In [169]:
```python
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, cross_val_score, GridSea
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

In [170]:
```python
# Load the dataset
Telco_df = pd.read_csv('Telco-Customer-Churn.csv')
```

In [171]:
```python
print(Telco_df.head())  # Display the first few rows
```

```
   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]
```

In [172]:
```
print (Telco_df.shape)
Telco_df.isnull().sum()
```

```
(7043, 21)
```

Out[172]:
```
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
```

In [173]:
```python
print(Telco_df.info())  # Summary of the dataset
```

```
<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
```

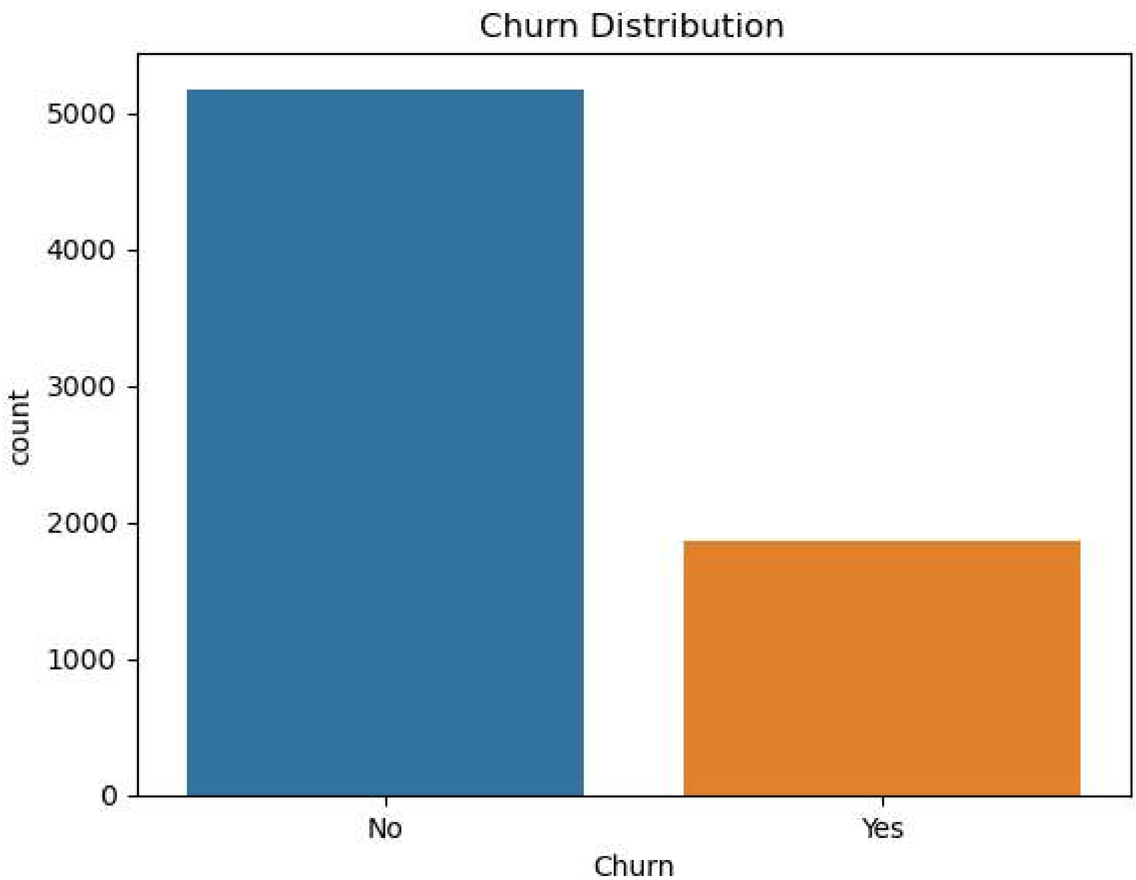
In [174]:
```python
# Check for duplicate rows
Telco_df.duplicated().sum()
```

Out[174]:
```
0
```

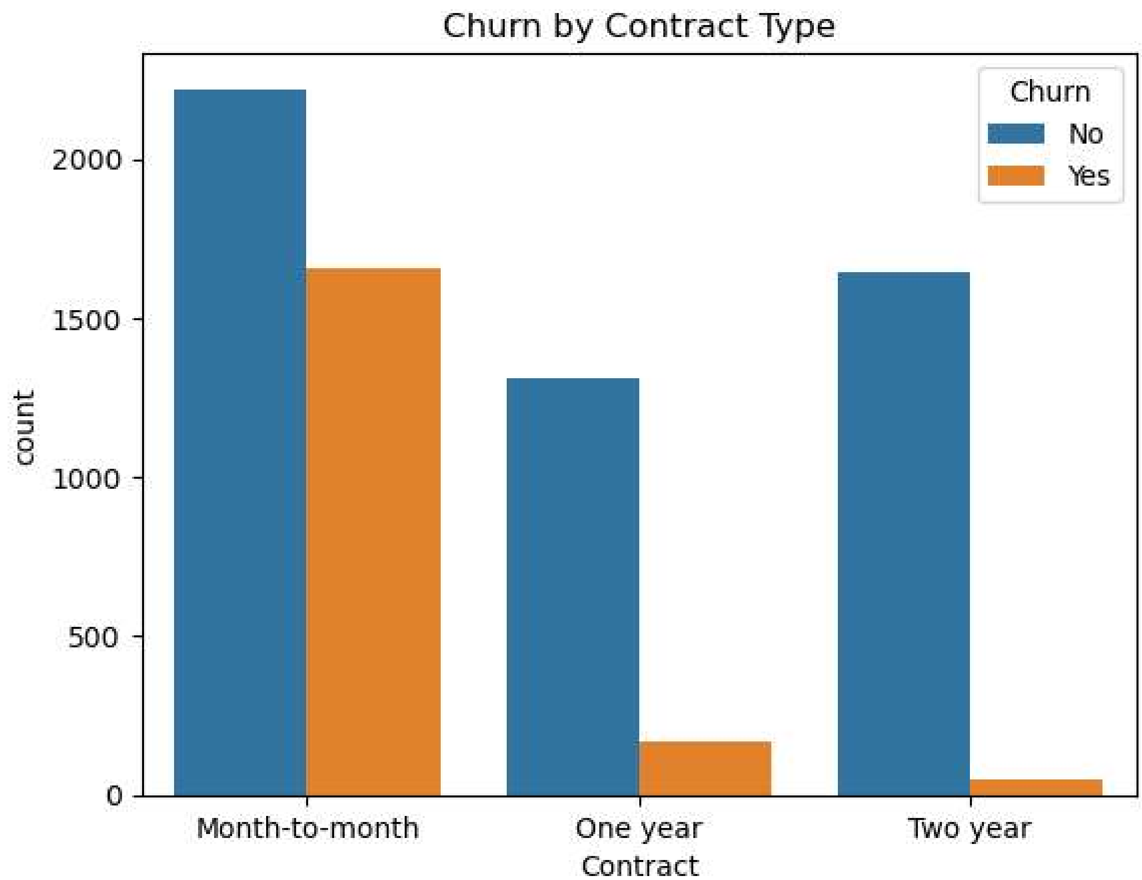
# VISUALIZATION

In [175]:

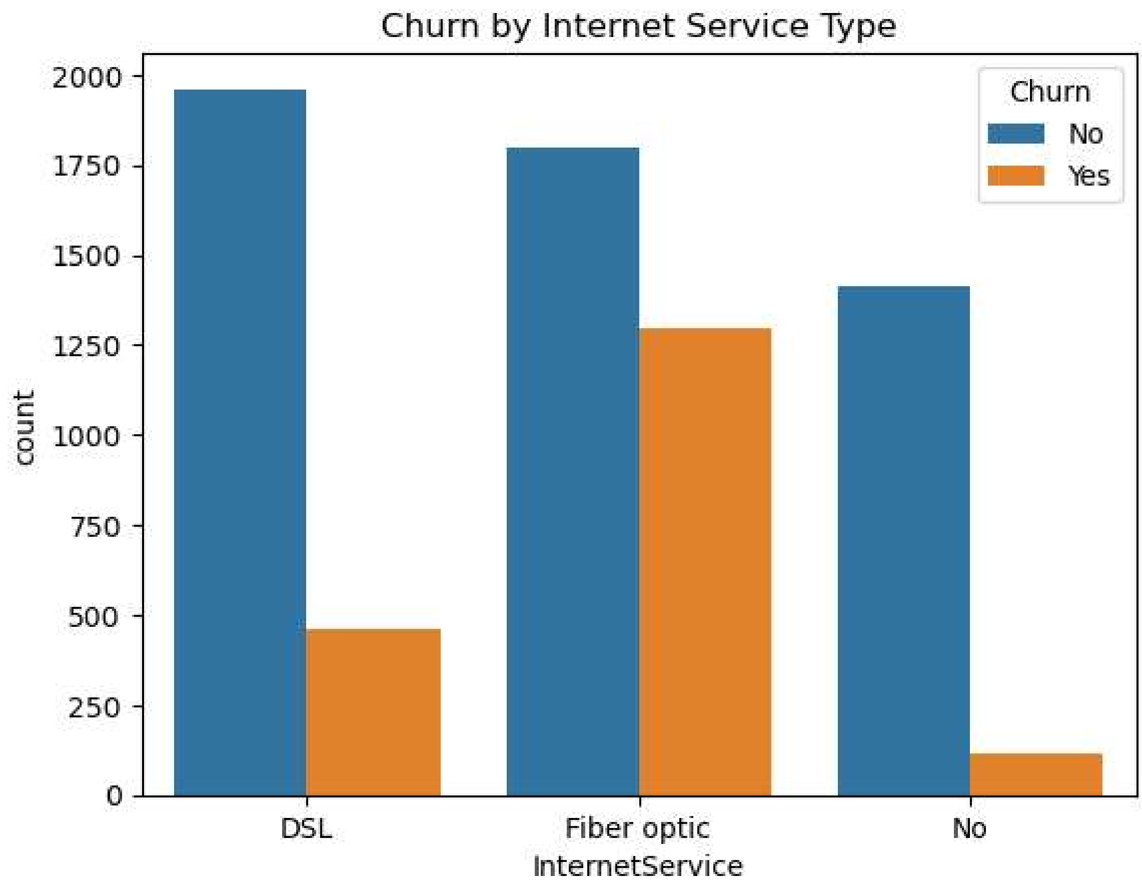
```
# Visualization 1: Churn distribution
sns.countplot(x='Churn', data=Telco_df)
plt.title('Churn Distribution')
plt.show()
```



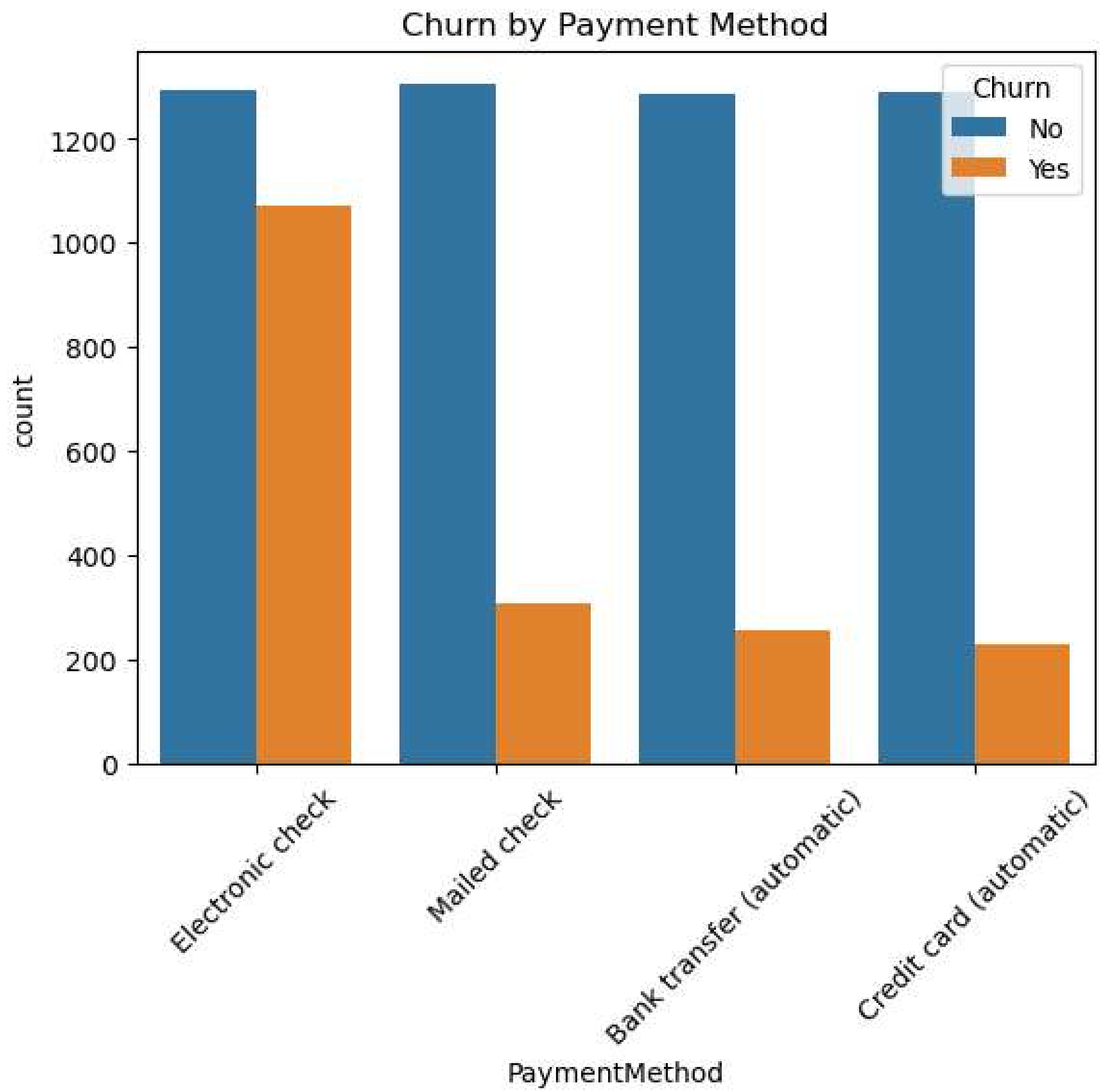
In [176]:
```
# Visualization 2: Churn by Contract type
sns.countplot(x='Contract', hue='Churn', data=Telco_df)
plt.title('Churn by Contract Type')
plt.show()
```



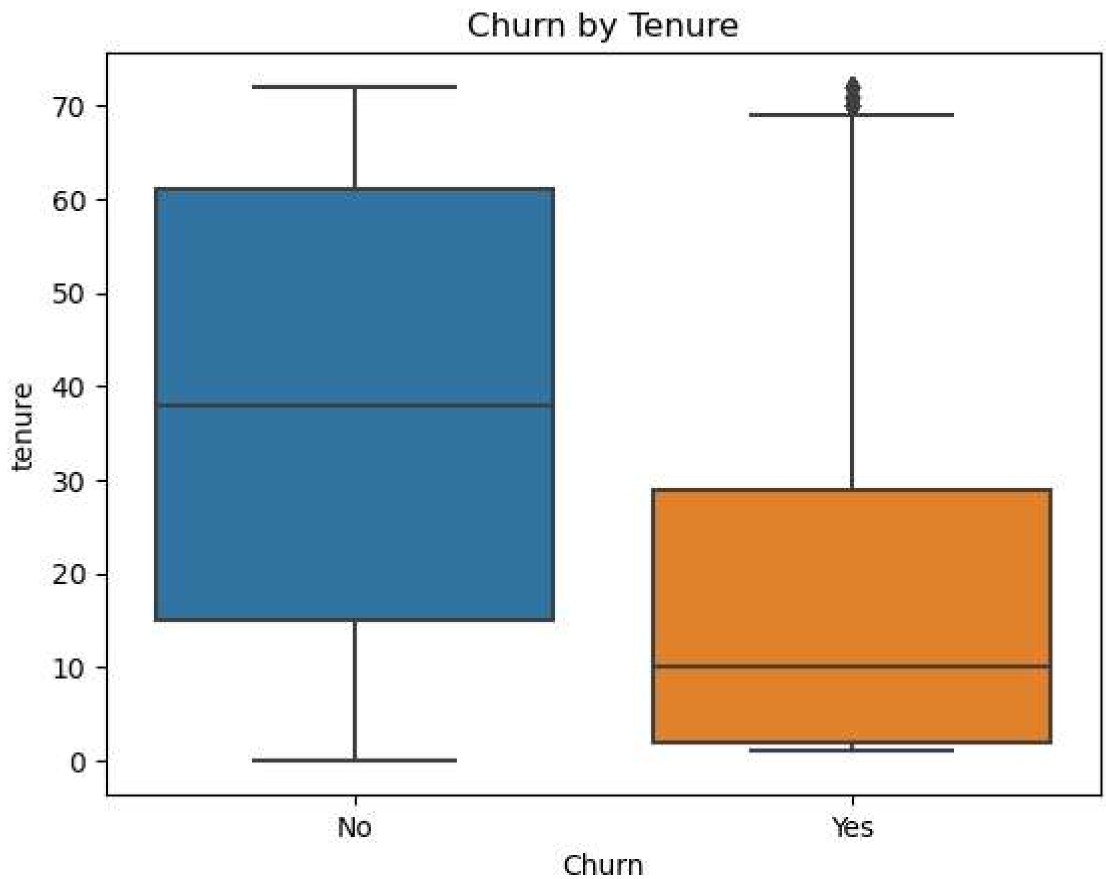
In [177]:
```
# Visualization 3: Churn by Internet Service type
sns.countplot(x='InternetService', hue='Churn', data=Telco_df)
plt.title('Churn by Internet Service Type')
plt.show()
```



In [178]:
```
# Visualization 4: Churn by Payment Method
sns.countplot(x='PaymentMethod', hue='Churn', data=Telco_df)
plt.title('Churn by Payment Method')
plt.xticks(rotation=45)
plt.show()
```



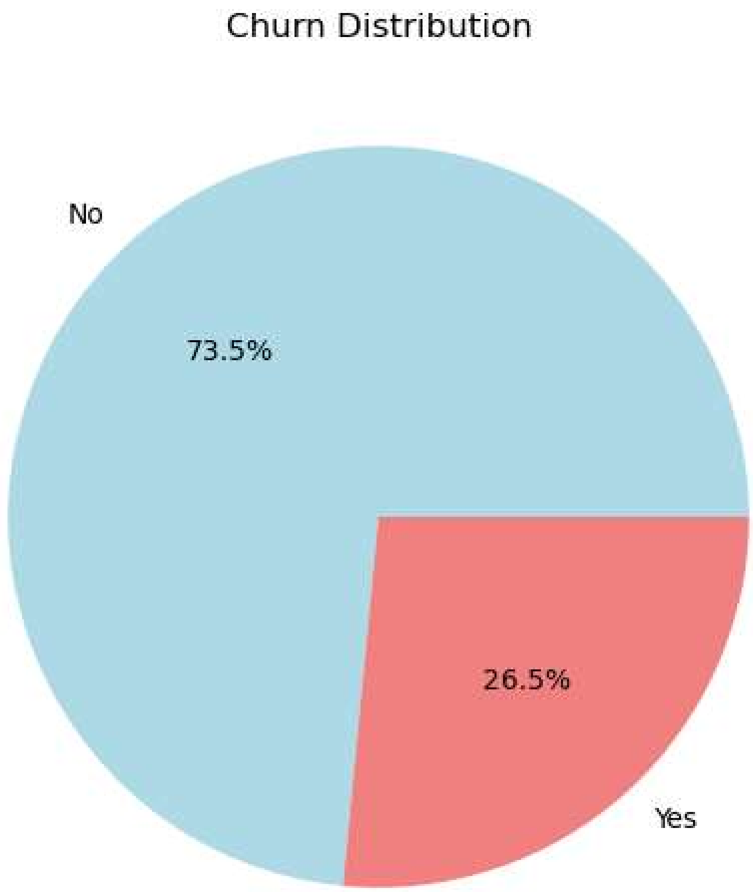
In [179]:
```
# Visualization 5: Churn by Tenure
sns.boxplot(x='Churn', y='tenure', data=Telco_df)
plt.title('Churn by Tenure')
plt.show()
```



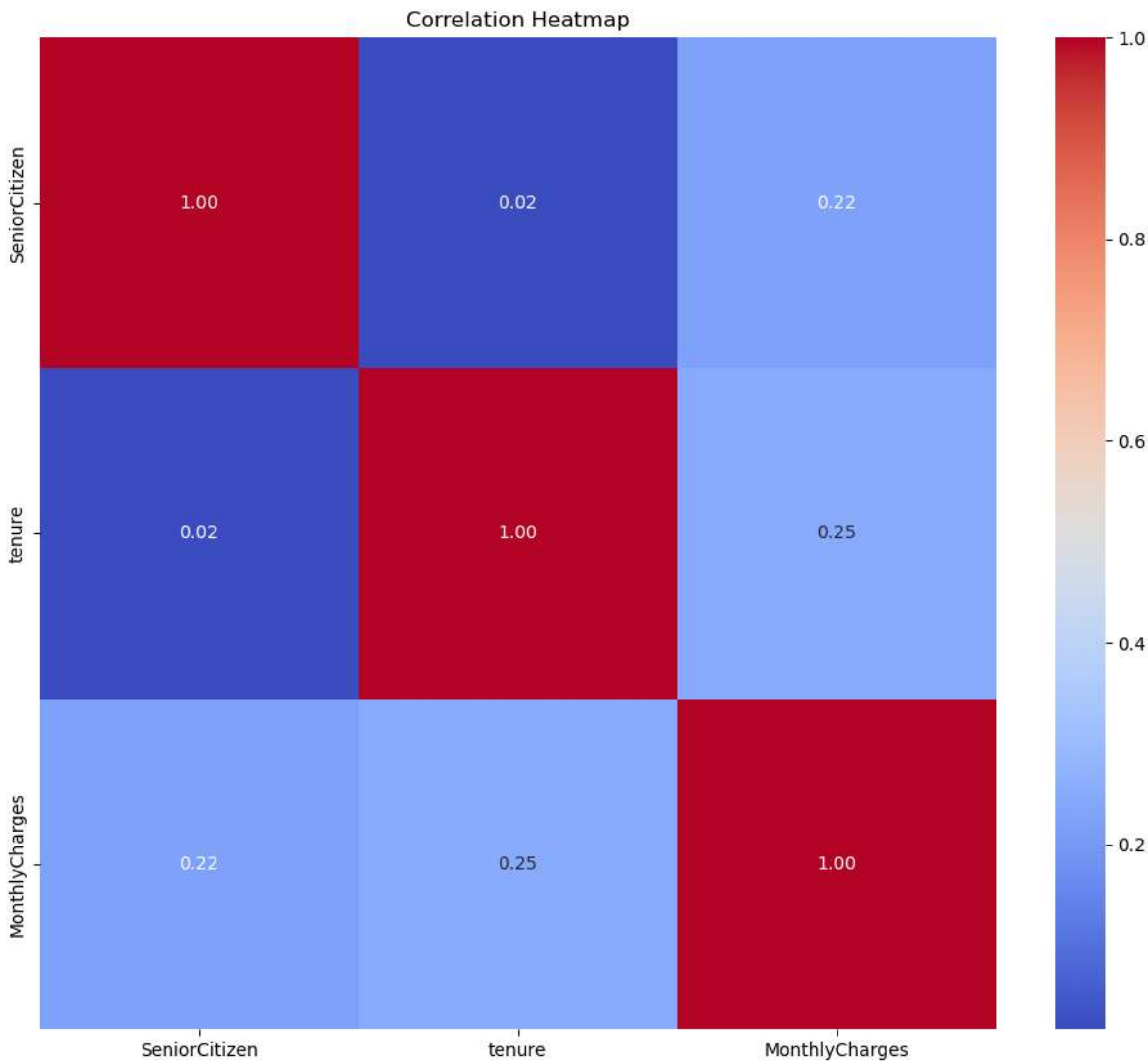

so we are not taking any action to handle the outliers,because a customer can be in the company for many months. Even though the customer had been in the company for a long time, we should consider them as part of our analysis.

In [180]:

```
# 6. Pie chart for churn distribution
plt.figure(figsize=(6, 6))
plt.pie(Telco_df['Churn'].value_counts(), labels=['No', 'Yes'], autopct='%1.1f
plt.title('Churn Distribution')
plt.show()
```



In [181]:
```
# Visualization 7: Correlation Heatmap
correlation_matrix = Telco_df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```




# DATA CLEANING

In [182]:
```
# Assuming 'No internet service' means the customer does not have that service
cols_fillna = ['MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProte
               'StreamingTV', 'StreamingMovies']
Telco_df[cols_fillna] = Telco_df[cols_fillna].replace('No internet service', '
```

In [183]:
```python
# Assuming 'No internet service' means the customer does not have that service
cols_fillna = ['MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProte
               'StreamingTV', 'StreamingMovies']
Telco_df[cols_fillna] = Telco_df[cols_fillna].replace('No phone service', 'No'
```

In [184]:
```python
Telco_df
```

Out[184]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLir |
|---|---|---|---|---|---|---|---|---|
| **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 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **7038** | 6840-RESVB | Male | 0 | Yes | Yes | 24 | Yes | |
| **7039** | 2234-XADUH | Female | 0 | Yes | Yes | 72 | Yes | |
| **7040** | 4801-JZAZL | Female | 0 | Yes | Yes | 11 | No | |
| **7041** | 8361-LTMKD | Male | 1 | Yes | No | 4 | Yes | |
| **7042** | 3186-AJIEK | Male | 0 | No | No | 66 | Yes | |

7043 rows × 21 columns

In [185]:
```python
# Drop irrelevant columns like customerID as it does not contribute to the pre
Telco_df.drop(columns=['customerID'], inplace=True)
```

# Feature Engineering

In [186]:
```python
# Convert 'Churn' column to binary values
Telco_df['Churn'] = Telco_df['Churn'].map({'Yes': 1, 'No': 0})
```

```
In [187]: # Create binary features for 'Partner', 'Dependents', 'PhoneService', 'PaperLe
          Telco_df['Partner'] = Telco_df['Partner'].map({'Yes': 1, 'No': 0})
          Telco_df['Dependents'] = Telco_df['Dependents'].map({'Yes': 1, 'No': 0})
          Telco_df['PhoneService'] = Telco_df['PhoneService'].map({'Yes': 1, 'No': 0})
          Telco_df['PaperlessBilling'] = Telco_df['PaperlessBilling'].map({'Yes': 1, 'No
```

```
In [188]: Telco_df
```

Out[188]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetS |
|---|---|---|---|---|---|---|---|---|
| **0** | Female | 0 | 1 | 0 | 1 | 0 | No | |
| **1** | Male | 0 | 0 | 0 | 34 | 1 | No | |
| **2** | Male | 0 | 0 | 0 | 2 | 1 | No | |
| **3** | Male | 0 | 0 | 0 | 45 | 0 | No | |
| **4** | Female | 0 | 0 | 0 | 2 | 1 | No | Fibe |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **7038** | Male | 0 | 1 | 1 | 24 | 1 | Yes | |
| **7039** | Female | 0 | 1 | 1 | 72 | 1 | Yes | Fibe |
| **7040** | Female | 0 | 1 | 1 | 11 | 0 | No | |
| **7041** | Male | 1 | 1 | 0 | 4 | 1 | Yes | Fibe |
| **7042** | Male | 0 | 0 | 0 | 66 | 1 | No | Fibe |

7043 rows × 20 columns

In [189]:
```
Telco_df
```

Out[189]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetS |
|---|---|---|---|---|---|---|---|---|
| **0** | Female | 0 | 1 | 0 | 1 | 0 | No | |
| **1** | Male | 0 | 0 | 0 | 34 | 1 | No | |
| **2** | Male | 0 | 0 | 0 | 2 | 1 | No | |
| **3** | Male | 0 | 0 | 0 | 45 | 0 | No | |
| **4** | Female | 0 | 0 | 0 | 2 | 1 | No | Fibe |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **7038** | Male | 0 | 1 | 1 | 24 | 1 | Yes | |
| **7039** | Female | 0 | 1 | 1 | 72 | 1 | Yes | Fibe |
| **7040** | Female | 0 | 1 | 1 | 11 | 0 | No | |
| **7041** | Male | 1 | 1 | 0 | 4 | 1 | Yes | Fibe |
| **7042** | Male | 0 | 0 | 0 | 66 | 1 | No | Fibe |

7043 rows × 20 columns

In [190]:
```
Telco_df = pd.get_dummies(Telco_df, columns=['gender', 'MultipleLines', 'Inter
                                             'OnlineSecurity', 'OnlineBackup',
                                             'TechSupport', 'StreamingTV', 'Str
                                             'Contract'], drop_first=True)
```

In [191]:
```
Telco_df
```

Out[191]:

| | SeniorCitizen | Partner | Dependents | tenure | PhoneService | PaperlessBilling | PaymentMetho |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 1 | 0 | 1 | Electronic chec |
| **1** | 0 | 0 | 0 | 34 | 1 | 0 | Mailed chec |
| **2** | 0 | 0 | 0 | 2 | 1 | 1 | Mailed chec |
| **3** | 0 | 0 | 0 | 45 | 0 | 0 | Bank transfe (automatic |
| **4** | 0 | 0 | 0 | 2 | 1 | 1 | Electronic chec |
| **...** | ... | ... | ... | ... | ... | ... | . |
| **7038** | 0 | 1 | 1 | 24 | 1 | 1 | Mailed chec |
| **7039** | 0 | 1 | 1 | 72 | 1 | 1 | Credit car (automatic |
| **7040** | 0 | 1 | 1 | 11 | 0 | 1 | Electronic chec |
| **7041** | 1 | 1 | 0 | 4 | 1 | 1 | Mailed chec |
| **7042** | 0 | 0 | 0 | 66 | 1 | 1 | Bank transfe (automatic |

7043 rows × 22 columns

In [192]:
```
# Convert TotalCharges to numeric
Telco_df['TotalCharges'] = pd.to_numeric(Telco_df['TotalCharges'], errors='coe
```

In [193]:
```
#calculate the ratio of MonthlyCharges to TotalCharges to see the average mont
Telco_df['AvgMonthlySpending'] = Telco_df['TotalCharges'] / Telco_df['tenure']
```

In [194]:
```
Telco_df
```

Out[194]:

| | SeniorCitizen | Partner | Dependents | tenure | PhoneService | PaperlessBilling | PaymentMetho |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 1 | 0 | 1 | Electronic chec |
| **1** | 0 | 0 | 0 | 34 | 1 | 0 | Mailed chec |
| **2** | 0 | 0 | 0 | 2 | 1 | 1 | Mailed chec |
| **3** | 0 | 0 | 0 | 45 | 0 | 0 | Bank transfe (automatic |
| **4** | 0 | 0 | 0 | 2 | 1 | 1 | Electronic chec |
| **...** | ... | ... | ... | ... | ... | ... | . |
| **7038** | 0 | 1 | 1 | 24 | 1 | 1 | Mailed chec |
| **7039** | 0 | 1 | 1 | 72 | 1 | 1 | Credit car (automatic |
| **7040** | 0 | 1 | 1 | 11 | 0 | 1 | Electronic chec |
| **7041** | 1 | 1 | 0 | 4 | 1 | 1 | Mailed chec |
| **7042** | 0 | 0 | 0 | 66 | 1 | 1 | Bank transfe (automatic |

7043 rows × 23 columns

# Feature Scaling

In [195]:
```
# Feature Scaling (if required)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
Telco_df[numerical_features] = scaler.fit_transform(Telco_df[numerical_feature
```

In [196]:
```python
Telco_df
```

Out[196]:

| | SeniorCitizen | Partner | Dependents | tenure | PhoneService | PaperlessBilling | PaymentMet |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | -1.277445 | 0 | 1 | Electronic c |
| **1** | 0 | 0 | 0 | 0.066327 | 1 | 0 | Mailed c |
| **2** | 0 | 0 | 0 | -1.236724 | 1 | 1 | Mailed c |
| **3** | 0 | 0 | 0 | 0.514251 | 0 | 0 | Bank trai (autom |
| **4** | 0 | 0 | 0 | -1.236724 | 1 | 1 | Electronic c |
| **...** | ... | ... | ... | ... | ... | ... | |
| **7038** | 0 | 1 | 1 | -0.340876 | 1 | 1 | Mailed c |
| **7039** | 0 | 1 | 1 | 1.613701 | 1 | 1 | Credit (autom |
| **7040** | 0 | 1 | 1 | -0.870241 | 0 | 1 | Electronic c |
| **7041** | 1 | 1 | 0 | -1.155283 | 1 | 1 | Mailed c |
| **7042** | 0 | 0 | 0 | 1.369379 | 1 | 1 | Bank trai (autom |

7043 rows × 23 columns

In [197]:
```python
# Check for missing values
Telco_df.isnull().sum()
```

Out[197]:
```
SeniorCitizen                0
Partner                      0
Dependents                   0
tenure                       0
PhoneService                 0
PaperlessBilling             0
PaymentMethod                0
MonthlyCharges               0
TotalCharges                11
Churn                        0
gender_Male                  0
MultipleLines_Yes            0
InternetService_Fiber optic  0
InternetService_No           0
OnlineSecurity_Yes           0
OnlineBackup_Yes             0
DeviceProtection_Yes         0
TechSupport_Yes              0
StreamingTV_Yes              0
StreamingMovies_Yes          0
Contract_One year            0
Contract_Two year            0
AvgMonthlySpending          11
dtype: int64
```

# DEALING WITH MISSING VALUES

In [198]:
```
total_charges_mean = Telco_df['TotalCharges'].mean()
#Replace the missing values in the 'TotalCharges' column with the calculated m
Telco_df['TotalCharges'].fillna(total_charges_mean, inplace=True)
```

In [199]:
```
#dealing with the missing values using the mean
avg_monthly_spending_mean = Telco_df['AvgMonthlySpending'].mean()
#Replace the missing values in the 'AvgMonthlySpending' column with the calcul
Telco_df['AvgMonthlySpending'].fillna(avg_monthly_spending_mean, inplace=True)
```

In [200]:
```
Telco_df.isnull().sum()
```

Out[200]:
```
SeniorCitizen                  0
Partner                        0
Dependents                     0
tenure                         0
PhoneService                   0
PaperlessBilling               0
PaymentMethod                  0
MonthlyCharges                 0
TotalCharges                   0
Churn                          0
gender_Male                    0
MultipleLines_Yes              0
InternetService_Fiber optic    0
InternetService_No             0
OnlineSecurity_Yes             0
OnlineBackup_Yes               0
DeviceProtection_Yes           0
TechSupport_Yes                0
StreamingTV_Yes                0
StreamingMovies_Yes            0
Contract_One year              0
Contract_Two year              0
AvgMonthlySpending             0
dtype: int64
```

# VALIDATION SPLIT

In [201]:
```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from dmba import classificationSummary
```

In [202]:
```
ta into features (X) and target (y)
'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'PaperlessE
Churn']
```

In [203]:
```
# Splitting the dataset into training and testing sets
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.2, rar
```

# NEURAL NETWORK

In [204]:
```
clf = MLPClassifier(hidden_layer_sizes=(200, 100), activation='logistic', solv
```

In [205]:
```
clf.fit(train_X, train_y.values)
```

Out[205]:
```
MLPClassifier(activation='logistic', batch_size=256,
              hidden_layer_sizes=(200, 100), max_iter=2000)
```

In [206]:
```
clf.predict(X)
```

Out[206]:
```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

In [207]:

```
#Network structure
print('Intercepts')
print(clf.intercepts_)
```

```
Intercepts
[array([-0.07711076,  0.04778391, -0.11437006,  0.16322386, -0.05399742,
        0.06280103, -0.06295765, -0.13327462, -0.07654465, -0.0741688 ,
        0.05018679, -0.01543338, -0.12097818, -0.08782992, -0.06460622,
       -0.08328986, -0.02982481, -0.01698305,  0.13822986, -0.0878224 ,
        0.07614872,  0.10424416,  0.02954134,  0.04933228,  0.14071913,
        0.04604924, -0.12765704,  0.09622789,  0.04664133,  0.04145939,
        0.06980371,  0.07832422,  0.10275634,  0.01520635, -0.12026865,
       -0.06718483, -0.05889039, -0.08507691, -0.12489526,  0.06690968,
       -0.04913522,  0.11136058, -0.06579893,  0.05574222,  0.07085085,
       -0.09586987, -0.07378966, -0.09442997,  0.01864218, -0.04196119,
        0.08233836,  0.09994046,  0.10435348, -0.01547792,  0.08910943,
       -0.14863354, -0.04997154, -0.05405346,  0.10181535,  0.07388298,
        0.07375984,  0.09568427, -0.02953724, -0.01568243, -0.00917402,
        0.06416815,  0.06843717,  0.03059337, -0.08253441,  0.05186856,
        0.05507195,  0.03297723, -0.09402142,  0.0479809 ,  0.02748668,
        0.01661871,  0.01167706,  0.04206024,  0.06373487,  0.14502593,
       -0.04502546,  0.16194255,  0.06799084,  0.01056557,  0.09549596,
       -0.05057188, -0.07249029, -0.06127656,  0.08602927, -0.03266914,
        0.03470659, -0.09969943,  0.10521663, -0.04670525, -0.0260195 ,
       -0.00987203,  0.03982132,  0.06858372,  0.07712721,  0.06821213,
       -0.05507857, -0.03856469,  0.0637783 , -0.10458279, -0.00100946,
       -0.12456497,  0.11352142,  0.01896289,  0.14389551,  0.07789877,
        0.00304002, -0.10607939,  0.03724744, -0.09502081, -0.0121915 ,
        0.01661286, -0.08669511, -0.05088586,  0.03881723,  0.0180952 ,
        0.1437362 , -0.09645999, -0.01274096, -0.00719962,  0.03376839,
        0.02546789, -0.12776966,  0.12671569,  0.05497335, -0.05079617,
       -0.0181749 ,  0.11890329,  0.04643214,  0.0549724 ,  0.0188724 ,
       -0.057389  , -0.02018207,  0.00730316,  0.11797376,  0.05259888,
       -0.0557688 , -0.08715235, -0.07190564, -0.05273055,  0.01922235,
       -0.00453224,  0.11673014,  0.08466116,  0.1150289 ,  0.04381728,
       -0.01404847, -0.07236474, -0.08352699,  0.11910235, -0.08244437,
        0.05187741,  0.03364077,  0.09214004,  0.03660464,  0.04965636,
       -0.06029192,  0.08509258, -0.05481139, -0.06163182, -0.10427193,
        0.02788885,  0.04705745,  0.05053703,  0.0755811 ,  0.09546765,
        0.00078261, -0.08677788, -0.05073877,  0.04878723,  0.11729899,
        0.0511977 ,  0.03651654, -0.0076614 ,  0.06618771,  0.12390693,
        0.03620347, -0.01704678,  0.10710408,  0.07250365, -0.03203895,
       -0.1169381 ,  0.05736459, -0.0535965 ,  0.02953488,  0.03588595,
        0.02550196, -0.09880053, -0.01919622, -0.03628041, -0.08358034,
        0.04978479,  0.13139801, -0.07146026,  0.02972915, -0.00810207]), arr
ay([ 5.32295116e-02, -8.32804305e-02,  2.10037985e-03,  4.53619117e-02,
       -9.14240261e-02, -7.31747962e-02,  6.10768135e-02, -8.30551147e-02,
       -3.70508351e-02,  5.03482994e-02,  1.13100221e-02, -7.94008044e-02,
       -7.10224817e-02, -7.21744390e-03, -4.76103620e-02,  4.00430714e-02,
       -4.25554549e-02,  5.78202683e-02, -6.28524628e-02, -5.38364907e-02,
       -6.92670029e-02, -1.12842435e-01, -1.39195298e-02, -5.78254708e-02,
        3.63953678e-02, -9.04994729e-02, -1.00320687e-02, -2.07142544e-02,
       -4.42791536e-02, -7.56208117e-02,  7.68847492e-02, -7.29476919e-02,
       -1.05801169e-01, -4.95628185e-02,  4.87847088e-02, -9.71934795e-02,
        3.88843209e-02, -1.08549199e-02, -2.20046808e-03,  4.27585415e-02,
        4.22117290e-02, -7.69699988e-02, -6.14362826e-02,  2.05567835e-02,
       -1.56535577e-02, -9.41029854e-03, -1.86458573e-02,  5.08265956e-02,
        5.43924321e-02,  2.51265149e-02, -6.77285137e-02, -8.55760263e-03,
       -7.45661044e-02,  3.83314009e-02,  1.75649614e-02,  1.60116824e-03,
        7.64290969e-02,  1.20938913e-03, -4.81318841e-02, -4.49948585e-02,
        5.42117901e-02, -5.65470492e-02, -3.74689535e-03, -5.52694032e-02,
```

```
         2.27156884e-02, -1.04961403e-01, -3.23709447e-02, -4.92558876e-02,
         2.13948375e-03, -7.97998621e-02, -2.55598297e-02,  8.38673717e-02,
        -1.69493803e-03, -2.39888721e-02, -9.73972094e-02, -2.92642318e-03,
         7.93858524e-03, -5.23213698e-02,  3.34637271e-02, -2.42982546e-02,
         5.70511069e-02, -4.16644537e-02, -6.34847691e-03, -7.02037905e-02,
         2.45368582e-02, -1.03574353e-01, -3.19790869e-02, -4.10134984e-02,
         7.81284562e-02, -4.51689461e-02, -3.72090685e-02, -6.41228594e-02,
        -8.50386348e-02, -5.33562435e-02,  8.88003827e-03, -8.31653411e-02,
         6.06990400e-05,  4.54496327e-02,  6.58351156e-02, -4.34009324e-02]),
array([-0.03900596])]
```

In [208]:

```python
# training performance
classificationSummary(train_y, clf.predict(train_X))
```

```
Confusion Matrix (Accuracy 0.7989)

       Prediction
Actual    0    1
     0 3850  288
     1  845  651
```

In [209]:

```python
# validation performance
classificationSummary(valid_y, clf.predict(valid_X))
```

```
Confusion Matrix (Accuracy 0.8098)

       Prediction
Actual   0   1
     0 960  76
     1 192 181
```

# RANDOM FOREST

In [210]:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

In [211]:

```python
rf = RandomForestClassifier(random_state=1)
rf.fit(train_X, train_y)
```

Out[211]:

```
RandomForestClassifier(random_state=1)
```

In [212]:

```python
rf = RandomForestClassifier(random_state=1)
rf.fit(train_X, train_y)
```

Out[212]:

```
RandomForestClassifier(random_state=1)
```

In [213]:
```
train_pred = rf.predict(train_X)
valid_pred = rf.predict(valid_X)
```

In [214]:
```
# Calculate Metrics for Training Data
train_cm = confusion_matrix(train_y, train_pred)
train_accuracy = accuracy_score(train_y, train_pred)
train_precision = precision_score(train_y, train_pred)
train_recall = recall_score(train_y, train_pred)
train_f1_score = f1_score(train_y, train_pred)
```

In [215]:
```
# Calculate Metrics for Validation Data
valid_cm = confusion_matrix(valid_y, valid_pred)
valid_accuracy = accuracy_score(valid_y, valid_pred)
valid_precision = precision_score(valid_y, valid_pred)
valid_recall = recall_score(valid_y, valid_pred)
valid_f1_score = f1_score(valid_y, valid_pred)
```

In [216]:
```
# Print the Metrics
print("Random Forest Metrics:")
print("Training Accuracy:", train_accuracy)
print("Training Precision:", train_precision)
print("Training Recall:", train_recall)
print("Training F1 Score:", train_f1_score)
print("\nValidation Accuracy:", valid_accuracy)
print("Validation Precision:", valid_precision)
print("Validation Recall:", valid_recall)
print("Validation F1 Score:", valid_f1_score)
```

```
Random Forest Metrics:
Training Accuracy: 0.9978700745473909
Training Precision: 0.9986559139784946
Training Recall: 0.9933155080213903
Training F1 Score: 0.9959785522788204

Validation Accuracy: 0.7977288857345636
Validation Precision: 0.6617647058823529
Validation Recall: 0.48257372654155495
Validation F1 Score: 0.5581395348837209
```

In [217]:
```
# Print the Confusion Matrices
print("\nTraining Confusion Matrix:")
print(train_cm)
print("\nValidation Confusion Matrix:")
print(valid_cm)
```

```
Training Confusion Matrix:
[[4136    2]
 [  10 1486]]

Validation Confusion Matrix:
[[944  92]
 [193 180]]
```

# LOGISTIC REGRESSION

In [218]:
```
#Import necessary models and evaluation metrics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

In [219]:
```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(train_X, train_y)
y_pred_logreg = logreg.predict(valid_X)
```

In [220]:
```
# Evaluate the models
print("Logistic Regression Metrics:")
print("Accuracy:", accuracy_score(valid_y, y_pred_logreg))
print("Precision:", precision_score(valid_y, y_pred_logreg))
print("Recall:", recall_score(valid_y, y_pred_logreg))
print("F1 Score:", f1_score(valid_y, y_pred_logreg))
```

```
Logistic Regression Metrics:
Accuracy: 0.815471965933286
Precision: 0.6805111821086262
Recall: 0.5710455764075067
F1 Score: 0.6209912536443148
```

# In summary, based on the provided metrics, the Logistic Regression model generally performs better than the other algorithms.

In [ ]:
```

```