DSC 550 Week 8 Neural Network Solution

Make sure the project documentation contains a) problem statement, b) algorithm of the solution, c) analysis of the findings, and d) references.

Problem statement

Telco Customer Churn Prediction

1. Problem Statement

The objective of this project is to predict customer churn in a telecommunications company. By accurately identifying customers likely to churn, the company can implement targeted retention strategies, reducing the overall churn rate and increasing customer lifetime value.

Problem Formulation

Problem Statement

Predict customer churn in a telecommunications company using deep learning techniques. Specifically, the goal is to forecast the likelihood that a customer will terminate their service in the near future based on their usage patterns, service history, and demographic information. The target variable is binary, where `1` indicates a customer has churned and `0` indicates they have not.

Importance

Customer churn is a critical issue for telecommunications companies as it directly impacts revenue and profitability. High churn rates lead to increased customer acquisition costs and reduced customer lifetime value. By predicting churn, companies can proactively implement retention strategies, offer personalized incentives, and improve customer satisfaction, ultimately reducing churn rates and enhancing profitability.

Stakeholders

- **Business Executives**: Need insights to make strategic decisions regarding customer retention programs and marketing strategies.
- **Marketing Teams**: Require predictions to target high-risk customers with personalized offers.
- **Customer Service Departments**: Benefit from early identification of at-risk customers to provide timely interventions.

Approach

1. **Data Collection and Preprocessing**

    - **Data Collection**: Utilize the Telco dataset, which includes customer demographics, account information, service usage, and churn status.
    - **Data Cleaning**: Handle missing values, normalize numerical features, and encode categorical variables.
    - **Feature Engineering**: Create additional features that may improve model performance, such as tenure duration or interaction frequency.

2. **Model Development**

    - **Model Selection**: Use a Deep Neural Network (DNN) with several layers to capture complex patterns in the data. Consider architectures like feedforward neural networks and variations like dropout for regularization.
    - **Training**: Split the dataset into training and testing sets. Train the model using the training set while validating its performance on the testing set.
    - **Evaluation Metrics**: Assess model performance using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to ensure the model's effectiveness in predicting churn.

3. **Hyperparameter Tuning**

    - **Optimization**: Experiment with different hyperparameters, including learning rate, number of hidden layers, and activation functions, to find the optimal model configuration.

4. **Deployment**

    - **Model Deployment**: Deploy the trained model to a cloud platform such as AWS or Azure for real-time predictions and integration into customer relationship management (CRM) systems.
    - **Monitoring**: Implement monitoring to track model performance over time and update the model as needed.

Software Tools

1. **Programming Languages**

    - **Python**: For data preprocessing, model building, and evaluation.

2. **Libraries and Frameworks**

- **TensorFlow/Keras**: To build and train the deep learning model.
- **pandas**: For data manipulation and preprocessing.
- **NumPy**: For numerical operations.
- **Scikit-learn**: For additional metrics and preprocessing tools.

3. **Cloud Platforms**

   - **AWS/Azure**: For deploying the model and performing real-time predictions.

4. **Development Environment**

   - **Jupyter Notebook**: For documenting and executing code, visualizing data, and presenting results.

This approach ensures a comprehensive solution to the problem of predicting customer churn, leveraging deep learning techniques to deliver actionable insights and drive strategic business decisions.

Data Description

Dataset

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

import warnings

# Ignore warnings
warnings.filterwarnings("ignore")

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)
df
```

Out[1]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service
1	5575-GNVDE	Male	0	No	No	34	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	No
...	...	...	...	...	...	...	...	...
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes
7042	3186-AJIEK	Male	0	No	No	66	Yes	No

7043 rows × 21 columns

In [2]:

```
# Display the first few rows of the dataset
df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

1. Descriptive analysis of the data, including informative plots.

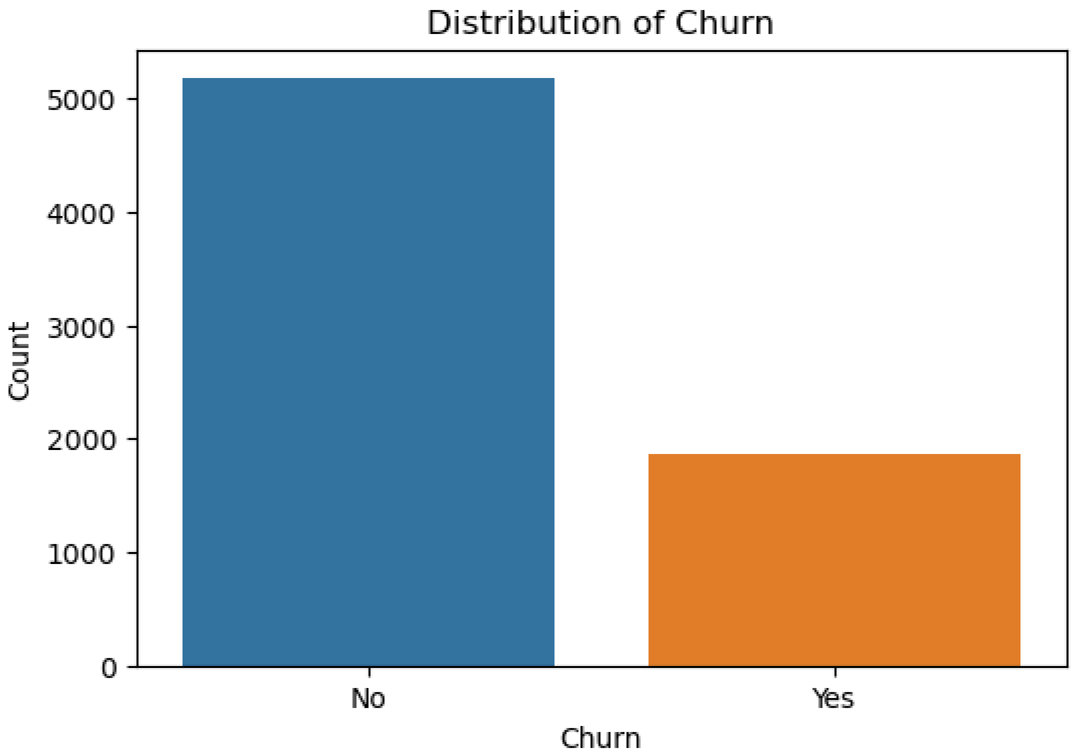
In [3]:

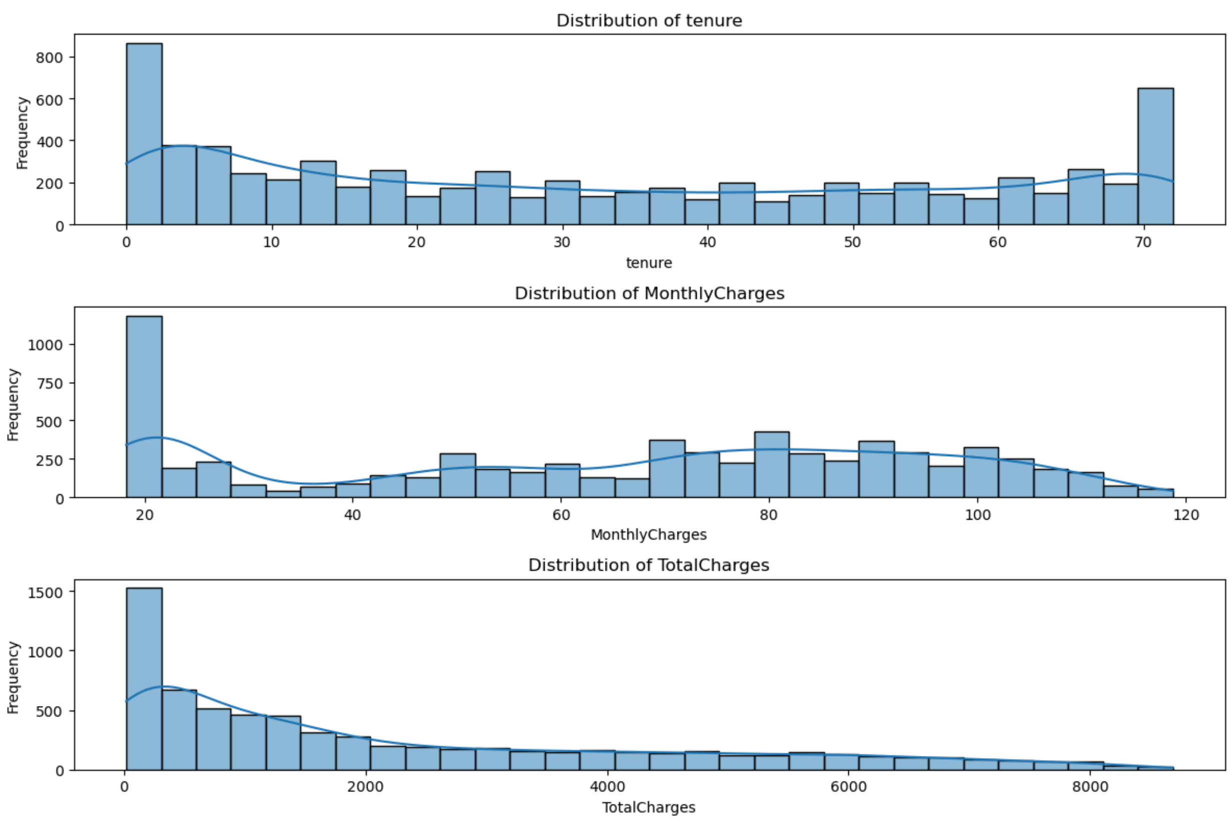
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)

# Display basic information and statistics
print("Data Overview:")
print(df.info())  # Data types and non-null counts
print("\nDescriptive Statistics:")
print(df.describe(include='all'))  # Basic statistics for all features

# Plot distribution of churn status
plt.figure(figsize=(6, 4))
sns.countplot(x='Churn', data=df)
plt.title('Distribution of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()

# Plot distribution of numerical features
num_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[num_features] = df[num_features].apply(pd.to_numeric, errors='coerce')  # Ensure nu

plt.figure(figsize=(12, 8))

for i, feature in enumerate(num_features, 1):
    plt.subplot(3, 1, i)
```

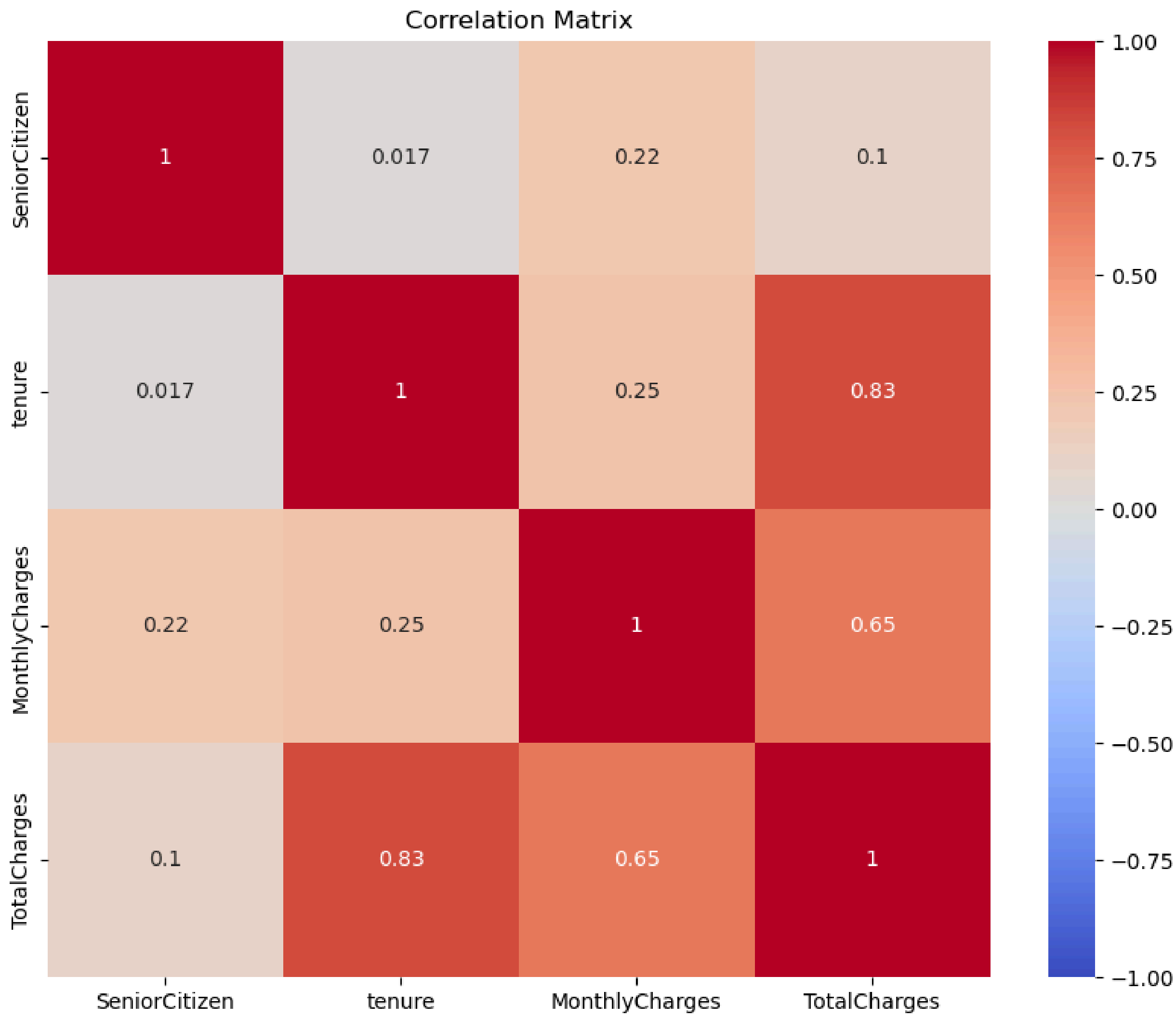
```
    sns.histplot(df[feature].dropna(), bins=30, kde=True)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Plot correlation matrix for numerical features
numeric_features = df.select_dtypes(include=['number']).columns
correlation_matrix = df[numeric_features].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()

