In [26]:
```
#Importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
```

In [27]:
```
df= pd.read_csv("C:\\Users\\skp18\\Downloads\\archive (1)\\tested.csv")
```

In [28]:
```
#Information about the dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

In [29]:
```
df.head()
```

Out[29]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	0	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	0	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	1	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [30]:
```
#descriptive statistics of the column

df.describe()
```

Out[30]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

In [31]:
```
#finding correlation
import seaborn as sns
import matplotlib.pyplot as plt

# Check for missing values in the DataFrame
if df.isnull().sum().any():
    print("Warning: The DataFrame contains missing values. Handle them before creating the heatmap.")
else:
    # Plot the heatmap
    sns.heatmap(df.corr(), cmap="YlGnBu")
    plt.show()
```

```
Warning: The DataFrame contains missing values. Handle them before creating the heatmap.
```

In [32]:

```python
import pandas as pd

# Assuming df is your DataFrame
# Check the data types of each column
print(df.dtypes)

# Convert non-numeric columns to numeric
for column in df.columns:
    if df[column].dtype == 'object':  # Check if the column is of object (string) type
        try:
            df[column] = pd.to_numeric(df[column])
        except ValueError:
            # Handle the case where conversion to numeric is not possible
            print(f"Warning: Unable to convert column '{column}' to numeric.")

# Check the data types again
print(df.dtypes)
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
Warning: Unable to convert column 'Name' to numeric.
Warning: Unable to convert column 'Sex' to numeric.
Warning: Unable to convert column 'Ticket' to numeric.
Warning: Unable to convert column 'Cabin' to numeric.
Warning: Unable to convert column 'Embarked' to numeric.
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [33]:
```
#calculate percentage of missing vallues for each column to understand completeness of the dataset

df.isna().sum()/len(df)*100
```

Out[33]:
```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            20.574163
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.239234
Cabin          78.229665
Embarked        0.000000
dtype: float64
```

In [34]:
```
#drop rows and columns with missing values(cabin)

df.drop('Cabin', axis=1,inplace=True)
```

In [35]:
```
df.drop('Name', axis=1,inplace=True) #As name is n
```

In [36]:
```
category_cols= df.select_dtypes(include='object').columns
print(category_cols)
```

```
Index(['Sex', 'Ticket', 'Embarked'], dtype='object')
```

