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	mo
0	56	housemaid	married	basic.4y	no	no	no	telephone	n
1	57	services	married	high.school	unknown	no	no	telephone	n
2	37	services	married	high.school	no	yes	no	telephone	n

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

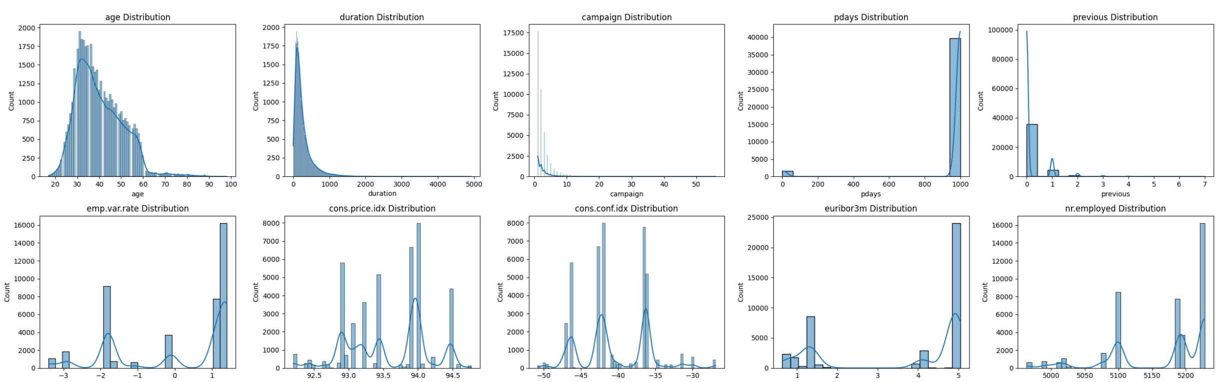
Feature	Description	Range
Nr.employed	Number of employees (quarterly indicator)	Numeric
y (Target)	Has the client subscribed to a term deposit?	"yes", "no"

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()
```

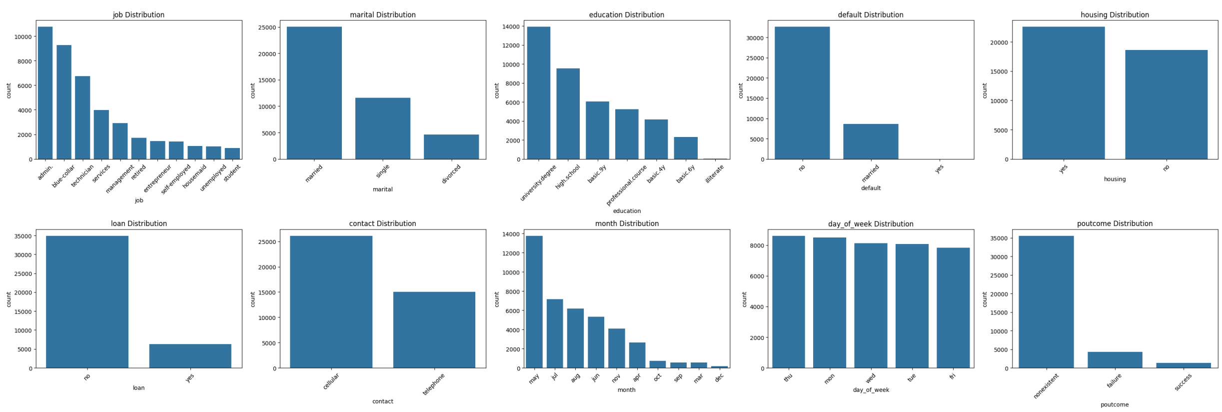


2- Categorical Data :

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().i
    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)
```

```python
# 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)
```

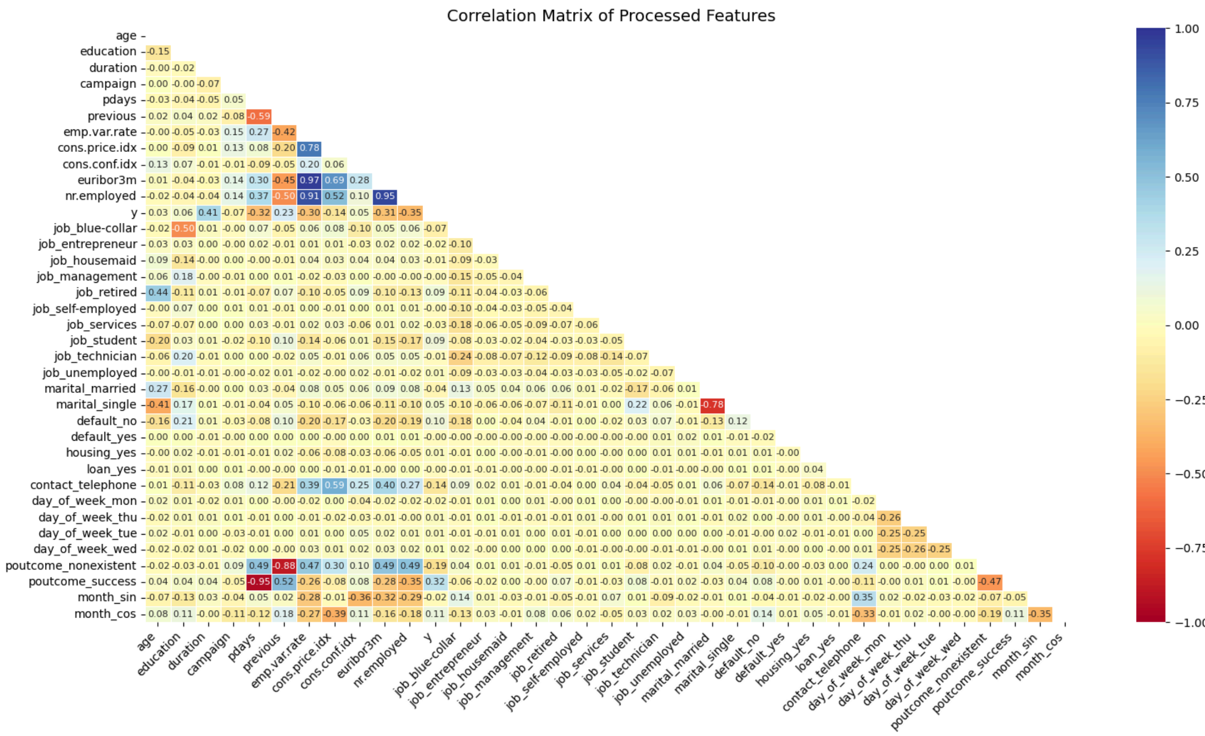
Feature Selection:

1- Correlation analysis:

In [15]:

```python
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', linewi
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

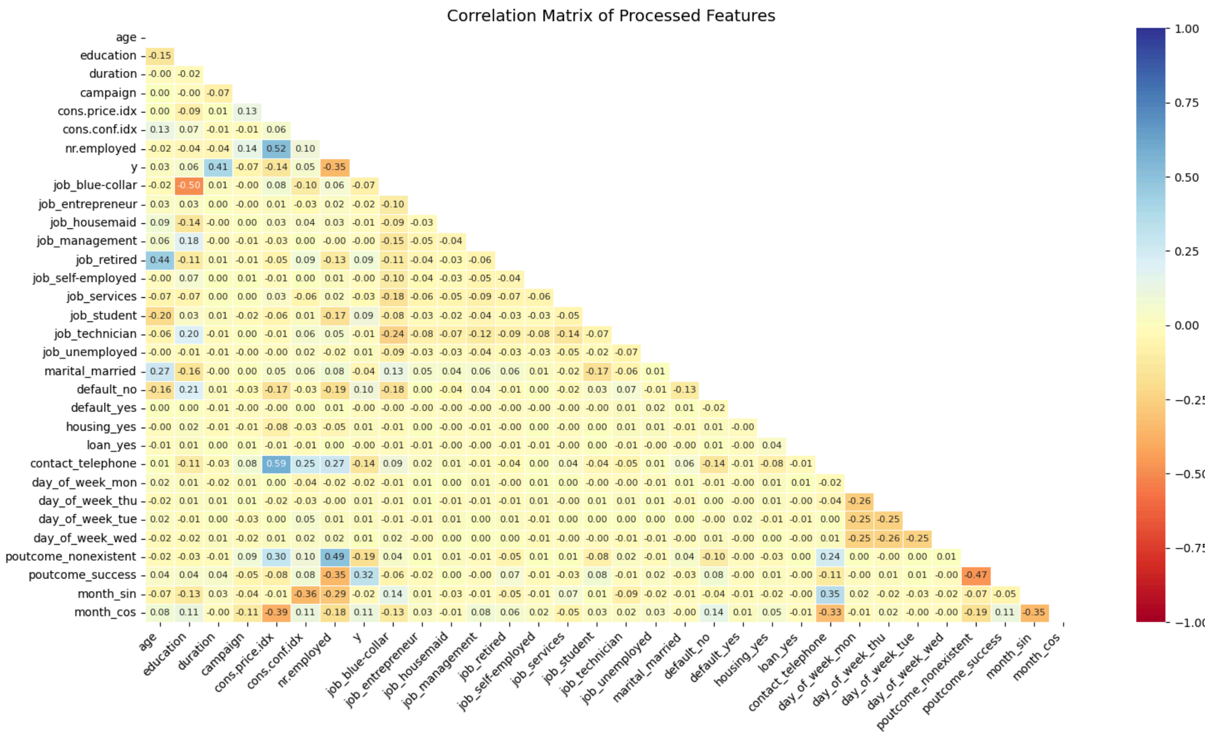
From analyzing the correlation matrix, features with high correlation coefficients (close to 1 or -1) suggest redundancy. To reduce multicollinearity, we are dropping the following features :

Feature	Highly Correlated With
previous	poutcome_nonexistent, nr.employed, euribor3m, ...
pdays	poutcome_success, poutcome_nonexistent, previous, ...
emp.var.rate	euribor3m, cons.price.idx, nr.employed, ...
euribor3m	pdays, previous, cons.price.idx, ...
marital_single	age, job_student, marital married, ...

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

# Visualising 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', linewi
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...
```

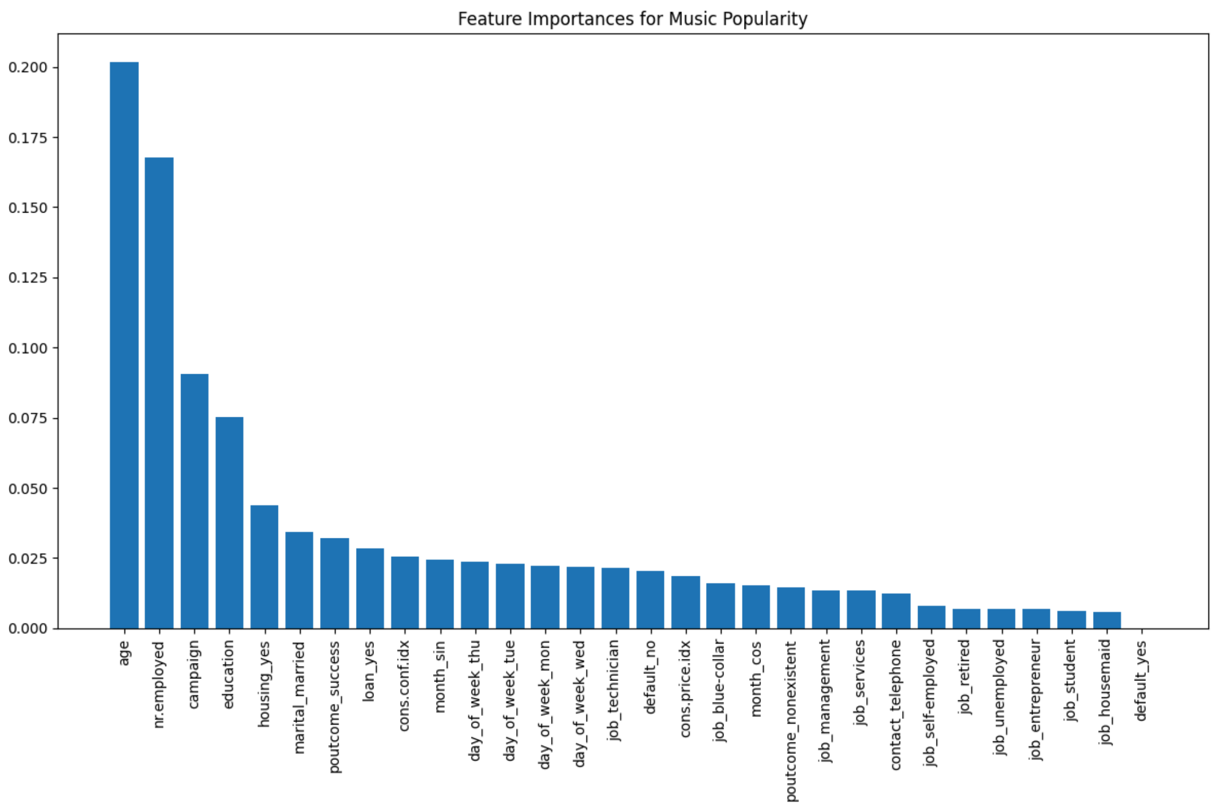


2- Feature Importance Analysis

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')
```

```python
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_acousticness_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_marr
ied', '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_tec
hnician', '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.employec
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 [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:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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
```

```python
# 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 - l
earning_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 - l
earning_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 - l
earning_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.
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 - l
earning_rate: 0.0010
Epoch 19/150
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 - l
earning_rate: 5.0000e-04
Epoch 20/150
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 - l
earning_rate: 5.0000e-04
Epoch 21/150
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 - l
earning_rate: 5.0000e-04
Epoch 22/150
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 - l
earning_rate: 5.0000e-04
Epoch 23/150
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 - l
earning_rate: 5.0000e-04
Epoch 24/150
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 - l
earning_rate: 5.0000e-04
Epoch 25/150
512/515 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - AUC: 0.8055 - accuracy: 0.9003 -
loss: 0.2722
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_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 - l
earning_rate: 6.2500e-05
```

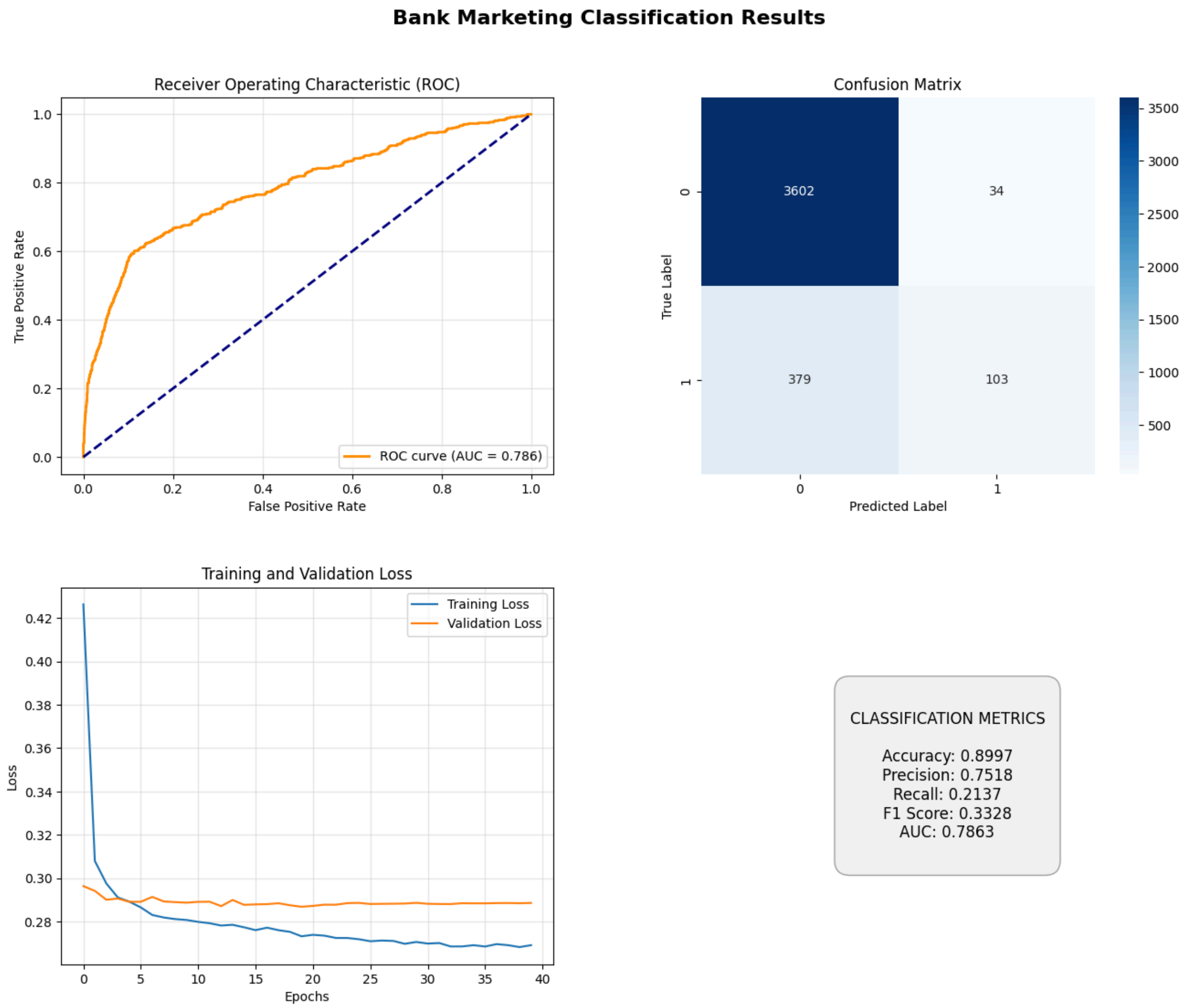
```
Epoch 40/150
496/515 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - AUC: 0.8082 - accuracy: 0.9010 -
loss: 0.2719
Epoch 40: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
515/515 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - AUC: 0.8082 - accuracy: 0.9011 -
loss: 0.2718 - val_AUC: 0.7890 - val_accuracy: 0.8936 - val_loss: 0.2885 - l
earning_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_cur

# 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, fontweigh

# 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_a
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')
```

```python
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()
```



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} (±{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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
CV Accuracy: 0.8992 (±0.0022)
```

5- Fine Tuning the hyper parameters :

In [38]:

```python
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_combina
    # Build model
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(X_train.s
        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_b
    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:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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, 'opt
imizer': 'adam', 'accuracy': 0.9023476839065552}
```

```
/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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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, 'opt
imizer': 'rmsprop', 'accuracy': 0.9024488925933838}
/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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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, 'opt
imizer': 'adam', 'accuracy': 0.9013357758522034}
/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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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, 'opt
imizer': 'rmsprop', 'accuracy': 0.9024488925933838}
/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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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, 'opti
mizer': 'adam', 'accuracy': 0.9007285833358765}
/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 laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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, 'opti
mizer': 'rmsprop', 'accuracy': 0.9013357758522034}

Best parameters: {'batch_size': 32, 'learning_rate': 0.001, 'dropout_rate':
0.2, 'optimizer': 'rmsprop', 'accuracy': 0.9024488925933838}
```

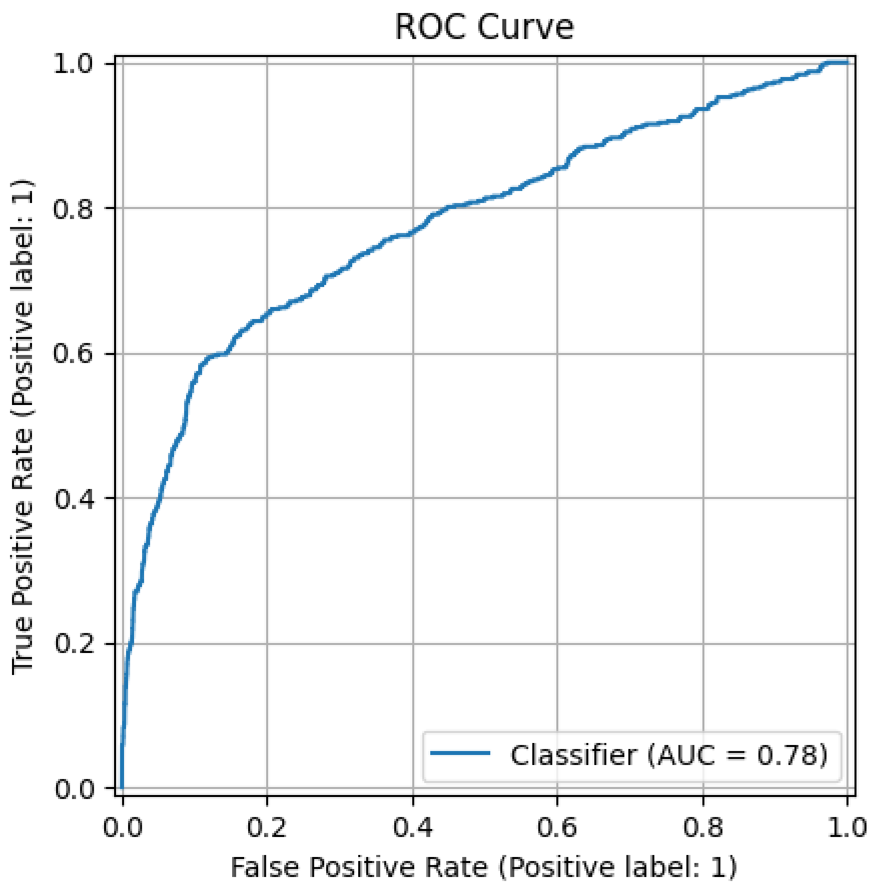
Making predictions and Final Evaluation :

In [40]:

```python
from sklearn.metrics import accuracy_score, classification_report, confusion
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
    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:8
7: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as th
e 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