VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES
VIVEKANANDA SCHOOL OF INFORMATION TECHNOLOGY

BACHELOR OF COMPUTER APPLICATION
Practical- Machine Learning with Python BCA-311

Guru Gobind Singh Indraprastha University
Sector - 16C Dwarka, Delhi – 110078

SUBMITTED TO:	**SUBMITTED BY:**
Dr. Priyanka Gupta	Dhruv Sharma
Assistant Professor	02229802021
VSIT	BCA 5EA

LIST OF PRACTICALS

1. Extract the data from the database using python.

Solution:

import pandas as pd
dataset = pd.read_csv('Data.csv')
 x = dataset.iloc[:, :-1].values
 y = dataset.iloc[:, -1].values
dataset

Output:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

2. Write a program to implement linear and logistic regression

Solution: # Linear

Regression

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Salary_Data.csv')
x = dataset.iloc[:,:-1].values
y = dataset.iloc[:,-1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, random_state = 0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train) y_pred =
regressor.predict(X_test)
print(regressor.predict([[5.6]]))

Output:

```
Prediction:  [79153.46992552]
```

 # Logisitic Regression

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
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 = 0)
from sklearn.preprocessing import StandardScaler sc = StandardScaler()
X_train = sc.fit_transform(X_train) X_test
= sc.transform(X_test)
from sklearn.linear_model import LogisticRegression classifier
= LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
print(f"Prediction of Purchased: ",classifier.predict(sc.transform([[30,87000]])))
```

Output:

```
Prediction of Purchased:  [0]
```

3. Write a program to implement the Naïve Bayesian Classifier for a sample training data set stored as a .csv file. Compute the accuracy of the classifier, considering few test data sets.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
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 = 0)
print(X_train) print(y_train) print(X_test) print(y_test)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train) print(X_test)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB() classifier.fit(X_train,
y_train)
print("Prediction of purchased: ",classifier.predict(sc.transform([[30,87000]])))
y_pred = classifier.predict(X_test)
```

```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
from sklearn.metrics import accuracy_score
print("Accuracy of Model: ",accuracy_score(y_test, y_pred))
```

Output:

```
Prediction of purchased:  [0]
```

```
Accuracy of Model:  0.9
```

4. Write a program to implement K-Nearest Neighbours (KNN) and Support Vector Machine (SVM) Algorithm for classification.

Solution:

 # SVM Algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
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 = 0)
print(X_train) print(y_train) print(X_test) print(y_test)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train) print(X_test)
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
print("Prediction of purchased: ",classifier.predict(sc.transform([[32,150000]])))
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
from sklearn.metrics import accuracy_score
print("Accuracy of Model: ",accuracy_score(y_test, y_pred))
```

Output:

```
Prediction of Purchase: [1]
```

```
Accuracy of Model:  0.9
```

KNN Algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
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 = 0)
print(X_train) print(y_train) print(X_test) print(y_test)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train) print(X_test)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
print("Prediction of purchased: ",classifier.predict(sc.transform([[32,150000]])))
```

Output:

```
Prediction of purchased:  [0]
```

5. Implement classification of a given dataset using random forest.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
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 = 0)
print(X_train) print(y_train) print(X_test) print(y_test)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train) print(X_test)
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
print(classifier.predict(sc.transform([[15,20000]])))
```

Output:

```
Prediction of Purchase: [1]
```

6. Build an Artificial Neural Network (ANN) by implementing the Back Propagation algorithm and test the same using appropriate data sets.

Solution:

```
import numpy as np
X = np.array(([2,9],[1,5],[3,6]),dtype=float)
y = np.array(([92],[86],[89]),dtype=float)
X = X/np.amax(X,axis=0)
y = y/100 y

def sigmoid(x):
   return 1/(1+np.exp(-x))

def derivatives_sigmoid(x):
   return x*(1-x)
epoch = 5000
lr = 0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh = np.random.uniform(size=(1,hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout = np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
 hinp1 = np.dot(X,wh)
```

```
hinp = hinp1+bh
hlayer_act = sigmoid(hinp)
outinp1 = np.dot(hlayer_act,wout)
outinp = outinp1+bout
output = sigmoid(outinp)
EO = y - output
   outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
   hiddengrad = derivatives_sigmoid(hlayer_act)
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) * lr
   wh += X.T.dot(d_hiddenlayer) *lr

print("Input: "+str(X))
print('Actual Output: '+str(y))
print('Predicted Output: ',output)
```

Output:

```
Input: [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output: [[0.92]
 [0.86]
 [0.89]]
Predicted Output:  [[0.89928189]
 [0.87483272]
 [0.89492914]]
```

7. Apply K-Means algorithm to cluster a set of data stored in .csv file. Use the same data for clustering using the Hierarchical algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML, library classes in the program.

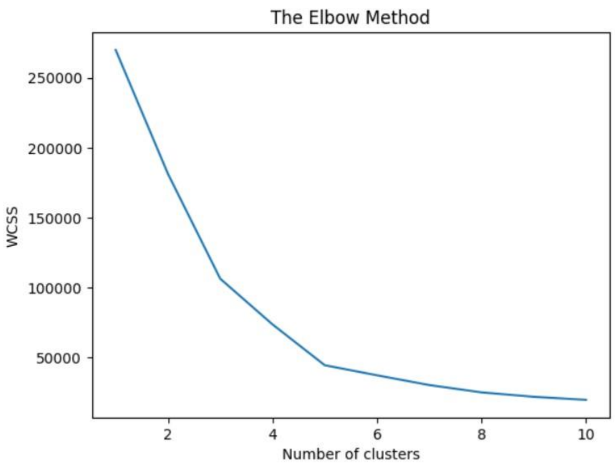
Solution:

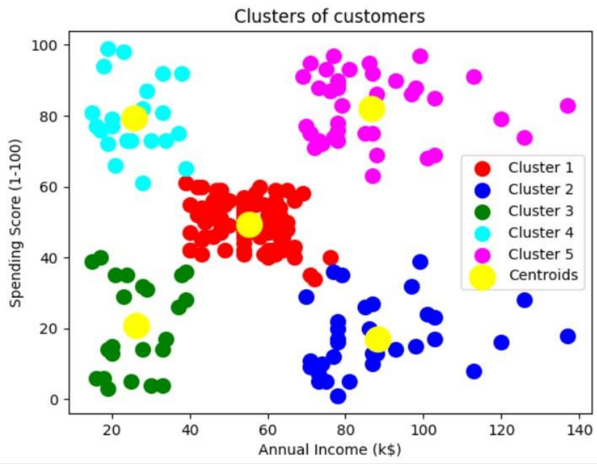
 # K-Means algorithm

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values X
from sklearn.cluster import KMeans
wcss = [] for i in range(1, 11):
kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
kmeans.fit(X)
```

```
wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss) plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') plt.show()
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label =
'Centroids') plt.title('Clusters of
customers') plt.xlabel('Annual
Income (k$)') plt.ylabel('Spending
Score (1-100)') plt.legend()
plt.show()
```

Output:



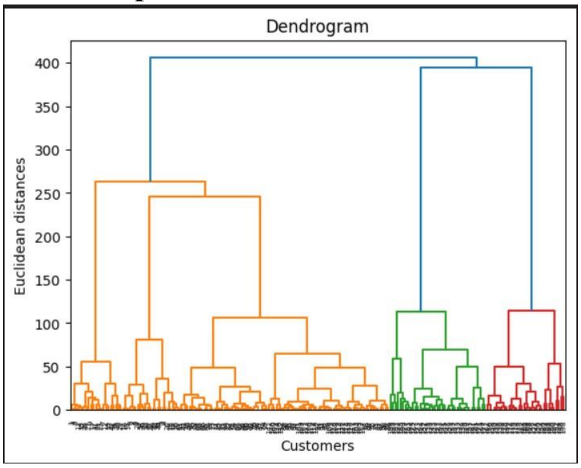


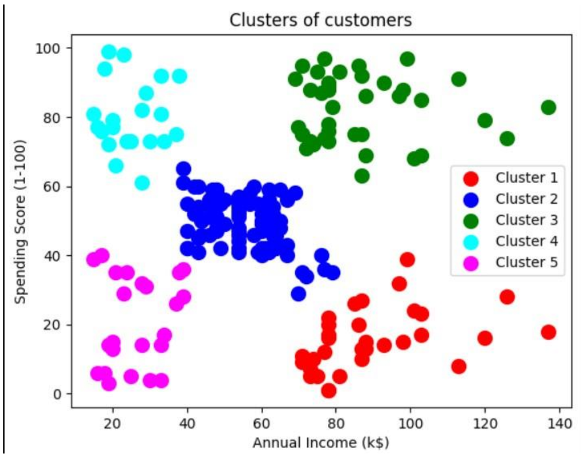
Hierarchical clustering

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram') plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)') plt.legend()
plt.show()
```

Output:





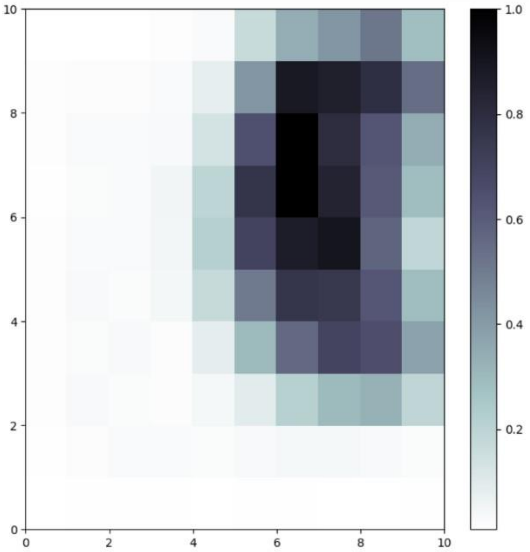
8. Write a program to implement Self-Organising Map (SOM).

Solution:

```
from minisom import MiniSom
import numpy as np import
matplotlib.pyplot as plt import
pandas as pd
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
som_grid_rows = 10
som_grid_columns = 10
input_len = 2
# Size of input data (number of features) sigma = 1.0
# Spread of neighborhood function learning_rate = 0.5
som = MiniSom(som_grid_rows, som_grid_columns, input_len, sigma=sigma,
learning_rate=learning_rate)
som.train_random(X, len(X))
plt.figure(figsize=(8, 8))
plt.pcolor(som.distance_map().T, cmap='bone_r')
# distance map as background plt.colorbar()
for i, x in enumerate(X):
    w = som.winner(x)
# getting the winner
```

```
    plt.plot(w[0] + 0.5, w[1] + 0.5, 'x', markerfacecolor='None', markeredgecolor='blue',
markersize=10, markeredgewidth=2)
plt.title('Self-Organizing Map') plt.show()
```

Output:



9. Write a program for empirical comparison of different supervised learning algorithm. **Solution:**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier from
sklearn.ensemble import RandomForestClassifier from
sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
models = {
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC()
}
results = {}
for name, model in models.items():
model.fit(X_train, y_train)
 y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
```

```
    f1 = f1_score(y_test, y_pred, average='macro')
results[name] = {
'Accuracy': accuracy,
'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    }
print("Results:")
for name, scores in results.items():
    print(f"{name}:")
for metric, score in scores.items():
print(f"    {metric}: {score:.4f}")
    print()
```

Output:

```
Results:
Logistic Regression:
   Accuracy: 1.0000
   Precision: 1.0000
   Recall: 1.0000
   F1 Score: 1.0000

Decision Tree:
   Accuracy: 1.0000
   Precision: 1.0000
   Recall: 1.0000
   F1 Score: 1.0000

Random Forest:
   Accuracy: 1.0000
   Precision: 1.0000
   Recall: 1.0000
   F1 Score: 1.0000

SVM:
   Accuracy: 1.0000
   Precision: 1.0000
   Recall: 1.0000
   F1 Score: 1.0000
```

10. Write a program for empirical comparison of different unsupervised learning algorithm. **Solution:**

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture from sklearn.metrics
import silhouette_score
data = load_iris()
X = data.data
y = data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
models = {
  'K-Means': KMeans(n_clusters=3),
  'Hierarchical': AgglomerativeClustering(n_clusters=3),
  'GaussianMixture': GaussianMixture(n_components=3),
  'DBSCAN': DBSCAN(eps=0.5, min_samples=5),
}
results = {}
for name, model in models.items():
if name == 'DBSCAN':
     # DBSCAN doesn't predict directly, so fit_predict is used
labels = model.fit_predict(X_scaled)
else:
     labels = model.fit_predict(X_scaled)
silhouette = silhouette_score(X_scaled, labels)
results[name] = silhouette print("Silhouette
Scores:")
for name, score in results.items():
print(f"{name}: {score:.4f}")
```

Output:

```
Silhouette Scores:
K-Means: 0.4594
Hierarchical: 0.4467
GaussianMixture: 0.3742
DBSCAN: 0.3565
```

11. Write a program to build Decision Trees using

i) Information Gain ii) Gini
Index

Using appropriate dataset. Visualize the trees and compare all the performance metrics.

Solution: # Information Gain

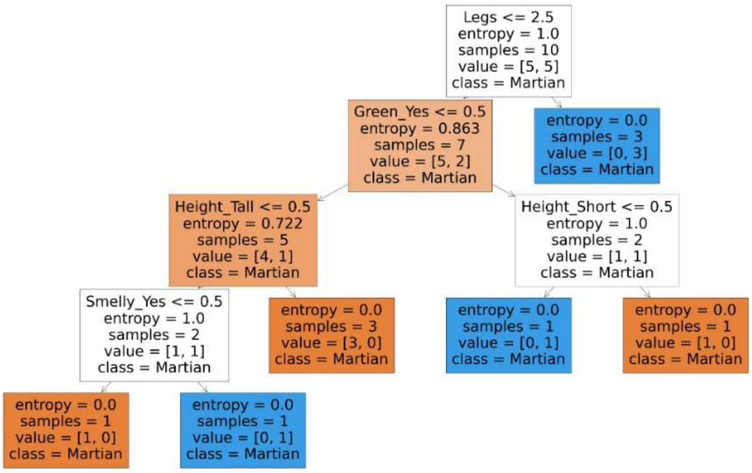
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd dataset =
pd.read_csv('species.csv')
df = pd.DataFrame(dataset)
X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values
print(X)
print(y)
df_encoded = pd.get_dummies(df.drop(columns=['Species']))
target = df['Species']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_encoded,y, test_size=0.33, random_state=324)
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='entropy',random_state=42)
clf.fit(df_encoded,target)
new_data_point = pd.DataFrame({
   'Green': ["Yes"],
   'Legs': [2],
   'Height': ["Tall"],
   'Smelly': ["No"]
})
new_data_encoded = pd.get_dummies(new_data_point)
missing_cols = set(df_encoded.columns) - set(new_data_encoded.columns)
for col in missing_cols:
   new_data_encoded[col] = 0
new_data_encoded = new_data_encoded[df_encoded.columns]
predicted_species = clf.predict(new_data_encoded)
print("Predicted Species:", predicted_species)
from sklearn import tree print(tree.export_text(clf))
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(50,30))
_ = tree.plot_tree(clf,feature_names=df_encoded.columns,class_names=target,filled=True)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Calculate precision
precision = precision_score(y_test, y_pred, average='weighted')
print(f'Precision: {precision:.4f}')
```

# Calculate recall
recall = recall_score(y_test, y_pred, average='weighted')
print(f'Recall: {recall:.4f}')

# Calculate F1-score
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'F1 Score: {f1:.4f}')

# Generate confusion matrix conf_matrix =
confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

Output:

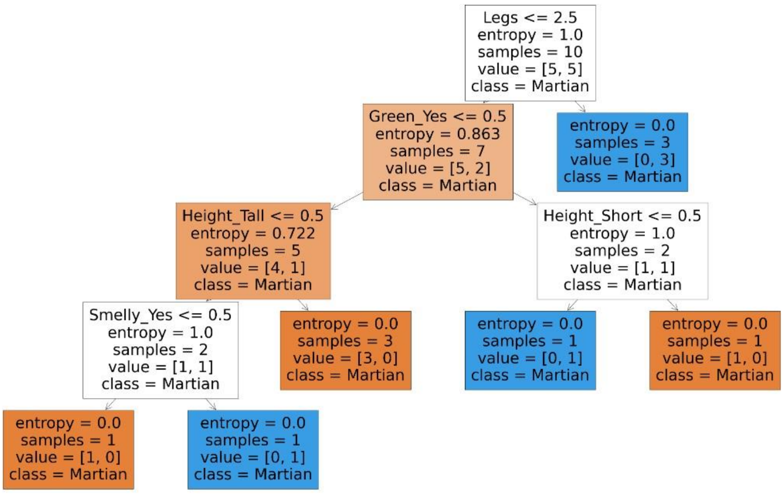



```
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
Confusion Matrix:
[[2 0]
 [0 2]]
```

