In [35]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = "/Users/anthonywinatasalim/Documents/UOL Y3S1/machine learning /
data_box = pd.read_csv(file_path)

# Display basic information about the dataset
print("Dataset Information:")
data_box.info()

# Display the first few rows of the dataset
print("\nFirst 5 rows of the dataset:")
display(data_box.head())

# Summary statistics of numerical features
print("\nSummary Statistics:")
display(data_box.describe())

# Checking for missing values
print("\nMissing Values:")
display(data_box.isnull().sum())

# Checking class distribution of the target variable
print("\nTarget Variable Distribution:")
outcome_counts = data_box['Target_Churn'].value_counts()
print(outcome_counts)

# Visualizing the distribution of the target variable
plt.figure(figsize=(6, 4))
sns.barplot(x=outcome_counts.index, y=outcome_counts.values, palette='coolwa
plt.xlabel("Churn Status")
plt.ylabel("Count")
plt.title("Distribution of Target Variable (Churn)")
plt.xticks(ticks=[0,1], labels=["Not Churned", "Churned"])
plt.show()

# Visualizing numerical feature distributions
data_box.hist(figsize=(12, 10), bins=20, edgecolor='black')
plt.suptitle("Distribution of Numerical Features", fontsize=14)
plt.show()

# Checking unique values in categorical columns
category_labels = ["Gender", "Promotion_Response"]
for label in category_labels:
    print(f"\nUnique values in {label}:")
    print(data_box[label].value_counts())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Customer_ID                  1000 non-null   int64
 1   Age                          1000 non-null   int64
 2   Gender                       1000 non-null   object
 3   Annual_Income                1000 non-null   float64
 4   Total_Spend                  1000 non-null   float64
 5   Years_as_Customer            1000 non-null   int64
 6   Num_of_Purchases             1000 non-null   int64
 7   Average_Transaction_Amount   1000 non-null   float64
 8   Num_of_Returns               1000 non-null   int64
 9   Num_of_Support_Contacts      1000 non-null   int64
 10  Satisfaction_Score           1000 non-null   int64
 11  Last_Purchase_Days_Ago       1000 non-null   int64
 12  Email_Opt_In                 1000 non-null   bool
 13  Promotion_Response           1000 non-null   object
 14  Target_Churn                 1000 non-null   bool
dtypes: bool(2), float64(3), int64(8), object(2)
memory usage: 103.6+ KB
```

First 5 rows of the dataset:

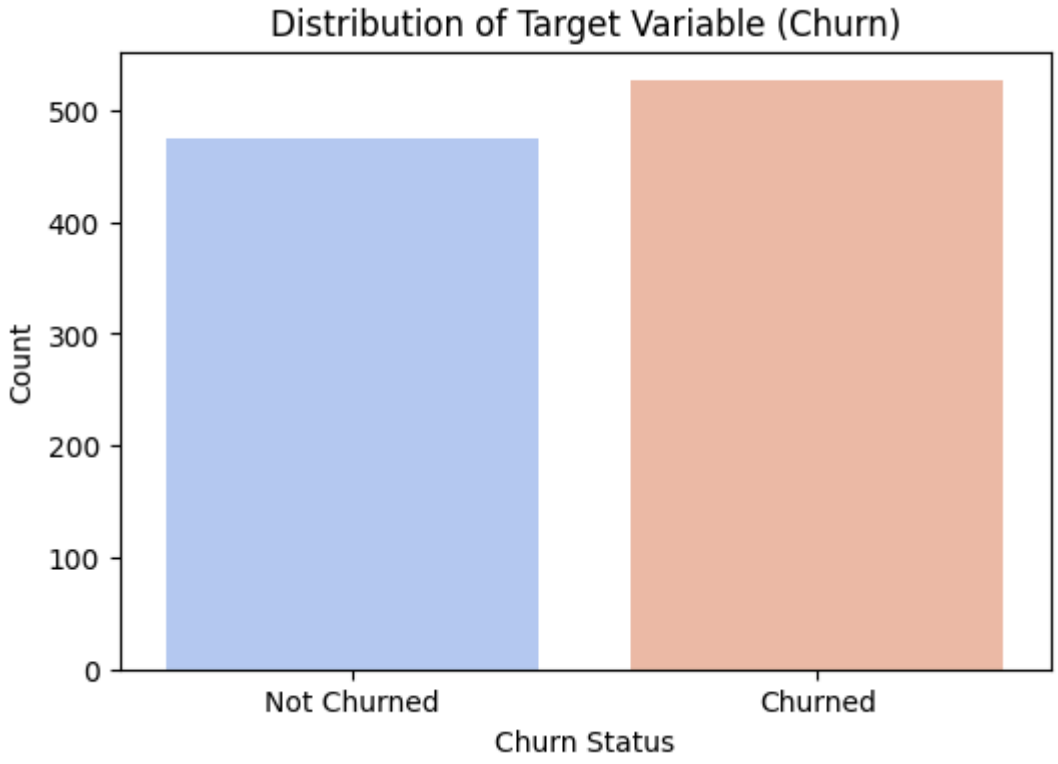
	Customer_ID	Age	Gender	Annual_Income	Total_Spend	Years_as_Customer	Num_of_P
0	1	62	Other	45.15	5892.58	5	
1	2	65	Male	79.51	9025.47	13	
2	3	18	Male	29.19	618.83	13	
3	4	21	Other	79.63	9110.30	3	
4	5	21	Other	77.66	5390.88	15	

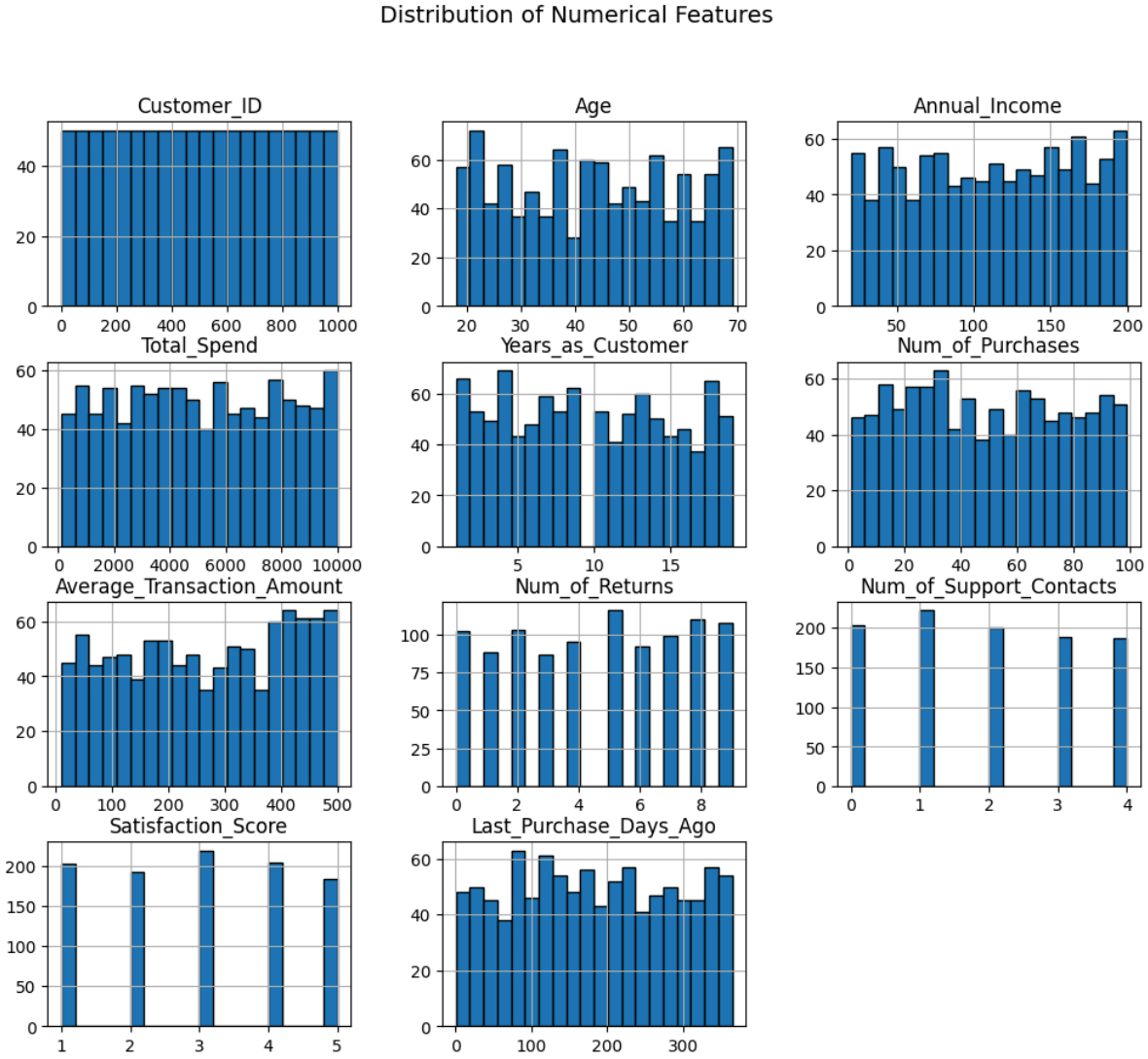
Summary Statistics:

	Customer_ID	Age	Annual_Income	Total_Spend	Years_as_Customer	Num_c
count	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	
mean	500.500000	43.267000	111.962960	5080.79265	9.727000	
std	288.819436	15.242311	52.844111	2862.12335	5.536346	
min	1.000000	18.000000	20.010000	108.94000	1.000000	
25%	250.750000	30.000000	67.800000	2678.67500	5.000000	
50%	500.500000	43.000000	114.140000	4986.19500	9.000000	
75%	750.250000	56.000000	158.452500	7606.47000	14.000000	
max	1000.000000	69.000000	199.730000	9999.64000	19.000000	

Missing Values:

```
Customer_ID                  0
Age                          0
Gender                       0
Annual_Income                0
Total_Spend                  0
Years_as_Customer            0
Num_of_Purchases             0
Average_Transaction_Amount   0
Num_of_Returns               0
Num_of_Support_Contacts      0
Satisfaction_Score           0
Last_Purchase_Days_Ago       0
Email_Opt_In                 0
Promotion_Response           0
Target_Churn                 0
dtype: int64
Target Variable Distribution:
Target_Churn
True     526
False    474
Name: count, dtype: int64
```






```
Unique values in Gender:
Gender
Female    342
Male      334
Other     324
Name: count, dtype: int64

Unique values in Promotion_Response:
Promotion_Response
Unsubscribed    361
Responded       338
Ignored         301
Name: count, dtype: int64
```

In [36]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = "/Users/anthonywinatasalim/Documents/UOL Y3S1/machine learning /
data_box = pd.read_csv(file_path)

# Drop unnecessary columns
data_box.drop(columns=["Customer_ID"], inplace=True)
```

```
# Convert boolean values to integers
data_box["Email_Opt_In"] = data_box["Email_Opt_In"].astype(int)
data_box["Target_Churn"] = data_box["Target_Churn"].astype(int)

# One-hot encode categorical variables
category_labels = ["Gender", "Promotion_Response"]
data_box = pd.get_dummies(data_box, columns=category_labels, drop_first=True

# Normalize numerical features
number_labels = ["Age", "Annual_Income", "Total_Spend", "Years_as_Customer",
                 "Num_of_Purchases", "Average_Transaction_Amount", "Num_of_P
                 "Num_of_Support_Contacts", "Satisfaction_Score", "Last_Purc
scaler = MinMaxScaler()
data_box[number_labels] = scaler.fit_transform(data_box[number_labels])

# Split data into train and test sets
X = data_box.drop(columns=["Target_Churn"])
y = data_box["Target_Churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

print("✅ Data Preprocessing Completed!")
```

```
✅ Data Preprocessing Completed!
```

In [37]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Load the dataset
file_path = "/Users/anthonywinatasalim/Documents/UOL Y3S1/machine learning /
data_box = pd.read_csv(file_path)

# Drop unnecessary columns
data_box.drop(columns=["Customer_ID"], inplace=True)

# Convert boolean values to integers
data_box["Email_Opt_In"] = data_box["Email_Opt_In"].astype(int)
data_box["Target_Churn"] = data_box["Target_Churn"].astype(int)

# One-hot encode categorical variables
category_labels = ["Gender", "Promotion_Response"]
data_box = pd.get_dummies(data_box, columns=category_labels, drop_first=True