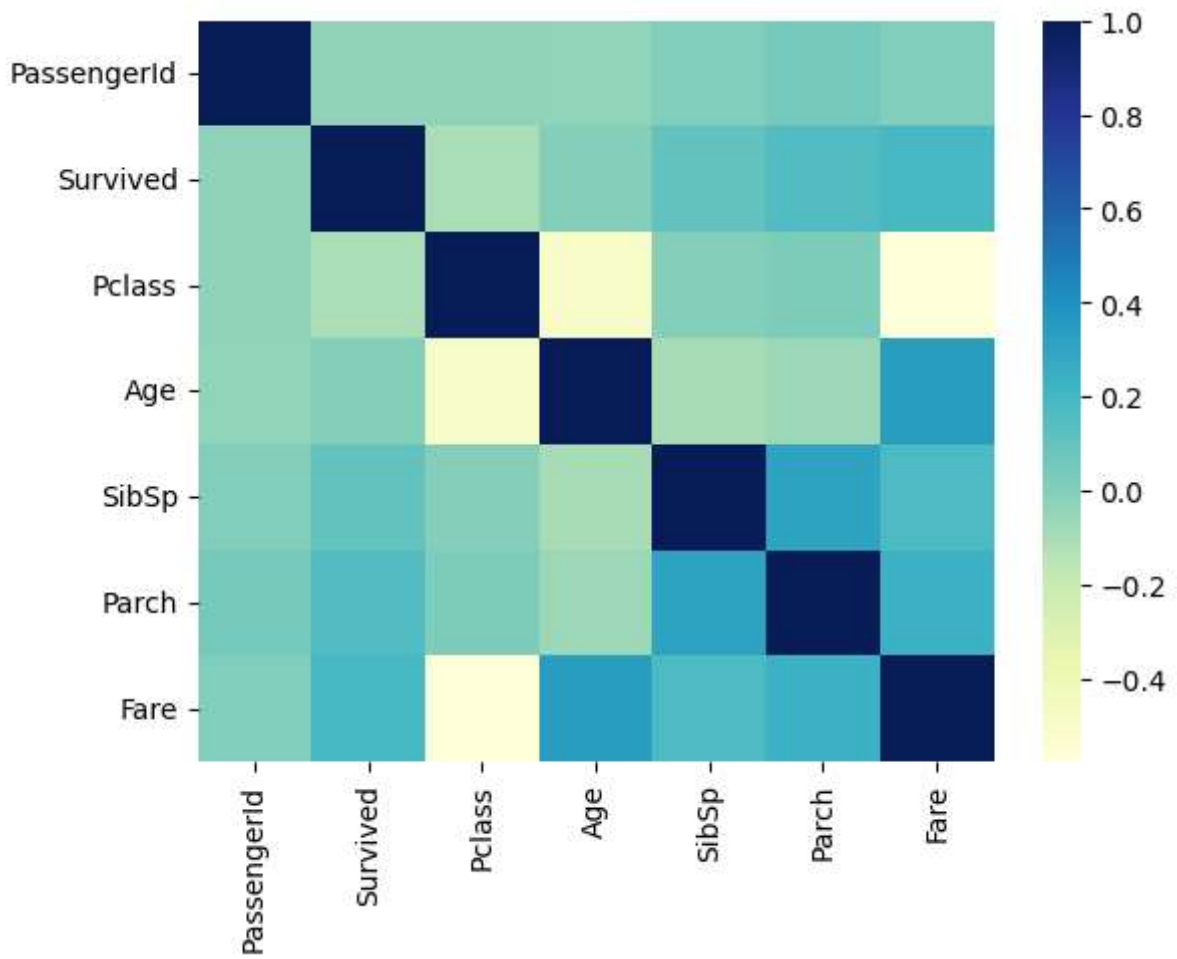
In [47]:

```
#converting categorical variables into numerical variables.
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming df is your DataFrame
# Display the column names to verify
print(df.columns)

# Label encode 'Sex' column
label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])

# One-hot encode the 'Title' column
df_converted = pd.get_dummies(df, columns=[ 'Ticket', 'Embarked'])

# Display the updated DataFrame
print(df_converted)
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Embarked'],
      dtype='object')
     PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch      Fare  \
0            892         0       3    1  34.5      0      0    7.8292
1            893         1       3    0  47.0      1      0    7.0000
2            894         0       2    1  62.0      0      0    9.6875
3            895         0       3    1  27.0      0      0    8.6625
4            896         1       3    0  22.0      1      1   12.2875
..           ...       ...     ...  ...   ...    ...    ...       ...
413         1305         0       3    1   NaN      0      0    8.0500
414         1306         1       1    0  39.0      0      0  108.9000
415         1307         0       3    1  38.5      0      0    7.2500
416         1308         0       3    1   NaN      0      0    8.0500
417         1309         0       3    1   NaN      1      1   22.3583

     Ticket_110469  Ticket_110489  ...  Ticket_STON/O2. 3101270  \
0            False          False  ...                    False
1            False          False  ...                    False
2            False          False  ...                    False
3            False          False  ...                    False
4            False          False  ...                    False
..             ...            ...  ...                      ...
413          False          False  ...                    False
414          False          False  ...                    False
415          False          False  ...                    False
416          False          False  ...                    False
417          False          False  ...                    False

     Ticket_STON/OQ. 369943  Ticket_W./C. 14260  Ticket_W./C. 14266  \
0                     False               False               False
1                     False               False               False
2                     False               False               False
3                     False               False               False
4                     False               False               False
..                      ...                 ...                 ...
413                   False               False               False
414                   False               False               False
415                   False               False               False
416                   False               False               False
417                   False               False               False
```

```
     Ticket_W./C. 6607  Ticket_W./C. 6608  Ticket_W.E.P. 5734  Embarked_C  \
0                False              False               False       False
1                False              False               False       False
2                False              False               False       False
3                False              False               False       False
4                False              False               False       False
..                 ...                ...                 ...         ...
413              False              False               False       False
414              False              False               False        True
415              False              False               False       False
416              False              False               False       False
417              False              False               False        True

     Embarked_Q  Embarked_S
0          True       False
1         False        True
2          True       False
3         False        True
4         False        True
..          ...         ...
413       False        True
414       False       False
415       False        True
416       False        True
417       False       False

[418 rows x 374 columns]
```

