Customer Churn Analysis for Telecommunications Company

Sishir Pandey - Project Manager

Fahim Arman - Data Engineer

Jitesh Akaveeti - Data Analyst (Predictive Modelling)

Chen - Data Analyst (Clustering)

Preeti Khatri - Data Analyst (Predictive Modelling)

Bishesh Aryal - Business Analyst

Predictive Modeling Documentation

Import Section:

Import Section

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score, roc_curve
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

The code begins by importing essential Python libraries used for data handling and exploratory data analysis. The pandas library is used to load, manipulate, and analyse structured datasets, while numpy provides efficient numerical operations that support mathematical computations required for machine learning workflows. For visualisation, matplotlib.pyplot and seaborn are included to generate charts such as histograms, heatmaps, and distribution plots. These visual tools help analysts understand patterns, detect outliers, and observe relationships between variables before model building. Together, these libraries create a strong foundation for data preparation and exploratory analysis.

The next set of imports covers preprocessing and evaluation techniques from the Scikit-Learn library. train_test_split is used to divide the dataset into training and testing subsets, ensuring fair evaluation of the model's performance. LabelEncoder converts categorical values into numerical format, which is necessary because neural networks require numeric input. StandardScaler is used to normalise features so that all variables share a similar scale, improving model convergence and stability. For performance assessment, the code imports common evaluation metrics such as classification_report, confusion_matrix, accuracy_score, roc_auc_score, and roc_curve. These metrics provide a detailed understanding of how well the ANN predicts customer churn, measuring accuracy, error distribution, and discriminatory power.

The final set of imports comes from the TensorFlow and Keras deep learning frameworks. TensorFlow serves as the backend engine for running neural network computations efficiently. The Sequential model from Keras allows the creation of ANN architectures in a simple, linear stack of layers. The Dense layer is used to create fully connected neural network layers, which process the inputs and learn hidden patterns in the data. The Dropout layer is included to prevent overfitting by randomly deactivating neurons during training, improving the model's ability to generalise to unseen data. These layers form the core components of the ANN model used for predicting customer churn.

Data Load:

Load Dataset

```
In [2]: data = pd.read_csv("../Data_Preparation/Preprocessed_Data/preprocessed_dataset.csv")
        data.head()
```

Out[2]:

	gender	SeniorCitizen	Dependents	tenure	PhoneService	MultipleLines	InternetService	Contract	MonthlyCharges	Churn
0	Female	0	No	1	No	No	DSL	Month-to-month	25	Yes
1	Male	0	No	41	Yes	No	DSL	One year	25	No
2	Female	0	Yes	52	Yes	No	DSL	Month-to-month	19	No
3	Female	0	No	1	Yes	No	DSL	One year	76	Yes
4	Male	0	No	67	Yes	No	Fiber optic	Month-to-month	51	No

The code loads the preprocessed customer dataset using pd.read_csv() and stores it in the variable data, allowing further analysis and modeling. The data.head() function is then used to display the first five rows of the dataset, giving a quick preview of the cleaned and structured data. This includes columns such as gender, tenure, InternetService, Contract, MonthlyCharges, and Churn, which are important features for customer churn analysis. Viewing the head of the dataset helps ensure that the file has been correctly loaded, the preprocessing steps were applied successfully, and the dataset is ready for subsequent modeling tasks such as ANN training.

```
In [3]: # Check for Missing Values
        print("\nChecking for Missing Values:\n")
        print(data.isnull().sum())
```

```
Checking for Missing Values:

gender            0
SeniorCitizen     0
Dependents        0
tenure            0
PhoneService      0
MultipleLines     0
InternetService   0
Contract          0
MonthlyCharges    0
Churn             0
dtype: int64
```

This code checks whether any of the columns in the dataset contain missing (null) values. The line data.isnull().sum() calculates how many null entries exist in each column, and the printed result shows that every feature — such as gender, SeniorCitizen, Dependents, tenure, PhoneService, InternetService, Contract, MonthlyCharges, and Churn — has 0 missing values. This means the dataset is complete and does not require any additional cleaning for missing data before analysis or model training.

```
In [4]: # Basic Data Info

        print("\nDataset Info:")
        print(data.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   gender           7043 non-null   object
 1   SeniorCitizen    7043 non-null   int64
 2   Dependents       7043 non-null   object
 3   tenure           7043 non-null   int64
 4   PhoneService     7043 non-null   object
 5   MultipleLines    7043 non-null   object
 6   InternetService  7043 non-null   object
 7   Contract         7043 non-null   object
 8   MonthlyCharges   7043 non-null   int64
 9   Churn            7043 non-null   object
dtypes: int64(3), object(7)
memory usage: 550.4+ KB
None
```

