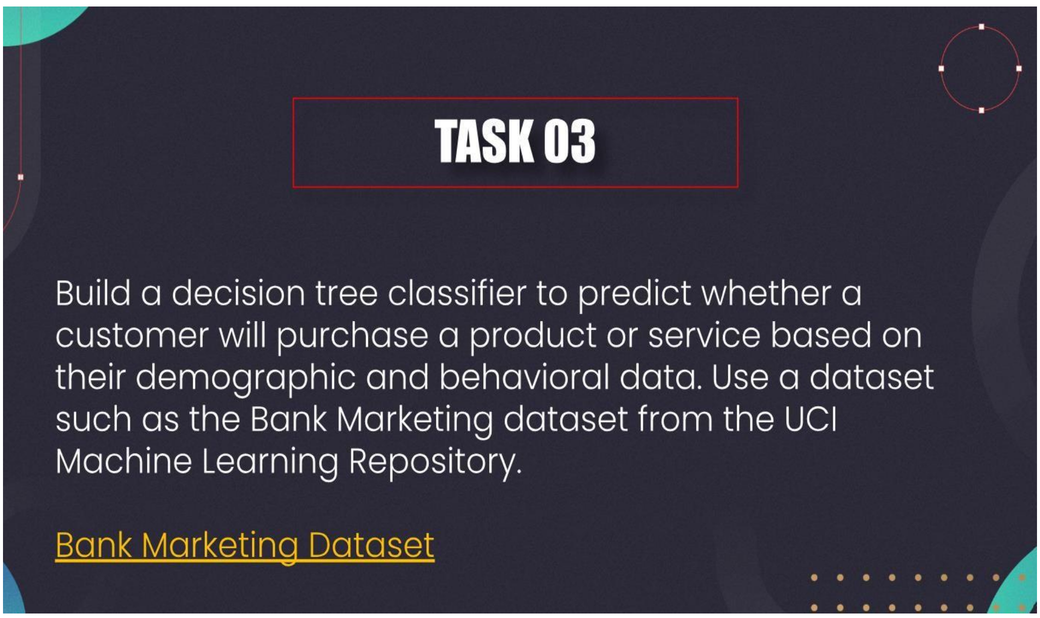
Project Name:Customer Purchase Prediction Using Decision Tree Classifier

By Shruti Thorat

Project Introduction

- The ability to predict customer purchase behavior is a critical aspect of business success, enabling organizations to tailor marketing strategies, optimize resources, and enhance customer experiences.
- This project aims to build a Decision Tree Classifier to predict whether a customer will purchase a product or service based on their demographic and behavioral data.
- The dataset used for this project is the Bank Marketing dataset, sourced from the UCI Machine Learning Repository. It includes diverse features such as age, job type, marital status, education level, and past interactions, providing a comprehensive view of customer profiles.
- 

Project Summary

- This project involves analyzing customer data, identifying patterns, and applying machine learning techniques to build a predictive model.
- The dataset is preprocessed to handle missing values, remove outliers, and encode categorical variables.
- Exploratory data analysis is conducted to understand feature distributions and correlations.
- Using a Decision Tree Classifier, the model predicts whether a customer will purchase a product or service.
- The model's performance is evaluated using accuracy scores, confusion matrices, and visualization of decision-making processes.
- The project also explores hyperparameter tuning to optimize the model's performance.

Business Objective

The primary objective is to assist businesses in:

- Predicting customer purchases based on their demographic and behavioral data.
- Enhancing marketing efficiency by identifying high-potential customers for targeted campaigns.
- Optimizing resources by focusing on customers most likely to convert, reducing marketing costs.
- Improving customer experience by personalizing offers and services based on predictive insights.

Importing Libraries

```python
#importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

Loading the Dataset

```python
#Loading the Dataset
df= pd.read_csv('/content/bank-additional-full.csv',delimiter=';')
df.rename(columns={'y':'deposit'},inplace=True)
df
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.em
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	0	nonexistent	-1.1	94.767	-50.8	1.028	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	1	failure	-1.1	94.767	-50.8	1.028	

41188 rows × 21 columns

Understanding the Data

```
df.head(10)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employ
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
5	45	services	married	basic.9y	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
6	59	admin.	married	professional.course	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
7	41	blue-collar	married	unknown	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
8	24	technician	single	professional.course	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519
9	25	services	single	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	519

10 rows × 21 columns

```
[6] df.tail()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.empl
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	49
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	49
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	0	nonexistent	-1.1	94.767	-50.8	1.028	49
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	49
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	1	failure	-1.1	94.767	-50.8	1.028	49

5 rows × 21 columns

```
[7] df.sample(5)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.emp
34800	20	blue-collar	single	high.school	no	yes	no	cellular	may	thu	...	3	999	0	nonexistent	-1.8	92.893	-46.2	1.266	
21745	31	technician	single	university.degree	no	no	no	cellular	aug	tue	...	5	999	0	nonexistent	1.4	93.444	-36.1	4.963	
28220	72	retired	married	basic.4y	no	unknown	unknown	cellular	apr	tue	...	1	999	0	nonexistent	-1.8	93.075	-47.1	1.453	
24211	55	blue-collar	married	basic.9y	no	no	no	cellular	nov	mon	...	1	999	0	nonexistent	-0.1	93.200	-42.0	4.191	
27483	32	admin.	divorced	university.degree	no	yes	no	cellular	nov	fri	...	3	999	0	nonexistent	-0.1	93.200	-42.0	4.021	

5 rows × 21 columns

```
print("\nInfo of the dataframe:")
df.info()
```

```
Info of the dataframe:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  deposit         41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

```
print("Shape of the dataframe:", df.shape)
```

```
Shape of the dataframe: (41188, 21)
```

[10]
```
print("\nColumns of the dataframe:")
print(df.columns)
```

```
Columns of the dataframe:
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'deposit'],
      dtype='object')
```

```
print("\nDatatypes of the columns:")
df.dtypes
```

Datatypes of the columns:

	0
age	int64
job	object
marital	object
education	object
default	object
housing	object
loan	object
contact	object
month	object
day_of_week	object
duration	int64
campaign	int64
pdays	int64
previous	int64
poutcome	object
emp.var.rate	float64
cons.price.idx	float64
cons.conf.idx	float64
euribor3m	float64

[11]

nr.employed	float64
deposit	object

dtype: object

```
print("\nValue counts of datatypes:")
df.dtypes.value_counts()
```

Value counts of datatypes:

	count
object	11
int64	5
float64	5

dtype: int64

[13]
```
print("\nNumber of duplicated rows:", df.duplicated().sum())
```

```
Number of duplicated rows: 12
```

[14]
```
# Check for missing values
print("\nNumber of missing values in each column:")
print(df.isnull().sum())
```

```
Number of missing values in each column:
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
deposit           0
dtype: int64
```

```python
[15] # Identifying categorical and numerical columns
     categorical_cols = df.select_dtypes(include=('object')).columns
     numerical_cols = df.select_dtypes(exclude=('object')).columns

     print("\nCategorical columns:")
     print(categorical_cols)

     print("\nNumerical columns:")
     print(numerical_cols)
```

```
Categorical columns:
Index(['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact',
       'month', 'day_of_week', 'poutcome', 'deposit'],
      dtype='object')

Numerical columns:
Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed'],
      dtype='object')
```

```python
[16] print("\nDescription of the dataframe:")
     df.describe()
```

```
Description of the dataframe:
```

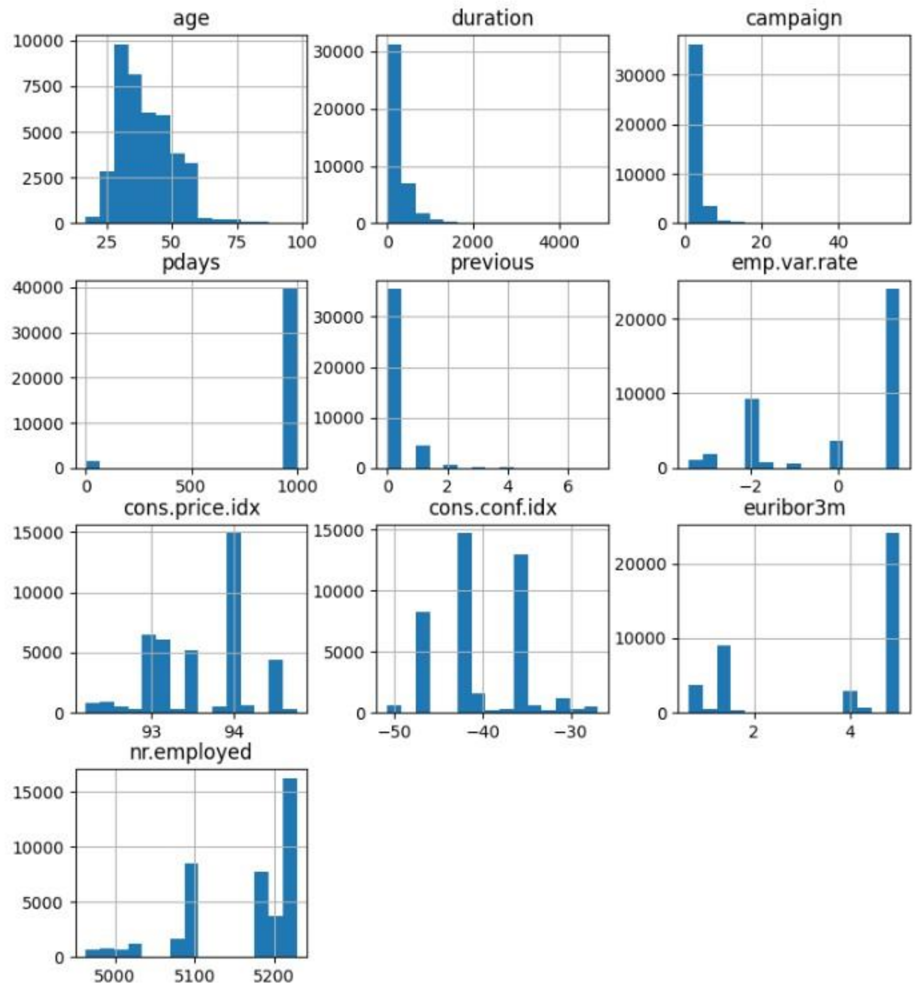
	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035911
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251528
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

```python
[17] df.describe(include='object')
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	deposit
count	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188
unique	12	4	8	3	3	3	2	10	5	3	2