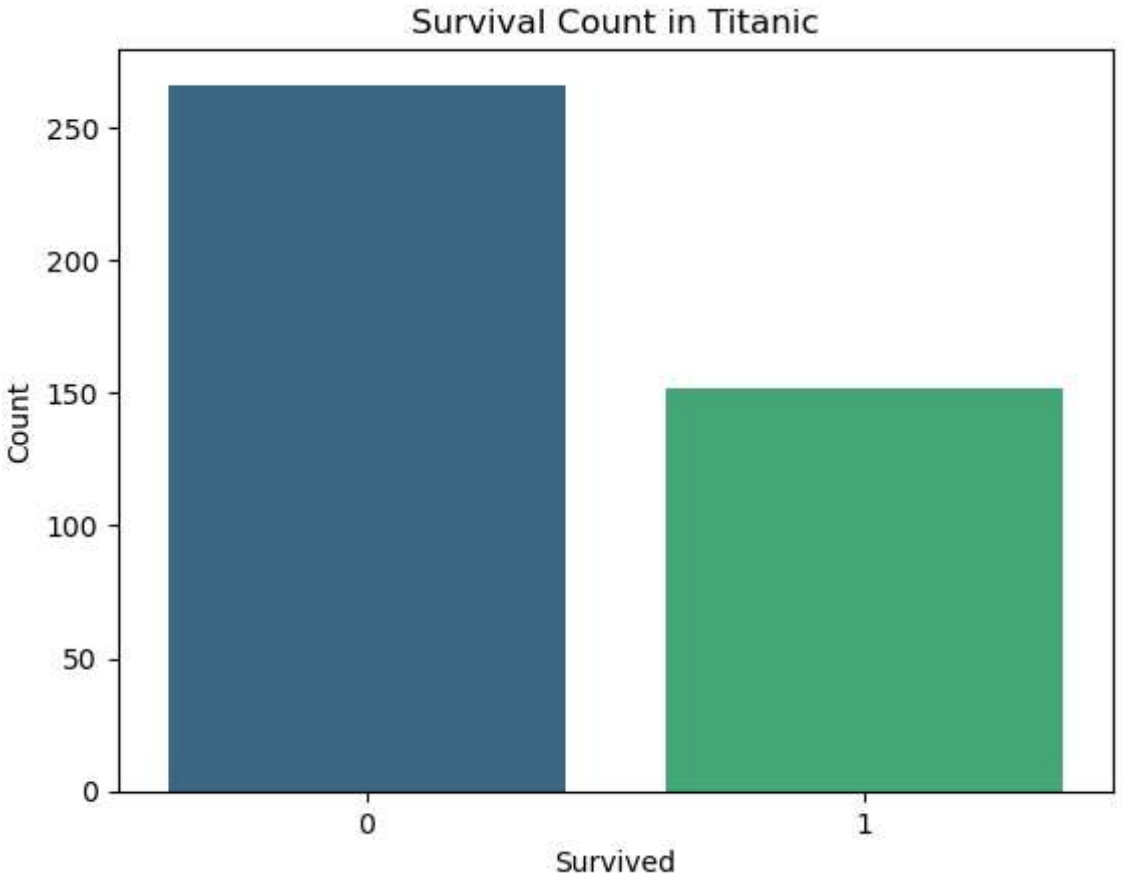
In [38]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Select only numeric columns for correlation
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
correlation_matrix = df[numeric_columns].corr()

