Banking Marketing using decision tree classifier

Step 1: Data Preparation

In [17]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('bank.csv')
```

In [18]:

```
data
```

Out[18]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	20	apr	257	1	-1	0	unknown	no
11158	39	services	married	secondary	no	733	no	no	unknown	16	jun	83	4	-1	0	unknown	no
11159	32	technician	single	secondary	no	29	no	no	cellular	19	aug	156	2	-1	0	unknown	no
11160	43	technician	married	secondary	no	0	no	yes	cellular	8	may	9	2	172	5	failure	no
11161	34	technician	married	secondary	no	0	no	no	cellular	9	jul	628	1	-1	0	unknown	no

11162 rows × 17 columns

Step 2: Train a Decision Tree Classifier

Import the necessary libraries and create a decision tree classifier. Train the classifier on the training data.

In [19]:

```
# Encode categorical variables
encoder = LabelEncoder()
data['job'] = encoder.fit_transform(data['job'])
data['marital'] = encoder.fit_transform(data['marital'])
data['education'] = encoder.fit_transform(data['education'])
data['default'] = encoder.fit_transform(data['default'])
data['housing'] = encoder.fit_transform(data['housing'])
data['loan'] = encoder.fit_transform(data['loan'])
data['contact'] = encoder.fit_transform(data['contact'])
```

```
data['month'] = encoder.fit_transform(data['month'])
data['day'] = encoder.fit_transform(data['day'])
data['poutcome'] = encoder.fit_transform(data['poutcome'])
data['deposit'] = encoder.fit_transform(data['deposit'])

# Split the dataset into training and testing sets
X = data.drop('deposit', axis=1)
y = data['deposit']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Evaluate the Model

In [20]:

```
## Train a Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier
clf.fit(X_train, y_train)
```

Out[20]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [21]:

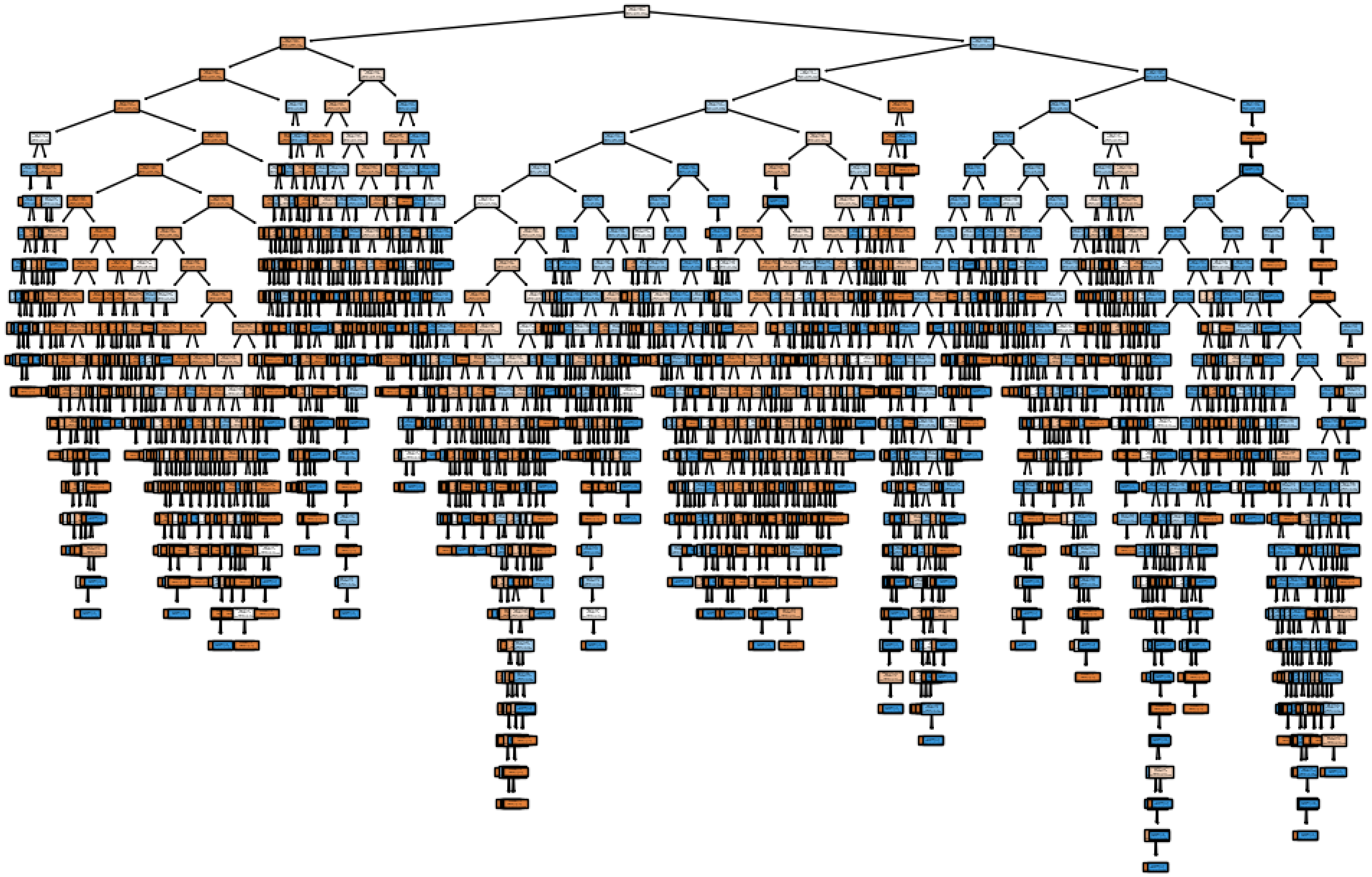
```
## Evaluate the Model
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print(f"Confusion Matrix:\n{conf_matrix}")
print(f"Classification Report:\n{classification_rep}")
```

```
Accuracy: 0.7630989699955217
Confusion Matrix:
[[915 251]
 [278 789]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.78      0.78      1166
           1       0.76      0.74      0.75      1067

    accuracy                           0.76      2233
   macro avg       0.76      0.76      0.76      2233
weighted avg       0.76      0.76      0.76      2233
```

Step 4: Visualize the Decision Tree(Optional)

In [22]:

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['Not Purchased', 'Purchased'])
plt.show()
```



Hyperparameter Tuning

In [23]:

```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters and their possible values
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Create GridSearchCV object with cross-validation
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Get the best model
best_model = grid_search.best_estimator_

# Evaluate the best model on the test data
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Best Hyperparameters:", best_params)
print(f"Accuracy with Best Model: {accuracy}")
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}
Accuracy with Best Model: 0.80653828929691
```

```
In [ ]:
```