 # Gini impurity
import numpy as np
import pandas as pd
dataset = pd.read_csv('species.csv')
df = pd.DataFrame(dataset)
df

```
X = dataset.iloc[:, 1:].values
y = dataset.iloc[:, 0].values
print(X) print(y)
df_encoded = pd.get_dummies(df.drop(columns=['Species']))
target = df['Species']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_encoded,y, test_size=0.33, random_state=324)
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='gini',random_state=42) clf.fit(df_encoded,target)
new_data_point = pd.DataFrame({
    'Green': ["Yes"],
    'Legs': [2],
    'Height': ["Tall"],
    'Smelly': ["No"]
})
new_data_encoded = pd.get_dummies(new_data_point)
missing_cols = set(df_encoded.columns) - set(new_data_encoded.columns)
for col in missing_cols:
    new_data_encoded[col] = 0
new_data_encoded = new_data_encoded[df_encoded.columns]
predicted_species = clf.predict(new_data_encoded)
print("Predicted Species:", predicted_species)
from sklearn import tree print(tree.export_text(clf))
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(50,30))
_ = tree.plot_tree(clf,feature_names=df_encoded.columns,class_names=target,filled=True)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred) print(f'Accuracy:
{accuracy:.4f}')

# Calculate precision
precision = precision_score(y_test, y_pred, average='weighted') print(f'Precision:
{precision:.4f}')

# Calculate recall
recall = recall_score(y_test, y_pred, average='weighted')
print(f'Recall: {recall:.4f}')

# Calculate F1-score
f1 = f1_score(y_test, y_pred, average='weighted')
print(f'F1 Score: {f1:.4f}')

# Generate confusion matrix conf_matrix =
confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

Output:




```
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1 Score: 1.0000
Confusion Matrix:
[[2 0]
 [0 2]]
```

12. Implement all the steps of Data Preprocessing on the appropriate dataset. Include handling missing data, encoding categorical data, and feature scaling in addition to the basic steps.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
dataset
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(x[:, 1:3])
```

```
x[:, 1:3] = imputer.transform(x[:,1:3])
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
x = np.array(ct.fit_transform(x)) print(x)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder() y= le.fit_transform(y)
print(y)
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, random_state = 1)
print(X_train) print(X_test) print(y_train) print(y_test)
from sklearn.preprocessing import StandardScaler sc
= StandardScaler()
X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
X_test[:, 3:] = sc.transform(X_test[:, 3:])
print(X_train) print(X_test)
```

Output:

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
[[0.0 1.0 0.0 0.0 44.0 72000.0]
 [1.0 0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 1.0 0.0 30.0 54000.0]
 [1.0 0.0 0.0 1.0 38.0 61000.0]
 [1.0 0.0 1.0 0.0 40.0 63777.77777777778]
 [0.0 1.0 0.0 0.0 35.0 58000.0]
 [1.0 0.0 0.0 1.0 38.77777777777778 52000.0]
 [0.0 1.0 0.0 0.0 48.0 79000.0]
 [1.0 0.0 1.0 0.0 50.0 83000.0]
 [0.0 1.0 0.0 0.0 37.0 67000.0]]
```

```
[[1.0 0.0 0.0 1.2909944487358056 -0.19159184384578545 -1.0781259408412425]
 [1.0 0.0 1.0 -0.7745966692414834 -0.014117293757057777
  -0.07013167641635372]
 [0.0 1.0 0.0 -0.7745966692414834 0.566708506533324 0.633562432710455]
 [1.0 0.0 0.0 1.2909944487358056 -0.30453019390224867
  -0.30786617274297867]
 [1.0 0.0 0.0 1.2909944487358056 -1.9018011447007988 -1.420463615551582]
 [0.0 1.0 0.0 -0.7745966692414834 1.1475343068237058 1.232653363453549]
 [1.0 0.0 1.0 -0.7745966692414834 1.4379472069688968 1.5749910381638885]
 [0.0 1.0 0.0 -0.7745966692414834 -0.7401495441200351 -0.5646194287757332]]
[[1.0 0.0 1.0 -0.7745966692414834 -1.4661817944830124 -0.9069571034860727]
 [0.0 1.0 0.0 -0.7745966692414834 -0.44973664397484414 0.2056403393225306]]
```