# Plot the heatmap
sns.heatmap(correlation_matrix, cmap="YlGnBu")
plt.show()
```



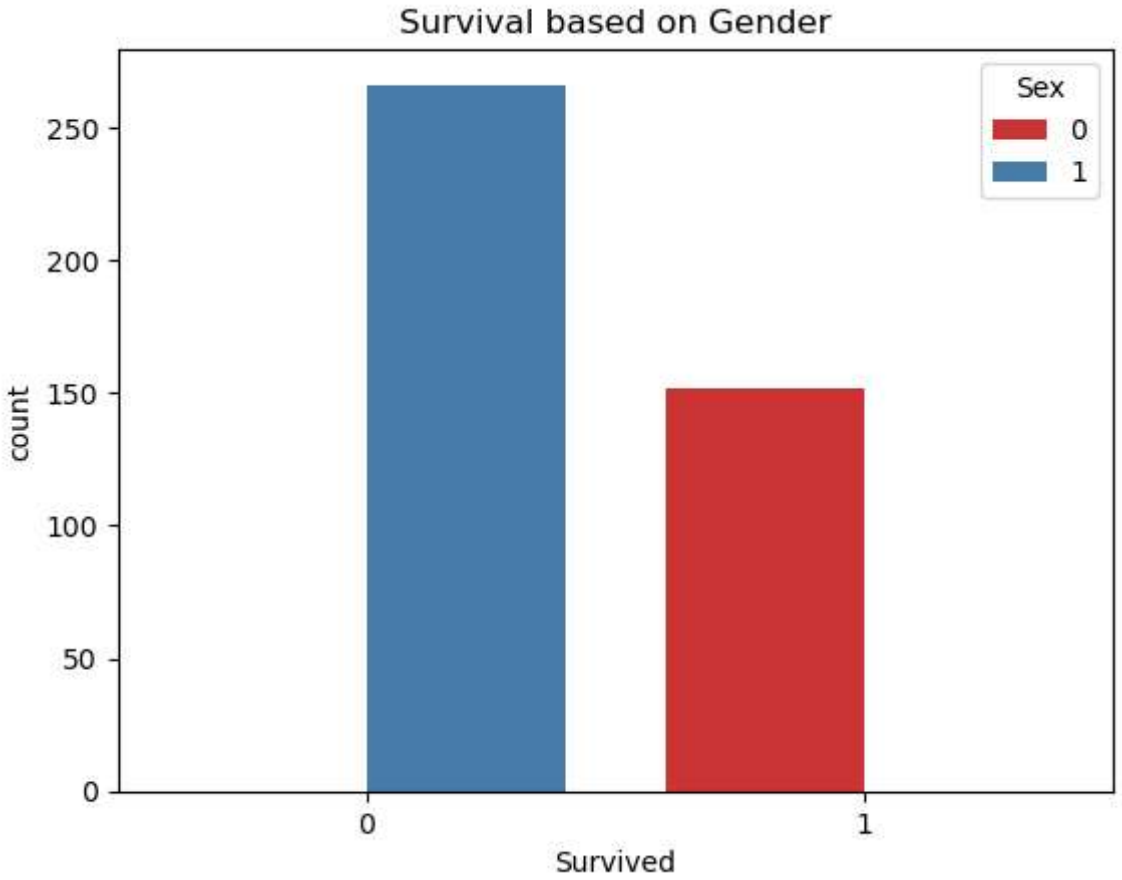
In [39]:
```
#percentage of survived passengers.

