

# DATA ANALYTICS & ML FINAL PROJECT : Bank Marketing

## 1. Loading Data :

```
In [2]: import pandas as pd
df = pd.read_csv("bank-additional-full.csv", sep=';')

df.head(3)
```

```
Out[2]:
```

	age	job	marital	education	default	housing	loan	contact	month
0	56	housemaid	married	basic.4y	no	no	no	telephone	nan
1	57	services	married	high.school	unknown	no	no	telephone	nan
2	37	services	married	high.school	no	yes	no	telephone	nan

3 rows × 21 columns

```
In [3]: print("Initial data info:")
df.info()
```

Initial data info:

<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	y	41188 non-null	object

dtypes: float64(5), int64(5), object(11)

memory usage: 6.6+ MB

## 2. Initial Cleaning & Preprocessing :

```
In [5]: # Checking for null values
print("Missing values before dropping:\n", df.isnull().sum())

# Checking for duplicates
print("\n\nDuplicate rows:", df.duplicated().sum())
print("\n")

# Checking for unknown values
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'lo
print("Unkown values : ")
for col in categorical_cols:
    unknown_count = df[col].value_counts().get('unknown', 0)
    print(f"{col:<12} | 'unknown' count: {unknown_count}")
```

Missing values before dropping:

```
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
y            0
dtype: int64
```

Duplicate rows: 12

Unkown values :

```
job          | 'unknown' count: 330
marital      | 'unknown' count: 80
education    | 'unknown' count: 1731
default      | 'unknown' count: 8597
housing      | 'unknown' count: 990
loan         | 'unknown' count: 990
contact      | 'unknown' count: 0
month        | 'unknown' count: 0
day_of_week  | 'unknown' count: 0
poutcome     | 'unknown' count: 0
```

```
In [6]: # Deleting duplicate values
df = df.drop_duplicates()

# Replacing Unknown with mode of the column
df['job'] = df['job'].replace('unknown', df['job'].mode()[0])
df['housing'] = df['housing'].replace('unknown', df['housing'].mode()[0])
df['loan'] = df['loan'].replace('unknown', df['loan'].mode()[0])
df['education'] = df['education'].replace('unknown', df['education'].mode()[0])
df['marital'] = df['marital'].replace('unknown', df['marital'].mode()[0])
df['default'] = df['default'].replace('unknown', df['marital'].mode()[0])
```

### 3. Feature Overview

Feature	Description	Range
<b>Age</b>	Age of the client	Numeric
<b>Job</b>	Type of job	"admin.", "blue-collar", "entrepreneur", "housemaid", "management", "retired", "self-employed", "services", "student", "technician", "unemployed", "unknown"
<b>Marital</b>	Marital status	"divorced", "married", "single", "unknown"
<b>Education</b>	Level of education	"basic.4y", "basic.6y", "basic.9y", "high.school", "illiterate", "professional.course", "university.degree", "unknown"
<b>Default</b>	Has credit in default?	"no", "yes", "unknown"
<b>Housing</b>	Has a housing loan?	"no", "yes", "unknown"
<b>Loan</b>	Has a personal loan?	"no", "yes", "unknown"
<b>Contact</b>	Type of contact communication	"cellular", "telephone"
<b>Month</b>	Last contact month	"jan", "feb", "mar", ..., "nov", "dec"
<b>Day_of_week</b>	Last contact day of the week	"mon", "tue", "wed", "thu", "fri"
<b>Duration</b>	Last contact duration in seconds ( <i>used only for benchmarking</i> )	Numeric
<b>Campaign</b>	Number of contacts in current campaign (includes last contact)	Numeric
<b>Pdays</b>	Days since last contact from previous campaign (999 means never contacted)	Numeric
<b>Previous</b>	Number of contacts before this campaign	Numeric
<b>Poutcome</b>	Outcome of previous marketing campaign	"failure", "nonexistent", "success"
<b>Emp.var.rate</b>	Employment variation rate (quarterly indicator)	Numeric
<b>Cons.price.idx</b>	Consumer price index (monthly indicator)	Numeric
<b>Cons.conf.idx</b>	Consumer confidence index (monthly indicator)	Numeric
<b>Euribor3m</b>	Euribor 3-month rate (daily indicator)	Numeric

Feature	Description	Range
<b>Nr.employed</b>	Number of employees (quarterly indicator)	Numeric
<b>y (Target)</b>	Has the client subscribed to a term deposit?	"yes", "no"

## 4. Exploratory Data Analysis :

### 1- Numerical Data :

```
In [9]: import matplotlib.pyplot as plt
import seaborn as sns

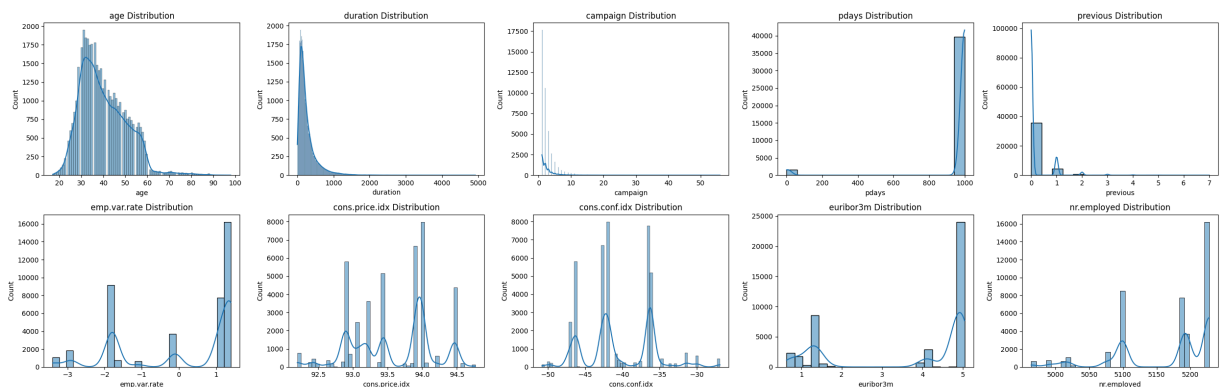
# Select numerical features
numerical_df = df.select_dtypes(include='number').columns

n_cols = 5
n_rows = -(-len(numerical_df) // n_cols) # Ceiling division

fig, axes = plt.subplots(n_rows, n_cols, figsize=(5*n_cols, 4*n_rows))
axes = axes.flatten()

# Plot each numerical feature
for i, col in enumerate(numerical_df):
    sns.histplot(data=df, x=col, kde=True, ax=axes[i])
    axes[i].set_title(f'{col} Distribution')

plt.tight_layout()
plt.show()
```



### 2- Categorical Data :

```
In [11]: # Select categorical features
categorical_df = df.select_dtypes(include='object').columns

# Exclude the target if needed
categorical_df = [col for col in categorical_cols if col != 'y']

# Grid size
```

```

n_cols = 5
n_rows = -(-len(categorical_df) // n_cols)

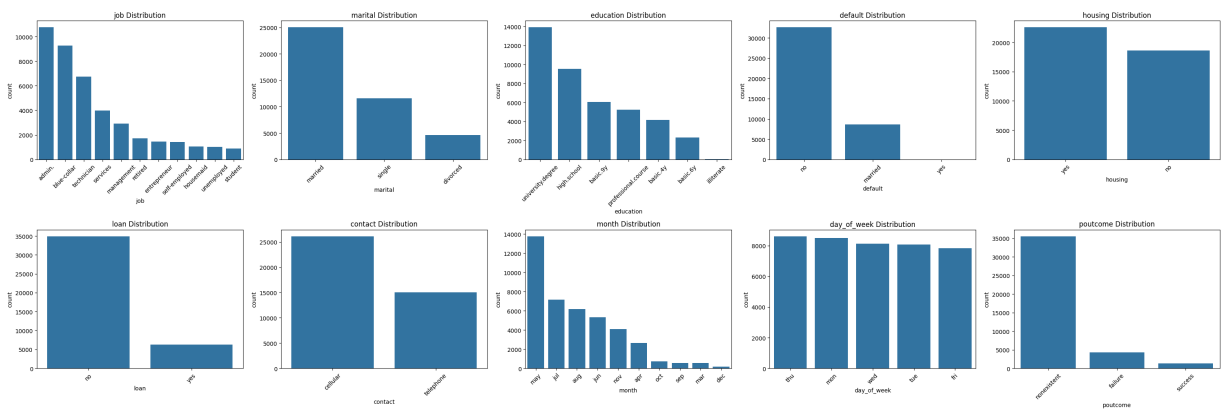
fig, axes = plt.subplots(n_rows, n_cols, figsize=(6*n_cols, 5*n_rows))
axes = axes.flatten()

for i, col in enumerate(categorical_df):
    sns.countplot(data=df, x=col, ax=axes[i], order=df[col].value_counts().index)
    axes[i].set_title(f'{col} Distribution')
    axes[i].tick_params(axis='x', rotation=45)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

```



## 5. Feature Engineering & Transformation:

```

In [13]: import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import OneHotEncoder

# 1- The target variable
df['y'] = df['y'].map({'yes': 1, 'no': 0})

# 2- Numerical values
standard_scaler_cols = ['age', 'emp.var.rate', 'cons.price.idx', 'cons.conf.
minmax_scaler_cols = ['duration', 'campaign', 'pdays', 'previous']

scaler_std = StandardScaler()
scaler_minmax = MinMaxScaler()

df[standard_scaler_cols] = scaler_std.fit_transform(df[standard_scaler_cols])
df[minmax_scaler_cols] = scaler_minmax.fit_transform(df[minmax_scaler_cols])

# 3- Categorical values
# One hot encoding for the categories :
onehot_cols = ['job', 'marital', 'default', 'housing', 'loan', 'contact', 'c
df = pd.get_dummies(df, columns=onehot_cols, drop_first=True)

```

```

# Cyclically encode months using sine and cosine transformations to capture
month_map = {'jan':1, 'feb':2, 'mar':3, 'apr':4, 'may':5, 'jun':6,
             'jul':7, 'aug':8, 'sep':9, 'oct':10, 'nov':11, 'dec':12}
df['month_num'] = df['month'].map(month_map)
df['month_sin'] = np.sin(2 * np.pi * df['month_num']/12)
df['month_cos'] = np.cos(2 * np.pi * df['month_num']/12)
df.drop(columns=['month', 'month_num'], inplace=True)

# Cyclic encoding for education levels :
edu_map = {
    'illiterate': 0,
    'basic.4y': 1,
    'basic.6y': 2,
    'basic.9y': 3,
    'high.school': 4,
    'professional.course': 5,
    'university.degree': 6,
}
df['education'] = df['education'].map(edu_map)

```

## Feature Selection:

### 1- Correlation analysis:

```

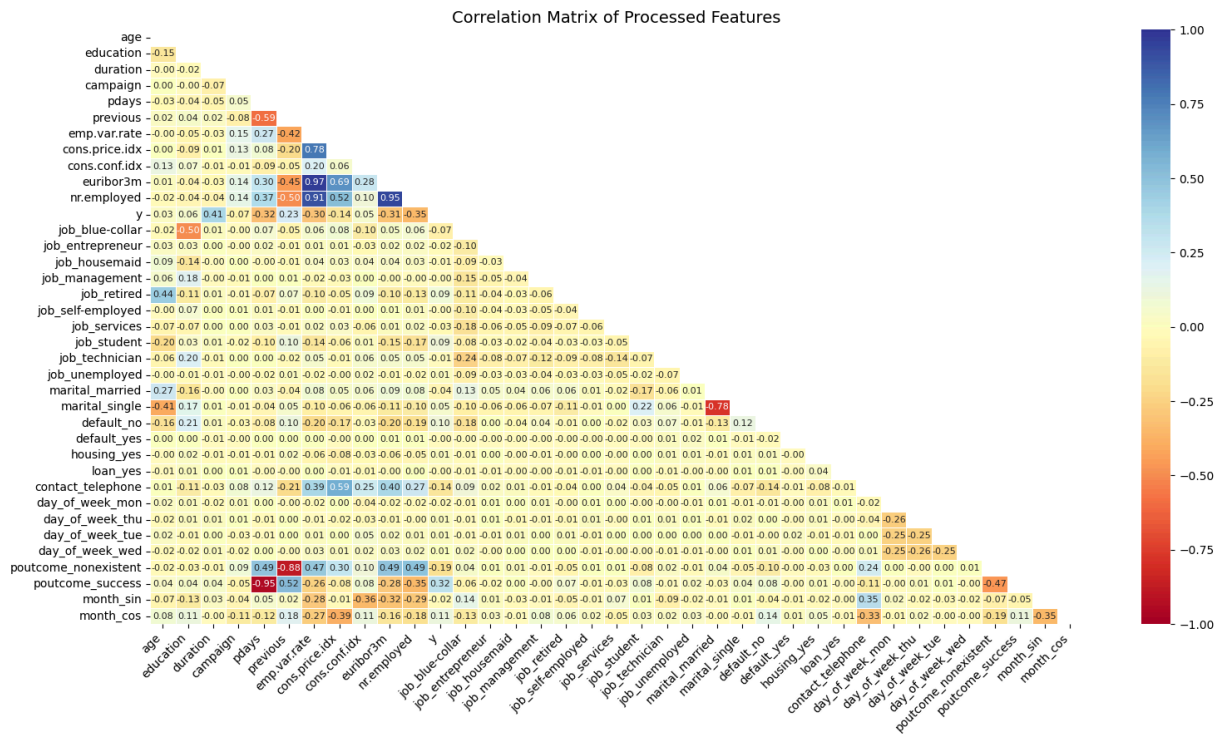
In [15]: import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

print("\nCalculating Correlation Matrix...")
correlation_matrix = df.corr()

plt.figure(figsize=(16, 9))
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, annot=True, cmap='RdYlBu', fmt='.2f', linewidths=.5)
plt.title('Correlation Matrix of Processed Features', fontsize=14)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

```

Calculating Correlation Matrix...



## Analysis of Correlation Matrix and Feature Selection

From analyzing the correlation matrix, features with high correlation coefficients (close to 1 or -1) suggest redundancy. To reduce multicollinearity, we are dropping the following features :