top	admin.	married	university.degree	no	yes	no	cellular	may	thu	nonexistent	no
freq	10422	24928	12168	32588	21576	33950	26144	13769	8623	35563	36548

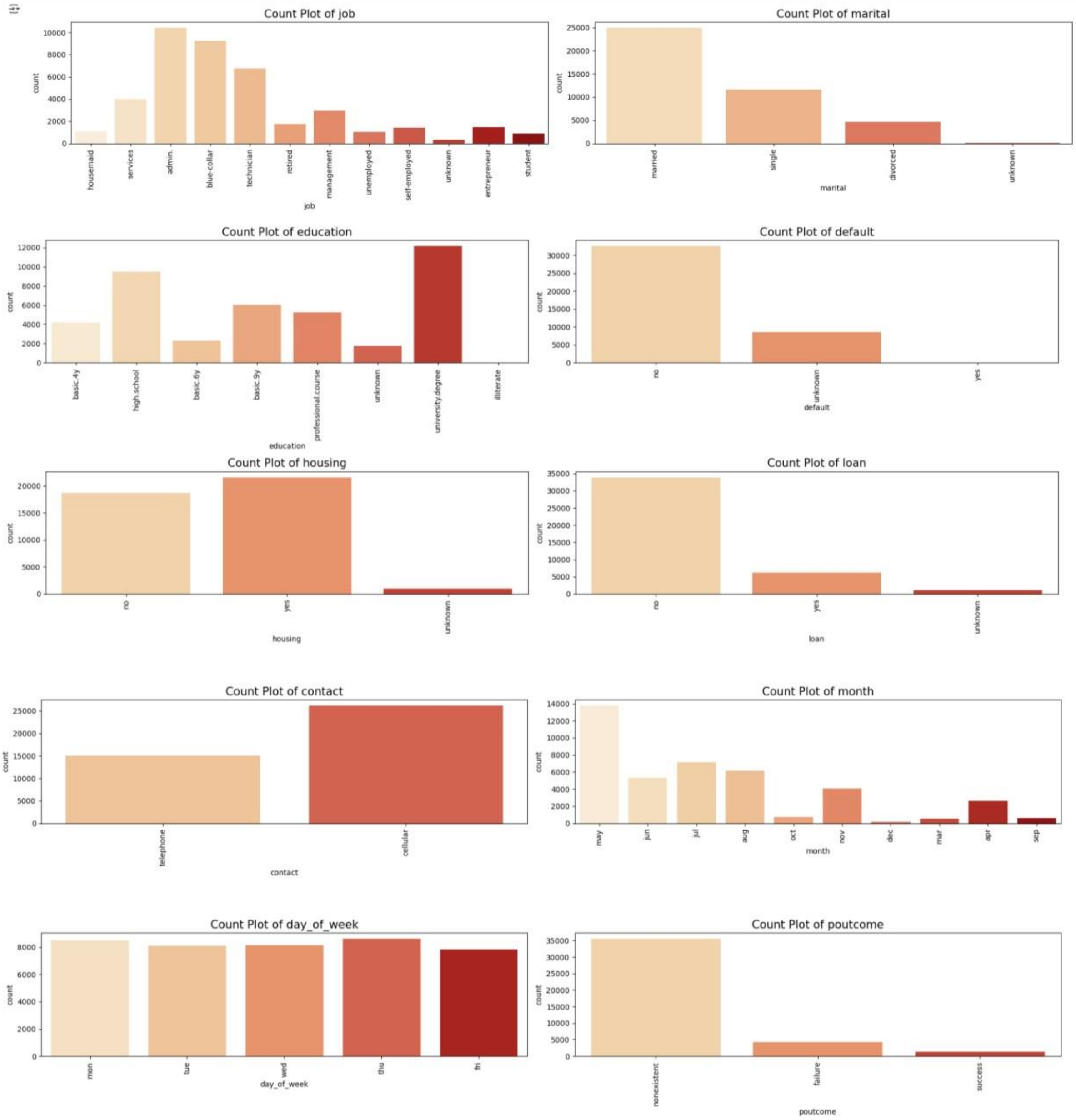
Histogram Plot for Numerical Columns

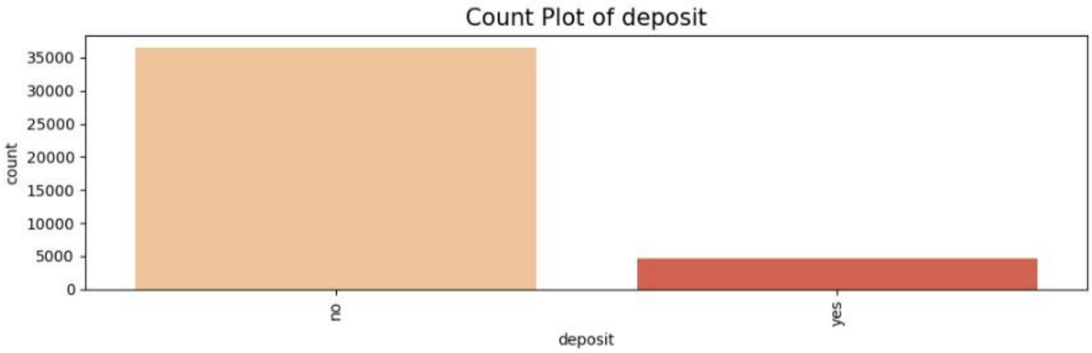
[18]
```
df[numerical_cols].hist(bins=15, figsize=(9, 10))
plt.show()
```



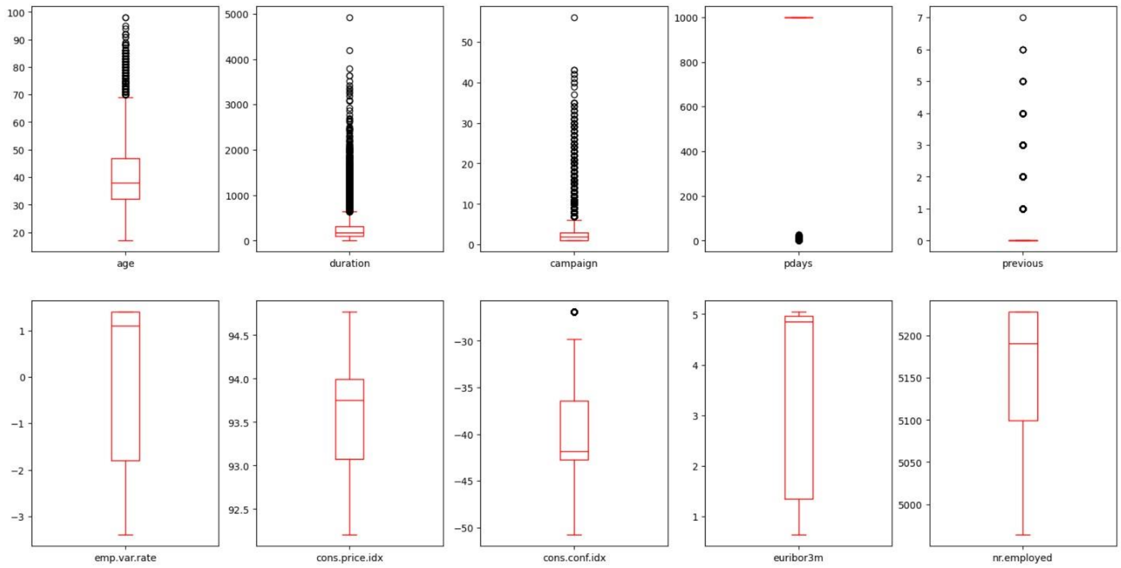
Countplot for Categorical Columns