df['Survived'].value_counts(normalize=True)*100
```

Out[39]:
```
Survived
0    63.636364
1    36.363636
Name: proportion, dtype: float64
```

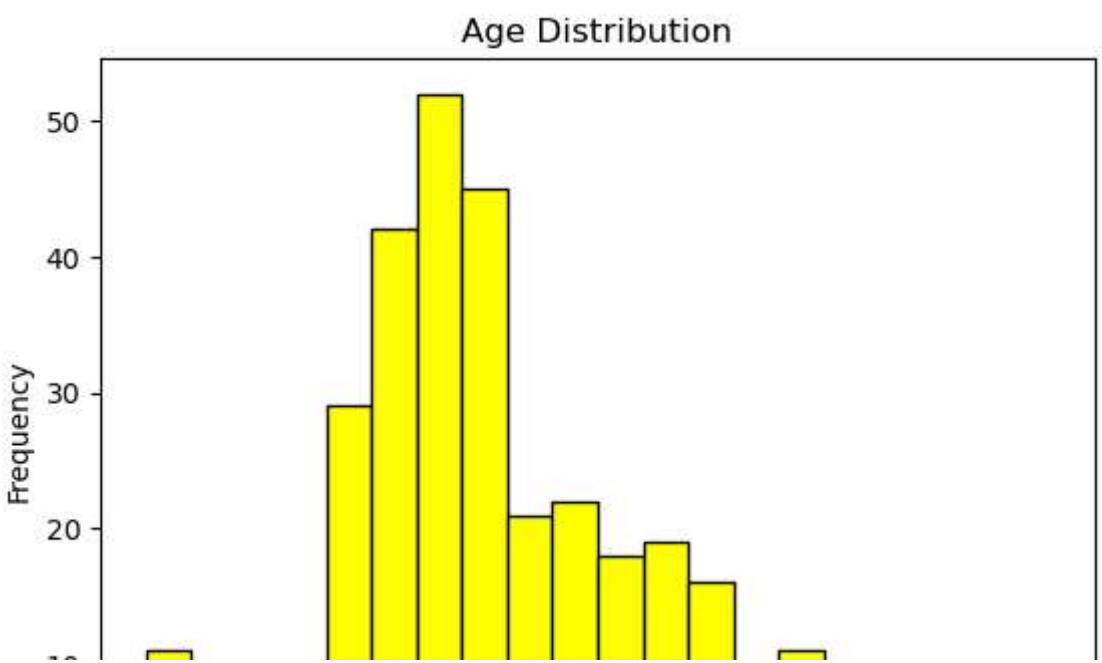
In [40]:
```
#survival rate visualization
import seaborn as sns
sns.countplot(x='Survived', data=df,palette='viridis')
plt.title('Survival Count in Titanic')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()
```



In [41]:
```
#Based on gender

sns.countplot(x='Survived', hue='Sex', data=df, palette='Set1')
plt.title('Survival based on Gender')
plt.show()
```



In [42]:
```
#based on age
plt.hist(df['Age'].dropna(), bins=20, color='yellow', edgecolor='black')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



