Problem and Dataset Description

You have been provided with a dataset containing various attributes about the behavior of an online shopper and whether they made a purchase or not. Your task is to build a decision tree model to predict whether a visitor to the webpage actually made a purchase or not based on the provided attributes.

Dataset columns:

- Electronic_Devices : the number of pages of electronic devices visited by the shopper in a session
- Electronic_Devices_Duration : the total time spent in electronic devices category by the shopper
- Groceries : the number of pages of groceries visited by the shopper
- Groceries_Duration : the total time spent in groceries category by the shopper
- Sports_Equipments : the number of pages of sports equipments visited by the shopper
- Sports_Equipments_Duration : the total time spent in sports equipments category by the shopper
- Bounce_Rates : this feature for a web page refers to the percentage of visitors who enter the site from that page and then leave ("bounce") without triggering any other requests
- Special_Day : this feature indicates the closeness of the site visiting time to a specific special day (e.g. Mother's Day, Independence Day)
- Month : the specific month of the year
- Browser : the browser used by the shopper
- Region : the region where the searches were made
- Type_of_visitor : this feature indicates whether the shopper is a returning or new visitor to the page
- Weekend : Boolean value indicating whether the date of the visit is weekend
- Purchase_made : Boolean value indicating whether the purchase was made or not

Problem : Decision Tree

1. Using the provided dataset, build a decision tree model that can predict whether a visitor will make a purchase during their online session. Additionally, evaluate the performance of your decision tree model using appropriate metrics such as accuracy, precision, recall, and F1-score.
2. Which attribute(s) did your decision tree identify as the most important for predicting whether a visitor will make a purchase or not?
3. What is the maximum depth of your decision tree and how did you estimate it?
4. What is the accuracy of your decision tree model in predicting purchase behavior, and did you employ any techniques to handle categorical features or missing values in the dataset?

```
# DO NOT EDIT

!pip install gdown
!gdown 18NuvJotUFiTAHW0YgaoLVu2blW_V6YX0
!unzip -o /content/assignment2.zip -d data

import pandas as pd

df1 = pd.read_csv('/content/data/assignment2-1.csv')
df2 = pd.read_csv('/content/data/assignment2-2.csv')
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.12.2)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.7
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7
Downloading...
From: https://drive.google.com/uc?id=18NuvJotUFiTAHW0YgaoLVu2blW_V6YX0
To: /content/assignment2.zip
100% 120M/120M [00:02<00:00, 56.7MB/s]
Archive:  /content/assignment2.zip
  inflating: data/assignment2-1.csv
  inflating: data/assignment2-2.csv
```

```
# Load libraries
import pandas as pd
```

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
```

```
#CREATED A COPY OF THE ORIG DATASET for later use if required
defdectree = df1[["ElectronicDevices", "ElectronicDevices_Duration", "Groceries", "Groceries_Duration", "SportsRelated", "Sports_Equipme
```

```
defdectree.head()
```

	ElectronicDevices	ElectronicDevices_Duration	Groceries	Groceries_Duration	SportsRelat
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1) # 80% training and 20% test
```

We need to identify the various values for category related columns and then assign numerical values

```
print(defdectree['Month'].unique())
```

```
['May' 'Mar']
```

```
d = {'May': 0, 'Mar': 1}
defdectree['Month'] = defdectree['Month'].map(d)
```

```
print(defdectree['Browser'].unique())
```

```
['Mozilla' 'Edge' 'Opera' 'Brave' 'Chrome' 'DuckDuckGo' '7' 'Mozilla0' '8'
 '9']
```

```
d = {'Mozilla': 0, 'Edge' : 1, 'Opera' : 2, 'Brave' : 3, 'Chrome': 4, 'DuckDuckGo': 5, '7': 6, 'Mozilla0': 7, '8': 8, '9': 9,}
defdectree['Browser'] = defdectree['Browser'].map(d)
```

```
print(defdectree['Type_of_visitor'].unique())
```

```
['Old_Visitor' 'New_Visitor']
```

```
d = {'Old_Visitor': 0, 'New_Visitor': 1}
defdectree['Type_of_visitor'] = defdectree['Type_of_visitor'].map(d)
```

```
defdectree['Weekend'] = defdectree['Weekend'].astype(int)
```

```
defdectree['Purchase_made'] = defdectree['Purchase_made'].astype(int)
```

```
defdectree.head()
```

	ElectronicDevices	ElectronicDevices_Duration	Groceries	Groceries_Duration	SportsRelat
0	0	0.0	0	0.0	
1	0	0.0	0	0.0	
2	0	0.0	0	0.0	
3	0	0.0	0	0.0	
4	0	0.0	0	0.0	

```
features = ['ElectronicDevices', 'ElectronicDevices_Duration', 'Groceries', 'Groceries_Duration', 'SportsRelated', 'Sports_Equipments_Du

X = defdectree[features]
y = defdectree['Purchase_made']
```

TRAIN THE MODEL

```
# Create Decision Tree classifier object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifier
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
```

find the metrics incl accuracy, prec Score, F1 Score and Recall

```
print('Accuracy:  %.3f' % accuracy_score(y_test, y_pred))
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
```

```
Accuracy:  0.846
Precision: 0.185
Recall: 0.233
F1 Score: 0.206
```

```
import matplotlib.pyplot as plt
import sys
import matplotlib
```

What is the maximum depth of your decision tree and how did you estimate it?

```
text_representation = tree.export_text(clf)

print('Max Depth of Dec Tree is :: %.3f' % clf.tree_.max_depth)

print(text_representation)
```

```
Max Depth of Dec Tree is :: 18.000
|--- feature_5 <= 750.96
|   |--- feature_5 <= 244.58
|   |   |--- feature_1 <= 14.50
|   |   |   |--- class: 0
|   |   |--- feature_1 >  14.50
|   |   |   |--- feature_1 <= 15.50
|   |   |   |   |--- feature_4 <= 6.50
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_4 >  6.50
|   |   |   |   |   |--- class: 1
|   |   |   |--- feature_1 >  15.50
|   |   |   |   |--- feature_0 <= 1.50
|   |   |   |   |   |--- feature_9 <= 2.00
|   |   |   |   |   |   |--- feature_1 <= 112.75
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_1 >  112.75
|   |   |   |   |   |   |   |--- feature_1 <= 162.50
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- feature_1 >  162.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- feature_9 >  2.00
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_0 >  1.50
|   |   |   |   |   |--- class: 0
|   |--- feature_5 >  244.58
|   |   |--- feature_5 <= 245.25
|   |   |   |--- class: 1
|   |   |--- feature_5 >  245.25
|   |   |   |--- feature_11 <= 0.50
|   |   |   |   |--- feature_1 <= 23.50
|   |   |   |   |   |--- feature_3 <= 14.17
|   |   |   |   |   |   |--- feature_4 <= 16.50
|   |   |   |   |   |   |   |--- feature_4 <= 8.50
```

```
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- feature_4 >  8.50
|   |   |   |   |   |   |   |   |   |--- feature_5 <= 323.46
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- feature_5 >  323.46
|   |   |   |   |   |   |   |   |   |   |--- feature_5 <= 332.96
|   |   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |   |   |--- feature_5 >  332.96
|   |   |   |   |   |   |   |   |   |   |   |--- feature_4 <= 10.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |   |--- feature_4 >  10.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |--- feature_4 >  16.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- feature_3 >  14.17
|   |   |   |   |   |   |   |--- feature_3 <= 43.50
|   |   |   |   |   |   |   |   |--- feature_4 <= 21.00
|   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |   |--- feature_4 >  21.00
|   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- feature_3 >  43.50
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- feature_1 >  23.50
|   |   |   |   |   |--- feature_4 <= 30.50
```

```
fig = plt.figure(figsize=(25,20))
tree.plot_tree(clf)
```

```
[Text(0.42057057899461403, 0.9736842105263158, 'x[5] <= 750.962\ngini = 0.166\nsamples = 1503\nvalue = [1366, 137]'),
 Text(0.11826750448833034, 0.9210526315789473, 'x[5] <= 244.583\ngini = 0.084\nsamples = 1005\nvalue = [961, 44]'),
 Text(0.02154398563734291, 0.868421052631579, 'x[1] <= 14.5\ngini = 0.011\nsamples = 558\nvalue = [555, 3]'),
 Text(0.01436265709156194, 0.8157894736842105, 'gini = 0.0\nsamples = 463\nvalue = [463, 0]'),
 Text(0.02872531418312388, 0.8157894736842105, 'x[1] <= 15.5\ngini = 0.061\nsamples = 95\nvalue = [92, 3]'),
 Text(0.01436265709156194, 0.7631578947368421, 'x[4] <= 6.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.00718132854578097, 0.7105263157894737, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.02154398563734291, 0.7105263157894737, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.04308797127468582, 0.7631578947368421, 'x[0] <= 1.5\ngini = 0.042\nsamples = 93\nvalue = [91, 2]'),
 Text(0.03590664272890485, 0.7105263157894737, 'x[9] <= 2.0\ngini = 0.208\nsamples = 17\nvalue = [15, 2]'),
 Text(0.02872531418312388, 0.6578947368421053, 'x[1] <= 112.75\ngini = 0.117\nsamples = 16\nvalue = [15, 1]'),
 Text(0.02154398563734291, 0.6052631578947368, 'gini = 0.0\nsamples = 14\nvalue = [14, 0]'),
 Text(0.03590664272890485, 0.6052631578947368, 'x[12] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.02872531418312388, 0.5526315789473685, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.04308797127468582, 0.5526315789473685, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.04308797127468582, 0.6578947368421053, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.05026929982046679, 0.7105263157894737, 'gini = 0.0\nsamples = 76\nvalue = [76, 0]'),
 Text(0.21499102333931777, 0.868421052631579, 'x[5] <= 245.25\ngini = 0.167\nsamples = 447\nvalue = [406, 41]'),
 Text(0.2078096947935368, 0.8157894736842105, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.22217235188509873, 0.8157894736842105, 'x[11] <= 0.5\ngini = 0.163\nsamples = 446\nvalue = [406, 40]'),
 Text(0.14631956912028726, 0.7631578947368421, 'x[1] <= 23.5\ngini = 0.131\nsamples = 368\nvalue = [342, 26]'),
 Text(0.08617594254937164, 0.7105263157894737, 'x[3] <= 14.167\ngini = 0.07\nsamples = 247\nvalue = [238, 9]'),
 Text(0.0718132854578097, 0.6578947368421053, 'x[4] <= 16.5\ngini = 0.042\nsamples = 232\nvalue = [227, 5]'),
 Text(0.06463195691202872, 0.6052631578947368, 'x[4] <= 8.5\ngini = 0.066\nsamples = 147\nvalue = [142, 5]'),
 Text(0.05745062836624776, 0.5526315789473685, 'gini = 0.0\nsamples = 56\nvalue = [56, 0]'),
 Text(0.0718132854578097, 0.5526315789473685, 'x[5] <= 323.458\ngini = 0.104\nsamples = 91\nvalue = [86, 5]'),
 Text(0.06463195691202872, 0.5, 'gini = 0.0\nsamples = 30\nvalue = [30, 0]'),
 Text(0.07899461400359066, 0.5, 'x[5] <= 332.958\ngini = 0.15\nsamples = 61\nvalue = [56, 5]'),
 Text(0.0718132854578097, 0.4473684210526316, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.08617594254937164, 0.4473684210526316, 'x[4] <= 10.5\ngini = 0.097\nsamples = 59\nvalue = [56, 3]'),
 Text(0.0718132854578097, 0.39473684210526316, 'x[9] <= 0.5\ngini = 0.298\nsamples = 11\nvalue = [9, 2]'),
 Text(0.06463195691202872, 0.34210526315789475, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.07899461400359066, 0.34210526315789475, 'x[5] <= 411.625\ngini = 0.18\nsamples = 10\nvalue = [9, 1]'),
 Text(0.0718132854578097, 0.2894736842105263, 'x[10] <= 1.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.06463195691202872, 0.23684210526315788, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.07899461400359066, 0.23684210526315788, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.08617594254937164, 0.2894736842105263, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.10053859964093358, 0.39473684210526316, 'x[4] <= 15.5\ngini = 0.041\nsamples = 48\nvalue = [47, 1]'),
 Text(0.0933572710951526, 0.34210526315789475, 'gini = 0.0\nsamples = 38\nvalue = [38, 0]'),
 Text(0.10771992818671454, 0.34210526315789475, 'x[1] <= 3.0\ngini = 0.18\nsamples = 10\nvalue = [9, 1]'),
 Text(0.10053859964093358, 0.2894736842105263, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.11490125673249552, 0.2894736842105263, 'x[8] <= 0.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.10771992818671454, 0.23684210526315788, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.12208258527827648, 0.23684210526315788, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.07899461400359066, 0.6052631578947368, 'gini = 0.0\nsamples = 85\nvalue = [85, 0]'),
 Text(0.10053859964093358, 0.6578947368421053, 'x[3] <= 43.5\ngini = 0.391\nsamples = 15\nvalue = [11, 4]'),
 Text(0.0933572710951526, 0.6052631578947368, 'x[1] <= 18.0\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.08617594254937164, 0.5526315789473685, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.10053859964093358, 0.5526315789473685, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.10771992818671454, 0.6052631578947368, 'gini = 0.0\nsamples = 10\nvalue = [10, 0]'),
 Text(0.20646319569120286, 0.7105263157894737, 'x[4] <= 30.5\ngini = 0.242\nsamples = 121\nvalue = [104, 17]'),
 Text(0.19210053859964094, 0.6578947368421053, 'x[6] <= 0.04\ngini = 0.205\nsamples = 112\nvalue = [99, 13]'),
 Text(0.18491921005385997, 0.6052631578947368, 'x[1] <= 24.75\ngini = 0.193\nsamples = 111\nvalue = [99, 12]'),
 Text(0.17773788150807898, 0.5526315789473685, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.19210053859964094, 0.5526315789473685, 'x[10] <= 4.5\ngini = 0.18\nsamples = 110\nvalue = [99, 11]'),
 Text(0.1615798922800718, 0.5, 'x[2] <= 3.5\ngini = 0.116\nsamples = 81\nvalue = [76, 5]'),
 Text(0.1436265709156194, 0.4473684210526316, 'x[1] <= 26.5\ngini = 0.096\nsamples = 79\nvalue = [75, 4]'),
 Text(0.12926391382405744, 0.39473684210526316, 'x[6] <= 0.005\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
```

```
clf2 = DecisionTreeClassifier(criterion='gini')
```

```
# Fit the decision tree classifier
clf2 = clf2.fit(X_train, y_train)
print('Max Depth of Dec Tree is :: %.3f' % clf2.tree_.max_depth)
```

```
Max Depth of Dec Tree is :: 17.000
```

```
feature_importances = clf2.feature_importances_
```

```
# WHICH FEATURE IS MOST IMPORTANT
```

WHICH FEATURE IS MOST IMPORTANT

```
import seaborn as sns

# Sort the feature importances from greatest to least using the sorted indices
sorted_indices = feature_importances.argsort()[::-1]
#sorted_feature_names = features[sorted_indices]
#print("SORTED FEATURE NAMES")
#print(sorted_feature_names)

print("SORTED INDICES")
sorted_importances = feature_importances[sorted_indices]
print(sorted_indices)

# Create a bar plot of the feature importances
sns.set(rc={'figure.figsize':(11.7,8.27)})
#sns.barplot(sorted_importances, sorted_feature_names)
print("SORTED IMPORTANCES")
print(sorted_importances)
```

```
SORTED INDICES
[ 5  4  1  3  0  9  6 10 12  8 11  7  2]
SORTED IMPORTANCES
[0.28962889 0.15910056 0.15147824 0.08414852 0.06816402 0.06807486
 0.06616778 0.04859415 0.0475849  0.00917867 0.00787941 0.
 0.        ]
```

```
#print('Highest Important Feature:  %.3f' % features[0])

print("MOST IMPORTANT FEATURE")
most_important = sorted_indices[0]
print(features[most_important])

print("SECOND MOST IMPORTANT FEATURE")
second_most_important = sorted_indices[1]
print(features[second_most_important])
```

```
MOST IMPORTANT FEATURE
Sports_Equipments_Duration
SECOND MOST IMPORTANT FEATURE
SportsRelated
```