**Report: Predicting Customer Purchase Using Decision Tree Classifier**

**Objective**

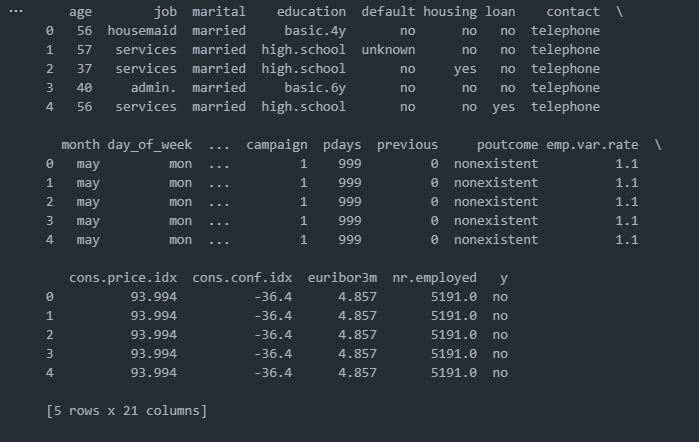
The goal of this project is to build a **Decision Tree Classifier** to predict whether a customer will purchase a product or service based on their demographic and behavioral data.

**Dataset**

The dataset used for this project is the **Bank Marketing dataset**, available from the [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/Bank+Marketing). The dataset contains information about direct marketing campaigns of a Portuguese bank, and it includes features such as age, job, marital status, education, and the contact information for customers.

**Key Variables**

* **age**: Age of the client.
* **job**: Type of job (e.g., admin, technician, etc.).
* **marital**: Marital status (e.g., married, single, divorced).
* **education**: Level of education (e.g., secondary, tertiary).
* **housing**: Whether the customer has a housing loan.
* **loan**: Whether the customer has a personal loan.
* **contact**: Communication type used for contact (e.g., cellular, telephone).
* **y**: Whether the client subscribed to a term deposit (target variable).



**Step 1: Install Required Libraries**

The first step is to install all the necessary libraries for data manipulation, machine learning, and model evaluation.

pip install pandas scikit-learn matplotlib seaborn

Libraries used:

* **pandas**: For data manipulation and analysis.
* **scikit-learn**: For building and training the decision tree classifier.
* **matplotlib** and **seaborn**: For visualizing the decision tree and evaluating model performance.

**Step 2: Download and Extract the Dataset**

Since the dataset is stored in a ZIP archive with multiple files, the first step is to download and extract the ZIP file.

import zipfile

import os

import requests

# Step 1: Download the ZIP file

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip"

zip\_file\_path = "bank\_marketing.zip"

# Download the file

response = requests.get(url)

with open(zip\_file\_path, 'wb') as file:

file.write(response.content)

# Step 2: Extract the ZIP file

with zipfile.ZipFile(zip\_file\_path, 'r') as zip\_ref:

zip\_ref.extractall("bank\_marketing")

Here, we use the requests library to download the dataset, then the zipfile library to extract its contents.

**Step 3: Load the Dataset**

After extracting the ZIP file, we load the appropriate CSV file into a pandas DataFrame for further analysis.

import pandas as pd

# Step 3: Locate the desired CSV file

csv\_file\_path = os.path.join("bank\_marketing", "bank-additional", "bank-additional-full.csv")

# Step 4: Load the CSV file

df = pd.read\_csv(csv\_file\_path, delimiter=';')

# Step 5: Display the first few rows

print(df.head())

Here, we read the CSV file (bank-additional-full.csv) into a pandas DataFrame and display the first few rows for inspection.

**Step 4: Data Preprocessing**

Data preprocessing is an important step to clean and prepare the data for training the model. This includes:

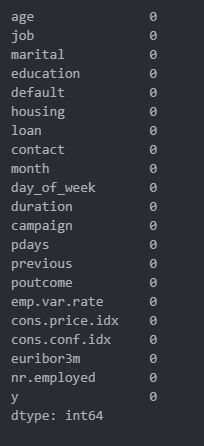
1. Checking for missing values
2. Encoding categorical features
3. Splitting the data into features (X) and the target variable (y)
4. Splitting the data into training and test sets

**Handling Missing Values**

# Check for missing values

print(df.isnull().sum())

We check if there are any missing values in the dataset. In this case, the dataset doesn't contain missing values.



**Encoding Categorical Features**

The dataset contains categorical features (e.g., job, marital, etc.), which need to be encoded into numerical values for machine learning models.

# Convert categorical columns to numerical using one-hot encoding (or label encoding)

df\_encoded = pd.get\_dummies(df, drop\_first=True)

We use **one-hot encoding** to convert categorical columns into numerical format, dropping the first column to avoid multicollinearity.

**Splitting Data into Features and Target**

We separate the dataset into features (X) and the target variable (y), where y is the target variable indicating whether the customer will purchase a product or service.

# Split the data into features (X) and target (y)

X = df\_encoded.drop('y\_yes', axis=1) # 'y\_yes' is the target variable for purchase (binary: yes/no)

y = df\_encoded['y\_yes'] # Target variable: whether the customer will purchase

Here, y\_yes is the target variable indicating whether the customer subscribed to a term deposit.

**Splitting Data into Training and Test Sets**

We split the dataset into training and test sets to evaluate model performance.

from sklearn.model\_selection import train\_test\_split

# Split the data into training and test sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Step 5: Building the Decision Tree Classifier**

We use the **DecisionTreeClassifier** from scikit-learn to train the model on the training data.

from sklearn.tree import DecisionTreeClassifier

# Initialize and train the Decision Tree model

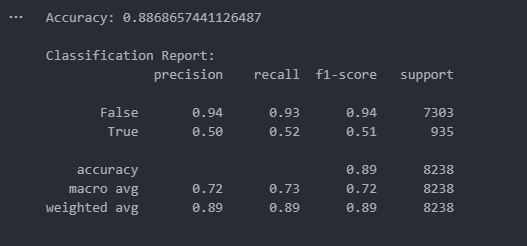
dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = dt\_model.predict(X\_test)

Here, we create an instance of the DecisionTreeClassifier, train it on the training set (X\_train and y\_train), and then make predictions on the test set (X\_test).

****

**Step 6: Evaluate the Model**

We evaluate the model by checking its accuracy and generating a classification report.

from sklearn.metrics import classification\_report, accuracy\_score

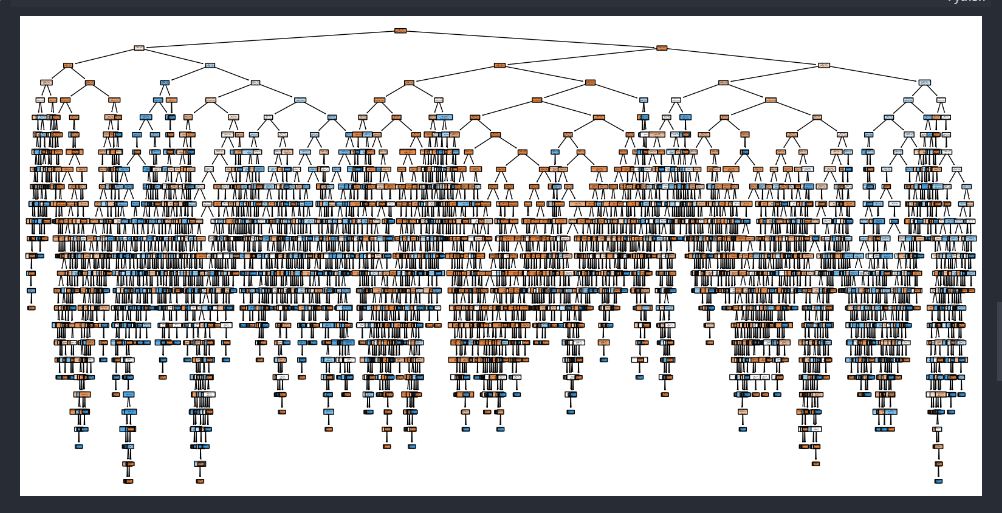
# Evaluate the model

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred)}")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

The classification report includes precision, recall, F1-score, and support for each class (Yes/No). Accuracy gives an overall measure of the model’s performance.



**Step 7: Visualize the Decision Tree**

Visualizing the decision tree helps to understand how the model is making predictions.

import matplotlib.pyplot as plt

from sklearn.tree import plot\_tree

# Plot the trained Decision Tree

plt.figure(figsize=(20,10))

plot\_tree(dt\_model, filled=True, feature\_names=X.columns, class\_names=['No', 'Yes'], rounded=True)

plt.show()

This step generates a graphical representation of the decision tree, showing how it splits the data based on feature values.

**Step 8: Fine-Tune the Model (Optional)**

To improve the model, we can fine-tune the hyperparameters of the decision tree, such as max\_depth and min\_samples\_leaf.

# Fine-tune the Decision Tree with max\_depth and min\_samples\_leaf

dt\_model\_tuned = DecisionTreeClassifier(random\_state=42, max\_depth=5, min\_samples\_leaf=10)

dt\_model\_tuned.fit(X\_train, y\_train)

# Make predictions with the tuned model

y\_pred\_tuned = dt\_model\_tuned.predict(X\_test)

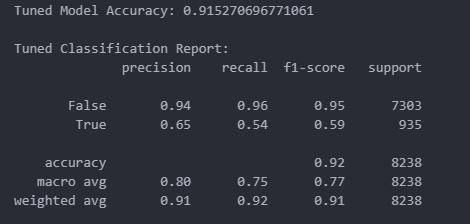
# Evaluate the tuned model

print(f"Tuned Model Accuracy: {accuracy\_score(y\_test, y\_pred\_tuned)}")

print("\nTuned Classification Report:")

print(classification\_report(y\_test, y\_pred\_tuned))

Fine-tuning helps improve the model's performance and prevents overfitting.



**Conclusion**

In this project, we built a **Decision Tree Classifier** to predict customer purchase behavior using the **Bank Marketing dataset**. The key steps included:

1. **Downloading and extracting the dataset**
2. **Preprocessing the data** (encoding categorical variables and splitting the dataset)
3. **Building and training the Decision Tree Classifier**
4. **Evaluating the model** using accuracy and classification reports
5. **Visualizing the decision tree** to understand the model's decision-making process
6. **Fine-tuning** the model for improved performance.

This model can be deployed to predict whether a customer will subscribe to a product based on their demographic and behavioral features.