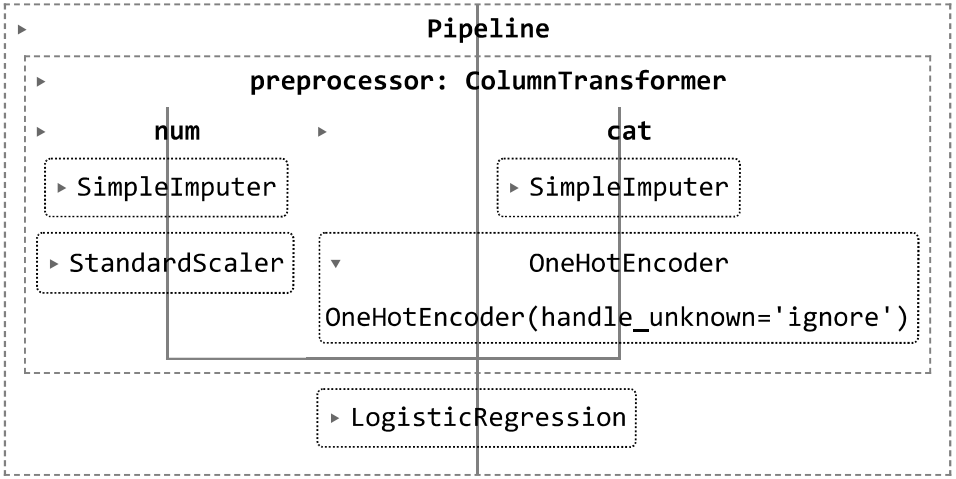
In [43]:
```
X = df.drop('Survived', axis = 1)
y = df['Survived']
```

In [44]:
```
#model selection
# 3. Splitting into training and testing
# Splitting the dataset into training and testing parts

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
```

In [53]:

```
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Assuming X_train and X_test have both numeric and categorical features

# Identify numeric and categorical columns
numeric_cols = X_train.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = X_train.select_dtypes(include=['object']).columns

# Create transformers for numeric and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Create a column transformer to apply different transformers to different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)

# Create a pipeline with the preprocessor and your model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', LogisticRegression())])

# Fit and transform the data using the pipeline
pipeline.fit(X_train, y_train)
```

Out[53]:



In [ ]:
```
print(X_train.isnull().sum())  # Check missing values in the training set
print(X_test.isnull().sum())   # Check missing values in the test set
```

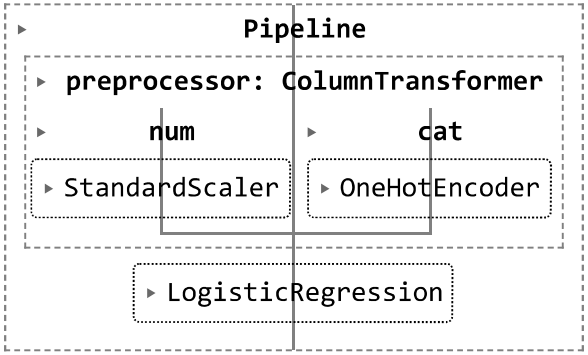
In [55]:
```
# Remove rows with missing values
X_train = X_train.dropna()
y_train = y_train[X_train.index]  # Adjust the target variable accordingly

X_test = X_test.dropna()
y_test = y_test[X_test.index]  # Adjust the target variable accordingly
```

In [56]:
```
numeric_cols = X.select_dtypes(include=['float64', 'int64']).columns
X_train_numeric = X_train[numeric_cols]
X_test_numeric = X_test[numeric_cols]

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_numeric)
X_test_scaled = scaler.transform(X_test_numeric)
```

In [57]:

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

# Assuming you have a dataframe with both numeric and categorical columns
# ...

# Separate numeric and categorical columns
numeric_cols = X.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Create transformers for numeric and categorical columns
numeric_transformer = StandardScaler()
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Create a column transformer to apply different transformers to different columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)

# Create a pipeline with the preprocessor and your model
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', LogisticRegression())])

# Fit and transform the data using the pipeline
pipeline.fit(X_train, y_train)
```

Out[57]:


Pipeline
preprocessor: ColumnTransformer
num
cat
StandardScaler
OneHotEncoder
LogisticRegression

In [58]:
```python
# Make predictions on the test set
y_pred = pipeline.predict(X_test)

# Evaluate the model
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Display evaluation metrics
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 0.6831683168316832
Confusion Matrix:
[[56 11]
 [21 13]]
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.84      0.78        67
           1       0.54      0.38      0.45        34

    accuracy                           0.68       101
   macro avg       0.63      0.61      0.61       101
weighted avg       0.66      0.68      0.67       101
```

In [ ]: