```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import pydotplus
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
```

```
data = pd.read_csv("/content/online_shoppers_intention.csv")
```

```
data.head(3)
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates
0	0	0.0	0	0.0	1	0.0	0.2
1	0	0.0	0	0.0	2	64.0	0.0
2	0	0.0	0	0.0	1	0.0	0.2

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.shape
```

```
(12330, 18)
```

```
data.isna().sum()
```

	0
Administrative	0
Administrative_Duration	0
Informational	0
Informational_Duration	0
ProductRelated	0
ProductRelated_Duration	0
BounceRates	0
ExitRates	0
PageValues	0
SpecialDay	0
Month	0
OperatingSystems	0
Browser	0
Region	0
TrafficType	0
VisitorType	0
Weekend	0
Revenue	0

dtype: int64

```
# Convert 'Revenue' to numerical (0 = No Purchase, 1 = Purchase)
data['Revenue'] = data['Revenue'].astype('category').cat.codes
```

```
# Convert categorical features to numeric using Label Encoding
label_encoders = {}
categorical_columns = ['Month', 'VisitorType', 'Weekend']
```

```
for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le


# Scale numeric features
numeric_features = ['Administrative', 'Administrative_Duration', 'Informational',
                    'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
                    'BounceRates', 'ExitRates', 'PageValues']
scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features])



# Split data into training (70%) and testing (30%) sets
np.random.seed(123)
trainData, testData = train_test_split(data, test_size=0.3, stratify=data['Revenue'])



# Define features and target
features = ['PageValues', 'BounceRates', 'ExitRates', 'ProductRelated', 'ProductRelated_Duration', 'Administrative', 'Month']
target = 'Revenue'


# Train a more balanced Decision Tree
model = DecisionTreeClassifier(
    random_state=123,
    max_depth=4,  # Increase depth slightly
    min_samples_split=100,  # Allow more splits
    min_samples_leaf=50,  # Reduce minimum leaf size
    ccp_alpha=0.005,  # Less aggressive pruning
    max_features=None,  # Consider all features
    class_weight={0.0: 1, 1.0: 2},  # Balance underrepresented class
    criterion='entropy'
)
model.fit(trainData[features], trainData[target])
```

```
DecisionTreeClassifier

DecisionTreeClassifier(ccp_alpha=0.005, class_weight={0.0: 1, 1.0: 2},
                       criterion='entropy', max_depth=4, min_samples_leaf=50,
                       min_samples_split=100, random_state=123)
```

```
# Visualize the improved tree
dot_data = export_graphviz(model, feature_names=features, class_names=['No Purchase', 'Purchase'], filled=True)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_png("decision_tree_improved.png")
```

```
True
```

```
# Show the tree image
img = Image.open("decision_tree_improved.png")
img.show()


# Make predictions
predictions = model.predict(testData[features])


conf_matrix = pd.DataFrame(confusion_matrix(testData[target], predictions),
                           index=['No Purchase', 'Purchase'], columns=['Predicted No Purchase', 'Predicted Purchase'])


print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
             Predicted No Purchase  Predicted Purchase
No Purchase                   2758                 369
Purchase                        95                 477
```

```
# Print classification report
print("\nClassification Report:")
print(classification_report(testData[target], predictions))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.88      0.92      3127
           1       0.56      0.83      0.67       572

    accuracy                           0.87      3699
   macro avg       0.77      0.86      0.80      3699
weighted avg       0.90      0.87      0.88      3699
```

```
# Display feature importance
feature_importance = pd.DataFrame({'Feature': features, 'Importance': model.feature_importances_})
feature_importance = feature_importance.sort_values(by='Importance', ascending=False)
print("\nFeature Importance:")
print(feature_importance)
```

```
Feature Importance:
                   Feature  Importance
0               PageValues    0.788243
6                    Month    0.138865
1              BounceRates    0.039360
3           ProductRelated    0.020022
5           Administrative    0.013510
2                ExitRates    0.000000
4  ProductRelated_Duration    0.000000
```