Feature	Highly Correlated With
<b>previous</b>	poutcome_nonexistent, nr.employed, euribor3m, ...
<b>pdays</b>	poutcome_success, poutcome_nonexistent, previous, ...
<b>emp.var.rate</b>	euribor3m, cons.price.idx, nr.employed, ...
<b>euribor3m</b>	pdays, previous, cons.price.idx, ...
<b>marital_single</b>	age, job_student, marital_married, ...

```
In [17]: # Dropping features based on correlation
df.drop(columns=['previous', 'pdays', 'emp.var.rate', 'euribor3m', 'marital_

# Visuallising the difference
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

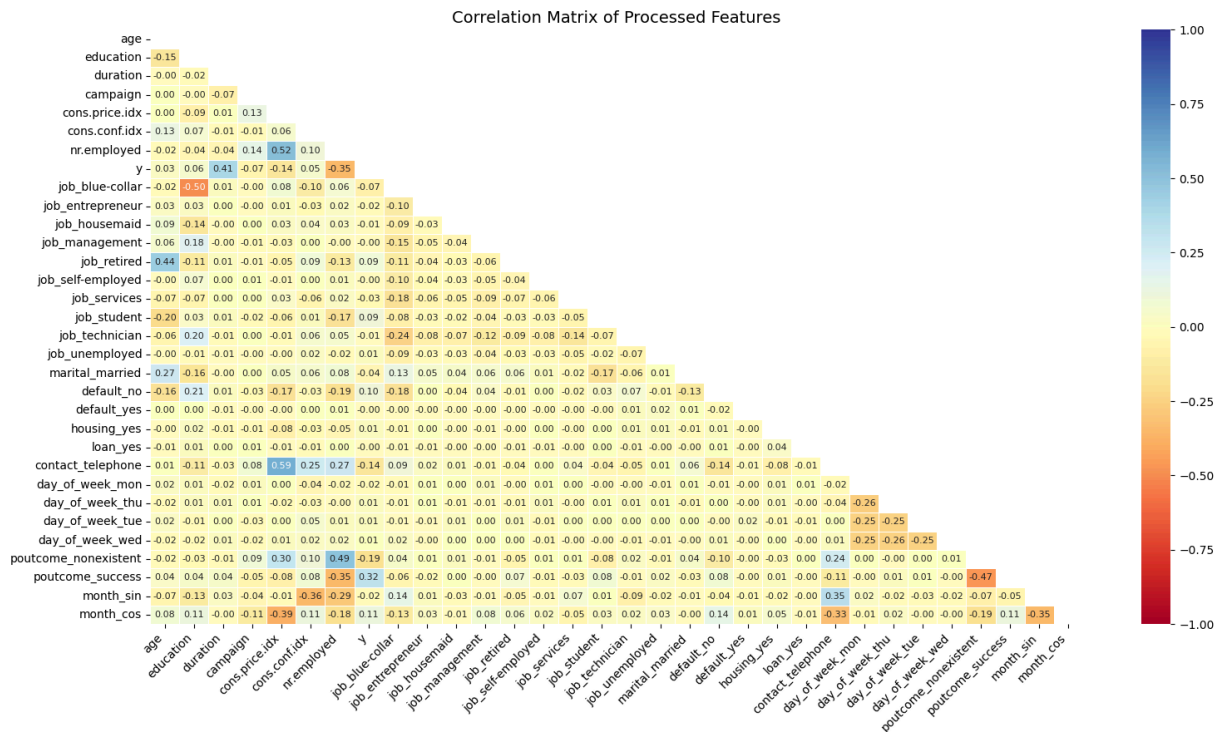
print("\nCalculating Correlation Matrix...")
correlation_matrix = df.corr()

plt.figure(figsize=(16, 9))
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, annot=True, cmap='RdYlBu', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of Processed Features', fontsize=14)
```



```
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

Calculating Correlation Matrix...



## 2- Feature Importance Analysis

```
In [19]: from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Split data
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns=['y', 'c

# Sample a subset
X_train_sample = X_train.sample(frac=0.2, random_state=42)
y_train_sample = y_train.loc[X_train_sample.index]

print("\nEvaluating feature importance using Random Forest...")
rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

# Get feature importances
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(12, 8))
plt.title('Feature Importances for Music Popularity')
plt.bar(range(X_train.shape[1]), importances[indices], align='center')
```

```

plt.xticks(range(X_train.shape[1]), [X_train.columns[i] for i in indices], r
plt.tight_layout()
plt.show()

# Select top features based on cumulative importance
cumulative_importance = np.cumsum(importances[indices])
n_features = np.where(cumulative_importance > 0.9)[0][0] + 1
top_features = [X_train.columns[i] for i in indices[:n_features]]
print(f"Top features (90% cumulative importance):\n\n{top_features}\n")

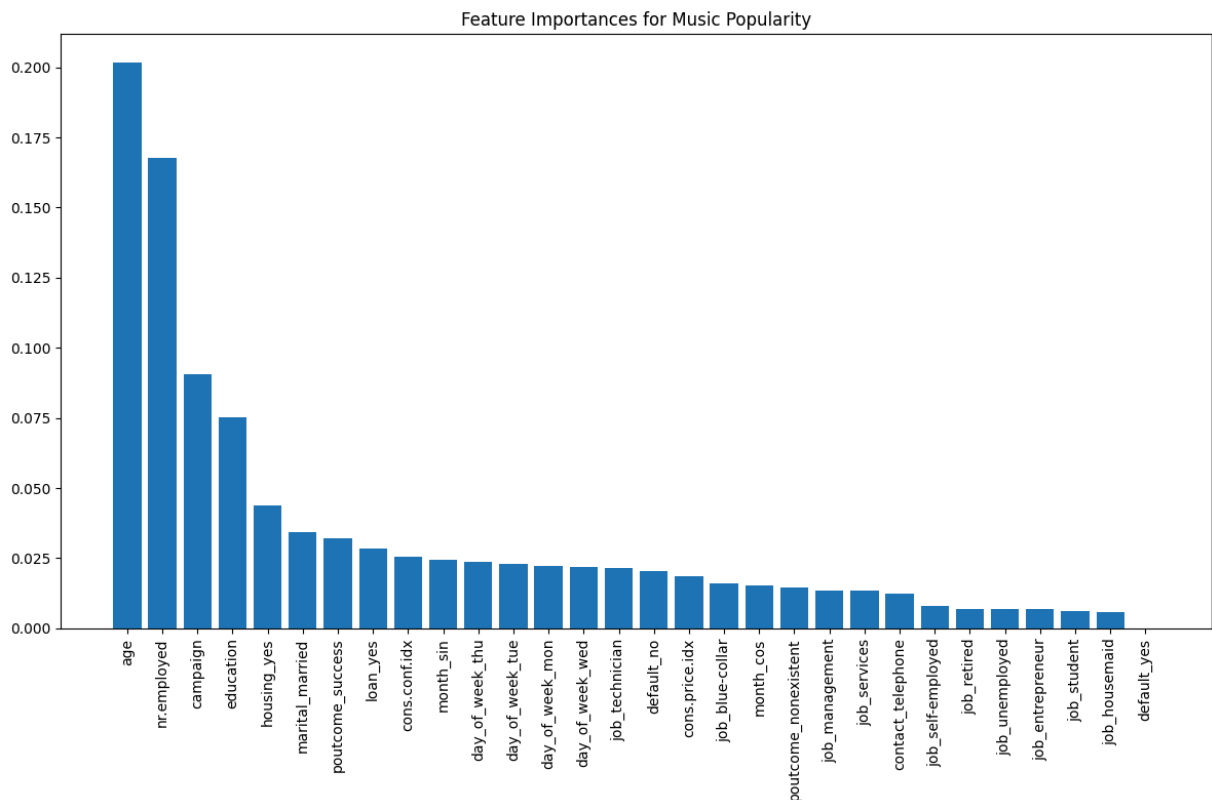
# Create refined dataset
df_important = X_train[top_features].copy()

# Retain custom features if not included
for feature in ['energy_to_acoustiness_ratio', 'vocal_character']:
    if feature not in df_important.columns and feature in X_train.columns:
        df_important[feature] = X_train[feature]

print(f"Refined feature set shape: {df_important.shape}")

```

Evaluating feature importance using Random Forest...



Top features (90% cumulative importance):

```

['age', 'nr.employed', 'campaign', 'education', 'housing_yes', 'marital_married', 'poutcome_success', 'loan_yes', 'cons.conf.idx', 'month_sin', 'day_of_week_thu', 'day_of_week_tue', 'day_of_week_mon', 'day_of_week_wed', 'job_technician', 'default_no', 'cons.price.idx', 'job_blue-collar', 'month_cos']

```

Refined feature set shape: (32940, 19)

ANN Model :

## 1- Scaling :

```
In [21]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming 'df' is your DataFrame and 'target' is the popularity column
X = df[top_features]
y = df['y']

# Standardize the features

# For boolean features
for column in X:
    if df[column].dtype == bool:
        df[column] = df[column].map({True: 1, False: 0})

# For numerical features
scaler = StandardScaler()
columns_to_exclude = ['age', 'cons.price.idx', 'cons.conf.idx', 'nr.employed']
X_scaled = X.copy()
X_scaled[X.columns.difference(columns_to_exclude)] = scaler.fit_transform(X[

# Split into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.

# Preview
print("Preview of final scaled X values : ")
X.head()
```

Preview of final scaled X values :

```
Out[21]:
```

	age	nr.employed	campaign	education	housing_yes	marital_married
0	1.533143	0.331695	0.0	1	False	True
1	1.629107	0.331695	0.0	4	False	True
2	-0.290177	0.331695	0.0	4	True	True
3	-0.002284	0.331695	0.0	2	False	True
4	1.533143	0.331695	0.0	4	False	True

## 2. Defining The Model

We decided to go ahead and experiment with an **Artificial Neural Network (ANN)** model, as we have many features to handle, introducing several complex, non-linear relationships. We'll begin with a simple architecture:

- **Input Layer:** 19 neurons (corresponding to the total number of final chosen features)
- **Hidden Layer 1:** 64 neurons, ReLU activation

- **Hidden Layer 2:** 32 neurons, ReLU activation
- **Output Layer:** 1 neuron, linear activation

In [26]: `pip install tensorflow`

Collecting tensorflow  
 Downloading tensorflow-2.16.2-cp312-cp312-macosx\_10\_15\_x86\_64.whl.metadata (4.1 kB)  
Collecting absl-py>=1.0.0 (from tensorflow)  
 Downloading absl\_py-2.2.2-py3-none-any.whl.metadata (2.6 kB)  
Collecting astunparse>=1.6.0 (from tensorflow)  
 Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)  
Collecting flatbuffers>=23.5.26 (from tensorflow)  
 Downloading flatbuffers-25.2.10-py2.py3-none-any.whl.metadata (875 bytes)  
Collecting gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 (from tensorflow)  
 Downloading gast-0.6.0-py3-none-any.whl.metadata (1.3 kB)  
Collecting google-pasta>=0.1.1 (from tensorflow)  
 Downloading google\_pasta-0.2.0-py3-none-any.whl.metadata (814 bytes)  
Requirement already satisfied: h5py>=3.10.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (3.11.0)  
Collecting libclang>=13.0.0 (from tensorflow)  
 Downloading libclang-18.1.1-py2.py3-none-macosx\_10\_9\_x86\_64.whl.metadata (5.2 kB)  
Collecting ml-dtypes~=0.3.1 (from tensorflow)  
 Downloading ml\_dtypes-0.3.2-cp312-cp312-macosx\_10\_9\_universal2.whl.metadata (20 kB)  
Collecting opt-einsum>=2.3.2 (from tensorflow)  
 Downloading opt\_einsum-3.4.0-py3-none-any.whl.metadata (6.3 kB)  
Requirement already satisfied: packaging in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (23.2)  
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (3.20.3)  
Requirement already satisfied: requests<3,>=2.21.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (2.32.2)  
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (75.2.0)  
Requirement already satisfied: six>=1.12.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (1.16.0)  
Collecting termcolor>=1.1.0 (from tensorflow)  
 Downloading termcolor-3.0.1-py3-none-any.whl.metadata (6.1 kB)  
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (4.11.0)  
Requirement already satisfied: wrapt>=1.11.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (1.14.1)  
Collecting grpcio<2.0,>=1.24.3 (from tensorflow)  
 Downloading grpcio-1.71.0-cp312-cp312-macosx\_10\_14\_universal2.whl.metadata (3.8 kB)  
Collecting tensorboard<2.17,>=2.16 (from tensorflow)  
 Downloading tensorboard-2.16.2-py3-none-any.whl.metadata (1.6 kB)  
Collecting keras>=3.0.0 (from tensorflow)  
 Downloading keras-3.9.2-py3-none-any.whl.metadata (6.1 kB)  
Requirement already satisfied: numpy<2.0.0,>=1.26.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (1.26.4)  
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (0.44.0)  
Requirement already satisfied: rich in /opt/anaconda3/lib/python3.12/site-packages (from tensorflow) (13.3.5)  
Collecting namex (from tensorflow)  
 Downloading namex-0.0.9-py3-none-any.whl.metadata (322 bytes)  
Collecting optree (from tensorflow)

Downloading optree-0.15.0-cp312-cp312-macosx\_10\_13\_universal2.whl.metadata (48 kB)

48.9/48.9 kB 931.3 kB/s eta 0:00

0:00 0:00:01

Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (2.0.4)

Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (3.7)

Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (2.2.2)

Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.12/site-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)

Requirement already satisfied: markdown>=2.6.8 in /opt/anaconda3/lib/python3.12/site-packages (from tensorboard<2.17,>=2.16->tensorflow) (3.4.1)

Collecting tensorboard-data-server<0.8.0,>=0.7.0 (from tensorboard<2.17,>=2.16->tensorflow)

Downloading tensorboard\_data\_server-0.7.2-py3-none-macosx\_10\_9\_x86\_64.whl.metadata (1.1 kB)

Requirement already satisfied: werkzeug>=1.0.1 in /opt/anaconda3/lib/python3.12/site-packages (from tensorboard<2.17,>=2.16->tensorflow) (3.0.3)

Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/anaconda3/lib/python3.12/site-packages (from werkzeug>=1.0.1->tensorboard<2.17,>=2.16->tensorflow) (2.1.3)

Requirement already satisfied: markdown-it-py<3.0.0,>=2.2.0 in /opt/anaconda3/lib/python3.12/site-packages (from rich->keras>=3.0.0->tensorflow) (2.2.0)

Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /opt/anaconda3/lib/python3.12/site-packages (from rich->keras>=3.0.0->tensorflow) (2.15.1)

Requirement already satisfied: mdurl~0.1 in /opt/anaconda3/lib/python3.12/site-packages (from markdown-it-py<3.0.0,>=2.2.0->rich->keras>=3.0.0->tensorflow) (0.1.0)

Downloading tensorflow-2.16.2-cp312-cp312-macosx\_10\_15\_x86\_64.whl (259.7 MB)

259.7/259.7 MB 1.7 MB/s eta 0:00

0:00 0:00:01

Downloading absl\_py-2.2.2-py3-none-any.whl (135 kB)

135.6/135.6 kB 2.0 MB/s eta 0:00

0:00 0:00:01

Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)

Downloading flatbuffers-25.2.10-py2.py3-none-any.whl (30 kB)

Downloading gast-0.6.0-py3-none-any.whl (21 kB)

Downloading google\_pasta-0.2.0-py3-none-any.whl (57 kB)

57.5/57.5 kB 1.6 MB/s eta 0:00:00

0

Downloading grpcio-1.71.0-cp312-cp312-macosx\_10\_14\_universal2.whl (11.3 MB)

11.3/11.3 MB 5.1 MB/s eta 0:00:00

0:00 0:00:01

Downloading keras-3.9.2-py3-none-any.whl (1.3 MB)

1.3/1.3 MB 4.0 MB/s eta 0:00:00a

0:00 0:01m

Downloading libclang-18.1.1-py2.py3-none-macosx\_10\_9\_x86\_64.whl (26.5 MB)

26.5/26.5 MB 4.0 MB/s eta 0:00:00

0:00 0:00:01

Downloading ml\_dtypes-0.3.2-cp312-cp312-macosx\_10\_9\_universal2.whl (393 kB)

393.6/393.6 kB 2.4 MB/s eta 0:00

0:00 0:00:01

Downloading opt\_einsum-3.4.0-py3-none-any.whl (71 kB)

71.9/71.9 kB 1.7 MB/s eta 0:00:00

0-:--:--

Downloading tensorboard-2.16.2-py3-none-any.whl (5.5 MB)

5.5/5.5 MB 1.9 MB/s eta 0:00:00

0:0100:010m

Downloading termcolor-3.0.1-py3-none-any.whl (7.2 kB)

Downloading tensorboard\_data\_server-0.7.2-py3-none-macosx\_10\_9\_x86\_64.whl (4.8 MB)

4.8/4.8 MB 5.3 MB/s eta 0:00:00

0:0100:01

Downloading namex-0.0.9-py3-none-any.whl (5.8 kB)

Downloading optree-0.15.0-cp312-cp312-macosx\_10\_13\_universal2.whl (639 kB)

639.5/639.5 kB 2.8 MB/s eta 0:0

0:00a 0:00:01

Installing collected packages: namex, libclang, flatbuffers, termcolor, tensorboard-data-server, optree, opt-einsum, ml-dtypes, grpcio, google-pasta, gast, astunparse, absl-py, tensorboard, keras, tensorflow

Successfully installed absl-py-2.2.2 astunparse-1.6.3 flatbuffers-25.2.10 gast-0.6.0 google-pasta-0.2.0 grpcio-1.71.0 keras-3.9.2 libclang-18.1.1 ml-dtypes-0.3.2 namex-0.0.9 opt-einsum-3.4.0 optree-0.15.0 tensorboard-2.16.2 tensorboard-data-server-0.7.2 tensorflow-2.16.2 termcolor-3.0.1

Note: you may need to restart the kernel to use updated packages.

```
In [30]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam

model = Sequential([
    # Input layer
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),

    # Hidden layers
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),

    Dense(32, activation='relu'),
    BatchNormalization(),
    Dropout(0.2),

    # Output layer - for binary classification
    Dense(1, activation='sigmoid') # Sigmoid for binary classification
])

# Compile with appropriate loss for binary classification
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy', # Suitable for binary classification
    metrics=['accuracy', 'AUC'] # Track accuracy and AUC
)
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```


### 3. Training & Evaluating the Model


We're training the model on the **scaled X values** with the following configuration:


- **Epochs:** Number of times the model sees the full dataset (e.g., 100)
- **Batch Size:** Number of samples processed before updating the model's weights (e.g., 32)
- **Validation Split:** A portion of the training data reserved to monitor performance and prevent overfitting


```
In [33]: from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
from tensorflow.keras.regularizers import l2  
  
# Define callbacks  
early_stopping = EarlyStopping(  
    monitor='val_loss',  
    patience=20, # More patience  
    restore_best_weights=True,  
    verbose=1  
)  
  
reduce_lr = ReduceLROnPlateau(  
    monitor='val_loss',  
    factor=0.5,  
    patience=5,  
    min_lr=0.00001,  
    verbose=1  
)  
  
# Train with better parameters  
history = model.fit(  
    X_train, y_train,  
    epochs=150,  
    batch_size=64, # Try different batch size  
    validation_data=(X_val, y_val),  
    callbacks=[early_stopping, reduce_lr],  
    verbose=1  
)
```





Epoch 1/150  
**515/515**  **6s** 3ms/step - AUC: 0.6467 - accuracy: 0.7521 - loss: 0.5560 - val\_AUC: 0.7864 - val\_accuracy: 0.8924 - val\_loss: 0.2963 - learning\_rate: 0.0010


Epoch 2/150  
**515/515**  **2s** 3ms/step - AUC: 0.7295 - accuracy: 0.8890 - loss: 0.3170 - val\_AUC: 0.7874 - val\_accuracy: 0.8936 - val\_loss: 0.2941 - learning\_rate: 0.0010


Epoch 3/150  
**515/515**  **1s** 3ms/step - AUC: 0.7663 - accuracy: 0.8975 - loss: 0.2918 - val\_AUC: 0.7907 - val\_accuracy: 0.8936 - val\_loss: 0.2900 - learning\_rate: 0.0010


Epoch 4/150  
**515/515**  **2s** 4ms/step - AUC: 0.7630 - accuracy: 0.8959 - loss: 0.2949 - val\_AUC: 0.7917 - val\_accuracy: 0.8934 - val\_loss: 0.2906 - learning\_rate: 0.0010


Epoch 5/150  
**515/515**  **3s** 5ms/step - AUC: 0.7687 - accuracy: 0.8942 - loss: 0.2927 - val\_AUC: 0.7901 - val\_accuracy: 0.8949 - val\_loss: 0.2891 - learning\_rate: 0.0010


Epoch 6/150  
**515/515**  **2s** 4ms/step - AUC: 0.7798 - accuracy: 0.8983 - loss: 0.2840 - val\_AUC: 0.7947 - val\_accuracy: 0.8910 - val\_loss: 0.2891 - learning\_rate: 0.0010


Epoch 7/150  
**515/515**  **1s** 3ms/step - AUC: 0.7749 - accuracy: 0.8990 - loss: 0.2846 - val\_AUC: 0.7917 - val\_accuracy: 0.8910 - val\_loss: 0.2913 - learning\_rate: 0.0010


Epoch 8/150  
**515/515**  **1s** 3ms/step - AUC: 0.7764 - accuracy: 0.8962 - loss: 0.2873 - val\_AUC: 0.7898 - val\_accuracy: 0.8936 - val\_loss: 0.2892 - learning\_rate: 0.0010


Epoch 9/150  
**515/515**  **1s** 3ms/step - AUC: 0.7830 - accuracy: 0.9004 - loss: 0.2788 - val\_AUC: 0.7885 - val\_accuracy: 0.8941 - val\_loss: 0.2889 - learning\_rate: 0.0010


Epoch 10/150  
**515/515**  **2s** 3ms/step - AUC: 0.7786 - accuracy: 0.8980 - loss: 0.2846 - val\_AUC: 0.7924 - val\_accuracy: 0.8924 - val\_loss: 0.2887 - learning\_rate: 0.0010


Epoch 11/150  
**515/515**  **1s** 3ms/step - AUC: 0.7836 - accuracy: 0.8992 - loss: 0.2795 - val\_AUC: 0.7929 - val\_accuracy: 0.8927 - val\_loss: 0.2891 - learning\_rate: 0.0010


Epoch 12/150  
**515/515**  **1s** 3ms/step - AUC: 0.7948 - accuracy: 0.9008 - loss: 0.2747 - val\_AUC: 0.7955 - val\_accuracy: 0.8927 - val\_loss: 0.2891 - learning\_rate: 0.0010


Epoch 13/150  
**515/515**  **1s** 3ms/step - AUC: 0.8004 - accuracy: 0.9010 - loss: 0.2697 - val\_AUC: 0.7994 - val\_accuracy: 0.8936 - val\_loss: 0.2871 - learning\_rate: 0.0010

Epoch 14/150  
**515/515**  **2s** 3ms/step - AUC: 0.7942 - accuracy: 0.8985 - loss: 0.2780 - val\_AUC: 0.7929 - val\_accuracy: 0.8927 - val\_loss: 0.2899 - learning\_rate: 0.0010


Epoch 15/150  
**515/515**  **2s** 3ms/step - AUC: 0.7962 - accuracy: 0.9016 - loss: 0.2736 - val\_AUC: 0.7939 - val\_accuracy: 0.8932 - val\_loss: 0.2877 - learning\_rate: 0.0010


Epoch 16/150  
**515/515**  **2s** 3ms/step - AUC: 0.7985 - accuracy: 0.9008 - loss: 0.2747 - val\_AUC: 0.7929 - val\_accuracy: 0.8934 - val\_loss: 0.2879 - learning\_rate: 0.0010


Epoch 17/150  
**515/515**  **2s** 3ms/step - AUC: 0.7943 - accuracy: 0.8977 - loss: 0.2812 - val\_AUC: 0.7920 - val\_accuracy: 0.8924 - val\_loss: 0.2880 - learning\_rate: 0.0010


Epoch 18/150  
**506/515**  **0s** 3ms/step - AUC: 0.7990 - accuracy: 0.8998 - loss: 0.2746


Epoch 18: ReduceLRonPlateau reducing learning rate to 0.0005000000237487257.


Epoch 19/150  
**515/515**  **2s** 4ms/step - AUC: 0.7989 - accuracy: 0.8998 - loss: 0.2746 - val\_AUC: 0.7942 - val\_accuracy: 0.8922 - val\_loss: 0.2884 - learning\_rate: 0.0010


Epoch 20/150  
**515/515**  **2s** 4ms/step - AUC: 0.7934 - accuracy: 0.8991 - loss: 0.2770 - val\_AUC: 0.7930 - val\_accuracy: 0.8936 - val\_loss: 0.2875 - learning\_rate: 5.0000e-04


Epoch 21/150  
**515/515**  **1s** 3ms/step - AUC: 0.7951 - accuracy: 0.8983 - loss: 0.2772 - val\_AUC: 0.7956 - val\_accuracy: 0.8932 - val\_loss: 0.2868 - learning\_rate: 5.0000e-04

Epoch 22/150  
**515/515**  **1s** 2ms/step - AUC: 0.8037 - accuracy: 0.8993 - loss: 0.2738 - val\_AUC: 0.7932 - val\_accuracy: 0.8963 - val\_loss: 0.2872 - learning\_rate: 5.0000e-04


Epoch 23/150  
**515/515**  **1s** 2ms/step - AUC: 0.8017 - accuracy: 0.9015 - loss: 0.2736 - val\_AUC: 0.7893 - val\_accuracy: 0.8927 - val\_loss: 0.2877 - learning\_rate: 5.0000e-04


Epoch 24/150  
**515/515**  **1s** 3ms/step - AUC: 0.7938 - accuracy: 0.9014 - loss: 0.2719 - val\_AUC: 0.7925 - val\_accuracy: 0.8934 - val\_loss: 0.2877 - learning\_rate: 5.0000e-04


Epoch 25/150  
**515/515**  **1s** 3ms/step - AUC: 0.8105 - accuracy: 0.9010 - loss: 0.2696 - val\_AUC: 0.7917 - val\_accuracy: 0.8949 - val\_loss: 0.2885 - learning\_rate: 5.0000e-04










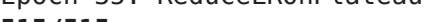




Epoch 26/150  
**512/515**  **0s** 3ms/step - AUC: 0.8055 - accuracy: 0.9003 - loss: 0.2722

Epoch 25: ReduceLRonPlateau reducing learning rate to 0.0002500000118743628.


Epoch 27/150  
**515/515**  **2s** 3ms/step - AUC: 0.8055 - accuracy: 0.9003 - loss: 0.2722 - val\_AUC: 0.7870 - val\_accuracy: 0.8941 - val\_loss: 0.2886 - learning\_rate: 5.0000e-04

Epoch 26/150  
**515/515**  **2s** 3ms/step - AUC: 0.8021 - accuracy: 0.9018 - loss: 0.2730 - val\_AUC: 0.7918 - val\_accuracy: 0.8941 - val\_loss: 0.2880 - learning\_rate: 2.5000e-04


Epoch 27/150  
**515/515**  **2s** 3ms/step - AUC: 0.8107 - accuracy: 0.9007 -

loss: 0.2717 - val\_AUC: 0.7898 - val\_accuracy: 0.8939 - val\_loss: 0.2881 - learning\_rate: 2.5000e-04  
Epoch 28/150  
**515/515**  1s 2ms/step - AUC: 0.8050 - accuracy: 0.9006 - loss: 0.2713 - val\_AUC: 0.7890 - val\_accuracy: 0.8939 - val\_loss: 0.2882 - learning\_rate: 2.5000e-04  
Epoch 29/150  
**515/515**  1s 3ms/step - AUC: 0.8076 - accuracy: 0.9022 - loss: 0.2720 - val\_AUC: 0.7903 - val\_accuracy: 0.8941 - val\_loss: 0.2883 - learning\_rate: 2.5000e-04  
Epoch 30/150  
**513/515**  0s 2ms/step - AUC: 0.8010 - accuracy: 0.9004 - loss: 0.2743  
Epoch 30: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.  
**515/515**  1s 2ms/step - AUC: 0.8010 - accuracy: 0.9004 - loss: 0.2743 - val\_AUC: 0.7909 - val\_accuracy: 0.8941 - val\_loss: 0.2886 - learning\_rate: 2.5000e-04  
Epoch 31/150  
**515/515**  1s 3ms/step - AUC: 0.7994 - accuracy: 0.9015 - loss: 0.2720 - val\_AUC: 0.7902 - val\_accuracy: 0.8936 - val\_loss: 0.2882 - learning\_rate: 1.2500e-04  
Epoch 32/150  
**515/515**  1s 3ms/step - AUC: 0.8040 - accuracy: 0.8989 - loss: 0.2750 - val\_AUC: 0.7907 - val\_accuracy: 0.8932 - val\_loss: 0.2880 - learning\_rate: 1.2500e-04  
Epoch 33/150  
**515/515**  1s 3ms/step - AUC: 0.8178 - accuracy: 0.9033 - loss: 0.2637 - val\_AUC: 0.7905 - val\_accuracy: 0.8927 - val\_loss: 0.2880 - learning\_rate: 1.2500e-04  
Epoch 34/150  
**515/515**  1s 3ms/step - AUC: 0.8190 - accuracy: 0.9020 - loss: 0.2657 - val\_AUC: 0.7900 - val\_accuracy: 0.8934 - val\_loss: 0.2884 - learning\_rate: 1.2500e-04  
Epoch 35/150  
**506/515**  0s 2ms/step - AUC: 0.8123 - accuracy: 0.9012 - loss: 0.2692  
Epoch 35: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.  
**515/515**  1s 2ms/step - AUC: 0.8122 - accuracy: 0.9012 - loss: 0.2692 - val\_AUC: 0.7902 - val\_accuracy: 0.8934 - val\_loss: 0.2884 - learning\_rate: 1.2500e-04  
Epoch 36/150  
**515/515**  1s 2ms/step - AUC: 0.8125 - accuracy: 0.9032 - loss: 0.2658 - val\_AUC: 0.7905 - val\_accuracy: 0.8932 - val\_loss: 0.2884 - learning\_rate: 6.2500e-05  
Epoch 37/150  
**515/515**  1s 2ms/step - AUC: 0.8081 - accuracy: 0.9039 - loss: 0.2645 - val\_AUC: 0.7904 - val\_accuracy: 0.8932 - val\_loss: 0.2885 - learning\_rate: 6.2500e-05  
Epoch 38/150  
**515/515**  1s 2ms/step - AUC: 0.8150 - accuracy: 0.9014 - loss: 0.2662 - val\_AUC: 0.7904 - val\_accuracy: 0.8929 - val\_loss: 0.2885 - learning\_rate: 6.2500e-05  
Epoch 39/150  
**515/515**  1s 2ms/step - AUC: 0.8140 - accuracy: 0.9004 - loss: 0.2695 - val\_AUC: 0.7896 - val\_accuracy: 0.8934 - val\_loss: 0.2884 - learning\_rate: 6.2500e-05

Epoch 40/150

**496/515**  **0s** 2ms/step - AUC: 0.8082 - accuracy: 0.9010 - loss: 0.2719

Epoch 40: ReduceLRonPlateau reducing learning rate to 3.125000148429535e-05.

**515/515**  **1s** 2ms/step - AUC: 0.8082 - accuracy: 0.9011 - loss: 0.2718 - val\_AUC: 0.7890 - val\_accuracy: 0.8936 - val\_loss: 0.2885 - learning\_rate: 6.2500e-05

Epoch 40: early stopping

Restoring model weights from the end of the best epoch: 20.

```
In [34]: import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, roc_curve

# Get probabilities
y_pred_prob = model.predict(X_test)
# Convert to binary predictions
y_pred_binary = (y_pred_prob > 0.5).astype(int)

# Calculate metrics
print(classification_report(y_test, y_pred_binary))

# Create confusion matrix
cm = confusion_matrix(y_test, y_pred_binary)

# Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Create visualization dashboard
fig = plt.figure(figsize=(16, 12))
fig.suptitle('Bank Marketing Classification Results', fontsize=16, fontweight='bold')

# Define grid for subplots
gs = fig.add_gridspec(2, 2, hspace=0.3, wspace=0.3)

# ROC curve
ax1 = fig.add_subplot(gs[0, 0])
ax1.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc})')
ax1.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('Receiver Operating Characteristic (ROC)')
ax1.legend(loc="lower right")
ax1.grid(True, alpha=0.3)

# Confusion Matrix
ax2 = fig.add_subplot(gs[0, 1])
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax2)
ax2.set_xlabel('Predicted Label')
ax2.set_ylabel('True Label')
ax2.set_title('Confusion Matrix')

# Training history
ax3 = fig.add_subplot(gs[1, 0])
ax3.plot(history.history['loss'], label='Training Loss')
```

```

ax3.plot(history.history['val_loss'], label='Validation Loss')
ax3.set_xlabel('Epochs')
ax3.set_ylabel('Loss')
ax3.set_title('Training and Validation Loss')
ax3.legend()
ax3.grid(True, alpha=0.3)

# Classification metrics table
ax4 = fig.add_subplot(gs[1, 1])
ax4.axis('off')
metrics_text = f"""
CLASSIFICATION METRICS

Accuracy: {accuracy_score(y_test, y_pred_binary):.4f}
Precision: {precision_score(y_test, y_pred_binary):.4f}
Recall: {recall_score(y_test, y_pred_binary):.4f}
F1 Score: {f1_score(y_test, y_pred_binary):.4f}
AUC: {roc_auc:.4f}
"""
ax4.text(0.5, 0.5, metrics_text, fontsize=12,
        ha='center', va='center',
        bbox=dict(boxstyle="round,pad=1", facecolor='#f0f0f0',
                  edgecolor='gray', alpha=0.7))

plt.tight_layout()
plt.subplots_adjust(top=0.9)
plt.show()

```

129/129 ————— 0s 2ms/step

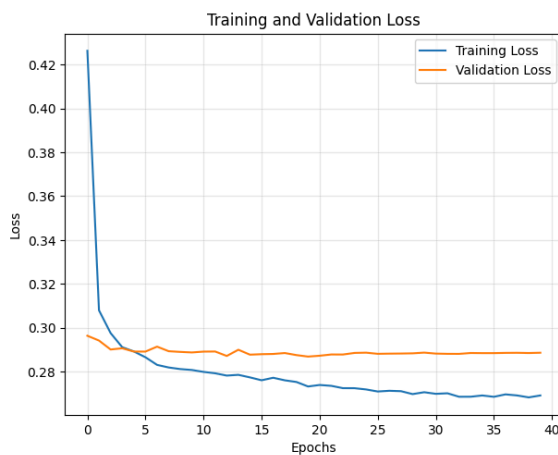
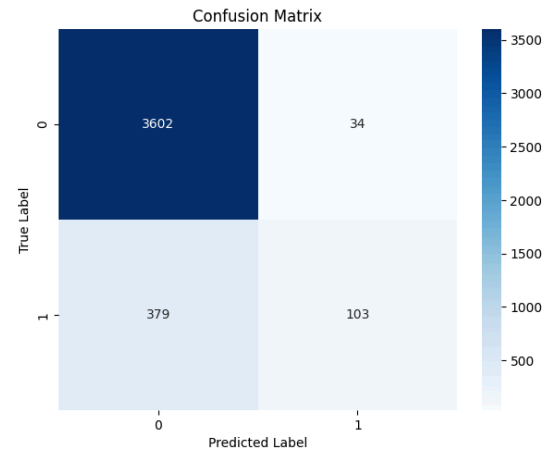
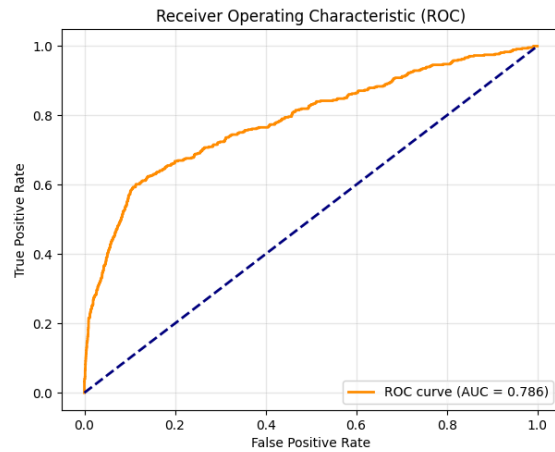
	precision	recall	f1-score	support
0	0.90	0.99	0.95	3636
1	0.75	0.21	0.33	482
accuracy			0.90	4118
macro avg	0.83	0.60	0.64	4118
weighted avg	0.89	0.90	0.87	4118

```

/var/folders/96/58r8hw9s4kjdr3zgpcmyjtn80000gn/T/ipykernel_65146/2964765037.
py:71: UserWarning: This figure includes Axes that are not compatible with t
ight_layout, so results might be incorrect.
plt.tight_layout()

```

## Bank Marketing Classification Results



### CLASSIFICATION METRICS

Accuracy: 0.8997  
Precision: 0.7518  
Recall: 0.2137  
F1 Score: 0.3328  
AUC: 0.7863

## 4- Cross Validation :

```
In [36]: from sklearn.model_selection import KFold
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping

# Set up cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = []

# Manual cross-validation loop
for train_idx, val_idx in kfold.split(X_scaled):
    # Split data
    X_train_cv, X_val_cv = X_scaled.iloc[train_idx], X_scaled.iloc[val_idx]
    y_train_cv, y_val_cv = y.iloc[train_idx], y.iloc[val_idx]

    # Build model
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(X_scaled.
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.BatchNormalization(),
```

```

        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    # Compile
    model.compile(
        optimizer=tf.keras.optimizers.Adam(0.001),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    # Train
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_
history = model.fit(
        X_train_cv, y_train_cv,
        epochs=50,
        batch_size=64,
        validation_data=(X_val_cv, y_val_cv),
        callbacks=[early_stopping],
        verbose=0
    )

    # Evaluate
    _, accuracy = model.evaluate(X_val_cv, y_val_cv, verbose=0)
    cv_scores.append(accuracy)

# Print results
print(f"CV Accuracy: {np.mean(cv_scores):.4f} ( $\pm$ {np.std(cv_scores):.4f})")

```

```

/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
CV Accuracy: 0.8992 ( $\pm 0.0022$ )

```

## 5- Fine Tuning the hyper parameters :

```

In [38]: import itertools
        from tensorflow.keras.optimizers import Adam, RMSprop

        # Define parameter grid
        param_combinations = list(itertools.product(
            [32, 64, 128], # batch_size
            [0.001, 0.01], # learning_rate
            [0.2, 0.3],    # dropout_rate
            ['adam', 'rmsprop'] # optimizer
        ))

        results = []

        # Use a sample of the data for quicker hyperparameter tuning
        train_idx, val_idx = train_test_split(np.arange(len(X_train)), test_size=0.3)
        X_train_sample, X_val_sample = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_train_sample, y_val_sample = y_train.iloc[train_idx], y_train.iloc[val_idx]

        # Try each combination
        for batch_size, learning_rate, dropout_rate, optimizer_name in param_combinations:
            # Build model
            model = tf.keras.Sequential([
                tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train_sample.shape[1],)),
                tf.keras.layers.Dropout(dropout_rate),
                tf.keras.layers.Dense(64, activation='relu'),
                tf.keras.layers.Dropout(dropout_rate),

```



```

        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dropout(dropout_rate),
        tf.keras.layers.Dense(1, activation='sigmoid')
    ])

    # Configure optimizer
    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=learning_rate)
    else:
        optimizer = RMSprop(learning_rate=learning_rate)

    # Compile
    model.compile(
        optimizer=optimizer,
        loss='binary_crossentropy',
        metrics=['accuracy']
    )

    # Train
    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    history = model.fit(
        X_train_sample, y_train_sample,
        epochs=30, # Reduced for speed
        batch_size=batch_size,
        validation_data=(X_val_sample, y_val_sample),
        callbacks=[early_stopping],
        verbose=0
    )

    # Evaluate
    _, accuracy = model.evaluate(X_val_sample, y_val_sample, verbose=0)

    # Store results
    params = {
        'batch_size': batch_size,
        'learning_rate': learning_rate,
        'dropout_rate': dropout_rate,
        'optimizer': optimizer_name,
        'accuracy': accuracy
    }
    results.append(params)
    print(f"Params: {params}")

    # Find best parameters
    best_params = max(results, key=lambda x: x['accuracy'])
    print(f"\nBest parameters: {best_params}")

```

/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```

    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Params: {'batch_size': 32, 'learning_rate': 0.001, 'dropout_rate': 0.2, 'optimizer': 'adam', 'accuracy': 0.9023476839065552}

```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Params: {'batch_size': 32, 'learning_rate': 0.001, 'dropout_rate': 0.2, 'optimizer': 'rmsprop', 'accuracy': 0.9024488925933838}
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Params: {'batch_size': 32, 'learning_rate': 0.001, 'dropout_rate': 0.3, 'optimizer': 'adam', 'accuracy': 0.9013357758522034}
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Params: {'batch_size': 32, 'learning_rate': 0.001, 'dropout_rate': 0.3, 'optimizer': 'rmsprop', 'accuracy': 0.9024488925933838}
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Params: {'batch_size': 32, 'learning_rate': 0.01, 'dropout_rate': 0.2, 'optimizer': 'adam', 'accuracy': 0.900728583358765}
```

```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Params: {'batch_size': 32, 'learning_rate': 0.01, 'dropout_rate': 0.2, 'optimizer': 'rmsprop', 'accuracy': 0.9013357758522034}
```

Best parameters: {'batch\_size': 32, 'learning\_rate': 0.001, 'dropout\_rate': 0.2, 'optimizer': 'rmsprop', 'accuracy': 0.9024488925933838}

## Making predictions and Final Evaluation :

```
In [40]: from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Extract best parameters
best_batch_size = best_params['batch_size']
best_learning_rate = best_params['learning_rate']
best_dropout_rate = best_params['dropout_rate']
best_optimizer_name = best_params['optimizer']

# Rebuild final model with best params
final_model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(best_dropout_rate),
```

```

tf.keras.layers.Dense(64, activation='relu'),
tf.keras.layers.Dropout(best_dropout_rate),
tf.keras.layers.Dense(32, activation='relu'),
tf.keras.layers.Dropout(best_dropout_rate),
tf.keras.layers.Dense(1, activation='sigmoid')
])

optimizer = Adam(learning_rate=best_learning_rate) if best_optimizer_name ==

final_model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics

# Train final model
final_model.fit(X_train, y_train, epochs=50, batch_size=best_params['batch_s
validation_split=0.2, callbacks=[EarlyStopping(patience=5, r

# Predict and evaluate
y_pred_probs = final_model.predict(X_test).ravel()
y_pred = (y_pred_probs > 0.5).astype(int)

print(f"\nFinal Test Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print(f"ROC AUC Score: {roc_auc_score(y_test, y_pred_probs):.4f}\n")
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# ROC Curve
RocCurveDisplay.from_predictions(y_test, y_pred_probs)
plt.title("ROC Curve")
plt.grid()
plt.show()

```

/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```

super().__init__(activity_regularizer=activity_regularizer, **kwargs)
129/129 ————— 0s 1ms/step

```

Final Test Accuracy: 0.8961

ROC AUC Score: 0.7777

Classification Report:

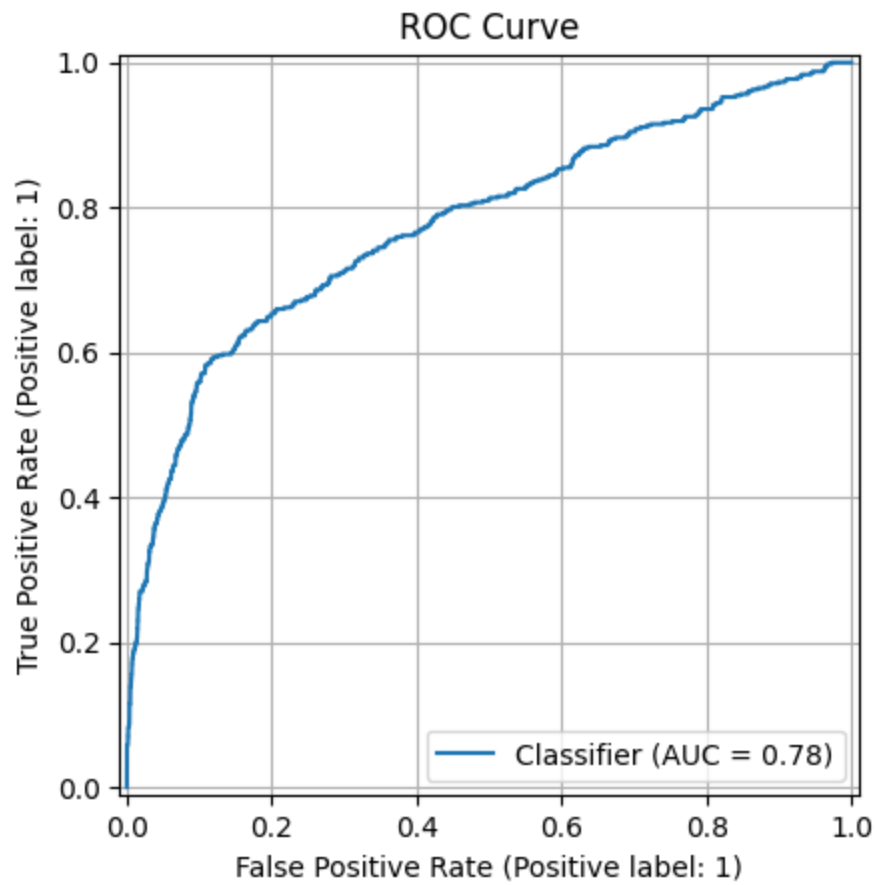
	precision	recall	f1-score	support
0	0.90	0.99	0.94	3636
1	0.76	0.16	0.27	482
accuracy			0.90	4118
macro avg	0.83	0.58	0.61	4118
weighted avg	0.88	0.90	0.87	4118

Confusion Matrix:

```

[[3611  25]
 [ 403  79]]

```



This notebook was converted with [convert.ploomber.io](https://convert.ploomber.io)