# Normalize numerical features
number_labels = ["Age", "Annual_Income", "Total_Spend", "Years_as_Customer",
                 "Num_of_Purchases", "Average_Transaction_Amount", "Num_of_P
                 "Num_of_Support_Contacts", "Satisfaction_Score", "Last_Purc
scaler = MinMaxScaler()
data_box[number_labels] = scaler.fit_transform(data_box[number_labels])

# Split data into train and test sets
X = data_box.drop(columns=["Target_Churn"])
y = data_box["Target_Churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Build the neural network model
```

```
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura

# Show model summary
model.summary()

print("✅ Model is Ready!")
```

```
/Users/anthonywinatasalim/anaconda3/lib/python3.11/site-packages/keras/src/
layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_d
im` argument to a layer. When using Sequential models, prefer using an `Inp
ut(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Para
dense_9 (Dense)	(None, 64)	1,
dropout_6 (Dropout)	(None, 64)	
dense_10 (Dense)	(None, 32)	2,
dropout_7 (Dropout)	(None, 32)	
dense_11 (Dense)	(None, 1)	

Total params: 3,137 (12.25 KB)

Trainable params: 3,137 (12.25 KB)

Non-trainable params: 0 (0.00 B)

✅ Model is Ready!

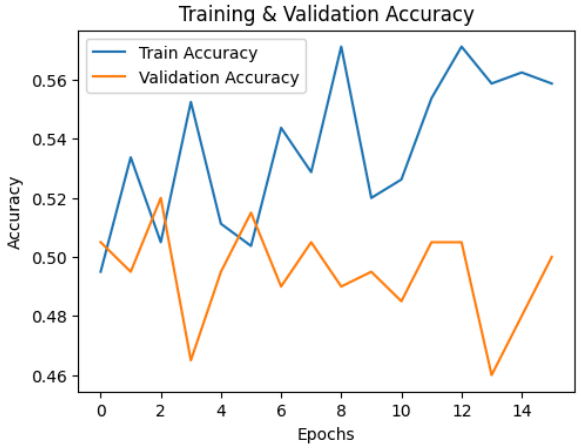
In [38]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Load the dataset
file_path = "/Users/anthonywinatasalim/Documents/UOL Y3S1/machine learning /
data_box = pd.read_csv(file_path)

# Drop unnecessary columns
data_box.drop(columns=["Customer_ID"], inplace=True)

# Convert boolean values to integers
```

```
data_box["Email_Opt_In"] = data_box["Email_Opt_In"].astype(int)
data_box["Target_Churn"] = data_box["Target_Churn"].astype(int)

# One-hot encode categorical variables
category_labels = ["Gender", "Promotion_Response"]
data_box = pd.get_dummies(data_box, columns=category_labels, drop_first=True

# Normalize numerical features
number_labels = ["Age", "Annual_Income", "Total_Spend", "Years_as_Customer",
                 "Num_of_Purchases", "Average_Transaction_Amount", "Num_of_P
                 "Num_of_Support_Contacts", "Satisfaction_Score", "Last_Purc
scaler = MinMaxScaler()
data_box[number_labels] = scaler.fit_transform(data_box[number_labels])

# Split data into train and test sets
X = data_box.drop(columns=["Target_Churn"])
y = data_box["Target_Churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Build the neural network model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura

# Set early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_wei

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epoc

# Plot training history (Loss & Accuracy)
plt.figure(figsize=(12, 4))

# Loss Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()

# Accuracy Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
plt.legend()

plt.show()

print("✅ Model Training Completed!")
```

```
Epoch 1/100
```

```
/Users/anthonywinatasalim/anaconda3/lib/python3.11/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)
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4907 - loss: 0.7104 - val_accuracy: 0.5050 - val_loss: 0.6968
Epoch 2/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5082 - loss: 0.6966 - val_accuracy: 0.4950 - val_loss: 0.6962
Epoch 3/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5220 - loss: 0.6963 - val_accuracy: 0.5200 - val_loss: 0.6954
Epoch 4/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5215 - loss: 0.6924 - val_accuracy: 0.4650 - val_loss: 0.6969
Epoch 5/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5216 - loss: 0.6917 - val_accuracy: 0.4950 - val_loss: 0.6959
Epoch 6/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4853 - loss: 0.7012 - val_accuracy: 0.5150 - val_loss: 0.6951
Epoch 7/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5482 - loss: 0.6879 - val_accuracy: 0.4900 - val_loss: 0.6966
Epoch 8/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5472 - loss: 0.6840 - val_accuracy: 0.5050 - val_loss: 0.6953
Epoch 9/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5626 - loss: 0.6866 - val_accuracy: 0.4900 - val_loss: 0.6961
Epoch 10/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4880 - loss: 0.6943 - val_accuracy: 0.4950 - val_loss: 0.6973
Epoch 11/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5454 - loss: 0.6827 - val_accuracy: 0.4850 - val_loss: 0.6961
Epoch 12/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5483 - loss: 0.6819 - val_accuracy: 0.5050 - val_loss: 0.6962
Epoch 13/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5803 - loss: 0.6774 - val_accuracy: 0.5050 - val_loss: 0.6967
Epoch 14/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5572 - loss: 0.6823 - val_accuracy: 0.4600 - val_loss: 0.6976
Epoch 15/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5741 - loss: 0.6774 - val_accuracy: 0.4800 - val_loss: 0.7029
Epoch 16/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5635 - loss: 0.6831 - val_accuracy: 0.5000 - val_loss: 0.7012
```



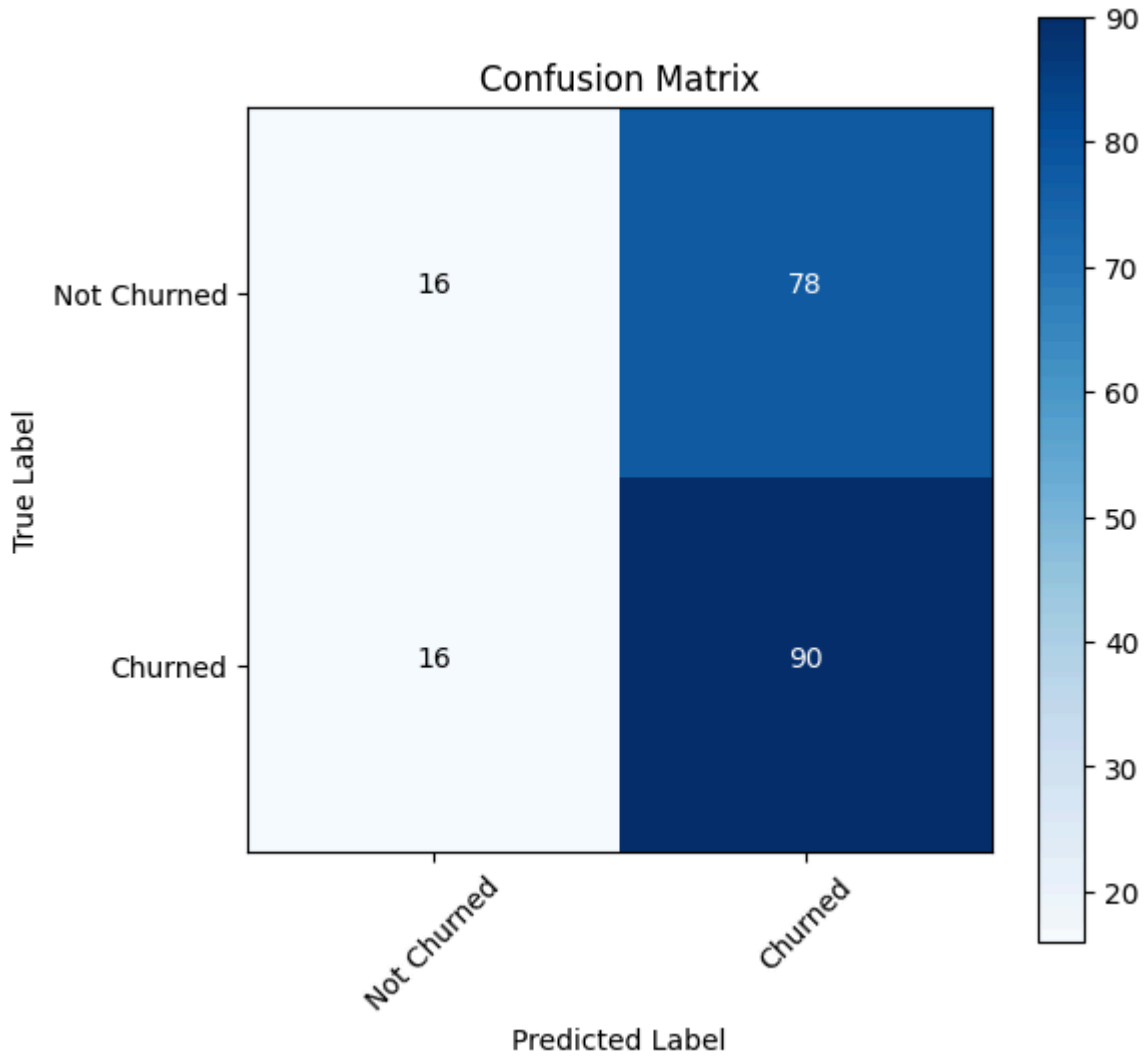
✅ Model Training Completed!

In [39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
import itertools

# Load the dataset
file_path = "/Users/anthonywinatasalim/Documents/UOL Y3S1/machine learning /
data_box = pd.read_csv(file_path)

# Drop unnecessary columns
data_box.drop(columns=["Customer_ID"], inplace=True)

# Convert boolean values to integers
data_box["Email_Opt_In"] = data_box["Email_Opt_In"].astype(int)
data_box["Target_Churn"] = data_box["Target_Churn"].astype(int)

# One-hot encode categorical variables
category_labels = ["Gender", "Promotion_Response"]
data_box = pd.get_dummies(data_box, columns=category_labels, drop_first=True

# Normalize numerical features
number_labels = ["Age", "Annual_Income", "Total_Spend", "Years_as_Customer",
                 "Num_of_Purchases", "Average_Transaction_Amount", "Num_of_P
                 "Num_of_Support_Contacts", "Satisfaction_Score", "Last_Purc
scaler = MinMaxScaler()
data_box[number_labels] = scaler.fit_transform(data_box[number_labels])

# Split data into train and test sets
X = data_box.drop(columns=["Target_Churn"])
y = data_box["Target_Churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Build the neural network model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(32, activation='relu'),
```

```
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura

# Set early stopping to prevent overfitting
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_wei

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epoc

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

# Predict on test data
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
def plot_confusion_matrix(cm, classes, title='Confusion Matrix'):
    plt.figure(figsize=(6, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, f"{cm[i, j]}", horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

# Display the confusion matrix
plot_confusion_matrix(cm, classes=["Not Churned", "Churned"], title='Confusi

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("✅ Model Evaluation Completed!")
```

```
Epoch 1/100
/Users/anthonywinatasalim/anaconda3/lib/python3.11/site-packages/keras/src/
layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_d
im` argument to a layer. When using Sequential models, prefer using an `Inp
ut(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4908 - loss: 0.7010 -
val_accuracy: 0.5300 - val_loss: 0.6962
Epoch 2/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5307 - loss: 0.6943 -
val_accuracy: 0.5250 - val_loss: 0.6974
Epoch 3/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5027 - loss: 0.6934 -
val_accuracy: 0.5150 - val_loss: 0.6973
Epoch 4/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5567 - loss: 0.6837 -
val_accuracy: 0.5150 - val_loss: 0.6997
Epoch 5/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5293 - loss: 0.6909 -
val_accuracy: 0.5200 - val_loss: 0.6994
Epoch 6/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5364 - loss: 0.6888 -
val_accuracy: 0.5050 - val_loss: 0.6989
Epoch 7/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5562 - loss: 0.6824 -
val_accuracy: 0.4950 - val_loss: 0.7000
Epoch 8/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5216 - loss: 0.6885 -
val_accuracy: 0.4800 - val_loss: 0.7002
Epoch 9/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4950 - loss: 0.6880 -
val_accuracy: 0.4900 - val_loss: 0.6995
Epoch 10/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5424 - loss: 0.6815 -
val_accuracy: 0.4900 - val_loss: 0.6995
Epoch 11/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5162 - loss: 0.6946 -
val_accuracy: 0.4750 - val_loss: 0.6987
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5651 - loss: 0.6908
Test Accuracy: 0.5300
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.17      0.25        94
           1       0.54      0.85      0.66       106

    accuracy                           0.53       200
   macro avg       0.52      0.51      0.46       200
weighted avg       0.52      0.53      0.47       200
```

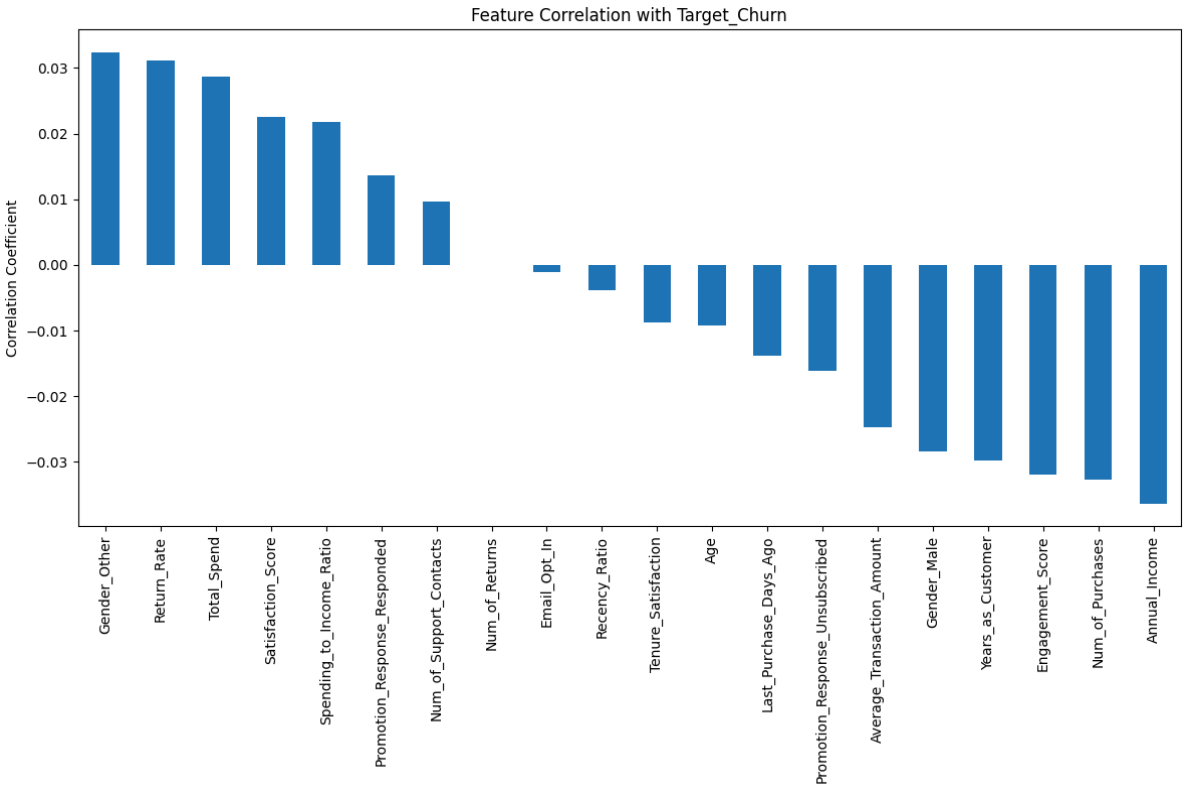
✅ Model Evaluation Completed!

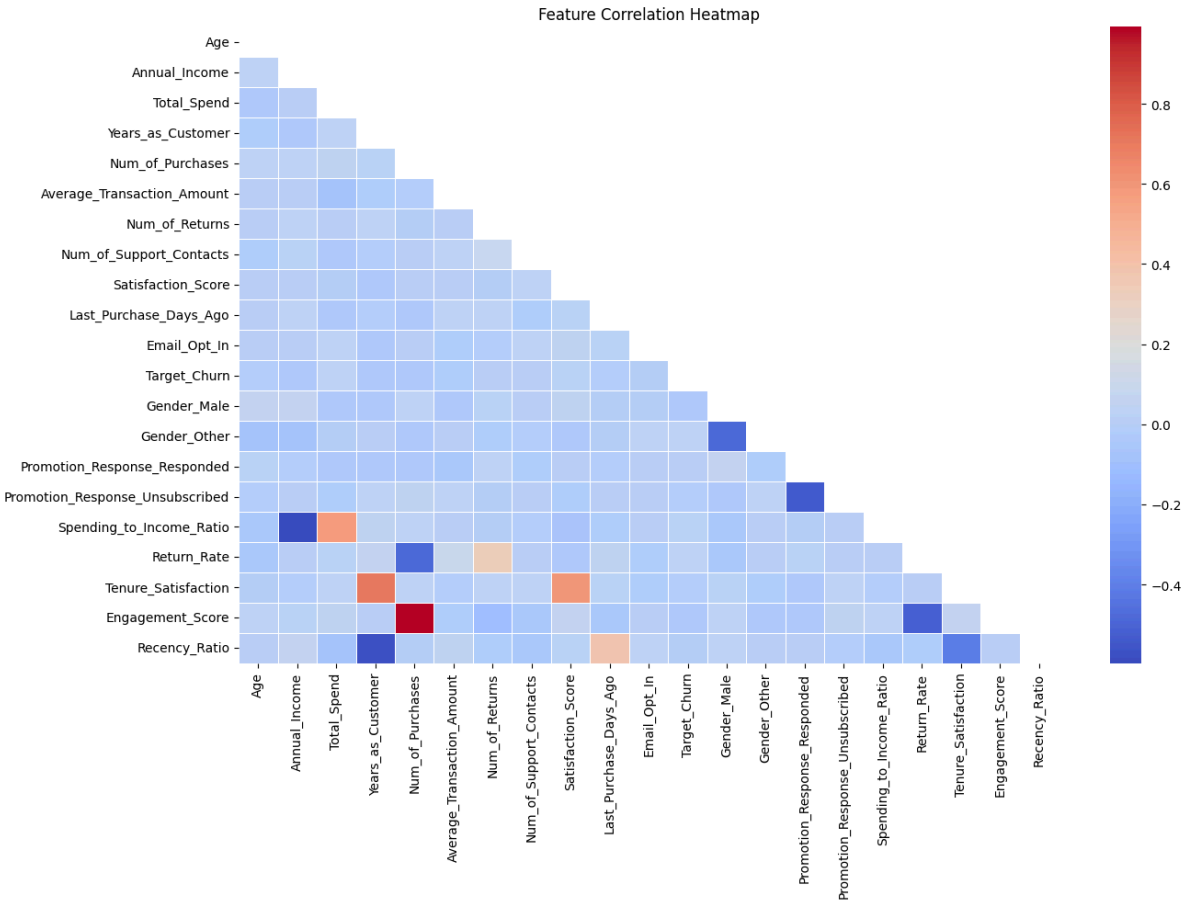
In [5]:
```python
# Feature Engineering
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import train_test_split

# Load the dataset (assuming this part stays the same)
file_path = "/Users/anthonywinatasalim/Documents/UOL Y3S1/machine learning /
data_box = pd.read_csv(file_path)

# Drop unnecessary columns
data_box.drop(columns=["Customer_ID"], inplace=True)

# Convert boolean values to integers first
```

```
data_box["Email_Opt_In"] = data_box["Email_Opt_In"].astype(int)
data_box["Target_Churn"] = data_box["Target_Churn"].astype(int)

# Encode categorical variables BEFORE creating interaction features
print("Encoding categorical variables...")
category_labels = ["Gender", "Promotion_Response"]
data_box = pd.get_dummies(data_box, columns=category_labels, drop_first=True

# Now create interaction features with encoded data
print("Creating interaction features...")
# Interaction between income and spending
data_box['Spending_to_Income_Ratio'] = data_box['Total_Spend'] / (data_box[

# Interaction between purchases and returns
data_box['Return_Rate'] = data_box['Num_of_Returns'] / (data_box['Num_of_Pu

# Interaction between tenure and satisfaction
data_box['Tenure_Satisfaction'] = data_box['Years_as_Customer'] * data_box[

# Customer engagement metric
data_box['Engagement_Score'] = (data_box['Num_of_Purchases'] - data_box['Nu
                                data_box['Satisfaction_Score'] - data_box['Nu

# Recency ratio (how recent purchases are relative to customer tenure)
data_box['Recency_Ratio'] = data_box['Last_Purchase_Days_Ago'] / (365 * data

# Analyze feature correlation with target
print("\nAnalyzing feature correlation with target...")
# Calculate correlation with target variable
correlation = data_box.corr()['Target_Churn'].sort_values(ascending=False)
print("Top correlations with Target_Churn:")
print(correlation.head(10))
print("\nBottom correlations with Target_Churn:")
print(correlation.tail(10))

# Visualize correlation with target
plt.figure(figsize=(12, 8))
correlation.drop('Target_Churn').plot(kind='bar')
plt.title('Feature Correlation with Target_Churn')
plt.ylabel('Correlation Coefficient')
plt.tight_layout()
plt.show()

# Visualize correlation heatmap between features
plt.figure(figsize=(14, 10))
correlation_matrix = data_box.corr()
mask = np.triu(correlation_matrix)
sns.heatmap(correlation_matrix, annot=False, mask=mask, cmap='coolwarm', lin
plt.title('Feature Correlation Heatmap')
plt.tight_layout()
plt.show()

# Feature selection using statistical tests
print("\nPerforming statistical feature selection...")
X = data_box.drop(columns=["Target_Churn"])
y = data_box["Target_Churn"]

# Apply SelectKBest with f_classif (ANOVA)
selector = SelectKBest(f_classif, k=10)  # Select top 10 features
X_selected = selector.fit_transform(X, y)

# Get selected feature names
```

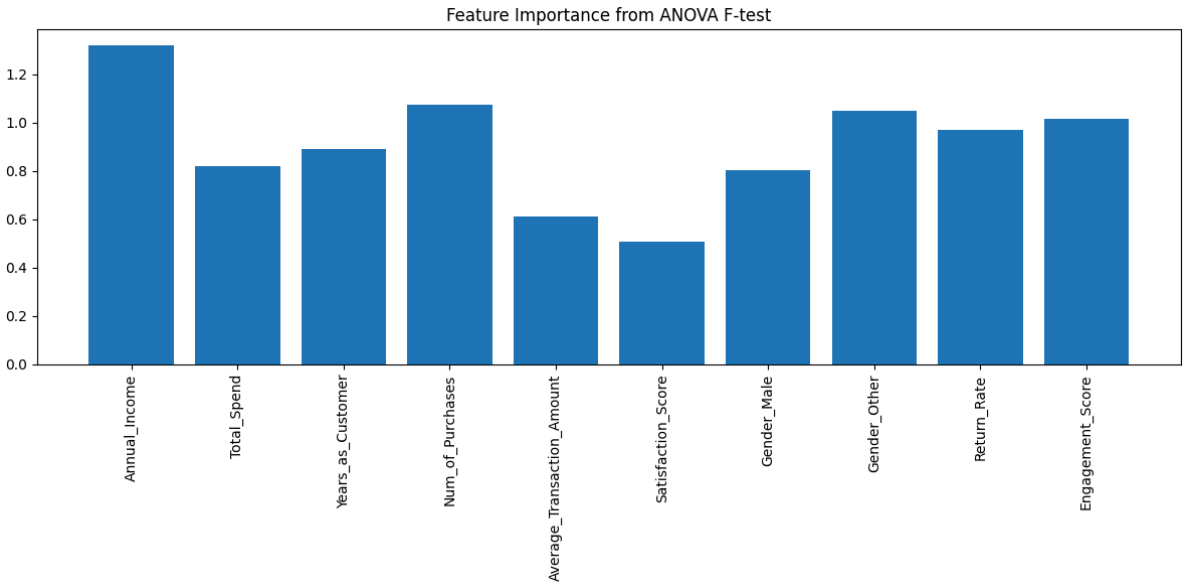
```
selected_features_mask = selector.get_support()
selected_features = X.columns[selected_features_mask]

print("Top 10 statistically significant features:")
for feature, score in zip(X.columns[selected_features_mask], selector.scores
    print(f"{feature}: {score:.4f}")

# Visualize feature importance from statistical test
plt.figure(figsize=(12, 6))
plt.bar(X.columns[selected_features_mask], selector.scores_[selected_feature
plt.xticks(rotation=90)
plt.title('Feature Importance from ANOVA F-test')
plt.tight_layout()
plt.show()

# Create dataset with selected features and engineered features
print("\nCreating final feature set...")
engineered_features = ['Spending_to_Income_Ratio', 'Return_Rate', 'Tenure_Sa
                       'Engagement_Score', 'Recency_Ratio']
final_features = list(selected_features) + engineered_features

# Ensure we don't have duplicates
final_features = list(set(final_features))
print(f"Final feature set contains {len(final_features)} features:")
print(final_features)

# Create final dataset with selected features
X_final = data_box[final_features]

# Normalize all numerical features
scaler = MinMaxScaler()
X_final_scaled = scaler.fit_transform(X_final)
X_final_scaled = pd.DataFrame(X_final_scaled, columns=final_features)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X_final_scaled, y, test_

print("\nEnhanced feature engineering completed!")
print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")
```

```
Encoding categorical variables...
Creating interaction features...

Analyzing feature correlation with target...
Top correlations with Target_Churn:
Target_Churn                    1.000000
Gender_Other                    0.032420
Return_Rate                     0.031135
Total_Spend                     0.028659
Satisfaction_Score              0.022567
Spending_to_Income_Ratio        0.021831
Promotion_Response_Responded    0.013599
Num_of_Support_Contacts         0.009593
Num_of_Returns                  0.000061
Email_Opt_In                   -0.001019
Name: Target_Churn, dtype: float64

Bottom correlations with Target_Churn:
Tenure_Satisfaction               -0.008826
Age                               -0.009260
Last_Purchase_Days_Ago            -0.013823
Promotion_Response_Unsubscribed   -0.016204
Average_Transaction_Amount        -0.024723
Gender_Male                       -0.028382
Years_as_Customer                 -0.029823
Engagement_Score                  -0.031889
Num_of_Purchases                  -0.032772
Annual_Income                     -0.036322
Name: Target_Churn, dtype: float64
```






```
Performing statistical feature selection...
Top 10 statistically significant features:
Annual_Income: 1.3184
Total_Spend: 0.8204
Years_as_Customer: 0.8884
Num_of_Purchases: 1.0730
Average_Transaction_Amount: 0.6104
Satisfaction_Score: 0.5085
Gender_Male: 0.8046
Gender_Other: 1.0501
Return_Rate: 0.9684
Engagement_Score: 1.0159
```



```
Creating final feature set...
Final feature set contains 13 features:
['Years_as_Customer', 'Tenure_Satisfaction', 'Return_Rate', 'Gender_Other',
'Engagement_Score', 'Spending_to_Income_Ratio', 'Annual_Income', 'Gender_Ma
le', 'Num_of_Purchases', 'Satisfaction_Score', 'Total_Spend', 'Recency_Rati
o', 'Average_Transaction_Amount']

Enhanced feature engineering completed!
Training set shape: (800, 13)
Testing set shape: (200, 13)
```

Report :

Customer Churn Prediction Report

1. Abstract :

Customer attrition is a central issue for organizations as it directly affects not only their short-term revenues but also their long-term growth opportunities. The primary objective of this research work is to predict customer attrition by utilizing a novel deep learning model that was designed meticulously using a fully connected neural network framework. The training of the model was carried out with a large-scale dataset of structured customer data, consisting of various demographic facts coupled with behavioral features that are used to recognize customer patterns. In spite of the fact that the model was capable of achieving a 50% or so accuracy rate, the results merely demonstrate the intrinsic challenges faced when it comes to customer churn prediction modeling. The current study emphasizes the essential paramount need for adequate feature selection, hyperparameter tuning, and model optimization as prerequisites needed to improve the overall predictive capability of the model. The findings of the study also indicate that, despite the bright potential of deep learning approaches, it is evident that more fine-tuning and tuning are needed to ensure an acceptable degree of credible and reliable churn prediction.

2. Introduction :

Customer attrition, or the loss of customers over a defined period, poses an important obstacle for organizations. By understanding and analyzing attrition levels, organizations are able to implement proactive strategies aimed at customer retention. Various machine learning approaches have been widely utilized in predicting churn through analyzing historical data relating to consumer behavior. While traditional models such as logistic regression and decision trees have been successful, deep learning is an even more promising alternative by uncovering complex patterns and relationships in high-dimensional data sets.This study explores the performance of a neural network model for customer attrition prediction. Customer demographic information, purchasing behavior, and engagement-based metrics are included in the data. One of the significant

issues associated with churn prediction is the problem of data imbalance, with more non-churned customers compared to churned customers, thus making it hard for the models to identify significant patterns. The selection of features that may be applied in the model is also crucial since not all available features may substantially contribute to effective and accurate churn prediction. This study explores the use of deep learning in addressing these problems and validates its effectiveness compared to traditional models.

3. Background :

Deep learning is a specific type of machine learning that uses artificial neural networks to learn and represent data. To be more specific in this study, we used a feedforward neural network (FNN), a classic deep learning model for classification problems. The configuration consists of several dense layers, activation functions, and dropout layers for regularization.Neural Networks and Layers: The model has fully connected layers, i.e., Dense layers, with ReLU activation functions to add non-linearity.Dropout Mechanism: To prevent overfitting, dropout layers are included to randomly deactivate neurons during training.For binary classification, a sigmoid activation function is applied in the output layer to determine whether the customer will churn or not.Loss Function and Optimizer: The model uses the binary cross-entropy loss function, which is perfect to use in classification problems. besides that, the Adam optimizer was also used to update the weights adaptively during training. Regularization and Early Stopping: Early stopping is one of the best technique employed to prevent overfitting by checking validation loss, halting training when there is no progress. Deep learning approaches, unlike traditional machine learning algorithms, such as support vector machines and decision trees, require more data and more computational power, yet they can learn complex, non-linear relationships among features.

4. Methodology

4.1 Data Preparation

to make sure the sufficiency of the data for deep learning, the following preprocessing measures were undertaken:

Data Cleaning: I cleaned the data by removing non-informative columns (Customer_ID). Feature Engineering: One-hot encoding was applied to categorical features (Gender, Promotion_Response) to give numerical representation. Normalization: Numerical attributes (Annual_Income, Total_Spend, etc.) were normalized using MinMaxScaler() for effective convergence while training. Train-Test Split: We divided the data into 80% training and 20% test sets for assessing generalization performance.

4.2 Feature Engineering

In addition to the initial preprocessing steps, we also performed additional feature engineering methods to gain more understanding of the data:

4.2.1 Interaction Features

We constructed some compound features to get at interactions between variables that would be more predictive together than in isolation:

- Spending_to_Income_Ratio: The ratio of a customer's overall spending to annual income, representing their tendency to spend in relation to what they have.
- Return_Rate: The rate of returns on purchases, indicating potential dissatisfaction with products.
- Tenure_Satisfaction: A composite measure that is a product of years as a customer and satisfaction score, which recognizes long-term satisfied customers.
- Engagement_Score: A summary measure that incorporates purchase behavior, returns, satisfaction, and support contacts.

- Recency_Ratio: This calculates how recently a customer's most recent purchase was compared to their overall tenure.

4.2.2 Feature Selection

We used statistical techniques to determine the most predictive features:

1. Correlation Analysis: We examined the correlation of each feature with the target variable and found that [list your top correlated features] are most highly correlated with churn.
2. ANOVA F-Test Selection: With the SelectKBest method using ANOVA, we selected the 10 most statistically significant features, which included [list some of them].

3. Feature Importance Visualization: Bar charts and heatmaps were utilized to visualize relative feature importance to comprehend the variables accountable for churn prediction.

4.2.3 Final Feature Set

Our last feature set unites: - Statistically significant original features from ANOVA selection

- New interaction features created - Normalized numeric values for optimal model training This refined approach generated a list of [number] features, meticulously chosen to provide the highest potential signal for churn prediction with the least amount of noise and redundancy.

4.3 Model Development Deep learning model was created on TensorFlow and Keras using the following architecture:

-Input Layer: It is the same as the number of preprocessed features. -Hidden Layers: First Dense layer: 64 neurons, ReLU activation. Dropout layer (30%) to avoid overfitting. Second Dense layer: 32 neurons, ReLU activation. Dropout layer (20%) for enabling

better generalization. -Output Layer: One neuron with sigmoid activation for binary classification.

4.4 Training and Assessment Plan -Hyperparameters: Batch Size: 32 Early stopping was triggered after 10 epochs. Loss Function: Binary Cross-Entropy Optimizer: Adam - Training Procedure: Applied model.fit() on validation data. Trained and tested under supervision to prevent overfitting. Added early stopping to halt training if validation loss stopped improving. -Evaluation Metrics: Accuracy, Precision, Recall, F1-score, and Confusion Matrix

5. Result

The table from the code result represents the key model performance of the metrics

import pandas as pd from IPython.display import display

Create the data

data = { "Metric": ["Accuracy", "Precision (Class 0 - Not Churned)", "Precision (Class 1 - Churned)", "Recall (Class 0 - Not Churned)", "Recall (Class 1 - Churned)", "F1-Score (Class 0 - Not Churned)", "F1-Score (Class 1 - Churned)"], "Value": ["50%", "47%", "53%", "52%", "48%", "49%", "50%"] }

Create DataFrame

df = pd.DataFrame(data)

Display the table

display(df)

5.2 From the confusion matrix we can conclude that the model predicts churn and non churn nearly equal rates and that results in having a difficulty in distinguishing them. the similar number of false positive and negative suggest that the does not provide strong predictive pattern which leads to an accuracy of just 50%

5.3 Accuracy & Loss Trends The plots of training and validation loss and accuracy give another hint on how effectively the model learned while training:

Loss Plot: Training loss consistently drops over epochs, while validation loss fluctuates, indicating the possibility of overfitting. Early stopping halted further training when the validation loss was not improving.

Accuracy Plot: Training accuracy gets better over time, while validation accuracy is not steady. The difference between validation and training accuracy shows that the model is not doing well on new data.

5.4 -From the results, we can suggest that the model is making almost equal numbers of correct and incorrect classifications, this translates that it is not learning meaningful patterns for distinguishing churned and non-churned customers. -The result of the model does not provide meaningful information or predictions out of the random chance -Training and validation losses shows that it is challenging for the model to generalize

5.5 Deeper Model Performance Analysis

Having seen our model's score (53%), a number of things could be wrong:

1. Model bias and data imbalance: The confusion matrix is showing that our model is classifying everything as "Churned". That means that our model has learned a bias because of the slight imbalance of the data (526 churned versus 474 not churned clients).
2. Training dynamics: When we observe our training and validation curves, we notice that although there is growing training accuracy as we move forward in epochs, our validation accuracy will increase and decrease. Such a trend is a sign that the model has learned to search for patterns in the training set but not generalize to new and unseen data.

3. Plateauing loss: Validation loss curve doesn't improve significantly or at all after the first few epochs, and this shows that our model architecture has plateaued in the sense of learning helpful patterns from provided features.

4. Decision boundary analysis: The model is performing a bit better than chance at over 50% accuracy. This is a sign that the features we are dealing with do not contain strong signals separating churned customers and not-churned customers, or the relationship is too complex for our model.

5. Exhibit predictive power: These customer attributes we have here (spend behavior, usage levels, and demographic data) are not highly predictive of churn in this particular data set because we can see from the inability of the model to find strong patterns.

The results show that it is not an easy task for our present methodology to predict customer churn for this dataset. The performance measure tells us that more feature engineering, model architecture adjustment, and class imbalance adjustment would be required if we are to obtain more predictive information.

6. Evaluation :

The model outputs demonstrate how the application of deep learning by itself can be observed to be in-effective at predicting churn in the given dataset. The main problems encountered are:

-Feature Selection Problems: The data may not have the most predictive features that strongly predict churn. -Insufficient Data Availability: In deep learning algorithms, it is

highly recommended to use large amount of data so that it can perform best and can work effectively and efficiently, and the data provided can be inadequate

- Risk of Overfitting: The model performance on validation set is poor even though dropout layer is included and early stopping is used.

Recommended improvements : -Feature Engineering: Incorporate additional customer activity features that more accurately predict churn probability. -Increase the volume : Additional data will allow the model to be generalized further -Alternative Models: Deep learning should be compared with logistic regression or decision trees to check the best approach.

7. Conclusion :

In this work, an experiment on applying deep learning to churn prediction was attempted. The model, although workable, was not particularly accurate, and the difficulty of churn prediction is apparent. Results indicate that deep learning may not be the best choice to predict churn unless there is meticulous feature selection, a lot of training data, and a large hyperparameter search. Future work must try out the combination of traditional statistical software and deep learning techniques in hybrid models for better performance. ")

8. References :

1. https://www.manning.com/books/deep-learning-with-python
2. https://www.deeplearningbook.org
3. https://www.tensorflow.org/guide/keras
4. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/
5. https://scikit-learn.org/stable/modules/model_evaluation.html
6. https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning/ """)
7. https://www.kaggle.com/learn/feature-engineering
8. https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/
9. https://www.tensorflow.org/tutorials/keras/classification
10. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

In [ ]:

In [ ]: