telcoCustomerChurn

November 18, 2023

[127]:
```
# a)
# Importing needed libraries for data preprocessing, analysis and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
from sklearn.preprocessing import MinMaxScaler
```

[103]:
```
# b)
#Loading the dataset using pd.read_csv
customerChurn = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

[104]:
```
#Displaying the first five rows. This is to confirm that the data has been␣
↪loaded
customerChurn.head()
```

[104]:
```
   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]
```

[105]:

```python
customerChurn.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
```

This output shows information about the dataset such as: total number of rows, data type of each column, the number of non-values for each column, and the amount of memory used by the dataset.

[106]:
```
# d)
#Checking for missing values
missingPercentageOfValues = (customerChurn.isnull().mean() * 100).round(2)
print(f'Percentage of Missing Values in Each Column:
↪\n{missingPercentageOfValues}')
```

```
Percentage of Missing Values in Each Column:
customerID          0.0
gender              0.0
SeniorCitizen       0.0
Partner             0.0
Dependents          0.0
tenure              0.0
PhoneService        0.0
MultipleLines       0.0
InternetService     0.0
OnlineSecurity      0.0
OnlineBackup        0.0
DeviceProtection    0.0
TechSupport         0.0
StreamingTV         0.0
StreamingMovies     0.0
Contract            0.0
PaperlessBilling    0.0
PaymentMethod       0.0
MonthlyCharges      0.0
TotalCharges        0.0
Churn               0.0
dtype: float64
```

Judging from the above output, the percentage of missing values in each column is 0.

[107]:
```
# c)
# Determining the number of features and observations
num_observations = customerChurn.shape[0]
num_features = customerChurn.shape[1]
print(num_observations)
```

```
7043
```

[108]:
```
print(num_features)
```

```
21
```

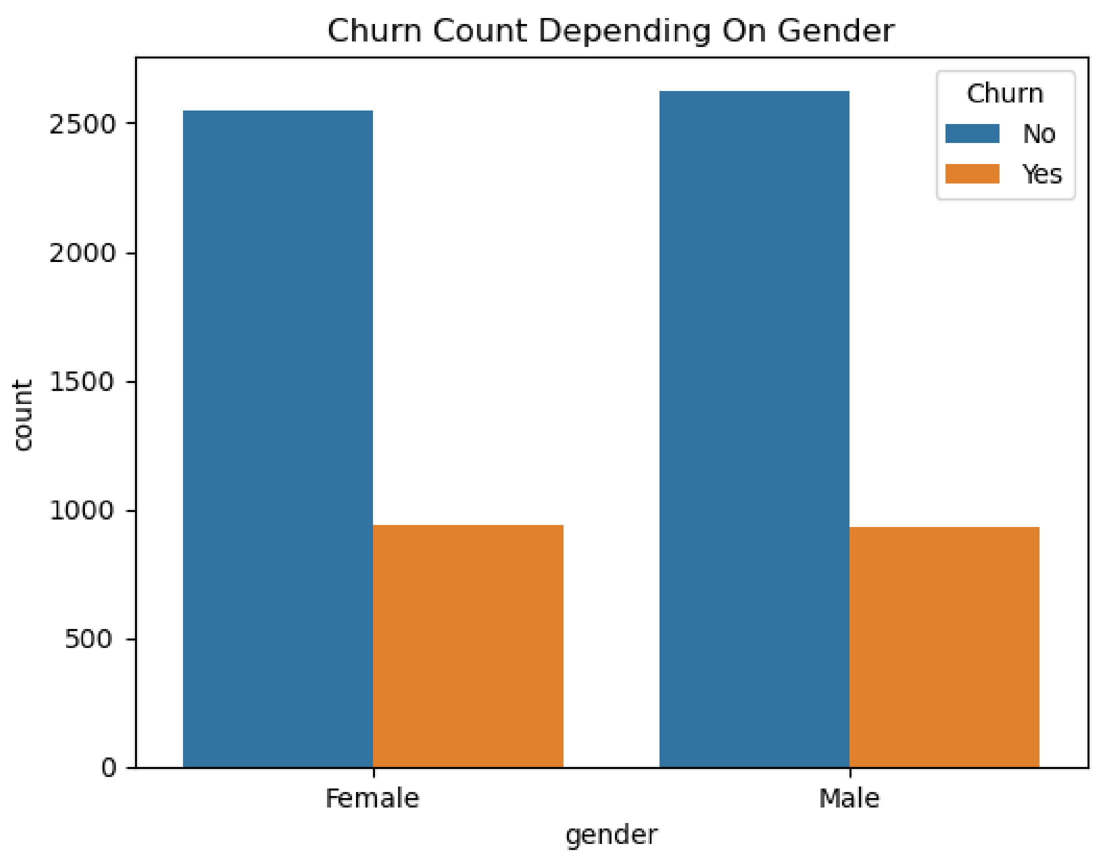
Therefore, the output indicates that there are 7043 observations and 21 features in the dataset.

[109]:
```
#Checking for duplicate rows on all columns and displaying them
customerDuplicate = customerChurn.duplicated(keep=False)
print(customerChurn[customerDuplicate])
```

```
Empty DataFrame
Columns: [customerID, gender, SeniorCitizen, Partner, Dependents, tenure,
PhoneService, MultipleLines, InternetService, OnlineSecurity, OnlineBackup,
DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract,
PaperlessBilling, PaymentMethod, MonthlyCharges, TotalCharges, Churn]
Index: []

[0 rows x 21 columns]
```

The output indicates that there are no duplicates in the dataset. The **.duplicated.(keep=False)** ensures that no instance of a duplicate is kept in the customerChurn dataset.

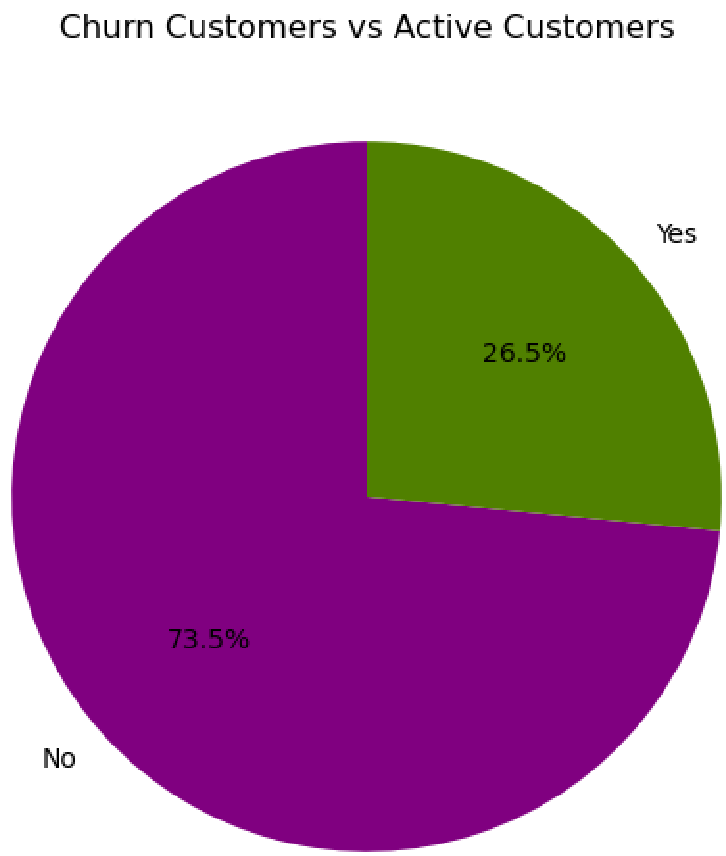
[110]:
```
# Using countplot and seaborn to visualize the number of churn instances based
↪on gender
sns.countplot(x='gender', hue='Churn', data=customerChurn)
plt.title('Churn Count Depending On Gender')
plt.show()
```



Judging by the output, the number of females and males who have churned, and those who have not, are almost the same. The only difference is the number of females who have not churned is a bit lower than the number of males who have not churned.

[111]:

```
#Computing percentage of churn customers and active customers
percentageChurn = customerChurn['Churn'].value_counts(normalize=True) * 100
#Using pie chart to visualize
plt.figure(figsize=(6, 6))
plt.pie(percentageChurn, labels=percentageChurn.index, autopct='%1.1f%%',
startangle=90, colors=['#800080', '#508000'])
plt.title('Churn Customers vs Active Customers')
plt.show()
```



Judging by the output, the percentage of customers who have churned is clearly less than the percentage of customers who are active (that is, customers who have not churned).

[112]:

```
# e)
columnLabel = 'Churn'
classPercentages = customerChurn[columnLabel].value_counts(normalize=True) * 100
print(f'Class Percentages:\n{classPercentages}')
```

```
Class Percentages:
```

```
Churn
No     73.463013
Yes    26.536987
Name: proportion, dtype: float64
```

The output above suggests that there is an imbalance in the dataset, with a higher number of instances belonging to the 'No' class compared to the 'Yes' class.

[113]:
```
customerChurn.describe()
```

[113]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

[114]:
```
#Transforming the customerChurn columns from categorical columns to numerical␣
 ↪columns by using label encoding which associates a unique
#numerical value to each category
labelEncoder = LabelEncoder()
customerChurn['Churn'] = labelEncoder.fit_transform(customerChurn['Churn'])
customerChurn['PaymentMethod'] = labelEncoder.
 ↪fit_transform(customerChurn['PaymentMethod'])
customerChurn['PaperlessBilling']  = labelEncoder.
 ↪fit_transform(customerChurn['PaperlessBilling'])
customerChurn['Contract'] = labelEncoder.
 ↪fit_transform(customerChurn['Contract'])
customerChurn['StreamingMovies'] = labelEncoder.
 ↪fit_transform(customerChurn['StreamingMovies'])
customerChurn['StreamingTV'] = labelEncoder.
 ↪fit_transform(customerChurn['StreamingTV'])
customerChurn['TechSupport'] = labelEncoder.
 ↪fit_transform(customerChurn['TechSupport'])
customerChurn['DeviceProtection'] = labelEncoder.
 ↪fit_transform(customerChurn['DeviceProtection'])
customerChurn['OnlineSecurity'] = labelEncoder.
 ↪fit_transform(customerChurn['OnlineSecurity'])
customerChurn['InternetService'] = labelEncoder.
 ↪fit_transform(customerChurn['InternetService'])
customerChurn['MultipleLines'] = labelEncoder.
 ↪fit_transform(customerChurn['MultipleLines'])
customerChurn['PhoneService'] = labelEncoder.
 ↪fit_transform(customerChurn['PhoneService'])
```

```
customerChurn['Dependents'] = labelEncoder.
 ↪fit_transform(customerChurn['Dependents'])
customerChurn['Partner'] = labelEncoder.fit_transform(customerChurn['Partner'])
customerChurn['SeniorCitizen'] = labelEncoder.
 ↪fit_transform(customerChurn['SeniorCitizen'])
customerChurn['gender'] = labelEncoder.fit_transform(customerChurn['gender'])
customerChurn['customerID'] = labelEncoder.
 ↪fit_transform(customerChurn['customerID'])
customerChurn['OnlineBackup'] = labelEncoder.
 ↪fit_transform(customerChurn['OnlineBackup'])
customerChurn['TotalCharges'] = labelEncoder.
 ↪fit_transform(customerChurn['TotalCharges'])
customerChurn['MonthlyCharges'] = labelEncoder.
 ↪fit_transform(customerChurn['MonthlyCharges'])
```

[115]:
```
#Displaying the last five rows of the dataset to confirm the conversion of␣
 ↪categorical columns to numerical columns
customerChurn.tail()
```

[115]:
```
      customerID  gender  SeniorCitizen  Partner  Dependents  tenure  \
7038        4853       1              0        1           1      24
7039        1525       0              0        1           1      72
7040        3367       0              0        1           1      11
7041        5934       1              1        1           0       4
7042        2226       1              0        0           0      66

      PhoneService  MultipleLines  InternetService  OnlineSecurity  …  \
7038             1              2                0               2  …
7039             1              2                1               0  …
7040             0              1                0               2  …
7041             1              2                1               0  …
7042             1              0                1               2  …

      DeviceProtection  TechSupport  StreamingTV  StreamingMovies  Contract  \
7038                 2            2            2                2         1
7039                 2            0            2                2         1
7040                 0            0            0                0         0
7041                 0            0            0                0         0
7042                 2            2            2                2         2

      PaperlessBilling  PaymentMethod  MonthlyCharges  TotalCharges  Churn
7038                 1              3             991          1597      0
7039                 1              1            1340          5698      0
7040                 1              2             137          2994      0
7041                 1              3             795          2660      1
7042                 1              0            1388          5407      0
```

```
[5 rows x 21 columns]
```

[116]:
```
columnLabel = 'Churn'
classPercentages = customerChurn[columnLabel].value_counts(normalize=True) * 100
print(f'Class Percentages:\n{classPercentages}')
```

```
Class Percentages:
Churn
0    73.463013
1    26.536987
Name: proportion, dtype: float64
```

[117]:
```
#Displaying the datatype of each column in the dataset
customerChurn.dtypes
```

[117]:
```
customerID          int32
gender              int32
SeniorCitizen       int64
Partner             int32
Dependents          int32
tenure              int64
PhoneService        int32
MultipleLines       int32
InternetService     int32
OnlineSecurity      int32
OnlineBackup        int32
DeviceProtection    int32
TechSupport         int32
StreamingTV         int32
StreamingMovies     int32
Contract            int32
PaperlessBilling    int32
PaymentMethod       int32
MonthlyCharges      int64
TotalCharges        int32
Churn               int32
dtype: object
```

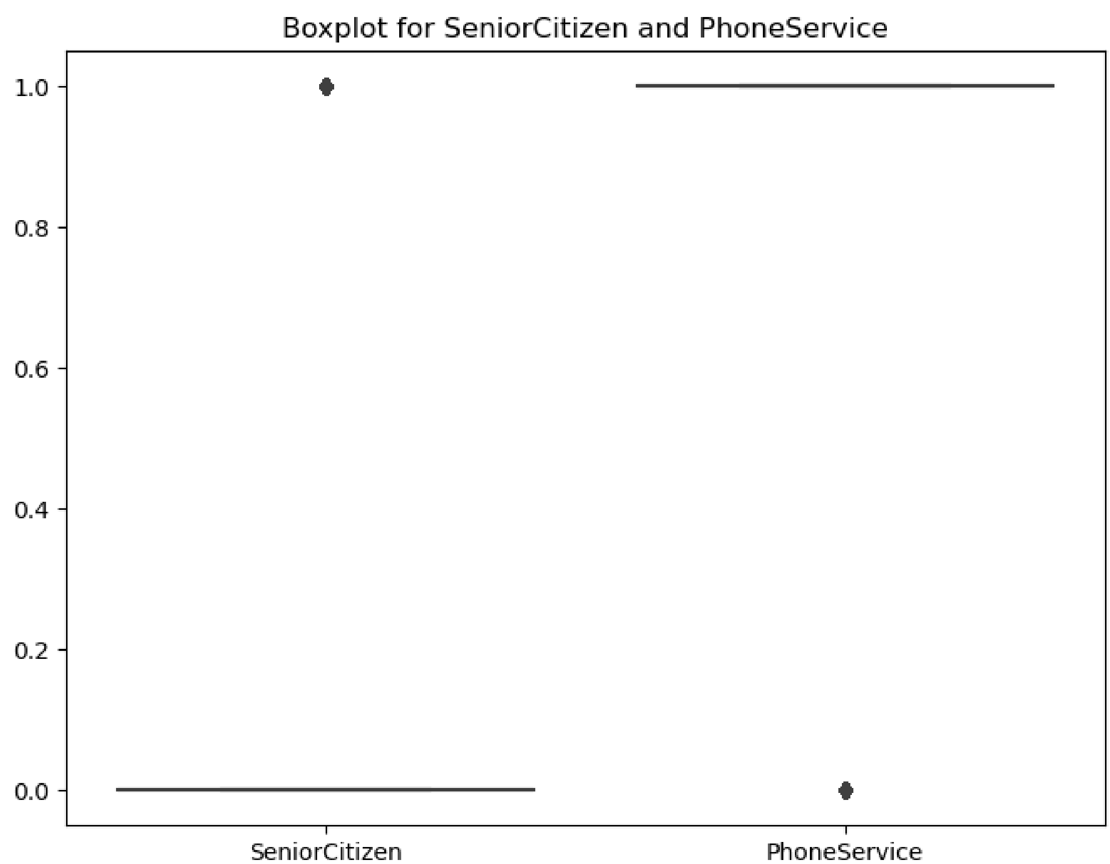
[118]:
```
#Checking for outliers
Q1 = customerChurn.quantile(0.25)
Q3 = customerChurn.quantile(0.75)
IQR = Q3 - Q1

lowerBound = Q1 - 1.5 * IQR
upperBound = Q3 + 1.5 * IQR

outliers = np.logical_or(np.isnan(customerChurn), np.logical_or(customerChurn <
 ↪lowerBound, customerChurn > upperBound))
```

```
# Displaying outliers for each column
columnOutliers = outliers.any()
print("Columns with outliers:\n")
print(columnOutliers[columnOutliers].index)
# #Removing outliers
# outlierID = (customerChurn < lowerBound) | (customerChurn > upperBound)
# no_outliers = customerChurn[~outlierID]
# #Displaying customerChurn without outliers
# print(no_outliers)
```

```
Columns with outliers:

Index(['SeniorCitizen', 'PhoneService'], dtype='object')
```

The output indicates that the columns that have outliers are the SeniorCitizen and the PhoneService columns. This to ensure that any errors made beforehand does not affect the quality of the dataset.

[119]:
```
#Visualizing the SeniorCitizen and the PhoneService via boxplot
plt.figure(figsize=(8, 6))
sns.boxplot(data=customerChurn[['SeniorCitizen', 'PhoneService']])
plt.title('Boxplot for SeniorCitizen and PhoneService')
plt.show()
```



[120]:
```
# normalizing the dataset via MinMaxScaler() to scale the features in a dataset
minmax_scaler = MinMaxScaler()
customerChurn_normalized = minmax_scaler.fit_transform(customerChurn)
```

Since the logistic function applied in logistic regression is sensitive to the scale of the input features, it helps to improve logistic regression models such as the one being applied to the telco customer dataset.

[121]:
```
#Pinpointing the independent variable (X) and the dependent variable (y)
X = customerChurn['InternetService'].values.reshape(-1, 1)
y = customerChurn['Churn'].values
#Dividing the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

[122]:
```
# h)
#Initializing the model
model = LogisticRegression()
```

[123]:
```
#Training the model
model.fit(X_train, y_train)
```

[123]:
```
LogisticRegression()
```

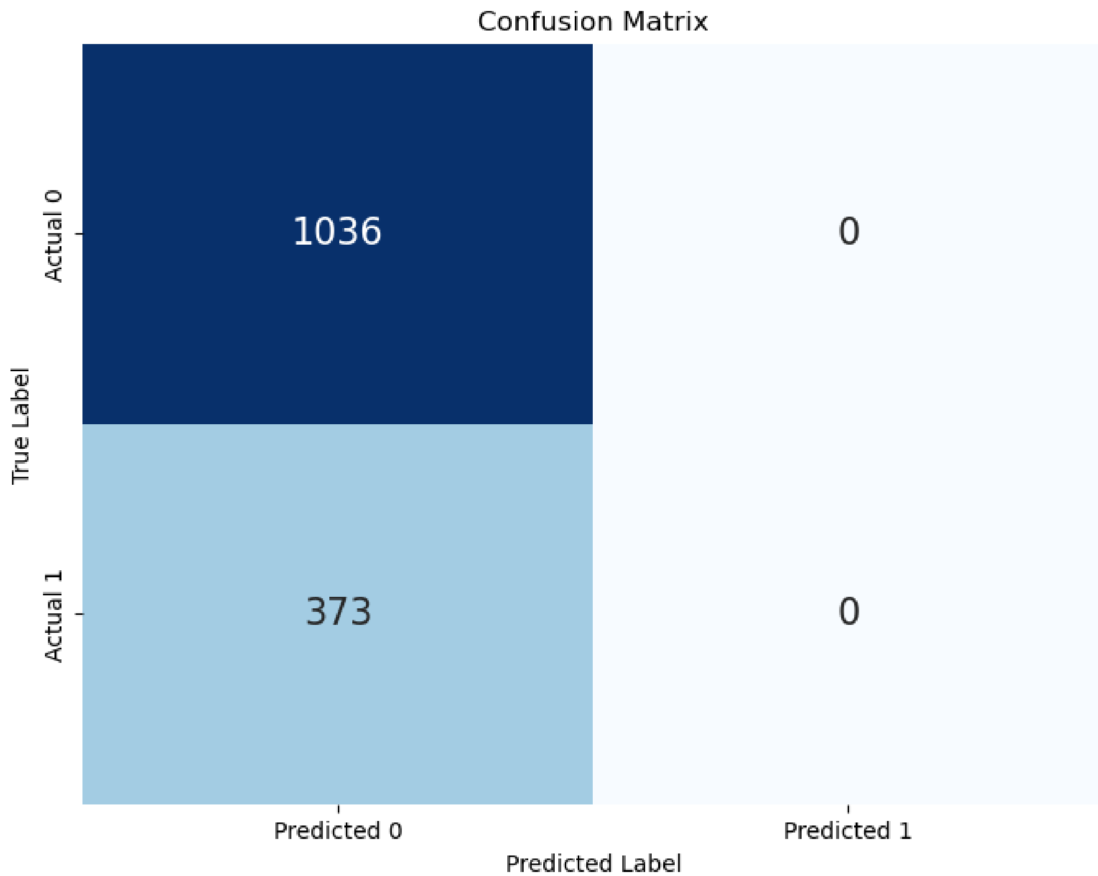
[124]:
```
#Prediction of the model
y_pred = model.predict(X_test)
# Evaluation of the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred, zero_division=1)
#Displaying results
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 0.7352732434350603
Confusion Matrix:
[[1036    0]
 [ 373    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      1.00      0.85      1036
           1       1.00      0.00      0.00       373

    accuracy                           0.74      1409
```

```
   macro avg       0.87      0.50      0.42      1409
weighted avg       0.81      0.74      0.62      1409
```

[125]:

```
#Visualizing the confusion matrix using heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size":
 ↪16}, cbar=False,
            xticklabels = ['Predicted 0', 'Predicted 1'],
            yticklabels = ['Actual 0', 'Actual 1'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



From the output above, the number of true positives (top-left cell) is 1036, the number of true negatives (bottom-right cell) is 0, the number of false positives (top-right cell) is 0 and the number of false negatives (bottom-left cell) is 373. In conclusion, even though, the model has strengths in terms of acccurately identifying positive instances, the same cannot be said in terms of identifying false negatives.