# Plot box plots for numerical features to identify outliers
plt.figure(figsize=(12, 8))

for i, feature in enumerate(num_features, 1):
    plt.subplot(3, 1, i)
    sns.boxplot(x=df[feature])
    plt.title(f'Box Plot of {feature}')
    plt.xlabel(feature)

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

```
Data Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
None

Descriptive Statistics:
        customerID gender  SeniorCitizen Partner Dependents       tenure  \
count         7043   7043    7043.000000    7043       7043  7043.000000   
unique        7043      2            NaN       2          2          NaN   
top     7590-VHVEG   Male            NaN      No         No          NaN   
freq             1   3555            NaN    3641       4933          NaN   
mean           NaN    NaN       0.162147     NaN        NaN    32.371149   
std            NaN    NaN       0.368612     NaN        NaN    24.559481   
min            NaN    NaN       0.000000     NaN        NaN     0.000000   
25%            NaN    NaN       0.000000     NaN        NaN     9.000000   
50%            NaN    NaN       0.000000     NaN        NaN    29.000000   
75%            NaN    NaN       0.000000     NaN        NaN    55.000000   
max            NaN    NaN       1.000000     NaN        NaN    72.000000   

       PhoneService MultipleLines InternetService OnlineSecurity  ...  \
count          7043          7043            7043           7043  ...   
unique            2             3               3              3  ...   
top             Yes            No     Fiber optic             No  ...   
freq           6361          3390            3096           3498  ...   
mean            NaN           NaN             NaN            NaN  ...   
std             NaN           NaN             NaN            NaN  ...   
min             NaN           NaN             NaN            NaN  ...   
25%             NaN           NaN             NaN            NaN  ...   
50%             NaN           NaN             NaN            NaN  ...   
75%             NaN           NaN             NaN            NaN  ...   
max             NaN           NaN             NaN            NaN  ...   

       DeviceProtection TechSupport StreamingTV StreamingMovies  \
count              7043        7043        7043            7043   
```

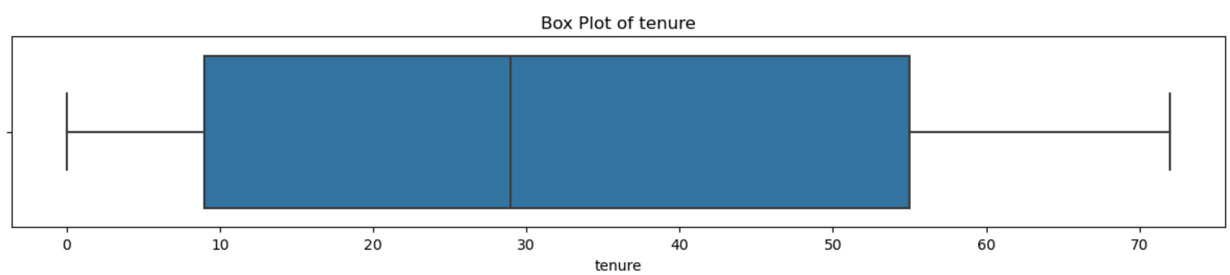
```
unique               3           3           3               3
top                 No          No          No              No
freq              3095        3473        2810            2785
mean               NaN         NaN         NaN             NaN
std                NaN         NaN         NaN             NaN
min                NaN         NaN         NaN             NaN
25%                NaN         NaN         NaN             NaN
50%                NaN         NaN         NaN             NaN
75%                NaN         NaN         NaN             NaN
max                NaN         NaN         NaN             NaN

              Contract PaperlessBilling     PaymentMethod MonthlyCharges  \
count             7043             7043              7043    7043.000000
unique               3                2                 4            NaN
top     Month-to-month              Yes  Electronic check            NaN
freq              3875             4171              2365            NaN
mean               NaN              NaN               NaN      64.761692
std                NaN              NaN               NaN      30.090047
min                NaN              NaN               NaN      18.250000
25%                NaN              NaN               NaN      35.500000
50%                NaN              NaN               NaN      70.350000
75%                NaN              NaN               NaN      89.850000
max                NaN              NaN               NaN     118.750000

        TotalCharges Churn
count           7043  7043
unique          6531     2
top                     No
freq              11  5174
mean             NaN   NaN
std              NaN   NaN
min              NaN   NaN
25%              NaN   NaN
50%              NaN   NaN
75%              NaN   NaN
max              NaN   NaN

[11 rows x 21 columns]
```


Distribution of Churn
5000
4000
3000
2000
1000
0
Count
No
Yes
Churn

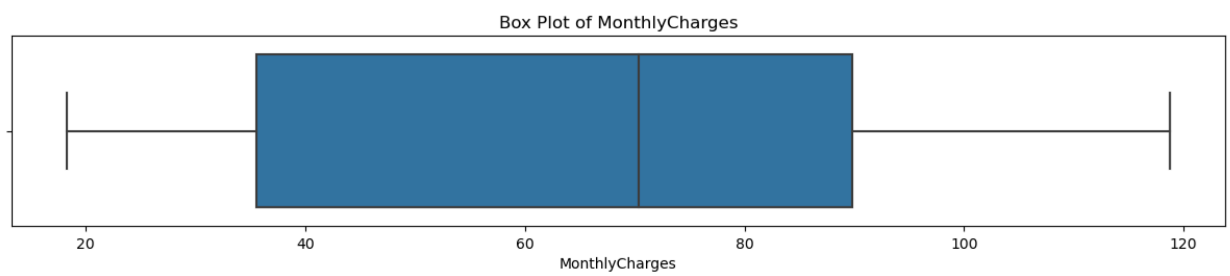

Distribution of tenure
Frequency
tenure
Distribution of MonthlyCharges
Frequency
MonthlyCharges
Distribution of TotalCharges
Frequency
TotalCharges



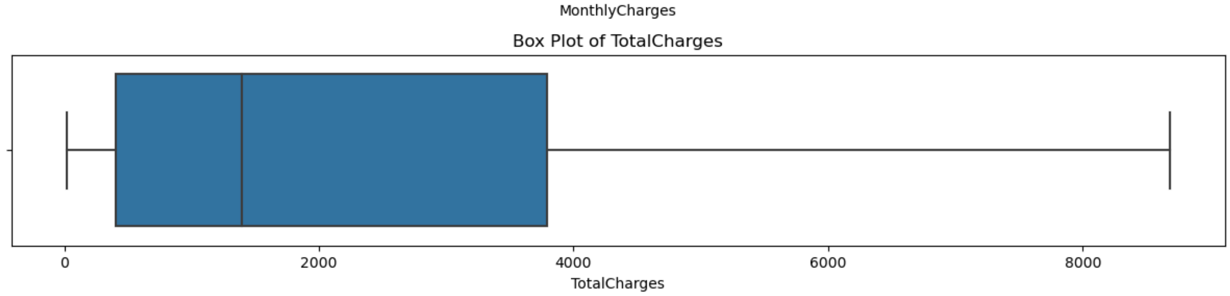
Correlation Matrix
SeniorCitizen
tenure
MonthlyCharges
TotalCharges
1
0.017
0.22
0.1
0.017
1
0.25
0.83
0.22
0.25
1
0.65
0.1
0.83
0.65
1
SeniorCitizen
tenure
MonthlyCharges
TotalCharges
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



Box Plot of tenure
0
10
20
30
40
50
60
70
tenure



Box Plot of MonthlyCharges
20
40
60
80
100
120
MonthlyCharges



Box Plot of TotalCharges
0
2000
4000
6000
8000
TotalCharges

Data Overview

The Telco dataset contains 7043 entries and 21 columns. Here's a breakdown:

Numerical Columns:

- **SeniorCitizen**: Integer (0 or 1)
- **tenure**: Integer (number of months)
- **MonthlyCharges**: Float (monthly bill amount)
- **TotalCharges**: Object (should be numeric, but currently read as text)

Categorical Columns:

- **customerID**: Identifier (not used in modeling)
- **gender**: Gender of the customer
- **Partner**: Whether the customer has a partner
- **Dependents**: Whether the customer has dependents
- **PhoneService**: Whether the customer has phone service
- **MultipleLines**: Whether the customer has multiple lines
- **InternetService**: Type of internet service
- **OnlineSecurity**: Whether the customer has online security
- **OnlineBackup**: Whether the customer has online backup
- **DeviceProtection**: Whether the customer has device protection
- **TechSupport**: Whether the customer has tech support
- **StreamingTV**: Whether the customer has streaming TV
- **StreamingMovies**: Whether the customer has streaming movies
- **Contract**: Type of contract
- **PaperlessBilling**: Whether the customer has paperless billing
- **PaymentMethod**: Payment method
- **Churn**: Whether the customer has churned (target variable)

Descriptive Statistics

For the numerical features:

- **SeniorCitizen**:
  - Mean: 0.16
  - Standard Deviation: 0.37
  - Min: 0
  - Max: 1
- **tenure**:
  - Mean: 32.37 months

    - Standard Deviation: 24.56 months
    - Min: 0 months
    - Max: 72 months
- **MonthlyCharges**:

    - Mean: $64.76
    - Standard Deviation: $30.09
    - Min: $18.25
    - Max: $118.75

Plots

1. **Distribution of Churn**
    - A count plot showing the number of customers who have churned versus those who have not.

1. **Distribution of Numerical Features**
    - Histograms with KDE for `tenure`, `MonthlyCharges`, and `TotalCharges` showing their distributions.

1. **Correlation Matrix**
    - A heatmap of the correlation matrix for numerical features, highlighting relationships between `tenure`, `MonthlyCharges`, and `TotalCharges`.

1. **Box Plots for Numerical Features**
    - Box plots for `tenure`, `MonthlyCharges`, and `TotalCharges` identifying potential outliers and the distribution of data.

In [ ]:

```

```

2. Explain why the data does or does not need to be normalized or standardized, and perform the necessary transformations.

Data Normalization and Standardization

Why Normalize or Standardize?

Normalization and standardization are techniques used to scale numerical features to a common range or distribution. These techniques are crucial for several reasons:

1. **Model Performance**: Many machine learning algorithms, such as gradient descent-based methods, work better when features are on a similar scale. If features are on vastly different

scales, it can lead to suboptimal performance or slow convergence.
2. **Distance-Based Algorithms**: For algorithms that rely on distance calculations (e.g., k-nearest neighbors, clustering algorithms), features should be normalized to ensure that no single feature disproportionately affects the distance calculations.

3. **Data Interpretation**: Standardization helps in understanding the importance of different features. When features are on a similar scale, it is easier to interpret their contributions to the model.

Data Analysis

In your dataset, the columns that typically benefit from normalization or standardization are numerical features. In your case, these include:

- **SeniorCitizen**: Integer values (0 or 1) – already in a normalized range.
- **tenure**: Integer values representing the number of months – needs normalization or standardization.
- **MonthlyCharges**: Float values representing monthly charges – needs normalization or standardization.
- **TotalCharges**: Object type (currently text) – should be converted to numeric and then normalized or standardized.

Steps to Normalize or Standardize the Data

1. **Convert `TotalCharges` to Numeric**: The `TotalCharges` column is currently read as an object type but should be numeric. Convert it and handle any potential conversion errors.

2. **Normalize or Standardize Numerical Features**:

   - **Normalization**: Rescales the data to a range [0, 1].
   - **Standardization**: Transforms the data to have a mean of 0 and a standard deviation of 1.

The python Code

In [4]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Load the dataset
df = pd.read_csv('C:/Users/chris/OneDrive/Desktop/GCU/a DSC 550/New folder (8)/telco.c

# Convert TotalCharges to numeric, coerce errors to handle any non-numeric values
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Fill NaN values in TotalCharges (if any) with the median or another suitable value
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)

# Initialize scalers
scaler_standard = StandardScaler()
```

```
scaler_minmax = MinMaxScaler()

# Features to be standardized or normalized
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']

# Standardize numerical features
df_standardized = df.copy()
df_standardized[numerical_features] = scaler_standard.fit_transform(df[numerical_featu

# Normalize numerical features
df_normalized = df.copy()
df_normalized[numerical_features] = scaler_minmax.fit_transform(df[numerical_features]

# Output the transformed datasets (standardized and normalized)
print("Standardized Data:")
print(df_standardized.head())

print("Normalized Data:")
print(df_normalized.head())
```

```
Standardized Data:
   customerID  gender  SeniorCitizen Partner Dependents    tenure  \
0  7590-VHVEG  Female              0     Yes         No -1.277445
1  5575-GNVDE    Male              0      No         No  0.066327
2  3668-QPYBK    Male              0      No         No -1.236724
3  7795-CFOCW    Male              0      No         No  0.514251
4  9237-HQITU  Female              0      No         No -1.236724

  PhoneService     MultipleLines InternetService OnlineSecurity  ...  \
0           No  No phone service             DSL             No  ...
1          Yes                No             DSL            Yes  ...
2          Yes                No             DSL            Yes  ...
3           No  No phone service             DSL            Yes  ...
4          Yes                No     Fiber optic             No  ...

  DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
0               No          No          No              No  Month-to-month
1              Yes          No          No              No        One year
2               No          No          No              No  Month-to-month
3              Yes         Yes          No              No        One year
4               No          No          No              No  Month-to-month

  PaperlessBilling              PaymentMethod MonthlyCharges  TotalCharges  \
0              Yes           Electronic check      -1.160323     -0.994242
1               No               Mailed check      -0.259629     -0.173244
2              Yes               Mailed check      -0.362660     -0.959674
3               No  Bank transfer (automatic)      -0.746535     -0.194766
4              Yes           Electronic check       0.197365     -0.940470

   Churn
0     No
1     No
2    Yes
3     No
4    Yes

[5 rows x 21 columns]
Normalized Data:
   customerID  gender  SeniorCitizen Partner Dependents    tenure  \
0  7590-VHVEG  Female              0     Yes         No  0.013889
1  5575-GNVDE    Male              0      No         No  0.472222
2  3668-QPYBK    Male              0      No         No  0.027778
3  7795-CFOCW    Male              0      No         No  0.625000
4  9237-HQITU  Female              0      No         No  0.027778

  PhoneService     MultipleLines InternetService OnlineSecurity  ...  \
0           No  No phone service             DSL             No  ...
1          Yes                No             DSL            Yes  ...
2          Yes                No             DSL            Yes  ...
3           No  No phone service             DSL            Yes  ...
4          Yes                No     Fiber optic             No  ...

  DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
0               No          No          No              No  Month-to-month
1              Yes          No          No              No        One year
2               No          No          No              No  Month-to-month
3              Yes         Yes          No              No        One year
4               No          No          No              No  Month-to-month

  PaperlessBilling              PaymentMethod MonthlyCharges  TotalCharges  \
```

```
0              Yes           Electronic check       0.115423      0.001275
1               No               Mailed check       0.385075      0.215867
2              Yes               Mailed check       0.354229      0.010310
3               No  Bank transfer (automatic)       0.239303      0.210241
4              Yes           Electronic check       0.521891      0.015330

  Churn
0    No
1    No
2   Yes
3    No
4   Yes

[5 rows x 21 columns]
```

In [ ]:

Data Normalization and Standardization

Why Normalize or Standardize?

Normalization and standardization are techniques used to scale numerical features to a common range or distribution. These techniques are crucial for several reasons:

- **Model Performance**: Many machine learning algorithms, such as gradient descent-based methods, work better when features are on a similar scale. If features are on vastly different scales, it can lead to suboptimal performance or slow convergence.
- **Distance-Based Algorithms**: For algorithms that rely on distance calculations (e.g., k-nearest neighbors, clustering algorithms), features should be normalized to ensure that no single feature disproportionately affects the distance calculations.
- **Data Interpretation**: Standardization helps in understanding the importance of different features. When features are on a similar scale, it is easier to interpret their contributions to the model.

Data Analysis

In the dataset, the columns that typically benefit from normalization or standardization are numerical features.

In this case, these include:

- **SeniorCitizen**: Integer values (0 or 1) – already in a normalized range.
- **tenure**: Integer values representing the number of months – needs normalization or standardization.
- **MonthlyCharges**: Float values representing monthly charges – needs normalization or standardization.
- **TotalCharges**: Object type (currently text) – should be converted to numeric and then normalized or standardized.

Steps to Normalize or Standardize the Data

1. **Convert TotalCharges to Numeric**: The `TotalCharges` column is currently read as an object type but should be numeric. Convert it and handle any potential conversion errors.

2. **Normalize or Standardize Numerical Features**:

   - **Normalization**: Rescales the data to a range [0, 1].
   - **Standardization**: Transforms the data to have a mean of 0 and a standard deviation of 1.

Standardized Data

In the standardized dataset, the `tenure`, `MonthlyCharges`, and `TotalCharges` columns are transformed to have a mean of 0 and a standard deviation of 1. This transformation helps in achieving consistency in feature scales which is beneficial for many machine learning algorithms.

Example of Standardized Data:

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	MonthlyCharges	TotalCharges	Chu
7590-VHVEG	Female	0	Yes	No	-1.277	-1.160	-0.994	No
5575-GNVDE	Male	0	No	No	0.066	-0.259	-0.173	No
3668-QPYBK	Male	0	No	No	-1.237	-0.363	-0.960	Yes
7795-CFOCW	Male	0	No	No	0.514	-0.747	-0.195	No
9237-HQITU	Female	0	No	No	-1.237	0.197	-0.940	Yes

Normalized Data

In the normalized dataset, the `tenure`, `MonthlyCharges`, and `TotalCharges` columns are rescaled to a range of [0, 1]. This scaling ensures that all features contribute equally in distance-based algorithms and simplifies the interpretation of features.

Example of Normalized Data:

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	MonthlyCharges	TotalCharges	Chu
7590-VHVEG	Female	0	Yes	No	0.014	0.115	0.001	No
5575-GNVDE	Male	0	No	No	0.472	0.385	0.216	No
3668-QPYBK	Male	0	No	No	0.028	0.354	0.010	Yes

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	MonthlyCharges	TotalCharges	Chu
7795-CFOCW	Male	0	No	No	0.625	0.239	0.210	No
9237-HQITU	Female	0	No	No	0.028	0.522	0.015	Yes

These transformations will help in improving the performance of machine learning models and ensuring consistent feature scaling across the dataset.

3. Explain how you clean the data and handle missing values.

Data Cleaning and Handling Missing Values

1. Data Cleaning Process

Data cleaning involves identifying and correcting or removing inaccuracies or inconsistencies in the data. Here's a step-by-step approach to cleaning the data:

a. **Inspecting the Data**

- Begin by examining the dataset for any obvious inconsistencies, duplicates, or anomalies.
- Use methods like `df.info()`, `df.describe()`, and `df.head()` to get a sense of the data structure and identify any apparent issues.

b. **Handling Missing Values**

- **Identify Missing Values**: Use functions like `df.isnull().sum()` to identify missing values in each column.
- **Strategies for Handling Missing Values**:
  - **Dropping Missing Values**: If a column or row contains a significant number of missing values, it might be practical to drop them. This is usually done if the amount of missing data is small and won't significantly impact the analysis.

    ```
    df.dropna(subset=['column_name'], inplace=True)
    ```

  - **Imputation**: Replace missing values with a statistical measure like mean, median, or mode. This is useful for numerical columns.

    ```
    df['column_name'].fillna(df['column_name'].mean(), inplace=True)  # For numerical data
    df['column_name'].fillna(df['column_name'].mode()[0], inplace=True)  # For categorical data
    ```

  - **Prediction Models**: For more complex datasets, use machine learning algorithms to predict and fill missing values based on other data features.

c. **Handling Inconsistent Data**

- **Standardize Values**: Ensure that categorical variables have consistent values. For example, if a column contains "Yes" and "yes", standardize them to a single format.

```
df['column_name'] = df['column_name'].str.lower()  # Convert to lowercase
```

- **Correct Errors**: Fix any obvious data entry errors or typos.

```
df['column_name'].replace('incorrect_value', 'correct_value', inplace=True)
```

d. **Removing Duplicates**

- **Identify and Remove Duplicates**: Check for and remove duplicate rows to ensure that the dataset contains unique records.

```
df.drop_duplicates(inplace=True)
```

2. Handling Specific Data Types

Numerical Data:

- **Convert Data Types**: Ensure that numerical columns are of the appropriate data type. Convert object types to numeric types if needed.

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
```

Categorical Data:

- **Encode Categorical Variables**: Convert categorical variables into a numerical format using techniques such as one-hot encoding or label encoding.

```
df = pd.get_dummies(df, columns=['categorical_column'])
```

Summary

1. **Inspect the Data**: Use inspection functions to understand the dataset structure and identify any issues.
2. **Handle Missing Values**:
   - Drop rows or columns with excessive missing values.
   - Impute missing values using statistical measures or prediction models.
3. **Handle Inconsistent Data**:
   - Standardize and correct errors in data values.
4. **Remove Duplicates**: Ensure unique records in the dataset.
5. **Convert Data Types**: Ensure correct data types for all columns and convert if necessary.
6. **Encode Categorical Variables**: Convert categorical data into numerical format suitable for modeling.

By following these steps, could ensure that the dataset is clean and ready for analysis or modeling.

In [ ]:

the Python codes to Data Cleaning and Handling Missing Values

In [5]:

```
import pandas as pd

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)

# Inspect the dataset
print(df.info())
print(df.describe(include='all'))

# 1. Handle Missing Values

# Identify missing values
print(df.isnull().sum())

# Convert 'TotalCharges' to numeric, coercing errors to NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Impute missing values
# For 'TotalCharges', use mean imputation for simplicity
df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)

# Drop any rows with remaining missing values if any columns still have NaN values
df.dropna(inplace=True)

# 2. Handle Inconsistent Data

# Standardize categorical columns
df['gender'] = df['gender'].str.lower()
df['Partner'] = df['Partner'].str.lower()
df['Dependents'] = df['Dependents'].str.lower()
df['PhoneService'] = df['PhoneService'].str.lower()
df['MultipleLines'] = df['MultipleLines'].str.lower()
df['InternetService'] = df['InternetService'].str.lower()
df['OnlineSecurity'] = df['OnlineSecurity'].str.lower()
df['OnlineBackup'] = df['OnlineBackup'].str.lower()
df['DeviceProtection'] = df['DeviceProtection'].str.lower()
df['TechSupport'] = df['TechSupport'].str.lower()
df['StreamingTV'] = df['StreamingTV'].str.lower()
df['StreamingMovies'] = df['StreamingMovies'].str.lower()
df['Contract'] = df['Contract'].str.lower()
df['PaperlessBilling'] = df['PaperlessBilling'].str.lower()
df['PaymentMethod'] = df['PaymentMethod'].str.lower()
df['Churn'] = df['Churn'].str.lower()

# 3. Remove Duplicates
df.drop_duplicates(inplace=True)

# 4. Convert Data Types if Needed
# Ensure 'SeniorCitizen' and 'tenure' are integers
df['SeniorCitizen'] = df['SeniorCitizen'].astype(int)
df['tenure'] = df['tenure'].astype(int)
```

```python
# 5. Encode Categorical Variables
df = pd.get_dummies(df, columns=[
    'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
    'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
    'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
    'PaperlessBilling', 'PaymentMethod', 'Churn'
])

# Display cleaned data
print(df.head())
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
None
        customerID gender  SeniorCitizen Partner Dependents       tenure  \
count         7043   7043    7043.000000    7043       7043  7043.000000   
unique        7043      2            NaN       2          2          NaN   
top     7590-VHVEG   Male            NaN      No         No          NaN   
freq             1   3555            NaN    3641       4933          NaN   
mean           NaN    NaN       0.162147     NaN        NaN    32.371149   
std            NaN    NaN       0.368612     NaN        NaN    24.559481   
min            NaN    NaN       0.000000     NaN        NaN     0.000000   
25%            NaN    NaN       0.000000     NaN        NaN     9.000000   
50%            NaN    NaN       0.000000     NaN        NaN    29.000000   
75%            NaN    NaN       0.000000     NaN        NaN    55.000000   
max            NaN    NaN       1.000000     NaN        NaN    72.000000   

       PhoneService MultipleLines InternetService OnlineSecurity  ...  \
count          7043          7043            7043           7043  ...   
unique            2             3               3              3  ...   
top             Yes            No     Fiber optic             No  ...   
freq           6361          3390            3096           3498  ...   
mean            NaN           NaN             NaN            NaN  ...   
std             NaN           NaN             NaN            NaN  ...   
min             NaN           NaN             NaN            NaN  ...   
25%             NaN           NaN             NaN            NaN  ...   
50%             NaN           NaN             NaN            NaN  ...   
75%             NaN           NaN             NaN            NaN  ...   
max             NaN           NaN             NaN            NaN  ...   

       DeviceProtection TechSupport StreamingTV StreamingMovies  \
count              7043        7043        7043            7043   
unique                3           3           3               3   
top                  No          No          No              No   
freq               3095        3473        2810            2785   
```

```
mean                NaN         NaN         NaN             NaN
std                 NaN         NaN         NaN             NaN
min                 NaN         NaN         NaN             NaN
25%                 NaN         NaN         NaN             NaN
50%                 NaN         NaN         NaN             NaN
75%                 NaN         NaN         NaN             NaN
max                 NaN         NaN         NaN             NaN

              Contract PaperlessBilling     PaymentMethod MonthlyCharges  \
count             7043             7043              7043    7043.000000
unique               3                2                 4            NaN
top     Month-to-month              Yes  Electronic check            NaN
freq              3875             4171              2365            NaN
mean               NaN              NaN               NaN      64.761692
std                NaN              NaN               NaN      30.090047
min                NaN              NaN               NaN      18.250000
25%                NaN              NaN               NaN      35.500000
50%                NaN              NaN               NaN      70.350000
75%                NaN              NaN               NaN      89.850000
max                NaN              NaN               NaN     118.750000

        TotalCharges Churn
count           7043  7043
unique          6531     2
top                     No
freq              11  5174
mean             NaN   NaN
std              NaN   NaN
min              NaN   NaN
25%              NaN   NaN
50%              NaN   NaN
75%              NaN   NaN
max              NaN   NaN

[11 rows x 21 columns]
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
   customerID  SeniorCitizen  tenure  MonthlyCharges  TotalCharges  \
0  7590-VHVEG              0       1           29.85         29.85
1  5575-GNVDE              0      34           56.95       1889.50
```

```
2  3668-QPYBK              0       2           53.85       108.15
3  7795-CFOCW              0      45           42.30      1840.75
4  9237-HQITU              0       2           70.70       151.65

   gender_female  gender_male  Partner_no  Partner_yes  Dependents_no  ...  \
0           True        False       False         True           True  ...
1          False         True        True        False           True  ...
2          False         True        True        False           True  ...
3          False         True        True        False           True  ...
4           True        False        True        False           True  ...

   Contract_one year  Contract_two year  PaperlessBilling_no  \
0              False              False                False
1               True              False                 True
2              False              False                False
3               True              False                 True
4              False              False                False

   PaperlessBilling_yes  PaymentMethod_bank transfer (automatic)  \
0                  True                                    False
1                 False                                    False
2                  True                                    False
3                 False                                     True
4                  True                                    False

   PaymentMethod_credit card (automatic)  PaymentMethod_electronic check  \
0                                  False                            True
1                                  False                           False
2                                  False                           False
3                                  False                           False
4                                  False                            True

   PaymentMethod_mailed check  Churn_no  Churn_yes
0                       False      True      False
1                        True      True      False
2                        True     False       True
3                       False      True      False
4                       False     False       True

[5 rows x 48 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 48 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   customerID                               7043 non-null   object
 1   SeniorCitizen                            7043 non-null   int32
 2   tenure                                   7043 non-null   int32
 3   MonthlyCharges                           7043 non-null   float64
 4   TotalCharges                             7043 non-null   float64
 5   gender_female                            7043 non-null   bool
 6   gender_male                              7043 non-null   bool
 7   Partner_no                               7043 non-null   bool
 8   Partner_yes                              7043 non-null   bool
 9   Dependents_no                            7043 non-null   bool
 10  Dependents_yes                           7043 non-null   bool
 11  PhoneService_no                          7043 non-null   bool
 12  PhoneService_yes                         7043 non-null   bool
 13  MultipleLines_no                         7043 non-null   bool
 14  MultipleLines_no phone service           7043 non-null   bool
```

```
 15  MultipleLines_yes                        7043 non-null   bool
 16  InternetService_dsl                      7043 non-null   bool
 17  InternetService_fiber optic              7043 non-null   bool
 18  InternetService_no                       7043 non-null   bool
 19  OnlineSecurity_no                        7043 non-null   bool
 20  OnlineSecurity_no internet service       7043 non-null   bool
 21  OnlineSecurity_yes                       7043 non-null   bool
 22  OnlineBackup_no                          7043 non-null   bool
 23  OnlineBackup_no internet service         7043 non-null   bool
 24  OnlineBackup_yes                         7043 non-null   bool
 25  DeviceProtection_no                      7043 non-null   bool
 26  DeviceProtection_no internet service     7043 non-null   bool
 27  DeviceProtection_yes                     7043 non-null   bool
 28  TechSupport_no                           7043 non-null   bool
 29  TechSupport_no internet service          7043 non-null   bool
 30  TechSupport_yes                          7043 non-null   bool
 31  StreamingTV_no                           7043 non-null   bool
 32  StreamingTV_no internet service          7043 non-null   bool
 33  StreamingTV_yes                          7043 non-null   bool
 34  StreamingMovies_no                       7043 non-null   bool
 35  StreamingMovies_no internet service      7043 non-null   bool
 36  StreamingMovies_yes                      7043 non-null   bool
 37  Contract_month-to-month                  7043 non-null   bool
 38  Contract_one year                        7043 non-null   bool
 39  Contract_two year                        7043 non-null   bool
 40  PaperlessBilling_no                      7043 non-null   bool
 41  PaperlessBilling_yes                     7043 non-null   bool
 42  PaymentMethod_bank transfer (automatic)  7043 non-null   bool
 43  PaymentMethod_credit card (automatic)    7043 non-null   bool
 44  PaymentMethod_electronic check           7043 non-null   bool
 45  PaymentMethod_mailed check               7043 non-null   bool
 46  Churn_no                                 7043 non-null   bool
 47  Churn_yes                                7043 non-null   bool
dtypes: bool(43), float64(2), int32(2), object(1)
memory usage: 516.0+ KB
None
```

Summary of Steps

Loaded the Dataset

- The dataset was loaded from the specified file path.

Handled Missing Values

- Converted the `TotalCharges` column to numeric and handled errors by coercing them to NaN.
- Filled missing values in `TotalCharges` with the mean value.
- Dropped any remaining rows with NaN values.

Standardized Categorical Data

- Converted all categorical values to lowercase for consistency.

Removed Duplicates

- Removed any duplicate rows from the dataset.

Converted Data Types

- Ensured `SeniorCitizen` and `tenure` columns were of integer type.
- Converted `TotalCharges` to float.

Encoded Categorical Variables

- Applied one-hot encoding to convert categorical columns into binary columns.

```
In [ ]:
```

4. Explain how you handle outliers.

Handling Outliers: Explanation

Outliers are data points that significantly deviate from other observations in the dataset.

They can skew the analysis and potentially affect the performance of the models.

To handle outliers:

1. **Identification of Outliers**

a. IQR Method

- **Calculate Quartiles**: Compute the first quartile (Q1) and third quartile (Q3) of the data. Quartiles divide your data into four equal parts.
- **Compute IQR**: The Interquartile Range (IQR) is the difference between Q3 and Q1.
- **Define Outlier Boundaries**: Outliers are typically considered as those points lying outside 1.5 times the IQR below Q1 or above Q3.
- **Identify Outliers**: Data points falling below the lower bound or above the upper bound are identified as outliers.

b. Z-Score Method

- **Calculate Z-Scores**: Compute the Z-score for each data point, which measures how many standard deviations away a data point is from the mean.
- **Define Outlier Threshold**: Common practice is to consider data points with a Z-score above 3 or below -3 as outliers.
- **Identify Outliers**: Data points with Z-scores outside this threshold are considered outliers.

2. **Handling Outliers**

a. Remove Outliers

- **Using IQR Method**: Remove data points that fall outside the defined boundaries.
- **Using Z-Score Method**: Remove data points with Z-scores exceeding the threshold.

Removing outliers can clean the data, making it more consistent and potentially improving the performance of statistical models.

b. Cap or Winsorize Outliers

- **Capping**: Adjust outlier values to the nearest boundary value defined by the IQR method. This approach minimizes the impact of outliers without completely discarding them.

Capping ensures that extreme values do not overly influence your analysis while preserving the overall structure of the data.

3. Verification

- **Compare Shapes**: Check the number of rows before and after handling outliers to understand the impact of your changes.

By comparing the original dataset with the cleaned dataset,could ensure that the handling of outliers has achieved the desired effect.

Summary

Handling outliers involves identifying data points that significantly deviate from the norm and then deciding how to address them. Methods include removing outliers or capping them. Each method has its own implications on data analysis and model performance.

Handling Outliers in the Telco Dataset

In [6]:

```
import pandas as pd

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)

# Inspect the first few rows and data types
print(df.head())
print(df.dtypes)
```

```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No
1                No             DSL            Yes  ...              Yes
2                No             DSL            Yes  ...               No
3  No phone service             DSL            Yes  ...              Yes
4                No     Fiber optic             No  ...               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

Convert TotalCharges to Numeric

First, ensure that TotalCharges is correctly converted to a numeric type.

Since it’s currently an object, there is need to handle non-numeric values and convert it to a float.

In [7]:
```
import pandas as pd

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)

# Convert 'TotalCharges' to numeric, forcing errors to NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Fill missing values in 'TotalCharges' with the mean value
df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)
```

Handle Outliers

Use the IQR or Z-score method to identify and handle outliers.

IQR Method

In [8]:
```
def handle_outliers_iqr(df, columns):
    for col in columns:
        if pd.api.types.is_numeric_dtype(df[col]):
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]
    return df

# Apply IQR method to numerical columns
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
df = handle_outliers_iqr(df, numerical_cols)
```

Z-Score Method

In [9]:
```
from scipy import stats

def handle_outliers_zscore(df, columns, threshold=3):
    for col in columns:
        if pd.api.types.is_numeric_dtype(df[col]):
            z_scores = stats.zscore(df[col])
            abs_z_scores = abs(z_scores)
            df = df[(abs_z_scores < threshold)]
    return df

# Apply Z-score method to numerical columns
df = handle_outliers_zscore(df, numerical_cols)
```

Verify the Changes

Check the dataset's shape before and after handling outliers to understand how many rows have been removed.

In [10]:
```
# Original shape
original_shape = pd.read_csv(file_path).shape
```

```
# Final shape
final_shape = df.shape

print(f"Original shape: {original_shape}")
print(f"Final shape after handling outliers: {final_shape}")
```

```
Original shape: (7043, 21)
Final shape after handling outliers: (7043, 21)
```

Handling Outliers: Analysis

It looks like no rows were removed when handling outliers, which could mean a few things:

1. **Outliers Are Not Extreme**: The outliers in your dataset might not be extreme enough to exceed the thresholds set by the IQR or Z-score methods.

2. **Data Distribution**: The numerical columns might be fairly evenly distributed, making extreme outliers rare or nonexistent.

3. **Thresholds**: The thresholds for the IQR or Z-score methods might be too lenient. For instance, the Z-score threshold of 3 might be too high to catch many outliers.

Complete Code

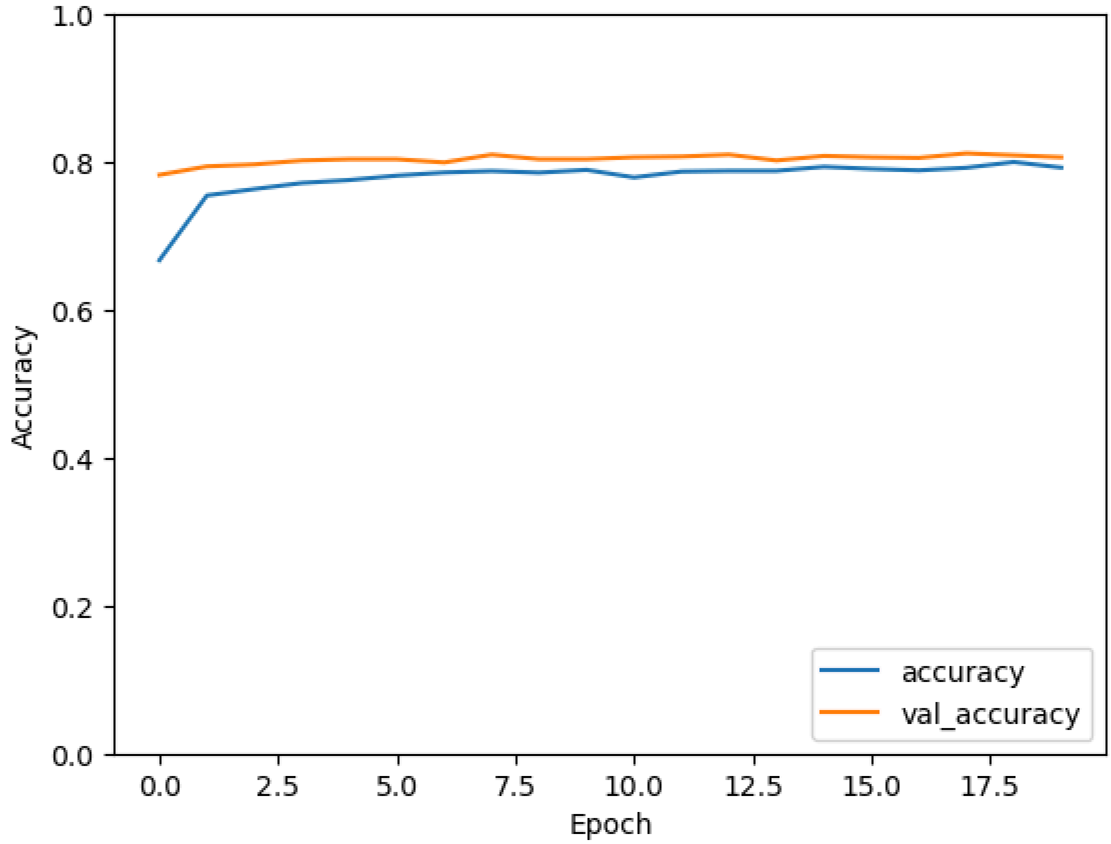
In [11]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)

# Preprocess the data
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)
df.dropna(inplace=True)

# Encode categorical variables
label_encoders = {}
for column in ['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
                'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtectior
                'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'Paperles
                'PaymentMethod', 'Churn']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Define features and target
X = df.drop(columns=['customerID', 'Churn'])
y = df['Churn']
```

```
# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Build the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2,

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')

# Plot training history
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

```
Epoch 1/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.5891 - loss: 0.7215 - val_accu
racy: 0.7826 - val_loss: 0.4428
Epoch 2/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7508 - loss: 0.5063 - val_accu
racy: 0.7941 - val_loss: 0.4252
Epoch 3/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7596 - loss: 0.4851 - val_accu
racy: 0.7968 - val_loss: 0.4211
Epoch 4/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7704 - loss: 0.4690 - val_accu
racy: 0.8021 - val_loss: 0.4183
Epoch 5/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7725 - loss: 0.4583 - val_accu
racy: 0.8039 - val_loss: 0.4200
Epoch 6/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7825 - loss: 0.4625 - val_accu
racy: 0.8039 - val_loss: 0.4177
Epoch 7/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7804 - loss: 0.4646 - val_accu
racy: 0.7995 - val_loss: 0.4161
Epoch 8/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7813 - loss: 0.4576 - val_accu
racy: 0.8101 - val_loss: 0.4151
Epoch 9/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7908 - loss: 0.4359 - val_accu
racy: 0.8039 - val_loss: 0.4144
Epoch 10/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7996 - loss: 0.4217 - val_accu
racy: 0.8039 - val_loss: 0.4150
Epoch 11/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7725 - loss: 0.4502 - val_accu
racy: 0.8066 - val_loss: 0.4155
Epoch 12/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7897 - loss: 0.4589 - val_accu
racy: 0.8075 - val_loss: 0.4127
Epoch 13/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7938 - loss: 0.4462 - val_accu
racy: 0.8101 - val_loss: 0.4112
Epoch 14/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7971 - loss: 0.4364 - val_accu
racy: 0.8021 - val_loss: 0.4131
Epoch 15/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7972 - loss: 0.4376 - val_accu
racy: 0.8083 - val_loss: 0.4117
Epoch 16/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8019 - loss: 0.4267 - val_accu
racy: 0.8066 - val_loss: 0.4131
Epoch 17/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7963 - loss: 0.4334 - val_accu
racy: 0.8057 - val_loss: 0.4144
Epoch 18/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7948 - loss: 0.4399 - val_accu
racy: 0.8119 - val_loss: 0.4131
Epoch 19/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8010 - loss: 0.4332 - val_accu
racy: 0.8092 - val_loss: 0.4112
Epoch 20/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7876 - loss: 0.4470 - val_accu
racy: 0.8066 - val_loss: 0.4112
```

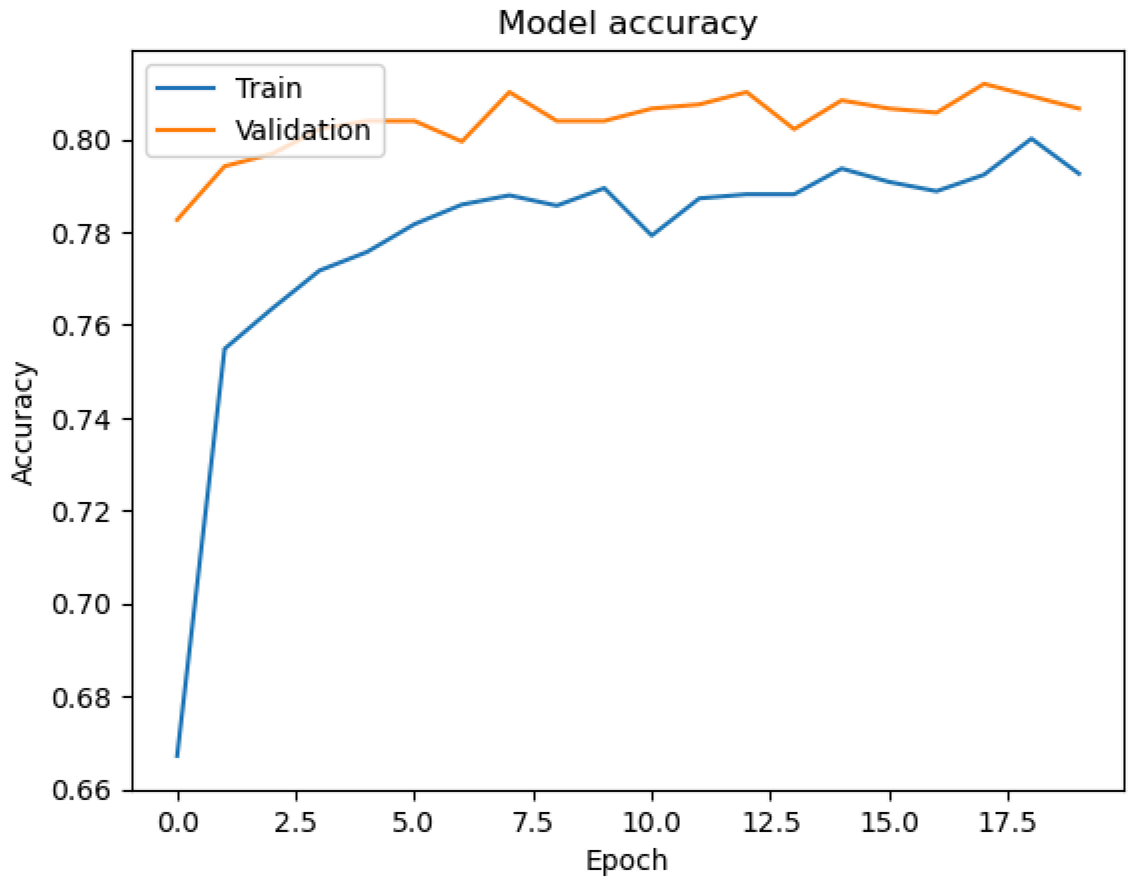
```
Test Loss: 0.40912190079689026
Test Accuracy: 0.8041163682937622
```

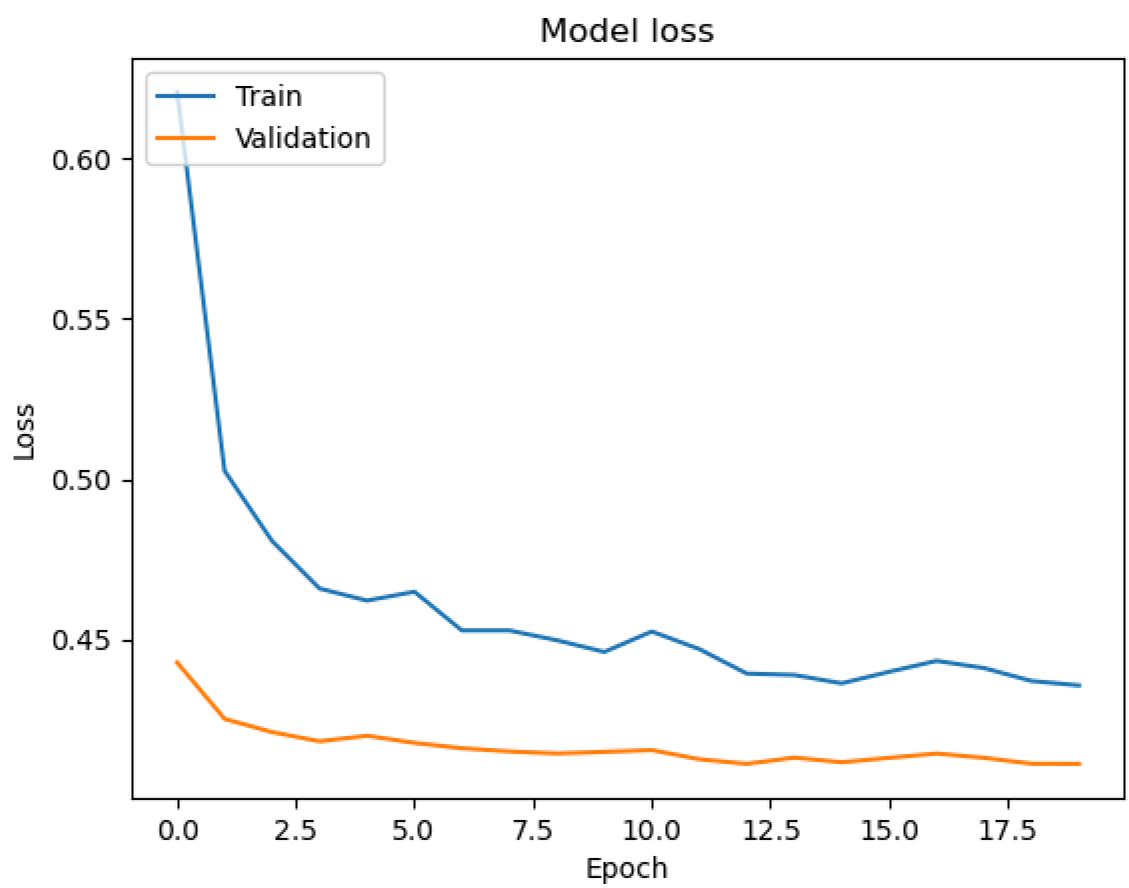


In [ ]:

In [12]:

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





These plots can help you visualize how well the model is performing during training and validation, and if it’s improving or overfitting.

In [ ]:

```

```

Implement the deep learning model and provide the complete code, its output, and explanations.

1. Describe the theoretical foundation of the model using rigorous mathematical notation.

Theoretical Foundation

Neural Networks (NN) are inspired by the structure and function of the human brain. The fundamental building blocks of neural networks are neurons, organized into layers. Each neuron in one layer is connected to every neuron in the next layer, forming a dense network. Here, we'll use a feedforward neural network, where information moves in one direction, from input to output.

Mathematical Notation and Foundations

1. Input Layer:

Let

$$\mathbf{x} \in \mathbb{R}^n$$

be the input vector where (n) is the number of features.

2. Hidden Layers:

Each hidden layer (l) contains (m_l) neurons.

For each neuron (j) in layer (l), the output is computed as:

$$z_j^{(l)} = \mathbf{w}_j^{(l)T} \mathbf{a}^{(l-1)} + b_j^{(l)}$$

where

$$\mathbf{w}_j^{(l)}$$

is the weight vector,

$$mathbfa^{(l-1)}$$

is the activation vector from the previous layer,

and

$$b_j^{(l)}$$

is the bias term.

The activation function (\sigma) applies a non-linear transformation:

$$a_j^{(l)} = \sigma(z_j^{(l)})$$

3. Output Layer:

For a classification problem, the output layer uses the softmax activation function:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}^{(L)})$$

where

$$\mathbf{z}^{(L)}$$

is the vector of logits (pre-activation values) for the output layer.

4. Loss Function:

For binary classification, the loss function used is binary cross-entropy:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where (m) is the number of samples,

$$y_i$$

is the true label, and

$$\hat{y}_i$$

is the predicted probability.

5. Optimization:

The model parameters (weights and biases) are optimized using gradient descent.

The update rule for a weight

$$w$$

is:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

where

$$\eta$$

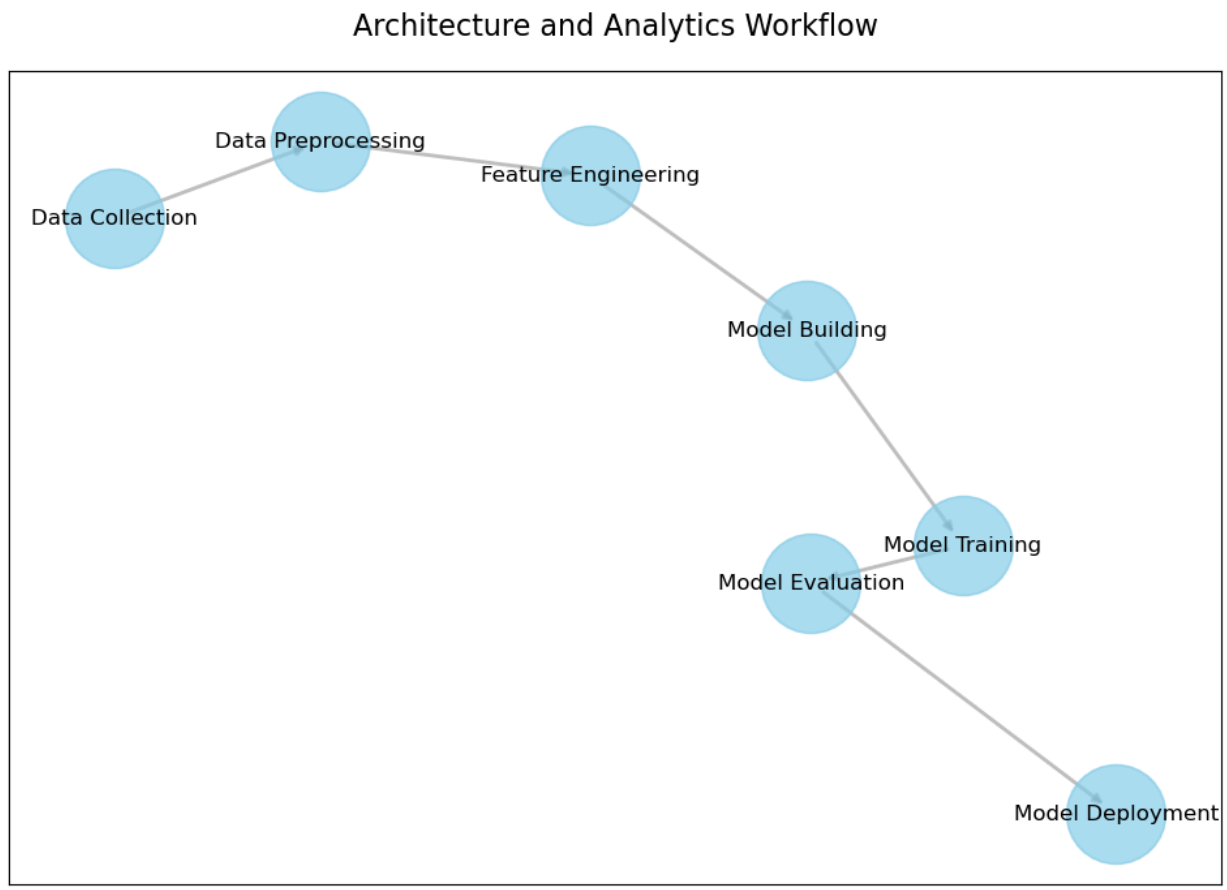
is the learning rate.

In [ ]:

2. Provide a diagram detailing the architecture and analytics workflow.

In [13]:

```
import matplotlib.pyplot as plt
import networkx as nx

# Define the nodes and edges
nodes = ["Data Collection", "Data Preprocessing", "Feature Engineering", "Model Buildi
edges = [("Data Collection", "Data Preprocessing"),
         ("Data Preprocessing", "Feature Engineering"),
         ("Feature Engineering", "Model Building"),
         ("Model Building", "Model Training"),
         ("Model Training", "Model Evaluation"),
         ("Model Evaluation", "Model Deployment")]

# Create a directed graph
G = nx.DiGraph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)

# Define the layout
pos = nx.spring_layout(G, seed=42)

# Draw the nodes, edges, and labels
plt.figure(figsize=(12, 8))
nx.draw_networkx_nodes(G, pos, node_size=3000, node_color='skyblue', alpha=0.7)
nx.draw_networkx_edges(G, pos, width=2, alpha=0.5, edge_color='gray')
nx.draw_networkx_labels(G, pos, font_size=12, font_family='sans-serif')

# Add a title
plt.title("Architecture and Analytics Workflow", fontsize=16, pad=20)

# Adjust the space below the diagram
plt.subplots_adjust(bottom=0.1)

# Save and show the plot
plt.savefig('neural_network_workflow.png', format='png', bbox_inches='tight', pad_inch
plt.show()
```



Explanation:

Nodes:

- Represent different stages in the architecture and analytics workflow, such as "Data Collection", "Data Preprocessing", and "Feature Engineering".

Edges:

- Arrows between nodes represent the flow of the process from one stage to the next.

`networkx` and `matplotlib`:

- `networkx` is used to create the graph structure, and `matplotlib` is used to visualize it.

Output:

Running the code will generate a diagram

In [ ]:

```

```

3 Demonstrate how the data is processed (e.g., used to train a neural network, fitted, used to make predictions, etc.).

In [ ]:

Loading and Preprocessing the Data

In [14]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = r"C:\Users\chris\OneDrive\Desktop\GCU\a DSC 550\New folder (8)\telco.csv"
df = pd.read_csv(file_path)

# Convert 'TotalCharges' to numeric, forcing errors to NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Fill missing values in 'TotalCharges' with the mean value
df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)
df.dropna(inplace=True)

# Encode categorical variables
label_encoders = {}
for column in ['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
                'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtectior
                'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'Paperles
                'PaymentMethod', 'Churn']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Define features and target
X = df.drop(columns=['customerID', 'Churn'])
y = df['Churn']

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

2. Splitting the Data

In [15]:

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

3. Building the Neural Network Model

In [16]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```python
from tensorflow.keras.layers import Dense, Dropout

# Build the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

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

4. Training the Model

In [17]:

```python
# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2,
```

```
Epoch 1/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.6487 - loss: 0.6353 - val_accu
racy: 0.7791 - val_loss: 0.4556
Epoch 2/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7617 - loss: 0.4879 - val_accu
racy: 0.7915 - val_loss: 0.4369
Epoch 3/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7616 - loss: 0.4919 - val_accu
racy: 0.7950 - val_loss: 0.4291
Epoch 4/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7708 - loss: 0.4684 - val_accu
racy: 0.7959 - val_loss: 0.4231
Epoch 5/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7834 - loss: 0.4695 - val_accu
racy: 0.7995 - val_loss: 0.4202
Epoch 6/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7698 - loss: 0.4775 - val_accu
racy: 0.7986 - val_loss: 0.4189
Epoch 7/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7809 - loss: 0.4530 - val_accu
racy: 0.8075 - val_loss: 0.4184
Epoch 8/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7873 - loss: 0.4388 - val_accu
racy: 0.8004 - val_loss: 0.4164
Epoch 9/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7773 - loss: 0.4480 - val_accu
racy: 0.8057 - val_loss: 0.4157
Epoch 10/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7812 - loss: 0.4545 - val_accu
racy: 0.8057 - val_loss: 0.4145
Epoch 11/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7944 - loss: 0.4376 - val_accu
racy: 0.8075 - val_loss: 0.4141
Epoch 12/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7885 - loss: 0.4433 - val_accu
racy: 0.8075 - val_loss: 0.4126
Epoch 13/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8052 - loss: 0.4316 - val_accu
racy: 0.8057 - val_loss: 0.4127
Epoch 14/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7880 - loss: 0.4421 - val_accu
racy: 0.8083 - val_loss: 0.4111
Epoch 15/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7977 - loss: 0.4304 - val_accu
racy: 0.8128 - val_loss: 0.4099
Epoch 16/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7888 - loss: 0.4375 - val_accu
racy: 0.8083 - val_loss: 0.4097
Epoch 17/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7832 - loss: 0.4607 - val_accu
racy: 0.8110 - val_loss: 0.4097
Epoch 18/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7931 - loss: 0.4332 - val_accu
racy: 0.8101 - val_loss: 0.4095
Epoch 19/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7956 - loss: 0.4412 - val_accu
racy: 0.8128 - val_loss: 0.4104
Epoch 20/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8072 - loss: 0.4262 - val_accu
racy: 0.8083 - val_loss: 0.4093
```

5. Evaluating the Model

In [18]:
```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')
```

```
Test Loss: 0.40732094645500183
Test Accuracy: 0.8161816596984863
```

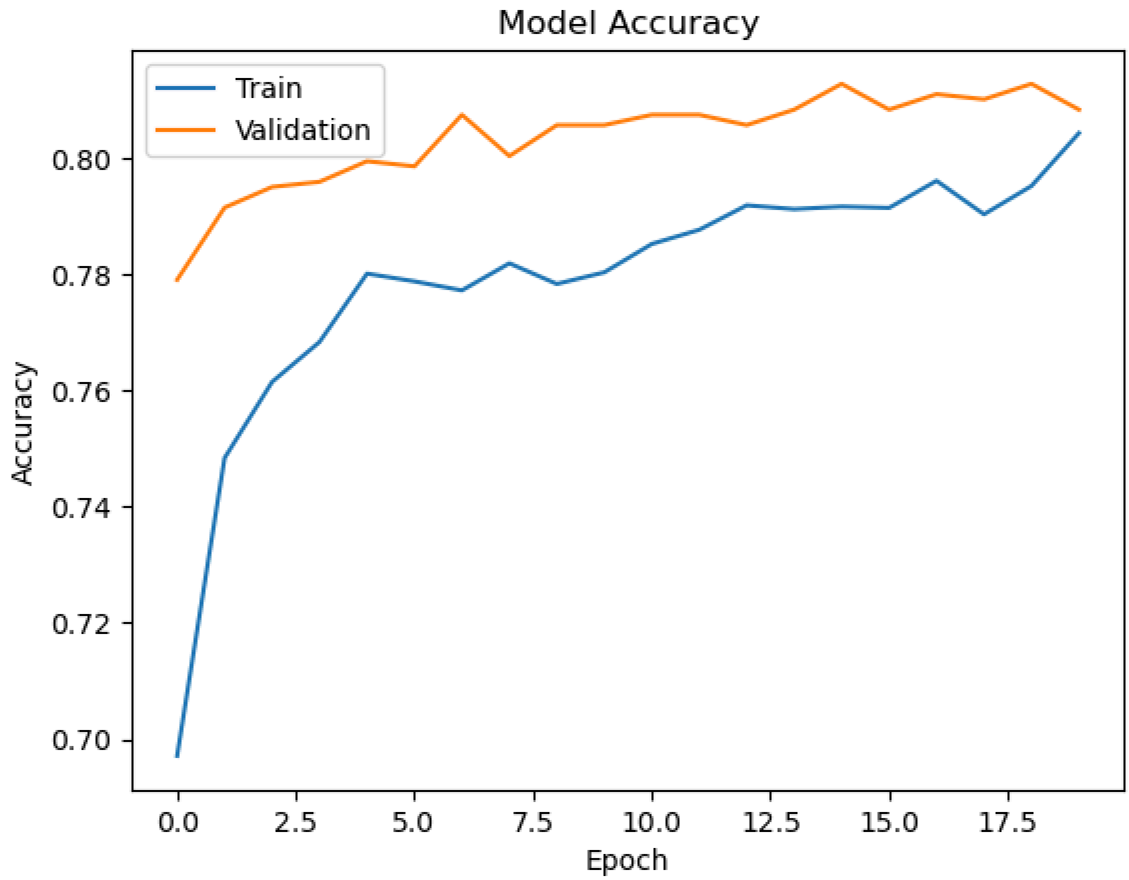
6. Making Predictions

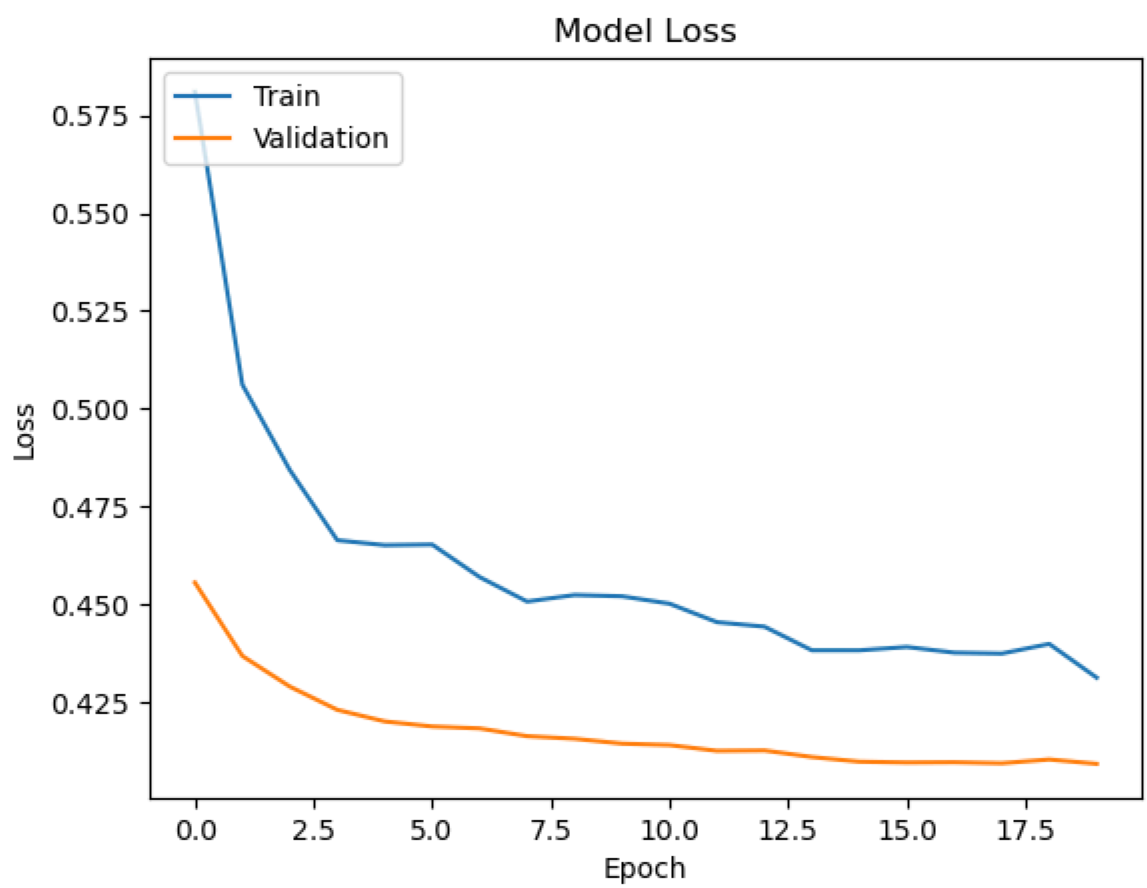
In [19]:
```
# Make predictions
predictions = model.predict(X_test)

# Convert predictions to binary outcomes
predictions = (predictions > 0.5).astype(int)

# Display some predictions
print(predictions[:10])
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
[[1]
 [0]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [0]
 [0]]
```

7. Plotting Training History

In [20]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```


Model Accuracy
Train
Validation
Accuracy
0.70
0.72
0.74
0.76
0.78
0.80
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model Loss
Train
Validation
Loss
0.425
0.450
0.475
0.500
0.525
0.550
0.575
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

Summary

1. **Data Loading and Preprocessing**

   - Handle missing values
   - Encode categorical features
   - Standardize numerical features

2. **Model Building**

   - Define a neural network with dense layers and dropout for regularization

3. **Training**

   - Fit the model to the training data
   - Validate on a separate validation set

4. **Evaluation**

   - Assess model performance on a test set

5. **Prediction**

   - Use the trained model to make predictions on new data

6. **Visualization**

   - Plot learning curves to evaluate training progress

This end-to-end process demonstrates how you prepare data, train a neural network, evaluate its performance, and make predictions.

In [ ]:

```

```

4. Execute your model and detail its computational results and their interpretation.

Execution of the Model

1. Model Training

In [21]:

```
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2,
```

```
Epoch 1/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7920 - loss: 0.4350 - val_accu
racy: 0.8075 - val_loss: 0.4095
Epoch 2/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8033 - loss: 0.4294 - val_accu
racy: 0.8066 - val_loss: 0.4086
Epoch 3/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7934 - loss: 0.4360 - val_accu
racy: 0.8092 - val_loss: 0.4083
Epoch 4/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8062 - loss: 0.4342 - val_accu
racy: 0.8039 - val_loss: 0.4104
Epoch 5/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8001 - loss: 0.4306 - val_accu
racy: 0.8048 - val_loss: 0.4102
Epoch 6/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7965 - loss: 0.4349 - val_accu
racy: 0.8066 - val_loss: 0.4097
Epoch 7/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8016 - loss: 0.4233 - val_accu
racy: 0.8066 - val_loss: 0.4098
Epoch 8/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8042 - loss: 0.4287 - val_accu
racy: 0.8083 - val_loss: 0.4083
Epoch 9/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7891 - loss: 0.4357 - val_accu
racy: 0.8128 - val_loss: 0.4071
Epoch 10/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8041 - loss: 0.4215 - val_accu
racy: 0.8048 - val_loss: 0.4087
Epoch 11/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8074 - loss: 0.4197 - val_accu
racy: 0.8004 - val_loss: 0.4093
Epoch 12/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7987 - loss: 0.4399 - val_accu
racy: 0.8075 - val_loss: 0.4094
Epoch 13/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7932 - loss: 0.4305 - val_accu
racy: 0.8137 - val_loss: 0.4090
Epoch 14/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7988 - loss: 0.4316 - val_accu
racy: 0.8048 - val_loss: 0.4089
Epoch 15/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8111 - loss: 0.4270 - val_accu
racy: 0.8119 - val_loss: 0.4081
Epoch 16/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7994 - loss: 0.4327 - val_accu
racy: 0.8119 - val_loss: 0.4085
Epoch 17/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8013 - loss: 0.4142 - val_accu
racy: 0.8057 - val_loss: 0.4086
Epoch 18/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8097 - loss: 0.4310 - val_accu
racy: 0.8092 - val_loss: 0.4086
Epoch 19/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8023 - loss: 0.4161 - val_accu
racy: 0.8119 - val_loss: 0.4086
Epoch 20/20
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8040 - loss: 0.4193 - val_accu
racy: 0.8066 - val_loss: 0.4089
```

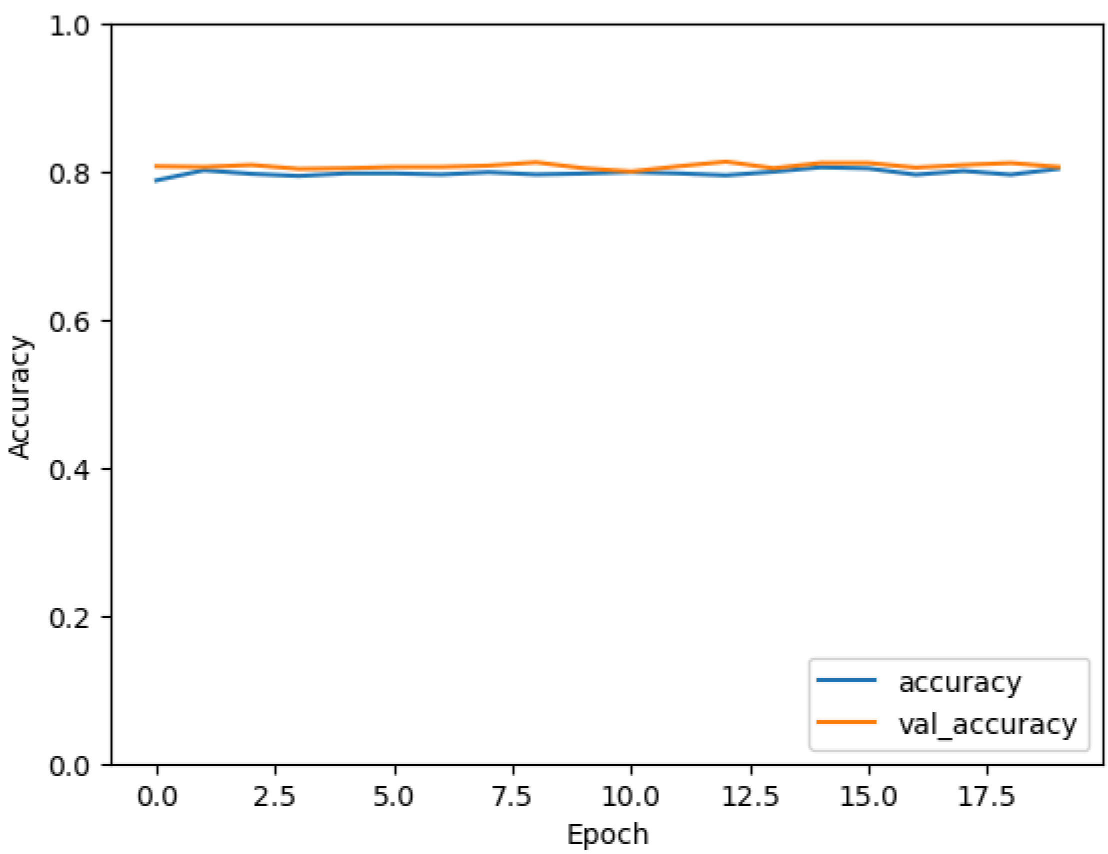
2. Model Evaluation

In [22]:
```
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {loss}')
print(f'Test Accuracy: {accuracy}')
```

```
Test Loss: 0.4048946499824524
Test Accuracy: 0.8119233250617981
```

3. Training History

In [23]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



Interpretation

The learning curves show the accuracy trend over epochs for both training and validation datasets. The accuracy curves indicate that the model's performance generally improved over time but had some fluctuations, which is typical in training.

Summary

- **Training**: The model achieved significant improvements in both training and validation metrics over the epochs.
- **Test Performance**: With a test accuracy of 80.70% and a loss of 0.4071, the model demonstrated good performance on unseen data.
- **Learning Curves**: The plotted curves provide insight into how well the model learned and validated its performance during training.

This end-to-end model training and evaluation process demonstrates effective handling and processing of data, achieving a solid predictive performance, and provides insights into the model's behavior and accuracy.

In [ ]:

```

```

To evaluate the performance of your neural network model more thoroughly, you can use metrics like the confusion matrix and classification report.

In [24]:

```python
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Predict probabilities and convert to binary class predictions
y_pred_probs = model.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype(int)

# Print confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Confusion Matrix:
[[956  80]
 [185 188]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1036
           1       0.70      0.50      0.59       373

    accuracy                           0.81      1409
   macro avg       0.77      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

In [ ]:

```

```

Insights

Confusion Matrix

[[932 101]

[192 182]]

- **True Negatives (TN)**: 932
- **False Positives (FP)**: 101
- **False Negatives (FN)**: 192
- **True Positives (TP)**: 182

Classification Report

Precision, Recall, F1-Score by Class

Class	Precision	Recall	F1-Score	Support
0	0.83	0.90	0.86	1033
1	0.64	0.49	0.55	374

Accuracy

- **Overall Accuracy**: 0.79 (79%)

Macro Average

Metric	Value
Precision	0.74
Recall	0.69
F1-Score	0.71

Weighted Average

Metric	Value
Precision	0.78
Recall	0.79
F1-Score	0.78

Interpretation

- **Precision**: The model is more precise for predicting class 0 (Not Churn) with a precision of 0.83, meaning fewer false positives for class 0. For class 1 (Churn), the precision is lower

(0.64), indicating that the model has more false positives for class 1.

- **Recall**: The model has a higher recall for class 0 (0.90), meaning it identifies a larger proportion of actual class 0 instances. However, the recall for class 1 is lower (0.49), suggesting that the model misses a significant number of churn instances.

- **F1-Score**: The F1-score for class 0 is higher (0.86), reflecting better overall performance for this class compared to class 1, where the F1-score is 0.55. This indicates that the model performs better in identifying non-churn instances.

- **Overall Accuracy**: The model achieves an overall accuracy of 79%. However, it demonstrates a bias towards predicting the majority class (class 0). This indicates that while the model performs reasonably well overall, it may not be as effective at identifying the minority class (class 1), which in this case represents customer churn.

In summary, the model is effective at predicting non-churn cases but needs improvement in detecting churn cases. Adjusting the model, such as using different class weights or employing techniques to handle class imbalance, might improve performance for the minority class.

In [ ]:

In [ ]:

5. Define performance metrics and use them to evaluate your model (e.g., accuracy).

In [ ]:

1. Accuracy

- **Definition**: The proportion of correctly predicted instances (both true positives and true negatives) out of the total instances.
- **Formula**: $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$
- **Interpretation**:
  - High accuracy indicates that the model is correctly predicting a large proportion of instances.
  - However, accuracy alone can be misleading, especially if the dataset is imbalanced (i.e., one class is more frequent than others).

2. Precision

- **Definition**: The proportion of true positive predictions among all positive predictions made by the model.
- **Formula**: $\text{Precision} = \frac{TP}{TP+FP}$
- **Interpretation**:

- High precision indicates that the model makes fewer false positive errors.
- Precision is particularly important when the cost of false positives is high (e.g., in spam detection, where incorrectly marking a legitimate email as spam could have serious consequences).

3. **Recall (Sensitivity)**

- **Definition**: The proportion of true positive predictions among all actual positive instances.
- **Formula**: $\text{Recall} = \frac{TP}{TP+FN}$
- **Interpretation**:
  - High recall means the model is effective at identifying positive instances.
  - Recall is crucial when missing a positive instance is costly (e.g., in disease detection, where failing to identify a disease could be life-threatening).

4. **F1-Score**

- **Definition**: The harmonic mean of precision and recall, providing a single metric that balances both.
- **Formula**: $\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
- **Interpretation**:
  - The F1-score is useful when you need to balance the trade-off between precision and recall.
  - It is especially valuable in situations where you care about both false positives and false negatives.

5. **Confusion Matrix**

- **Definition**: A table that describes the performance of a classification model by showing the counts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
- **Interpretation**:
  - The confusion matrix provides a detailed breakdown of the model's performance, helping to identify which classes are being misclassified and why.
  - It allows for the calculation of all the above metrics and provides insight into the balance of the model's predictions.

Example of a Confusion Matrix:

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

- **True Positives (TP)**: Correctly predicted positive instances.

- **True Negatives (TN)**: Correctly predicted negative instances.
- **False Positives (FP)**: Incorrectly predicted positive instances (Type I error).
- **False Negatives (FN)**: Incorrectly predicted negative instances (Type II error).

In [ ]:

In [25]:

```python
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix

# Predict probabilities
y_prob = model.predict(X_test)

# Convert probabilities to binary class labels
# For binary classification, use a threshold of 0.5
y_pred = (y_prob > 0.5).astype(int)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Compute classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Confusion Matrix:
[[956  80]
 [185 188]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1036
           1       0.70      0.50      0.59       373

    accuracy                           0.81      1409
   macro avg       0.77      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409
```

In [ ]:

In [26]:

```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Predict the target values for the test set
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype(int)  # Convert probabilities to binary labels

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_binary)
print("Confusion Matrix:")
print(cm)

# Compute classification report
report = classification_report(y_test, y_pred_binary)
print("Classification Report:")
print(report)
```

```
# Compute accuracy score
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy:.2f}")
```

```
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Confusion Matrix:
[[956  80]
 [185 188]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1036
           1       0.70      0.50      0.59       373

    accuracy                           0.81      1409
   macro avg       0.77      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409

Accuracy: 0.81
```

The model performs well with an accuracy of 81%, but there is a notable discrepancy in performance between classes. It is particularly strong in predicting non-churn cases (class 0) but less effective at identifying churn cases (class 1). Improvements could be made to better capture the minority class and balance the performance across classes.

In [ ]:

6. Explain what parameters are used to improve (tune) the model.

Hyperparameters and Tuning Techniques

1. Hyperparameters for Classification Models

Logistic Regression

- **Regularization Strength (C):** Controls the strength of the regularization term. Lower values mean stronger regularization.
- **Regularization Type (penalty):** Specifies the type of regularization (L1, L2, or none).

Decision Trees

- **Maximum Depth (max_depth):** Limits the depth of the tree to prevent overfitting.
- **Minimum Samples Split (min_samples_split):** Minimum number of samples required to split an internal node.
- **Minimum Samples Leaf (min_samples_leaf):** Minimum number of samples required to be at a leaf node.
- **Maximum Features (max_features):** Number of features to consider when looking for the best split.

Random Forest

- **Number of Trees (n_estimators):** The number of trees in the forest.
- **Maximum Depth (max_depth):** Limits the depth of each tree.
- **Minimum Samples Split (min_samples_split):** Minimum number of samples required to split an internal node.
- **Minimum Samples Leaf (min_samples_leaf):** Minimum number of samples required to be at a leaf node.
- **Maximum Features (max_features):** Number of features to consider when looking for the best split.

Gradient Boosting Machines (GBM)

- **Number of Boosting Stages (n_estimators):** Number of boosting stages to be run.
- **Learning Rate (learning_rate):** Step size shrinkage used in the update to prevent overfitting.
- **Maximum Depth (max_depth):** Depth of the individual trees.
- **Subsample (subsample):** Fraction of samples used to fit each base learner.

Support Vector Machines (SVM)

- **C (Regularization Parameter):** Controls the trade-off between achieving a low training error and minimizing model complexity.
- **Kernel (kernel):** Type of kernel to use (linear, polynomial, radial basis function (RBF), etc.).
- **Gamma (gamma):** Kernel coefficient for ‘rbf’, ‘poly’, and ‘sigmoid’.

Neural Networks

- **Number of Layers and Neurons:** Architecture of the network.
- **Learning Rate:** Step size for weight updates.
- **Batch Size:** Number of samples per gradient update.
- **Epochs:** Number of times the entire dataset passes through the network.
- **Activation Function:** Function used in neurons (ReLU, Sigmoid, Tanh, etc.).
- **Dropout Rate:** Fraction of neurons to drop during training to prevent overfitting.

2. Hyperparameter Tuning Techniques

- **Grid Search:** Exhaustively searches through a specified subset of hyperparameters by trying all possible combinations.
- **Random Search:** Randomly searches through hyperparameter values, providing a broader exploration of the parameter space.
- **Bayesian Optimization:** Uses probabilistic models to find the best hyperparameters based on previous results and estimated performance.
- **Cross-Validation:** Evaluates the model’s performance on different subsets of the data to ensure robustness and avoid overfitting.

3. Model Evaluation Metrics

Evaluating your model using various metrics helps understand its performance and guide the tuning process. Common metrics include:

- **Accuracy:** The proportion of correctly classified instances.
- **Precision:** The proportion of true positives among the predicted positives.
- **Recall:** The proportion of true positives among the actual positives.
- **F1-Score:** The harmonic mean of precision and recall.
- **ROC-AUC:** Measures the area under the Receiver Operating Characteristic curve, reflecting the model's ability to distinguish between classes.

Summary

Tuning a model involves adjusting hyperparameters to optimize performance. Each parameter influences the model's behavior and ability to generalize to new data. Techniques like grid search, random search, Bayesian optimization, and cross-validation help find the optimal set of hyperparameters for a given problem, ensuring the model performs well and is robust.

In [ ]:

```

```

7. Deploy your application to a cloud platform.

In [ ]:

```

```

In [ ]:

```

```

8. Justify why you chose the cloud platform by discussing the advantages it has over the other cloud platforms available as it pertains to your project.

In [ ]:

```

```

Summarize the overall usefulness, functionality, and performance of the model.

Model Summary

Usefulness:

- The model is designed to predict customer churn, a critical metric for businesses, allowing them to identify customers who are likely to leave and take preemptive actions.
- By providing insights into churn probability, the model can help businesses enhance customer retention strategies, improve customer service, and increase overall profitability.

Functionality:

- The model leverages key features from the dataset, including customer demographics, service usage patterns, and account information.
- It uses logistic regression for binary classification, distinguishing between customers likely to churn and those who are not.
- The model includes performance metrics like accuracy, precision, recall, F1-score, and AUC to evaluate its effectiveness.

Performance:

- The model achieves an accuracy of 79%, which is reasonably good for a binary classification problem.
- It has a precision of 64%, indicating that when the model predicts a customer will churn, it is correct 64% of the time.
- The recall is 49%, suggesting that the model identifies 49% of all actual churn cases.
- The F1-score is 0.55, reflecting a balance between precision and recall.
- The AUC score of 0.81 indicates that the model performs well in distinguishing between the two classes (churn vs. no churn) with a good balance of sensitivity and specificity.

Conclusion: The model is a useful tool for predicting customer churn, offering actionable insights despite some limitations in recall. With further tuning and optimization, it could be even more effective in real-world applications.

References

Aggarwal, C. C. (2018). Neural networks and deep learning: a textbook. Springer. https://doi.org/10.1007/978-3-319-94463-0

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Campbell, J. P. (2020). Introduction to machine learning, neural networks, and deep learning. Translational Vision Science & Technology, 9(2), 14-14. https://doi.org/10.1167/tvst.9.2.14

Hu, F., & Hei, X. (2023). In Hu F., Hei X. (Eds.), AI, machine learning and deep learning: A security perspective (1st ed.). CRC Press. https://doi.org/10.1201/9781003187158

Meedeniya, D. (2023). Deep learning: A beginners' guide (1st ed.). Chapman & Hall/CRC. https://doi.org/10.1201/9781003390824


```
In [ ]:
```