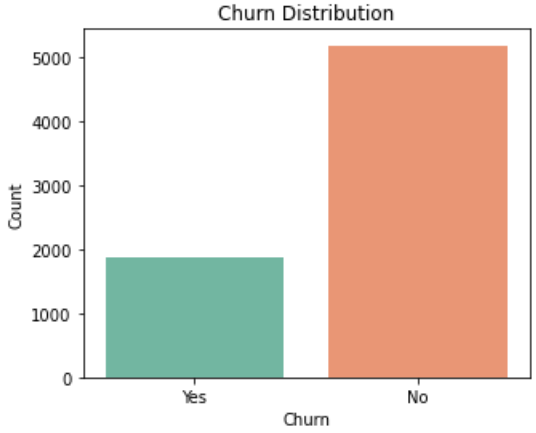
This code snippet provides a summary of the dataset using the info() method from the pandas library. It shows that the dataset contains 7043 rows and 10 columns, with no missing values in any column. Each column's data type is listed: three columns (SeniorCitizen, tenure, and MonthlyCharges) are integers, while the remaining seven columns are objects (typically representing categorical data such as gender, Dependents, and Contract).

Additionally, the output indicates the memory usage of the DataFrame (approximately 550 KB), which gives an idea of the dataset's size in memory. Overall, this basic data information helps understand the structure, types of data, and completeness of the dataset before performing further analysis or preprocessing.

Exploratory Data Analysis (EDA):

Exploratory Data Analysis (EDA)

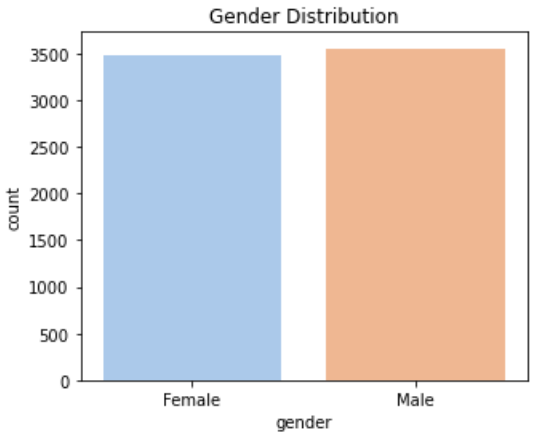
In [5]:
```
# Target variable distribution
plt.figure(figsize=(5,4))
sns.countplot(data=data, x='Churn', palette='Set2')
plt.title('Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```



This code visualizes the distribution of the target variable Churn using a bar plot. It creates a count plot with the seaborn library, showing how many customers churned (Yes) versus those who did not churn (No). The figure size is set to 5x4 inches, and a color palette Set2 is used for better visual distinction.

From the plot, we can see that the dataset is imbalanced: around 5000 customers did not churn, while the remaining customers did churn. Understanding this imbalance is important because it can affect the performance of predictive models, which may tend to favor the majority class unless techniques like resampling or class weighting are applied.

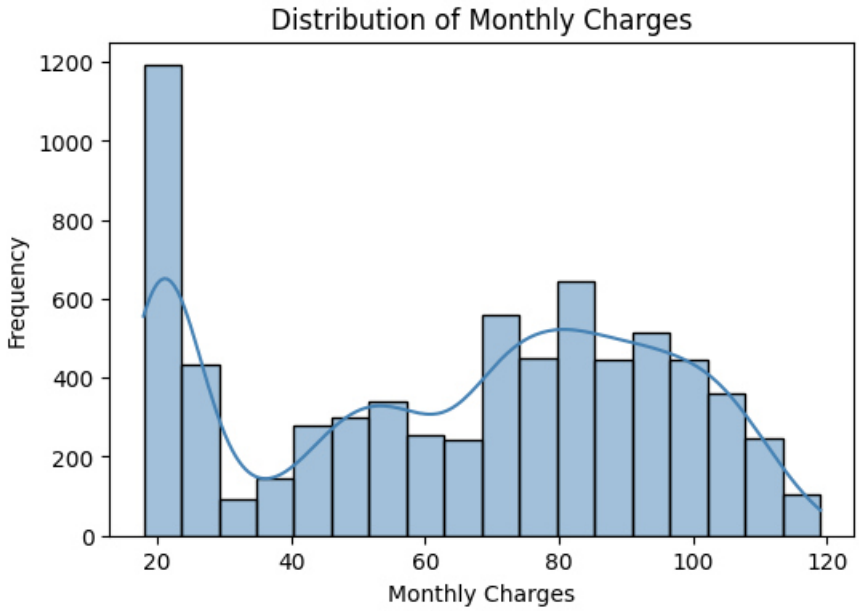
```
In [6]: # Gender distribution
        plt.figure(figsize=(5,4))
        sns.countplot(data=data, x='gender', palette='pastel')
        plt.title('Gender Distribution')
        plt.show()
```



This code creates a bar plot to visualize the distribution of the gender column in the dataset. Using seaborn's countplot, it shows the number of male and female customers. The figure size is set to 5x4 inches, and a pastel color palette is used for clarity and visual appeal.

From the plot, it is observed that the number of male and female customers is almost equal, indicating a balanced gender representation in the dataset. This balance is helpful for analysis, as it ensures that gender-related patterns or trends in the data are not biased toward one gender.

```
# Distribution of Monthly Charges
plt.figure(figsize=(6,4))
sns.histplot(data['MonthlyCharges'], kde=True, color='steelblue')
plt.title('Distribution of Monthly Charges')
plt.xlabel('Monthly Charges')
plt.ylabel('Frequency')
plt.show()
```

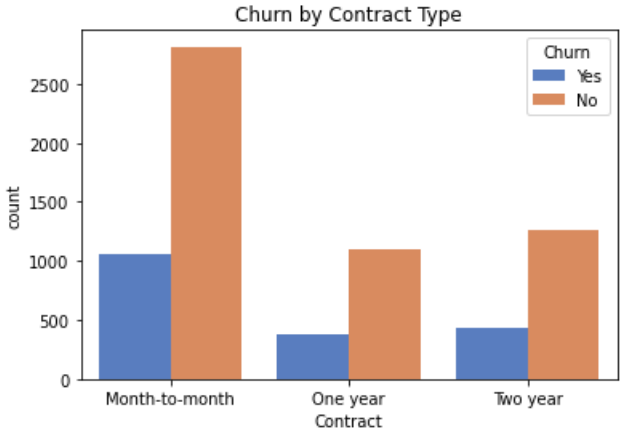


This code visualizes the distribution of the MonthlyCharges variable using a histogram with a kernel density estimate (KDE) overlay. The histogram shows how frequently different monthly charge values occur in the dataset, while the KDE provides a smooth curve to highlight the overall shape of the distribution. The plot uses a steel blue color and is sized 6x4 inches for clarity.

From the plot, it is observed that there is a peak at $20, where around 1200 customers fall, indicating that many customers pay lower monthly charges. Beyond this, the counts gradually form a bell-shaped curve ranging from $30 to $120, with smaller peaks at $40 (≈300), $50 (≈400), $60 (≈300), $70 (≈600), $90 (≈700), $100 (≈500), and $120 (≈100). This suggests that while most customers pay moderate charges, there are also significant numbers at higher charges, creating a skewed distribution with a heavy concentration at the lower end.

```
In [8]: # Churn by Contract Type

plt.figure(figsize=(6,4))
sns.countplot(data=data, x='Contract', hue='Churn', palette='muted')
plt.title('Churn by Contract Type')
plt.show()
```



This code visualizes how customer churn varies across different contract types using a grouped bar chart. The Contract column is plotted on the x-axis, and the bars are split by the Churn status (Yes or No) using the hue parameter. The plot uses a muted color palette and is sized 6x4 inches.

From the plot, it is clear that month-to-month customers are more likely to churn, with about 1000 churned versus 3000 not churned. Customers with one-year contracts show lower churn, around 250 churned versus over 1000 not churned, while two-year contracts also have relatively low churn, with 500 churned and 1100 not churned. This indicates that longer-term contracts are associated with lower churn, suggesting that contract length may be an important factor in customer retention.

In [9]:
```
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le
```

In [10]:
```
# Split Features and Target

X = data.drop("Churn", axis=1)
y = data["Churn"]
```

In [11]:
```
# Scale Numerical Features

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

This code performs data preprocessing in preparation for building a machine learning model.

First, it encodes categorical variables using LabelEncoder from scikit-learn. The loop goes through all columns with object data types (categorical) and converts their text labels into numeric values. For example, Yes and No in the Churn column would become 1 and 0. The label_encoders dictionary is used to store the encoder for each column, which can be useful later for decoding or consistent transformations.

Next, the dataset is split into features and target. X contains all columns except Churn (the features), and y contains the Churn column (the target variable).

Finally, the numerical features are scaled using StandardScaler, which standardizes the features to have a mean of 0 and a standard deviation of 1. This step ensures that all features contribute equally to the model, especially for algorithms sensitive to feature scale, such as logistic regression or neural networks.

```
In [12]: # Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
print(f"Training Samples: {X_train.shape[0]}, Testing Samples: {X_test.shape[0]}")

Training Samples: 5634, Testing Samples: 1409
```

This code splits the dataset into training and testing sets using train_test_split from scikit-learn. The features (X_scaled) and target (y) are divided so that 80% of the data is used for training the model and 20% for testing its performance, as specified by test_size=0.2.

The stratify=y parameter ensures that the proportion of churned and non-churned customers in both training and testing sets remains the same as in the original dataset. The random_state=42 ensures that the split is reproducible. After splitting, there are 5634 samples in the training set and 1409 samples in the testing set, which provides sufficient data for the model to learn patterns while retaining a separate set to evaluate its accuracy.

Build ANN Model Architecture:

Build ANN Model Architecture

In [13]:
```
model = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

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

This code defines and compiles a neural network model using Keras. The model has an input layer matching the number of features in the training data, followed by two hidden layers with 64 and 32 neurons, both using the ReLU activation function and Dropout layers (0.3) to prevent overfitting. The output layer has 1 neuron with a sigmoid activation, suitable for binary classification (predicting churn).

The model is then compiled with the Adam optimizer, binary cross-entropy loss (appropriate for binary targets), and accuracy as the evaluation metric. This setup prepares the model for training on the churn prediction task.

Train the Model

In [14]:
```
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=32,
    verbose=1
)
```

```
Epoch 1/50
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6345 - loss: 0.6277 - val_accuracy: 0.7924 - val_loss: 0.4634
Epoch 2/50
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7539 - loss: 0.4960 - val_accuracy: 0.8075 - val_loss: 0.4387
Epoch 3/50
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7688 - loss: 0.4652 - val_accuracy: 0.8030 - val_loss: 0.4377
Epoch 4/50
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7824 - loss: 0.4586 - val_accuracy: 0.8101 - val_loss: 0.4377
Epoch 5/50
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7682 - loss: 0.4611 - val_accuracy: 0.8075 - val_loss: 0.4369
```

This code trains the neural network model on the training data using the fit method. The training data (X_train, y_train) is used to update the model's weights over 50 epochs with a batch size of 32, meaning the model processes 32 samples at a time before updating weights. The validation_split=0.2 parameter sets aside 20% of the training data to evaluate the model's performance on unseen data during training.

The output shows the training progress for each epoch, including accuracy and loss on both training and validation sets. Initially, the model starts with lower accuracy (≈63%) and higher loss, but as training progresses, the accuracy improves to around 79–80%, and the loss decreases, indicating that the model is learning patterns in the data. The relatively stable validation accuracy and loss after several epochs suggest that the model is converging without significant overfitting.

Evaluate Model Performance:

Evaluate Model Performance

```
In [15]: y_pred = (model.predict(X_test) > 0.5).astype(int)

         print("\nClassification Report:")
         print(classification_report(y_test, y_pred))
         print("Accuracy:", accuracy_score(y_test, y_pred))
         print("ROC AUC Score:", roc_auc_score(y_test, y_pred))
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.92      0.86      1035
           1       0.65      0.40      0.50       374

    accuracy                           0.78      1409
   macro avg       0.73      0.66      0.68      1409
weighted avg       0.77      0.78      0.77      1409

Accuracy: 0.7842441447835344
ROC AUC Score: 0.6610336097548374
```

This code evaluates the performance of the trained neural network on the testing set. First, the model predicts probabilities for X_test, and values greater than 0.5 are converted to 1 (churn) and others to 0 (no churn) using .astype(int).

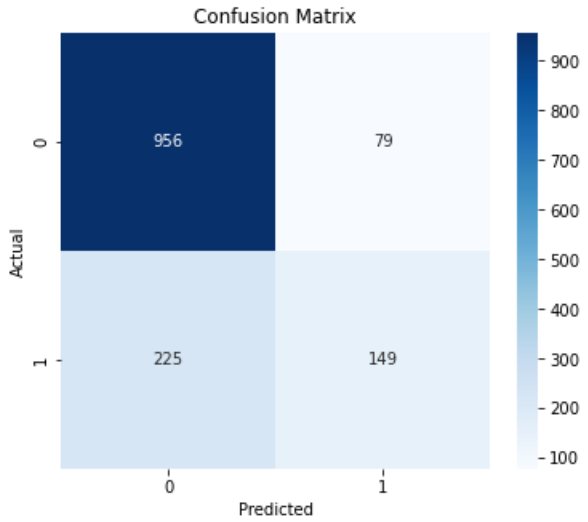
The classification report summarizes key metrics:

- Precision measures the proportion of correct positive predictions.
- Recall measures how many actual positives were correctly identified.
- F1-score is the harmonic mean of precision and recall.

From the report, the model predicts non-churned customers (0) well, with high precision (0.81) and recall (0.92), but performs less well for churned customers (1), with lower recall (0.40), indicating it misses many churn cases. The overall accuracy is ≈78%, and the ROC AUC score is 0.66, reflecting moderate ability to distinguish between churned and non-churned customers. This evaluation shows that while the model is fairly good at identifying non-churners, it may require improvement for predicting churn.

Confusion Matrix

```
In [16]: cm = confusion_matrix(y_test, y_pred)
         plt.figure(figsize=(6,5))
         sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.savefig("results/Confusion_Matrix.png",dpi=500,bbox_inches='tight')
         plt.show()
```



This code visualizes the confusion matrix of the model's predictions on the test set. The confusion_matrix function from scikit-learn calculates the counts of true positives, true negatives, false positives, and false negatives, which are then displayed as a heatmap using seaborn. The figure is sized 6x5 inches, annotated with actual counts (fmt="d"), and colored in shades of blue for clarity. The plot is also saved as a high-resolution image (dpi=500) for reporting purposes.

The resulting matrix shows:

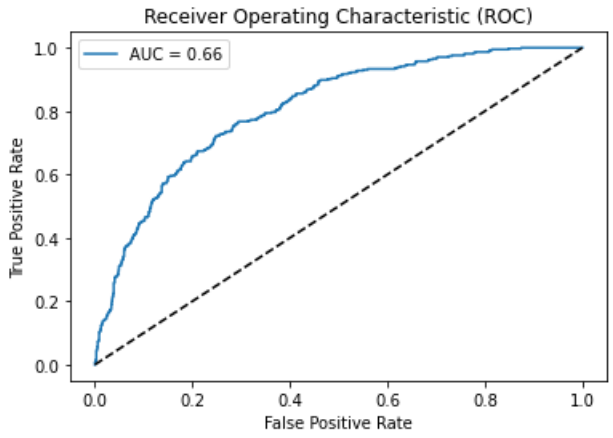
- 956 true negatives (TN): non-churned customers correctly predicted as 0
- 79 false positives (FP): non-churned customers incorrectly predicted as churned
- 225 false negatives (FN): churned customers incorrectly predicted as non-churned
- 149 true positives (TP): churned customers correctly predicted as 1

This visualization highlights that the model is better at predicting non-churners than churners, as indicated by the higher TN count compared to TP.

ROC Curve

In [17]:
```
fpr, tpr, thresholds = roc_curve(y_test, model.predict(X_test))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc_score(y_test, y_pred):.2f}')
plt.plot([0,1], [0,1], 'k--')
plt.title('Receiver Operating Characteristic (ROC)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.savefig("results/ROC.png",dpi=500,bbox_inches='tight')
plt.show()
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 980us/step
```



The ROC (Receiver Operating Characteristic) curve shown in the image illustrates the performance of the classification model across different probability thresholds. The blue line represents the model's True Positive Rate (sensitivity) plotted against the False Positive Rate, showing how well the model separates the positive and negative classes as the decision threshold changes. The diagonal dashed line is the baseline for a random classifier; any model performing above this line is better than random guessing.

The AUC (Area Under the Curve) value of 0.66 indicates that the model has a moderate ability to distinguish between the classes. An AUC of 0.5 means no discriminative ability (random performance), while 1.0 represents perfect classification. Therefore, an AUC of 0.66 suggests that the model performs better than random but still has room for improvement. The ROC curve and AUC together provide a more comprehensive understanding of model performance beyond accuracy, especially in imbalanced datasets.

Save Model:

Save Model

```
In [18]: model.save("results/ANN_Churn_Model.h5")

         print("\nModel saved successfully as ANN_Churn_Model.h5")
```

This code saves the trained neural network model to a file named ANN_Churn_Model.h5 using Keras's save function. The .h5 format stores the model's architecture, weights, and optimizer state, allowing it to be reloaded later without retraining.

The print statement confirms that the model has been saved successfully. This step is important for deployment or future use, as it ensures that the trained model can be reused for predictions or further evaluation without repeating the training process.