```python
# Calculate number of plots, rows, and columns
num_plots = len(categorical_cols)
num_rows = (num_plots+1)//2
num_cols = 2

# create new figure
plt.figure(figsize=(20,25))

# Create count plots for each categorical column
for i, col in enumerate(categorical_cols,1):
    plt.subplot(num_rows,num_cols,i)
    sns.countplot(x= col,data=df, palette = 'OrRd')
    plt.title(f"Count Plot of {col}", fontsize=15)
    plt.xlabel(col)
    plt.ylabel('count')
    plt.xticks(rotation=90)

#Adjust layout to prevent overlap of subplots
plt.tight_layout()
plt.show()
```






```
[ ] df.plot(kind='box',subplots= True,layout=(2,5),figsize=(20,10),color='r')
    plt.show()
```




```
for col in numerical_cols:
    Q1 = df[col].quantile(0.25)
    Q2 = df[col].quantile(0.5)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    print(f"{col}: Q1={Q1}, Q2={Q2}, Q3={Q3}, IQR={IQR}, Lower Bound={lower_bound}, Upper Bound={upper_bound}")
```

```
age: Q1=32.0, Q2=38.0, Q3=47.0, IQR=15.0, Lower Bound=9.5, Upper Bound=69.5
duration: Q1=102.0, Q2=180.0, Q3=319.0, IQR=217.0, Lower Bound=-223.5, Upper Bound=644.5
campaign: Q1=1.0, Q2=2.0, Q3=3.0, IQR=2.0, Lower Bound=-2.0, Upper Bound=6.0
pdays: Q1=999.0, Q2=999.0, Q3=999.0, IQR=0.0, Lower Bound=999.0, Upper Bound=999.0
previous: Q1=0.0, Q2=0.0, Q3=0.0, IQR=0.0, Lower Bound=0.0, Upper Bound=0.0
emp.var.rate: Q1=-1.8, Q2=1.1, Q3=1.4, IQR=3.2, Lower Bound=-6.6000000000000005, Upper Bound=6.200000000000001
cons.price.idx: Q1=93.075, Q2=93.749, Q3=93.994, IQR=0.9189999999999969, Lower Bound=91.69650000000001, Upper Bound=95.3725
cons.conf.idx: Q1=-42.7, Q2=-41.8, Q3=-36.4, IQR=6.300000000000004, Lower Bound=-52.150000000000006, Upper Bound=-26.949999999999992
euribor3m: Q1=1.344, Q2=4.857, Q3=4.961, IQR=3.617, Lower Bound=-4.081499999999999, Upper Bound=10.3865
nr.employed: Q1=5099.1, Q2=5191.0, Q3=5228.1, IQR=129.0, Lower Bound=4905.6, Upper Bound=5421.6
```

```
[ ] #skewness and Data Distribution
    skewness = df[numerical_cols].skew()
    print("\nSkewness of numerical columns:\n", skewness)
```

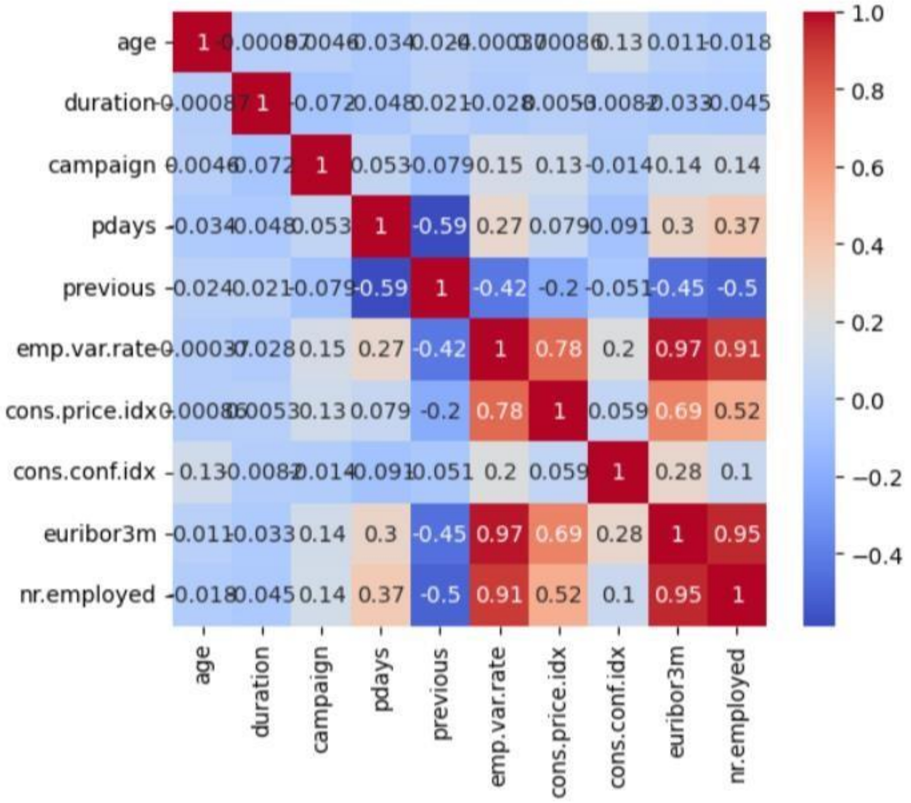
```
Skewness of numerical columns:
 age               0.784697
duration          3.263141
campaign          4.762507
pdays            -4.922190
previous          3.832042
emp.var.rate     -0.724096
cons.price.idx   -0.230888
cons.conf.idx     0.303180
euribor3m        -0.709188
nr.employed      -1.044262
dtype: float64
```

Correlation Matrix and Heatmap

```
# Exclude non-numeric columns for correlation matrix
corr_matrix = df[numerical_cols].corr()
print(corr_matrix)
plt.figure(figsize=(6, 5))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

```
                     age  duration  campaign     pdays  previous  \
age             1.000000 -0.000866  0.004594 -0.034369  0.024365
duration       -0.000866  1.000000 -0.071699 -0.047577  0.020640
campaign        0.004594 -0.071699  1.000000  0.052584 -0.079141
pdays          -0.034369 -0.047577  0.052584  1.000000 -0.587514
previous        0.024365  0.020640 -0.079141 -0.587514  1.000000
emp.var.rate   -0.000371 -0.027968  0.150754  0.271004 -0.420489
cons.price.idx  0.000857  0.005312  0.127836  0.078889 -0.203130
cons.conf.idx   0.129372 -0.008173 -0.013733 -0.091342 -0.050936
euribor3m       0.010767 -0.032897  0.135133  0.296899 -0.454494
nr.employed    -0.017725 -0.044703  0.144095  0.372605 -0.501333

                emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
age                -0.000371        0.000857       0.129372   0.010767
duration           -0.027968        0.005312      -0.008173  -0.032897
campaign            0.150754        0.127836      -0.013733   0.135133
pdays               0.271004        0.078889      -0.091342   0.296899
previous           -0.420489       -0.203130      -0.050936  -0.454494
emp.var.rate        1.000000        0.775334       0.196041   0.972245
cons.price.idx      0.775334        1.000000       0.058986   0.688230
cons.conf.idx       0.196041        0.058986       1.000000   0.277686
euribor3m           0.972245        0.688230       0.277686   1.000000
nr.employed         0.906970        0.522034       0.100513   0.945154

                nr.employed
age               -0.017725
duration          -0.044703
campaign           0.144095
pdays              0.372605
previous          -0.501333
emp.var.rate       0.906970
cons.price.idx     0.522034
cons.conf.idx      0.100513
euribor3m          0.945154
nr.employed        1.000000
```

```
euribor3m          0.945154
nr.employed        1.000000
```



High Correlation Columns

```
# Filter highly correlated columns
high_corr_cols = [col for col in corr_matrix.columns if any(corr_matrix[col] > 0.75)]
print("\nHighly correlated columns:", high_corr_cols)

# Create a copy and drop high correlation columns
data_copy = df.drop(high_corr_cols, axis=1)
print("\nShape after dropping high correlation columns:", data_copy.shape)
```

```
Highly correlated columns: ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']

Shape after dropping high correlation columns: (41188, 11)
```

Encode Categorical Data

```python
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df_new= data_copy.apply(encoder.fit_transform)
df_new
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	deposit
0	3	1	0	0	0	0	1	6	1	1	0
1	7	1	3	1	0	0	1	6	1	1	0
2	7	1	3	0	2	0	1	6	1	1	0
3	0	1	1	0	0	0	1	6	1	1	0
4	7	1	3	0	0	2	1	6	1	1	0
...	...	...	...	...	...	...	...	...	...	...	...
41183	5	1	5	0	2	0	0	7	0	1	1
41184	1	1	5	0	0	0	0	7	0	1	0
41185	5	1	6	0	2	0	0	7	0	1	0
41186	9	1	5	0	0	0	0	7	0	1	1
41187	5	1	5	0	2	0	0	7	0	0	0

41188 rows × 11 columns

```python
# Check the values in the target column
print(df_new['deposit'].value_counts())
```

```
deposit
0    36548
1     4640
Name: count, dtype: int64
```

Drop Independent Variable and Check Shape and Type

```
[ ] # Drop target variable 'deposit' from features
    X = df_new.drop('deposit', axis=1)
    y = df_new['deposit']

    print("\nShape of X:", X.shape)
    print("\nShape of y:", y.shape)
    print("\nType of X:", type(X))
    print("\nType of y:", type(y))
```

```
Shape of X: (41188, 10)

Shape of y: (41188,)

Type of X: <class 'pandas.core.frame.DataFrame'>

Type of y: <class 'pandas.core.series.Series'>
```

```
[ ] from sklearn.model_selection import train_test_split


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    print("Training Shape:", X_train.shape)
    print("Testing Shape:", X_test.shape)
```

```
Training Shape: (28831, 10)
Testing Shape: (12357, 10)
```

Build Decision Tree Classifier

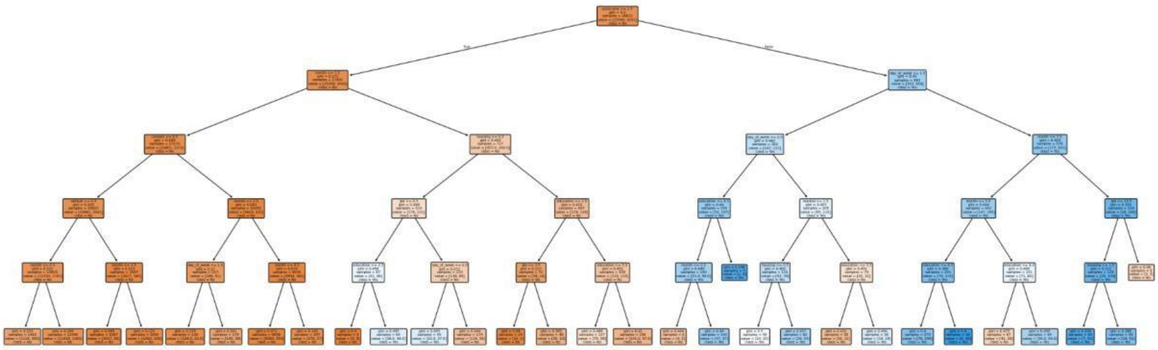
Model Training and Evaluation

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

dt = DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_split=10)
dt.fit(X_train, y_train)

# Predictions
y_pred_train = dt.predict(X_train)
y_pred_test = dt.predict(X_test)

# Scores
print("Training Accuracy:", accuracy_score(y_train, y_pred_train))
print("Testing Accuracy:", accuracy_score(y_test, y_pred_test))

# Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_test))
print("Classification Report:\n", classification_report(y_test, y_pred_test))
```

```
Training Accuracy: 0.899066976518331
Testing Accuracy: 0.8955248037549567
Confusion Matrix:
 [[10813   155]
 [ 1136   253]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94     10968
           1       0.62      0.18      0.28      1389

    accuracy                           0.90     12357
   macro avg       0.76      0.58      0.61     12357
weighted avg       0.87      0.90      0.87     12357
```

Visualize the Decision Tree

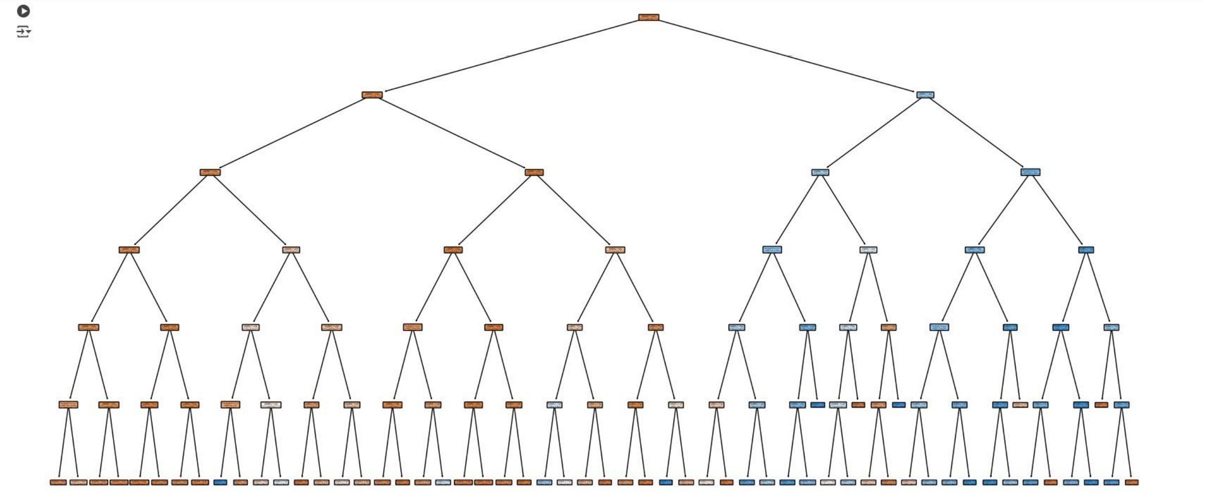
```python
from sklearn.tree import plot_tree

fn = X_train.columns
cn = ['No', 'Yes']


plt.figure(figsize=(30, 10))
plot_tree(dt, feature_names=fn, class_names=cn, filled=True, rounded=True)
plt.show()
```



Decision Tree with Specific Parameters

```python
# Initialize Decision Tree with specific parameters
dt1 = DecisionTreeClassifier(criterion='entropy', max_depth=6, min_samples_split=5)
dt1.fit(X_train, y_train)

# Predict on the test set
y_pred1 = dt1.predict(X_test)

# Updated Scores
print("Training Accuracy:", dt1.score(X_train, y_train))
print("Testing Accuracy:", dt1.score(X_test, y_test))


# Calculate accuracy
accuracy1 = accuracy_score(y_test, y_pred1)
print(f"Accuracy with specific parameters: {accuracy1}")

# Confusion Matrix
conf_matrix1 = confusion_matrix(y_test, y_pred1)
print(f"Confusion Matrix with specific parameters:\n {conf_matrix1}")


# Updated Tree Plot
plt.figure(figsize=(20, 10))
plot_tree(dt1, feature_names=fn, class_names=cn, filled=True, rounded=True)
plt.show()
```

```
Training Accuracy: 0.8989976067427422
Testing Accuracy: 0.896495913247552
Accuracy with specific parameters: 0.896495913247552
Confusion Matrix with specific parameters:
 [[10831   137]
 [ 1142   247]]
```



Conclusion

- The Decision Tree Classifier successfully predicts customer purchase behavior with a significant accuracy rate.
- The model provides interpretable insights into key factors influencing customer decisions, such as contact duration, marital status, and education level.
- Businesses can leverage these insights to develop focused marketing strategies and improve decisionmaking processes.
- The project highlights the power of machine learning in deriving actionable insights from data and emphasizes the importance of continuous evaluation and tuning for optimal performance