jsledd@ssdc.sas.upenn.edu) wrote:
: Are the Serbs doing the work of God?  Hmm...

: I've been wondering if anyone would ever ask the question,

: Are the governments of the United States and Europe not moving
: to end the ethnic cleansing by the Serbs because the targets are
: muslims?

: Can/Does God use those who are not following him to accomplish
: tasks for him?  Esp those tasks that are punative?

: James Sledd
: no cute sig....  but I'm working on it.

Are you suggesting that God supports genocide?
Perhaps the Germans were "punishing" Jews on God's behalf?

Any God who works that way is indescribably evil, and unworthy of
my worship or faith.

revdak@netcom.com
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer # to use data, con
tent of each string is converted into a vector of numbers
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

# create a pipeline that attaches it to a multinomial naive bayes classifier
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
```

In [ ]:
```
model.fit(train.data, train.target) # train model
labels = model.predict(test.data) # predict labels for test data
```

In [ ]:

```
from sklearn.metrics import confusion_matrix
mat = confusion_matrix(test.target, labels) # confusion matrix between the tru
e and predicted labels for the test data to evaluate the performance of the es
timator
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names, yticklabels=train.target_names, cm
ap='Blues')
plt.xlabel('true label')
plt.ylabel('predicted label');

# from the matrix, we observe that the estimator can successfully separate spa
ce talk from computer talk, but gets confused between talk about religion and
 talk about Christianity
```



In [ ]:

```
# define a function that returns the prediction of a single string
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
```

In [ ]:

```
predict_category('sending a payload to the ISS') #prediction for single string
using function predict_category defined above
```

Out[ ]: 'sci.space'

In [ ]:

```
predict_category('discussing islam vs atheism')
```

Out[ ]: 'soc.religion.christian'

In [ ]:

```
predict_category('determining the screen resolution')
```

Out[ ]: 'comp.graphics'

Experiement 8- Decision Tree

AIM- Apply Decision Tree on iris dataset

In [1]:
```
!wget https://raw.githubusercontent.com/towardsai/tutorials/master/decision_tree_learning/Iris.csv
```

```
--2022-11-10 17:57:13--  https://raw.githubusercontent.com/towardsai/tutorial
s/master/decision_tree_learning/Iris.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.10
8.133, 185.199.110.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.1
08.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5103 (5.0K) [text/plain]
Saving to: ‘Iris.csv’

Iris.csv            100%[===================>]   4.98K  --.-KB/s    in 0s

2022-11-10 17:57:14 (57.3 MB/s) - ‘Iris.csv’ saved [5103/5103]
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
```

In [3]:
```python
data = pd.read_csv('Iris.csv')
data
```

Out[3]:

	Id	sepal_length	sepal_width	petal_length	petal_width	species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [4]:
```python
data.shape
```

Out[4]: (150, 6)

In [6]:
```python
data = data.drop(['Id'], axis=1) # remove useless column
```

In [7]:
```python
data.head()
```

Out[7]:

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

In [8]:
```
data.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    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [9]:
```
# check for missing values
data.isnull().sum()
```

Out[9]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [12]:
```
# selecting featuers and labels
X = data.drop(['species'], axis=1)
y = data['species']
```

In [14]:
```
from sklearn.model_selection import train_test_split
# split data intro train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, ra
ndom_state = 42)
```

Decision tree classification on GINI Index

In [16]:
```
from sklearn.tree import DecisionTreeClassifier
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=
0)
clf_gini.fit(X_train, y_train) # train model
```

Out[16]:
```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [17]:
```
y_pred_gini = clf_gini.predict(X_test) # predict using model
y_pred_gini
```

Out[17]:
```
array(['Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica'], dtype=object)
```

In [19]:
```
# testing set accuracy
clf_gini.score(X_test, y_test)
```

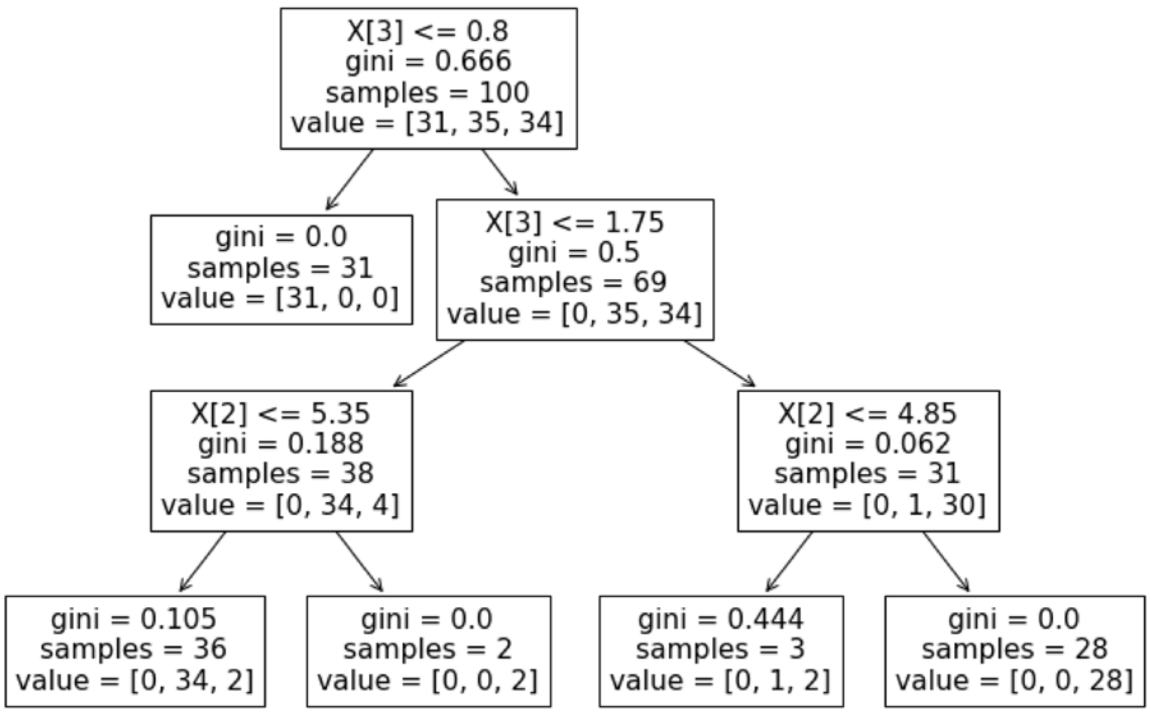
Out[19]:
```
0.98
```

In [21]:
```
# training set accuracy
clf_gini.score(X_train, y_train)
```

Out[21]:
```
0.97
```

In [23]:
```
# pictorial representation of decision trees
plt.figure(figsize=(12,8))
tree.plot_tree(clf_gini.fit(X_train, y_train))
```

Out[23]:
```
[Text(0.375, 0.875, 'X[3] <= 0.8\ngini = 0.666\nsamples = 100\nvalue = [31, 3
5, 34]'),
 Text(0.25, 0.625, 'gini = 0.0\nsamples = 31\nvalue = [31, 0, 0]'),
 Text(0.5, 0.625, 'X[3] <= 1.75\ngini = 0.5\nsamples = 69\nvalue = [0, 35, 3
4]'),
 Text(0.25, 0.375, 'X[2] <= 5.35\ngini = 0.188\nsamples = 38\nvalue = [0, 34,
4]'),
 Text(0.125, 0.125, 'gini = 0.105\nsamples = 36\nvalue = [0, 34, 2]'),
 Text(0.375, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.75, 0.375, 'X[2] <= 4.85\ngini = 0.062\nsamples = 31\nvalue = [0, 1,
30]'),
 Text(0.625, 0.125, 'gini = 0.444\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.875, 0.125, 'gini = 0.0\nsamples = 28\nvalue = [0, 0, 28]')]
```



Experiment 9- KMeans Clustering

AIM- Demonstrate k means clustering on iris dataset

In [10]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn import datasets
sns.set()
```

In [11]:
```
iris_data = pd.read_csv('iris.csv')
iris_data
```

Out[11]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

In [12]:
```
iris_data['Species'].value_counts()
```

Out[12]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

K-Means Clustering (Silhouette Coefficients)

In [29]:
```
data = iris_data.copy()
X = data.drop(columns=['Species'])
y = iris_data['Species']
```

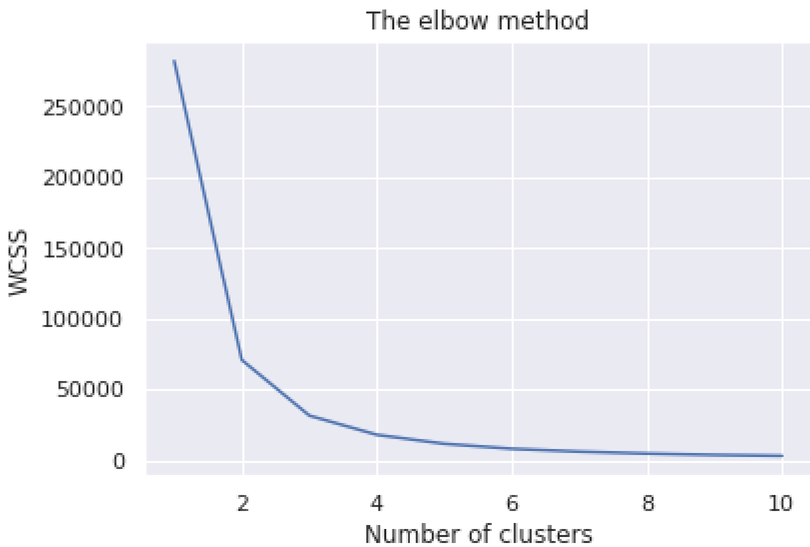
In [30]:
```
from sklearn.metrics import silhouette_score
for n_cluster in range(2, 11):
    kmeans = KMeans(n_clusters=n_cluster).fit(x)
    label = kmeans.labels_
    sil_coeff = silhouette_score(x, label, metric='euclidean')
    print(f"{n_cluster} n_clusters: Silhouette Coefficient = {sil_coeff}")
```

```
2 n_clusters: Silhouette Coefficient = 0.6205786765196579
3 n_clusters: Silhouette Coefficient = 0.5820898597618552
4 n_clusters: Silhouette Coefficient = 0.5568960211268352
5 n_clusters: Silhouette Coefficient = 0.5411170082028827
6 n_clusters: Silhouette Coefficient = 0.5322001264106738
7 n_clusters: Silhouette Coefficient = 0.5191196113307869
8 n_clusters: Silhouette Coefficient = 0.5087651863986412
9 n_clusters: Silhouette Coefficient = 0.5096484169182929
10 n_clusters: Silhouette Coefficient = 0.49474898379667565
```

K-Means Clustering (Elbow Method)

In [32]:
```
from sklearn.cluster import KMeans
x = iris_data.iloc[:, [0, 1, 2, 3]].values
wcss = [] # WCSS is the sum of squared distance between each point and the cen
troid in a cluster

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init
= 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
```

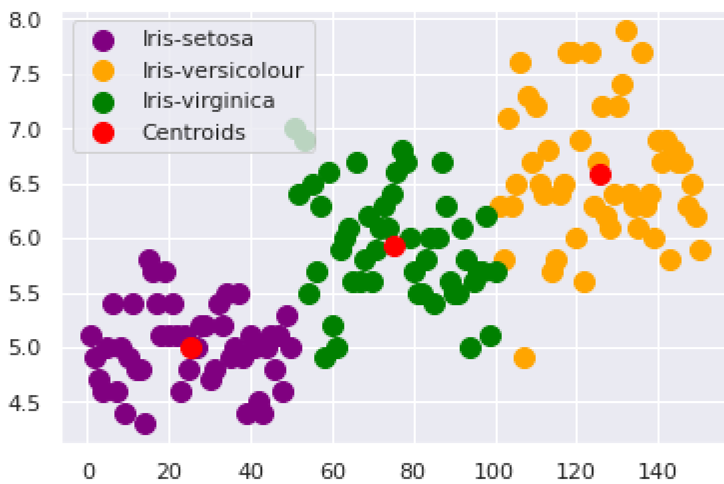
In [33]:
```
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```



In [35]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 1
0, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

In [36]:
```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'purple', l
abel = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'orange', l
abel = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', la
bel = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 1
00, c = 'red', label = 'Centroids')

plt.legend()
```

Out[36]: <matplotlib.legend.Legend at 0x7fca7b15fcd0>



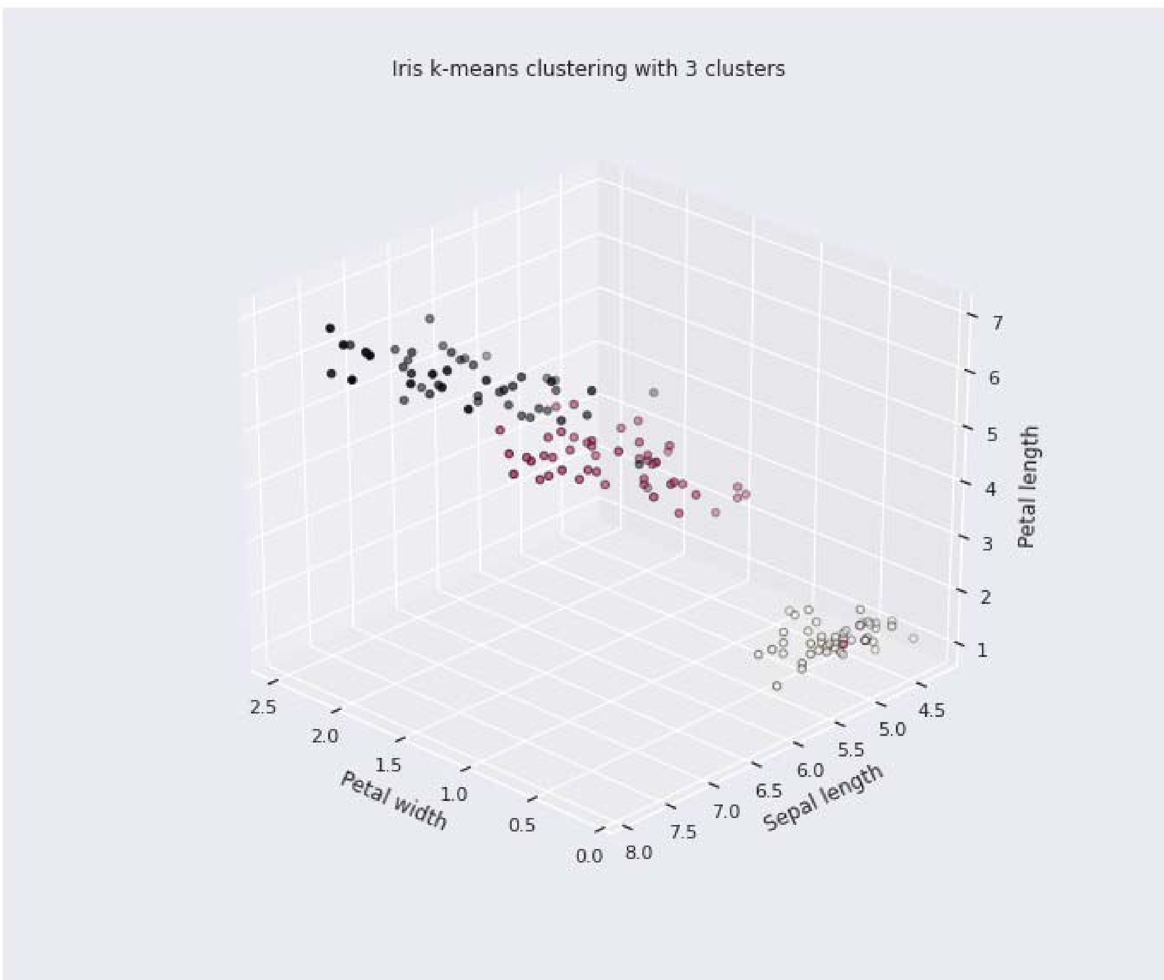
Training Model

In [38]:
```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3,n_init=1,init='random', max_iter=10000, random_sta
te=21,algorithm="auto")
fitted_model = model.fit(X)
labels = model.labels_
centers = pd.DataFrame(fitted_model.cluster_centers_)
print(f'Cluster centers: \n {centers}')
```

```
Cluster centers:
         0         1         2         3         4
0    125.0  6.570588  2.970588  5.523529  2.011765
1     74.5  5.922000  2.780000  4.206000  1.304000
2     25.0  5.006122  3.420408  1.465306  0.244898
```

Visualize Predictions K-Means

In [47]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 8))
ax = Axes3D(fig,
            rect=[0, 0, .95, 1],
            elev=30,
            azim=134)
model.fit(X)
labels = model.labels_

ax.set_title("Iris k-means clustering with 3 clusters")
ax.scatter(X["PetalWidthCm"], X["SepalLengthCm"], X["PetalLengthCm"],c=labels.
astype(np.float), edgecolor='k')
ax.set_xlabel('Petal width',labelpad=10)
ax.set_ylabel('Sepal length',labelpad=10)
ax.set_zlabel('Petal length',labelpad=10)
ax.dist = 13
plt.savefig("kclusters.png")
```



Experiment 10- Breadth First Search

AIM- Write a program in Python to Solve 3X3 magic square problem using breadth first search

In [1]:

```python
def generateSquare(n):
    magicSquare = [[0 for x in range(n)]
                   for y in range(n)]
    i = n / 2
    j = n - 1
    num = 1
    while num <= (n * n):
        if i == -1 and j == n: # 3rd condition
            j = n - 2
            i = 0
        else:
            if j == n:
                j = 0
            if i < 0:
                i = n - 1

        if magicSquare[int(i)][int(j)]: # 2nd condition
            j = j - 2
            i = i + 1
            continue
        else:
            magicSquare[int(i)][int(j)] = num
            num = num + 1

        j = j + 1
        i = i - 1 # 1st condition
    print("Magic Square for n =", n)
    print("Sum of each row or column",
        n * (n * n + 1) / 2, "\n")

    for i in range(0, n):
        for j in range(0, n):
            print('%2d ' % (magicSquare[i][j]),
                end='')
            if j == n - 1:
                print()
n = 7
generateSquare(n)
```

```
Magic Square for n = 7
Sum of each row or column 175.0

20 12  4 45 37 29 28
11  3 44 36 35 27 19
 2 43 42 34 26 18 10
49 41 33 25 17  9  1
40 32 24 16  8  7 48
31 23 15 14  6 47 39
22 21 13  5 46 38 30
```

Experiment 11- Depth First Search

AIM- Write a program in Python to Solve 8 queens problem using depth first search

In [1]:
```
import numpy as np
import pandas as pd

class NQueensSolver:
    def __init__(self):
        pass

    def set_queen(self, i, j, chessboard):
        new_chessboard = chessboard.copy()
        new_chessboard[i, j] = 1
        return new_chessboard

    def remove_queen(self, i, j, chessboard):
        new_chessboard = chessboard.copy()
        new_chessboard[i, j] = 0
        return new_chessboard

    def check_queen(self, i, j, n, chessboard):

        def in_boundary(row, col):
            return 0 <= row < n and 0 <= col < n

        # check row
        for col in range(n):
            if col != j and chessboard[i, col] == 1:
                return False

        # check column
        for row in range(n):
            if row != i and chessboard[row, j] == 1:
                return False

        # check diagonals
        diff = 0
        while in_boundary(i - diff, j - diff):
            if chessboard[i - diff, j - diff] == 1:
                return False
            diff += 1

        diff = 0
        while in_boundary(i - diff, j + diff):
            if chessboard[i - diff, j + diff] == 1:
                return False
            diff += 1

        diff = 0
        while in_boundary(i + diff, j - diff):
            if chessboard[i + diff, j - diff] == 1:
                return False
            diff += 1

        diff = 0
        while in_boundary(i + diff, j + diff):
            if chessboard[i + diff, j + diff] == 1:
                return False
            diff += 1
```

```
        return True

    def solve(self, n): # Depth Tree Search with Stack
        solutions = []
        root_data = {
            'row':-1,
            'chessboard': np.zeros((n, n), dtype=int)
        }

        stack = [root_data]
        while len(stack) > 0:
            data = stack.pop()

            row = data['row']
            chessboard = data['chessboard']

            if row == n - 1:
                solutions.append(chessboard)
            else:
                row += 1

                for col in range(n-1, -1, -1): # Iterate through n-1, n-2,
 ..., 3, 2, 1, 0
                    if self.check_queen(row, col, n, chessboard):
                        data = {
                            'row': row,
                            'chessboard': self.set_queen(row, col, chessboard)
                        }
                        stack.append(data)

        return solutions

solver = NQueensSolver()
for soln in solver.solve(5):
    print(soln, '\n')
```

```
[[1 0 0 0 0]
 [0 0 1 0 0]
 [0 0 0 0 1]
 [0 1 0 0 0]
 [0 0 0 1 0]]

[[1 0 0 0 0]
 [0 0 0 1 0]
 [0 1 0 0 0]
 [0 0 0 0 1]
 [0 0 1 0 0]]

[[0 1 0 0 0]
 [0 0 0 1 0]
 [1 0 0 0 0]
 [0 0 1 0 0]
 [0 0 0 0 1]]

[[0 1 0 0 0]
 [0 0 0 0 1]
 [0 0 1 0 0]
 [1 0 0 0 0]
 [0 0 0 1 0]]

[[0 0 1 0 0]
 [1 0 0 0 0]
 [0 0 0 1 0]
 [0 1 0 0 0]
 [0 0 0 0 1]]

[[0 0 1 0 0]
 [0 0 0 0 1]
 [0 1 0 0 0]
 [0 0 0 1 0]
 [1 0 0 0 0]]

[[0 0 0 1 0]
 [1 0 0 0 0]
 [0 0 1 0 0]
 [0 0 0 0 1]
 [0 1 0 0 0]]

[[0 0 0 1 0]
 [0 1 0 0 0]
 [0 0 0 0 1]
 [0 0 1 0 0]
 [1 0 0 0 0]]

[[0 0 0 0 1]
 [0 1 0 0 0]
 [0 0 0 1 0]
 [1 0 0 0 0]
 [0 0 1 0 0]]

[[0 0 0 0 1]
 [0 0 1 0 0]
 [1 0 0 0 0]
```

```
[0 0 0 1 0]
[0 1 0 0 0]]
```