CSE 351 Project #2: Titanic - Who will survive?

--> Zian Shang

Introduction:

In this individual data science project, I will analyze the factors influencing survival in the tragic sinking of the RMS Titanic on 15 April 1912. With a focus on the train set, which includes information such as passenger ticket class, gender, age, and the presence of family members on board, I aim to uncover patterns and insights into survival determinants. Leveraging machine learning techniques, my objective is to develop prediction models using the train set. This is to accurately predict survival using the test set data. By understanding the dynamics of survival during this historic event, this project contributes to our knowledge of past tragedies. It may offer valuable insights into future safety measures.

Reference:

1. https://en.wikipedia.org/wiki/Second-_and_third-class_facilities_on_the_Titanic#:~:text=Accommodation%20for%20third%20class%20v
2. https://en.wikipedia.org/wiki/First-class_facilities_of_the_Titanic#:~:text=First%20Class%20accommodation%20occupiec

Libraries and dataset:

In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.metrics import roc_curve
from sklearn.neighbors import KNeighborsClassifier

# Load csv files
df_train=pd.read_csv("train.csv")
df_test=pd.read_csv("test.csv")
```

Clean dataset:

1. While observing the train set, I observed that some of the values in the 'Age' and 'Cabin' columns are missing. Since the 'Cabin' column are non-numeric and meaningless to perform imputation, I decided the only input values for the 'Age' column. I used the method of random imputation, that is inserting random values within one stardard deviation of the mean age into the missing cells. This method involved the use of numpy functions and helps to preserve the variability in the data but may introduce some randomness.

2. I also noticed that some of the 'Ticket' cells have values of different forms. Some of them are having prefixes while the other are just strings of numbers. While having no insight on whether this column will affect one's survival, I decided to extract the ticket numbers from the cells for latter analysis. I wrote an extract_ticket_number(ticket) function that takes in a 'Ticket' string object and returns the 'TicketNumber' float object. I applyed this function to each cell of the 'Ticket' column using .apply() and converted them into of type integer using .astype().

*** Caution: do not run the cell below twice without rerunning the last code cell ***

3. I ensured every column in the train set is retaining its correct data type using the .astype() function

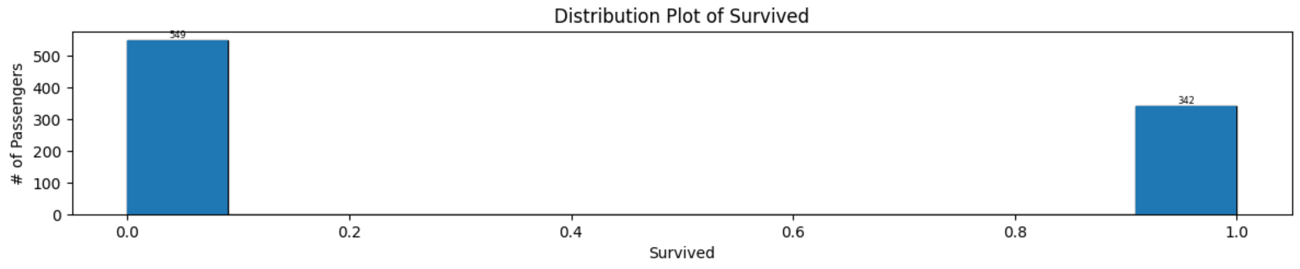
In [2]:
```
# ---------------------------------Inputing missing 'Age' values------------
# Obtain cells of the 'Age' column that misses values
missing_Ages = df_train['Age'].isnull()

# Calculate the mean and std dev of the available ages
mean_Age = df_train['Age'].mean()
std_Age = df_train['Age'].std()

# Generate random values within one standard deviation of the mean
random_values = np.random.normal(mean_Age, std_Age, missing_Ages.sum())

# Replace the missing Age values with the randomly generated values
df_train.loc[missing_Ages, 'Age'] = random_values

# -------------------------Extract 'Ticket' number from 'Ticket' column-----
# Function to extract ticket number that is always after an empty space
def extract_ticket_number(ticket):
    # Check if the ticket value is a string of numbers
    if ticket.isdigit():
        return int(ticket)
    else:
        # Split the ticket value by empty spaces
        parts = ticket.split()
        if len(parts) > 1:
            # Extract the last part after the empty space
            ticket_number = parts[-1]
            try:
                # Convert the ticket number to an integer
                return int(ticket_number)
            except ValueError:
                # Return None if the conversion fails
                return None
        else:
            return None

# Apply the extract function to the 'Ticket' column
df_train['TicketNumber'] = df_train['Ticket'].apply(extract_ticket_number).a

# Drop the original 'Ticket' column
df_train = df_train.drop('Ticket', axis=1)
# -------------------------Ensure columns retain correct data types--------
df_train=df_train.astype(df_train.dtypes)
```

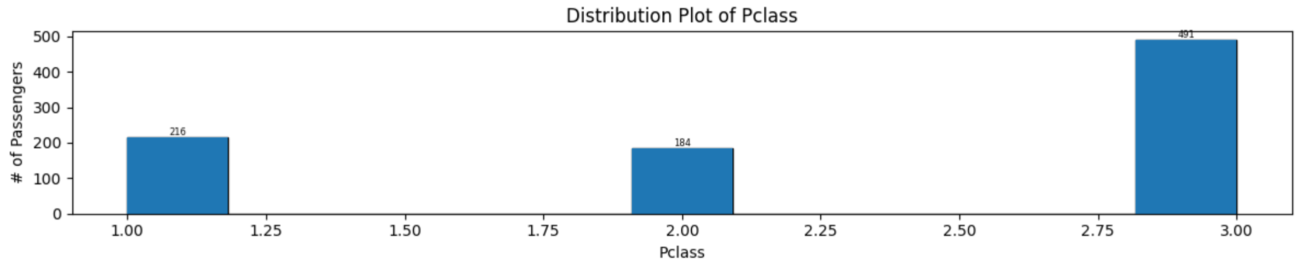
In [3]:
```
# Check the cleaned train data
# print(df_train.head(10))
```

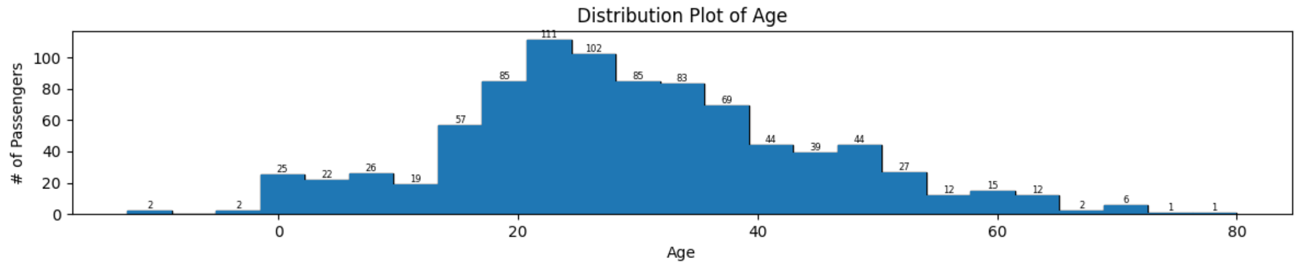
Distributions:

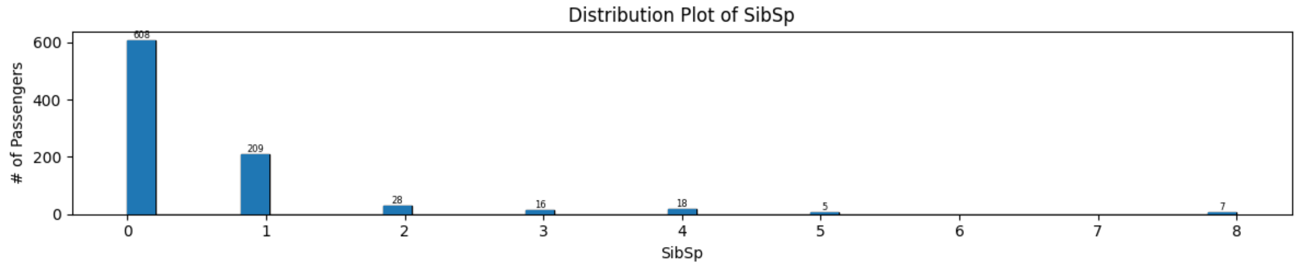
In order to determine if there are outliers exist in the dataset and to analyze any pattern, I ploted the 'Suvrived' coclumn and other numerical columns that possibly affect the future prediction of survival. The result is shown below.

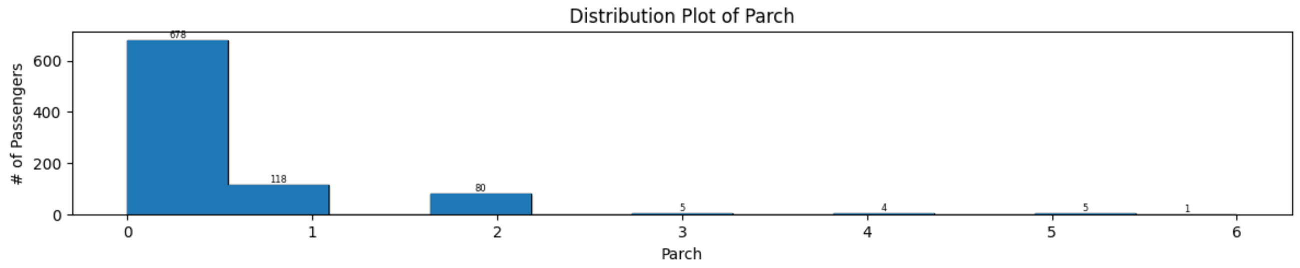
In [4]:
```
# -------------------------Plot numerical columns with potential outliers----
columns_to_plot = ['Survived','Pclass','Age', 'SibSp', 'Parch','Fare']

# Define the number of rows and columns for the subplots grid
num_rows = len(columns_to_plot)
num_cols = 1

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 15))

# Iterate over the columns and create individual histogram plots
for i, column in enumerate(columns_to_plot):
    ax = axes[i] if num_rows > 1 else axes
    sns.histplot(data=df_train, x=column, ax=ax)
    ax.set_xlabel(column)
    ax.set_ylabel('# of Passengers')
    ax.set_title(f'Distribution Plot of {column}')

    # Label the y-value on top of each column (excluding 0 counts)
    bin_counts, bin_edges, _ = ax.hist(df_train[column], bins='auto')
    for count, edge in zip(bin_counts, bin_edges):
        if count > 0:
            ax.text(edge + (bin_edges[1] - bin_edges[0]) / 2, count, int(cou

# Adjust the spacing between subplots
plt.tight_layout()

# Show the plots
plt.show()
```

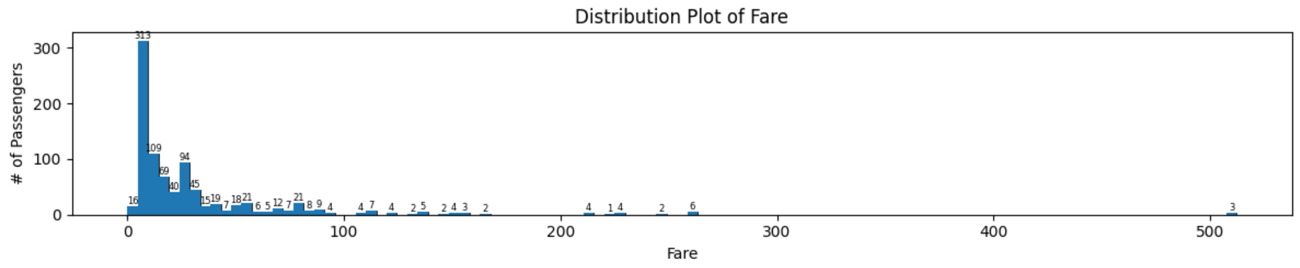

Distribution Plot of Survived
549
342
# of Passengers
Survived
0 100 200 300 400 500
0.0 0.2 0.4 0.6 0.8 1.0


Distribution Plot of Pclass
216
184
491
# of Passengers
Pclass
0 100 200 300 400 500
1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00


Distribution Plot of Age
2 2 25 22 26 19 57 85 111 102 85 83 69 44 39 44 27 12 15 12 2 6 1 1
# of Passengers
Age
0 20 40 60 80 100
0 20 40 60 80


Distribution Plot of SibSp
608
209
28
16
18
5
7
# of Passengers
SibSp
0 200 400 600
0 1 2 3 4 5 6 7 8


Distribution Plot of Parch
678
118
80
5
4
5
1
# of Passengers
Parch
0 200 400 600
0 1 2 3 4 5 6


Distribution Plot of Fare
16 313 109 69 40 94 45 15 19 7 18 21 6 5 12 7 21 8 9 4 4 7 4 2 5 2 4 3 2 4 1 4 2 6 3
# of Passengers
Fare
0 100 200 300
0 100 200 300 400 500

Based on the above histograms, I do not believe any numerical column should be removed due to outliers. First of all, the plots of 'Survived' and 'Pclass' both reflect real world situations: more people died in this tragedy, and most of the people were in the 3rd class of the Titanic. A second observation was that the range of ages of passengers ranged from 0 to 80, with one male passenger of age 74 and another of age 80 (I searched through the file to find the addresses). While the whole plot showed a slightly skewed distribution, removing two older men doesn't make sense. Additionally, the last three plots all showed possible power law distributions in which rare events would constantly happen. I do not believe it is appropriate to dismiss those with a large family or are richer than others. Therefore, I decided not to remove any data from the train set.

Socio-economic status:

Here, I used the passenger class 'Pclass' as a brief indicator of the socio-economic status of Titanic passengers. This works because 'Pclass' shows spending power. Passengers in class 1 typically have a higher socio-economic status than passengers in class 3, since class 3 was located in the least desirable part of the ship(1) while class 1 cabins were located on the boat deck between the forward grand staircase and officer's quarters(2) according to Wikipedia. I analyzed the relationship between 'Pclass' and other factors such as 'Age' and 'Sex' to see if any relationship exists between them. I also used plots and statistical analysis to do this.

In [5]:
```
# Obtain the names of columns, convert into a list
column_names = df_train.columns.tolist()
print(column_names)
```

```
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch',
'Fare', 'Cabin', 'Embarked', 'TicketNumber']
```

```
In [6]: # Group the dataset by passenger class
        grouped_by_pclass = df_train.groupby('Pclass')

        # Obtain the number of passenges by class
        passenger_count_by_pclass = df_train['Pclass'].value_counts().sort_index()

        # Obtain mean age of each class
        average_age_by_pclass = grouped_by_pclass['Age'].mean()

        # Obtain gender distribution by class
        gender_distribution_by_pclass = grouped_by_pclass['Sex'].value_counts().unst

        # Calculate the ratio of male to female passengers
        gender_distribution_by_pclass['gender_ratio'] = gender_distribution_by_pclas

        # Obtain family members by class
        family_members_by_pclass = grouped_by_pclass[['SibSp', 'Parch']].mean()

        print(f'--> Mean age of each passenger class:\n{average_age_by_pclass}\n')
        print(f'--> Passenger count of each passenger class: \n{passenger_count_by_p
        print(f'--> Gender distribution of each passenger class: \n{gender_distribut
        print(f'--> Mean family members count of each passenger class(SibSp & Parch)
```

```
--> Mean age of each passenger class:
Pclass
1    37.560298
2    30.511004
3    25.463930
Name: Age, dtype: float64

--> Passenger count of each passenger class:
Pclass
1    216
2    184
3    491
Name: count, dtype: int64

--> Gender distribution of each passenger class:
Sex     female  male  gender_ratio
Pclass
1           94   122      1.297872
2           76   108      1.421053
3          144   347      2.409722

--> Mean family members count of each passenger class(SibSp & Parch):
           SibSp     Parch
Pclass
1       0.416667  0.356481
2       0.402174  0.380435
3       0.615071  0.393075
```

The above statistics helped us gain insights into the characteristics and demographics of different socio-economic groups within the Titanic dataset. Below are some observations.

1. The mean age of each passenger class showed a decreasing pattern as one moved to the lower passenger classes. This might be because of people accumulating wealth as they grow older or of the health conditions of older people leading to their needs for better cabins.

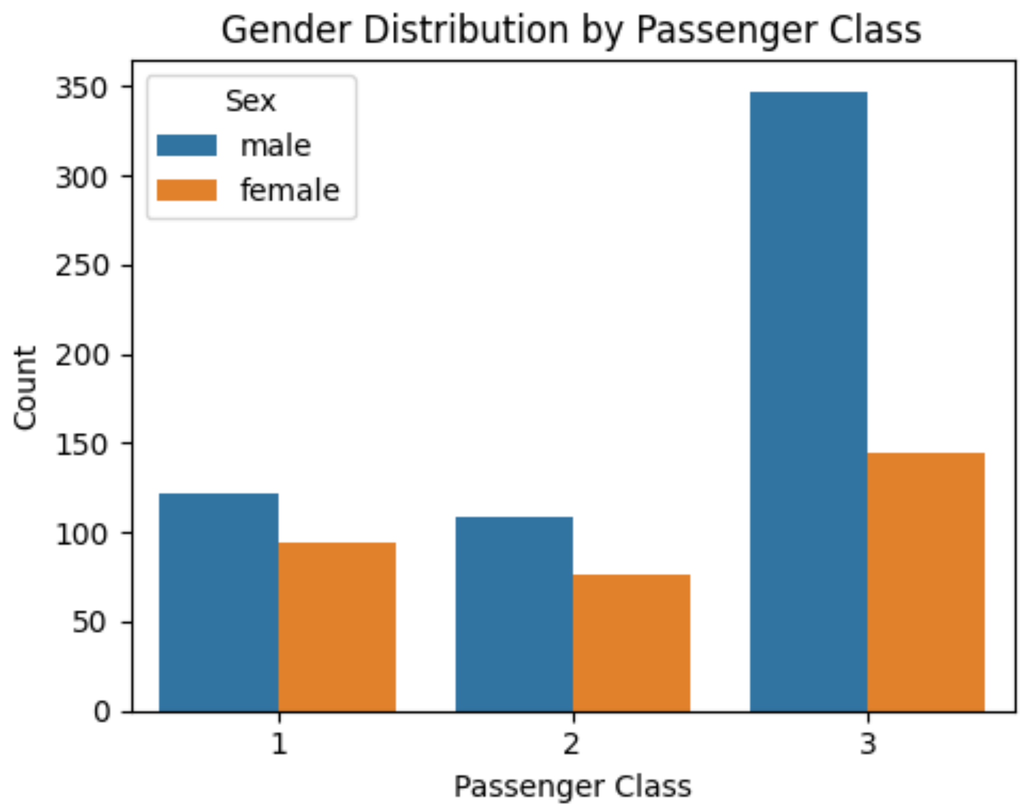
2. The total number of passengers in each class increased as classes decreased, which indicates that the majority of passengers belong to the lower passenger class. I also noticed that class 1 has more passengers than class 2. This might be potential indicator of the wide gap between the rich and the poor.

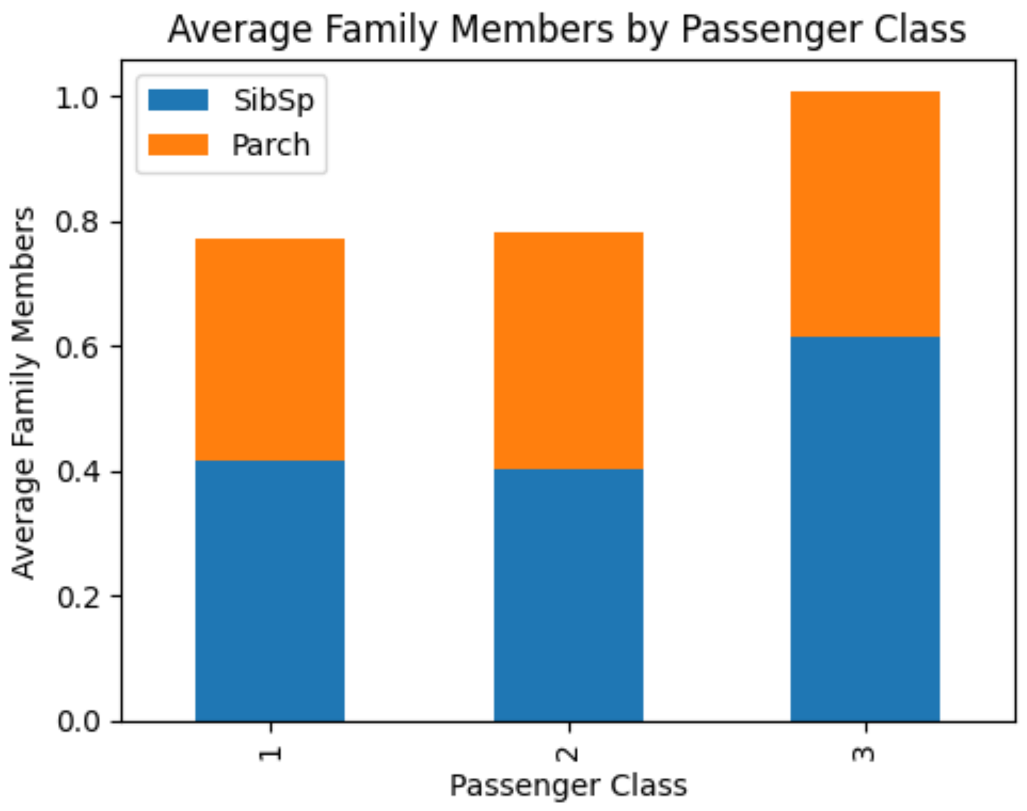
3. When looking at the gender distributions of each class directly, there was no obvious trend in how classes are associated with gender directly. However, when I calculated the ratio between males and females, I noticed that there was a steady upward trend in the ratio of males coupled with a decline in classes. This means the gender ratio might be a strongly associated factor with one's socio-economic status.

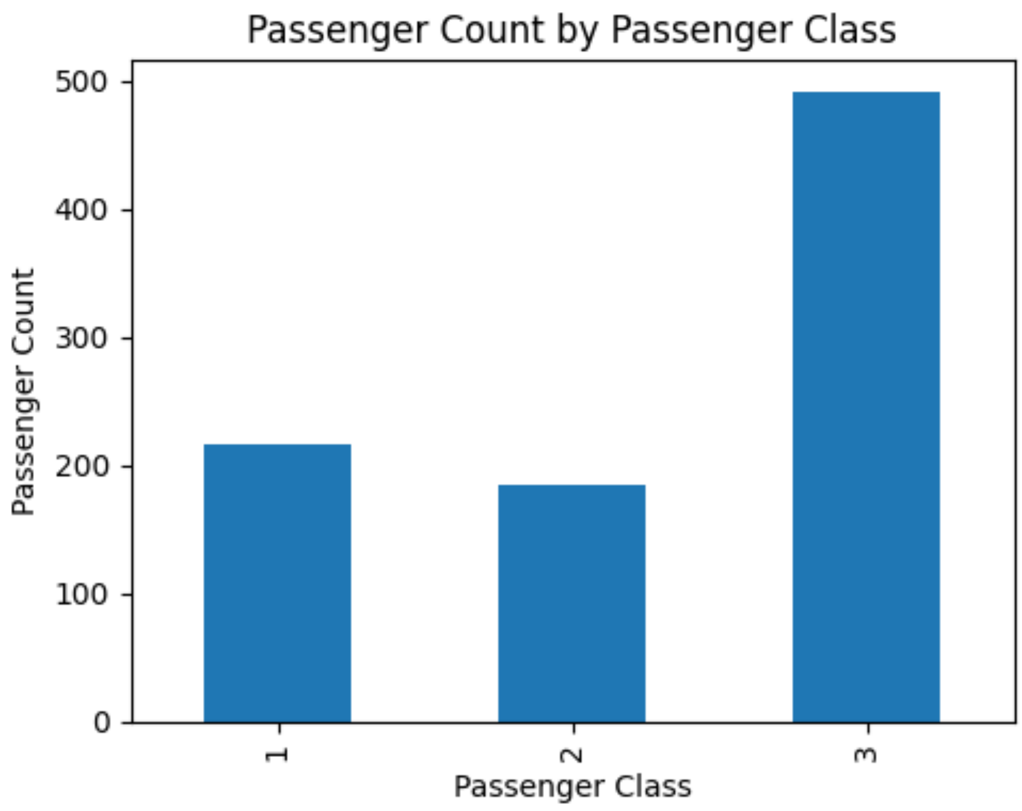
4. The number of family members also showed that more people were in the lower class by demonstrating an increasing trend of siblings/spouses and parents/children. It also showed that people in lower classes tend to have more children than people in higher classes, which is a complex social issue.

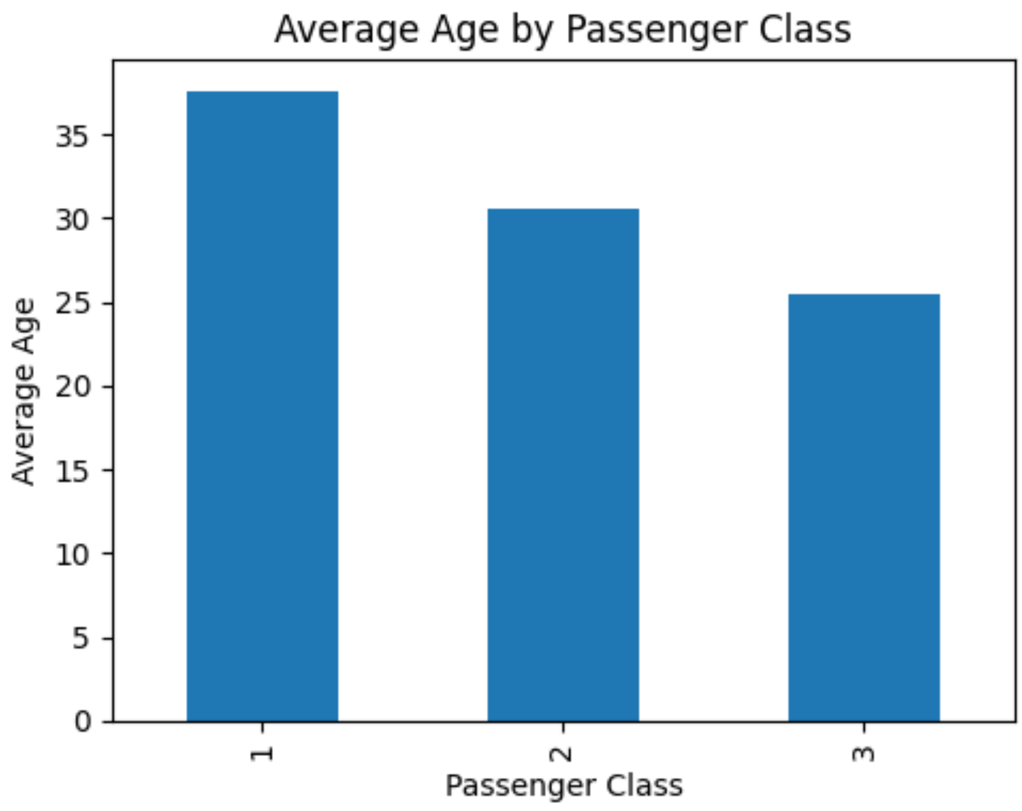
Below are bar plots for these four features:

```
In [7]: # Bar plot for gender distribution by passenger class
        plt.figure(figsize=(5, 4))
        sns.countplot(data=df_train, x='Pclass', hue='Sex')
        plt.xlabel('Passenger Class')
        plt.ylabel('Count')
        plt.title('Gender Distribution by Passenger Class')
        plt.tight_layout()
        plt.show()

        # Stacked bar plot for family members by passenger class
        fig, ax = plt.subplots(figsize=(5, 4))
        family_members_by_pclass.plot(kind='bar', stacked=True, ax=ax)
        plt.xlabel('Passenger Class')
        plt.ylabel('Average Family Members')
        plt.title('Average Family Members by Passenger Class')
        plt.tight_layout()
        plt.show()

        # Bar plot for passenger count by passenger class
        plt.figure(figsize=(5, 4))
        passenger_count_by_pclass.plot(kind='bar')
        plt.xlabel('Passenger Class')
        plt.ylabel('Passenger Count')
        plt.title('Passenger Count by Passenger Class')
        plt.tight_layout()
        plt.show()

        # Bar plot for average age by passenger class
        plt.figure(figsize=(5, 4))
        average_age_by_pclass.plot(kind='bar')
        plt.xlabel('Passenger Class')
        plt.ylabel('Average Age')
        plt.title('Average Age by Passenger Class')
        plt.tight_layout()
        plt.show()
```


Gender Distribution by Passenger Class
Sex
male
female
Count
Passenger Class
0
50
100
150
200
250
300
350
1
2
3


Average Family Members by Passenger Class
SibSp
Parch
Average Family Members
Passenger Class
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3

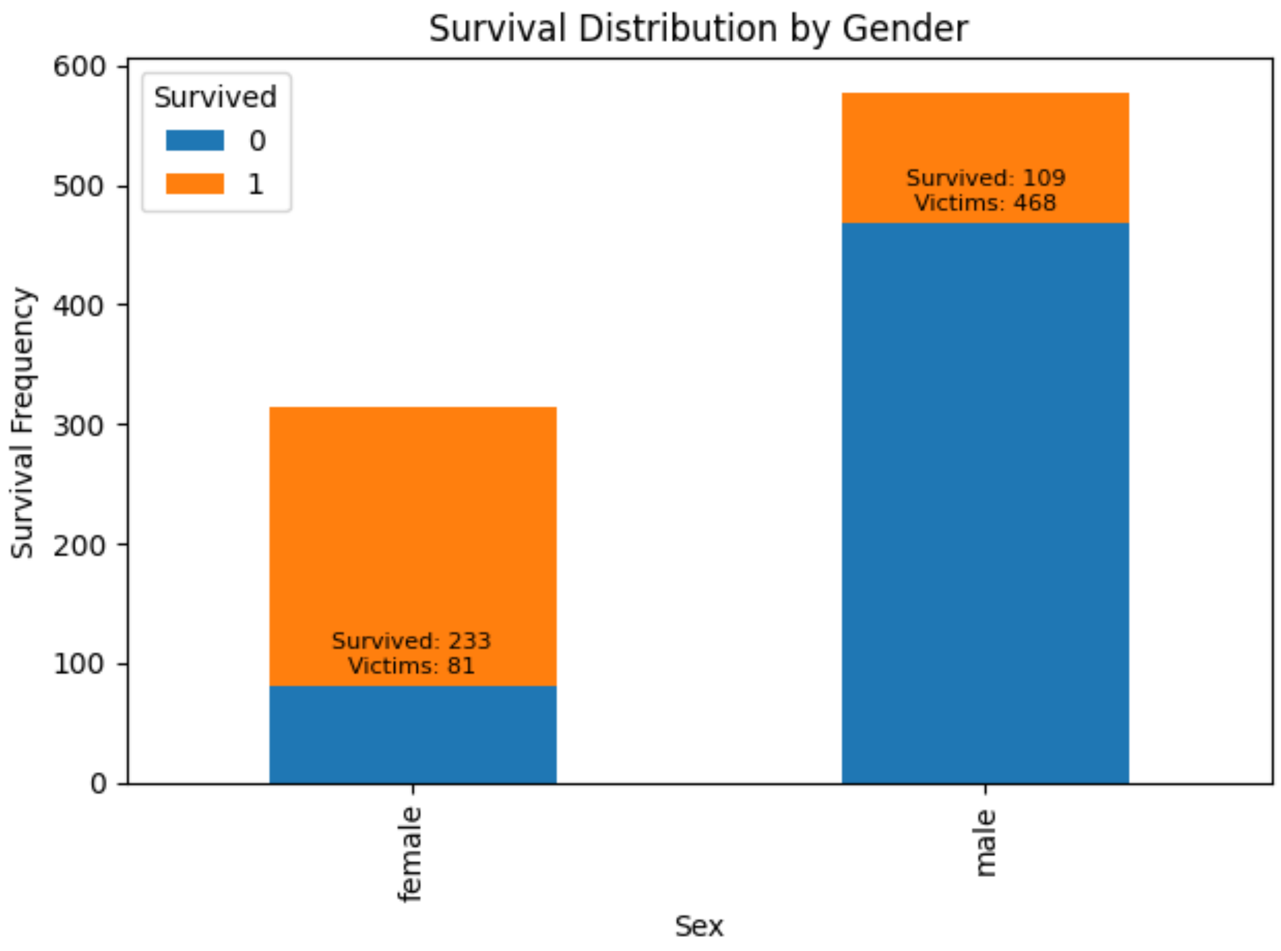

Passenger Count by Passenger Class
Passenger Count
0
100
200
300
400
500
1
2
3
Passenger Class


Average Age by Passenger Class
Average Age
0
5
10
15
20
25
30
35
1
2
3
Passenger Class

Distributions of suvivals by other features:

In the plots below, I created bar plots with the y-axis being the frequency of survivals/victims and the x-axis being features: gender, age, class, siblings/spouses, parents/children. These plots all showed some reasonable associations between features and survival rates. I also calculated percentages of survivals corresponding to each feature for better insights.

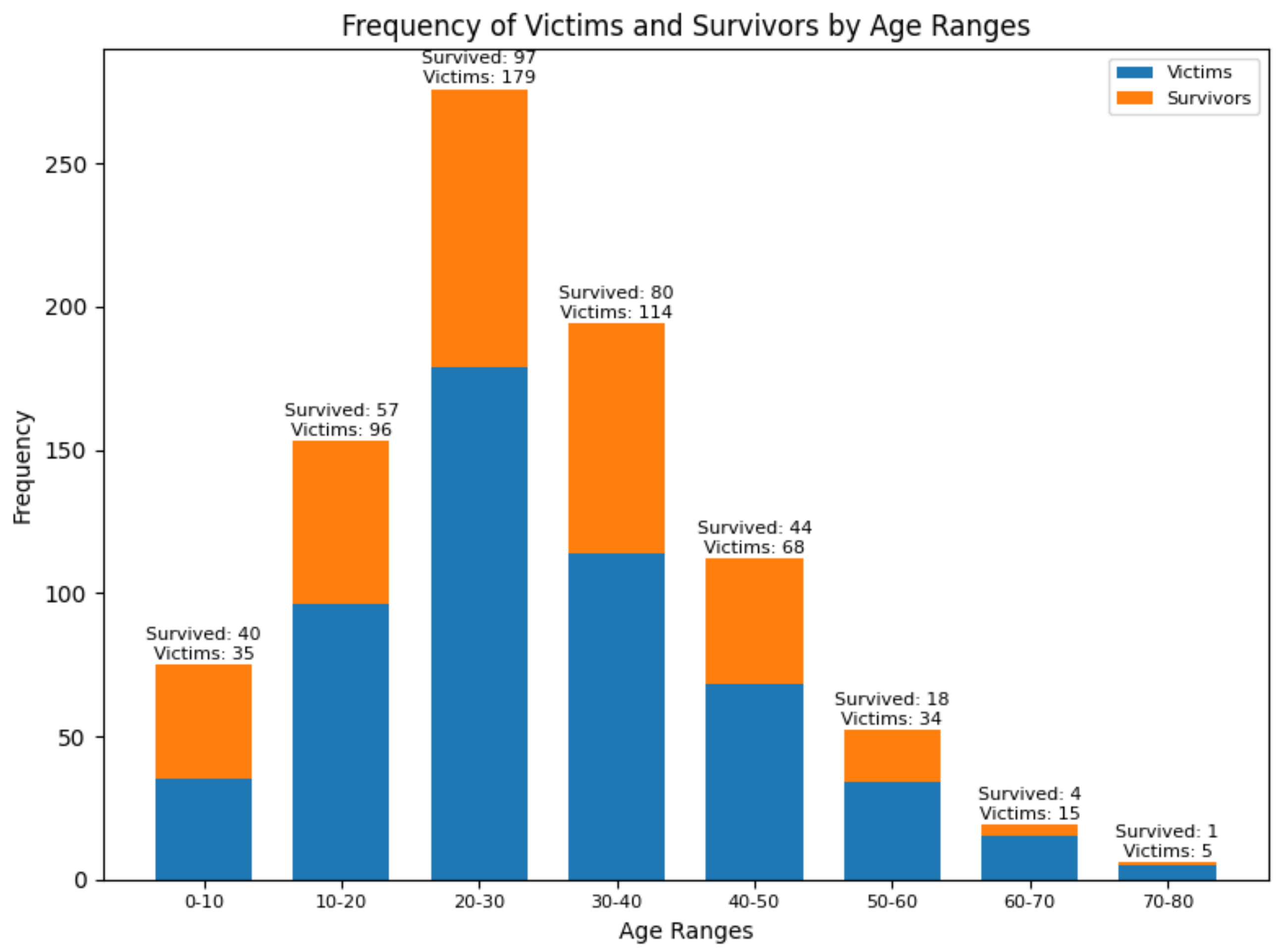
In [8]:
```
# ---------------------------Plot the frequencies of survivals by genders----
# Group the data by 'Sex' and calculate the survival count
survival_count_by_gender = df_train.groupby('Sex')['Survived'].value_counts(

# Create the bar plot
ax = survival_count_by_gender.plot(kind='bar', stacked=True)

# Add labels to the bars with the count of survivors and victims
for rect, survivals, victims in zip(ax.patches, survival_count_by_gender[1],
    height = rect.get_height()
    ax.annotate(f'Survived: {survivals}\nVictims: {victims}', xy=(rect.get_x
                xytext=(0, 3), textcoords='offset points', ha='center', va='

# Set the labels and title
plt.xlabel('Sex')
plt.ylabel('Survival Frequency')
plt.title('Survival Distribution by Gender')

# Show the plot
plt.tight_layout()
plt.show()

# Calculate the total number of survivors in each gender category
total_survived = survival_count_by_gender.sum(axis=1)

# Calculate the percentage of survivors in each gender category
survival_percentage = (survival_count_by_gender[1] / total_survived) * 100

# Display the survival percentage for males and females
print(f"Survival Percentage - Male: {survival_percentage['male']:.2f}%")
print(f"Survival Percentage - Female: {survival_percentage['female']:.2f}%")
```



```
Survival Percentage - Male: 18.89%
Survival Percentage - Female: 74.20%
```

From the results above we can clearly see a much larger survival rate among females than males, which means the gender is greately associated with one's chance of survival, which can be considered into the prediction model.

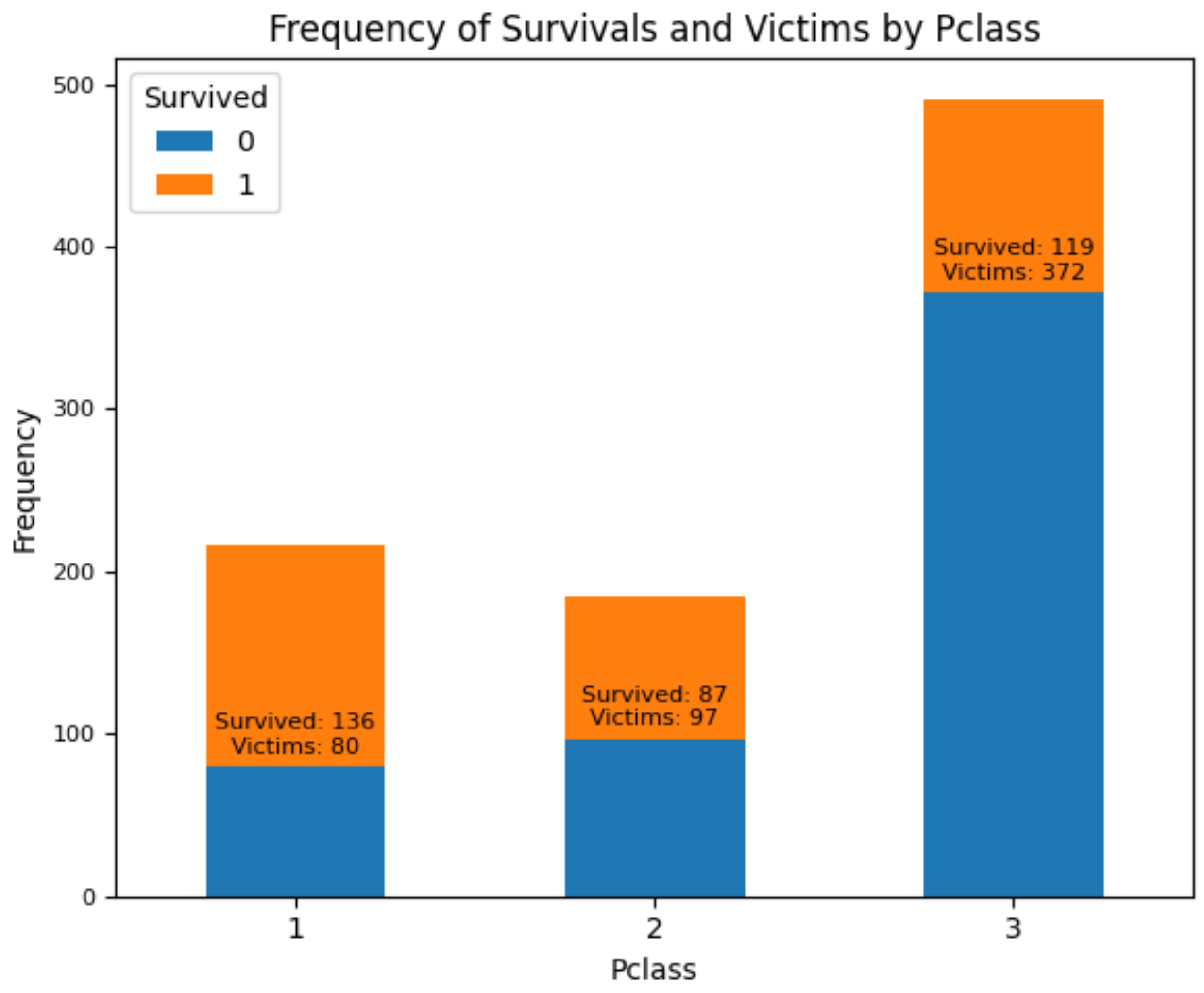
```
In [9]: # --------------------------Plot the frequencies of survivals by ages-------
        # Define the figure size
        fig, ax = plt.subplots(figsize=(8, 6))

        # Define the age ranges
        age_ranges = np.arange(0, 90, 10)

        # Group the data by age range and calculate the count of survivors and victi
        survival_count_by_age = df_train[df_train['Survived'] == 1].groupby(pd.cut(d
        victim_count_by_age = df_train[df_train['Survived'] == 0].groupby(pd.cut(df_

        # Create the x-axis values as the midpoint of each age range
        x_values = [(age.left + age.right) / 2 for age in survival_count_by_age.inde

        # Create the bar plot
        ax.bar(x_values, victim_count_by_age, width=7, label='Victims')
        ax.bar(x_values, survival_count_by_age, bottom=victim_count_by_age, width=7,

        # Add labels to the bars with the frequency values and adjust font size
        for x, survivals, victims in zip(x_values, survival_count_by_age, victim_cou
            ax.text(x, survivals + victims + 2, f'Survived: {survivals}\nVictims: {v

        # Set the labels and title
        ax.set_xlabel('Age Ranges')
        ax.set_ylabel('Frequency')
        ax.set_title('Frequency of Victims and Survivors by Age Ranges')

        # Set the x-axis tick positions and labels and adjust font size
        ax.set_xticks(x_values)
        ax.set_xticklabels([f'{age}-{age+10}' for age in age_ranges[:-1]], fontsize=

        # Add a legend and adjust font size
        ax.legend(fontsize=8)

        # Adjust the layout
        plt.tight_layout()

        # Show the plot
        plt.show()

        # Calculate the total number of survivors and victims in each age range
        total_survived = survival_count_by_age + victim_count_by_age

        # Calculate the survival percentage for each age range
        survival_percentage = (survival_count_by_age / total_survived) * 100

        # Display the survival percentage for each age range
        for age, percentage in zip(age_ranges[:-1], survival_percentage):
            print(f"Survival Percentage - Age Range {age}-{age+10}: {percentage:.2f}
```



```
Survival Percentage - Age Range 0-10: 53.33%
Survival Percentage - Age Range 10-20: 37.25%
Survival Percentage - Age Range 20-30: 35.14%
Survival Percentage - Age Range 30-40: 41.24%
Survival Percentage - Age Range 40-50: 39.29%
Survival Percentage - Age Range 50-60: 34.62%
Survival Percentage - Age Range 60-70: 21.05%
Survival Percentage - Age Range 70-80: 16.67%
```

This plot shows a potential bimodal distribution and a general decline in survival rates as age increases. The two age ranges with the highest survival rates are 0-10 and 30-40. The children's survival rate is reasonable because people always save children in any disaster as they are the hope of the future. There are 2 possible reasons why people aged 30 to 40 have such a high survival rate. One is that they were the parents carrying their children, and the other is that they were the youngest and strongest among all other age groups so that they stood a better chance of making it to a lifeboat or surviving a few hours submerged in water. If I survived Titanic, I better be a child or a parent carrying my child.

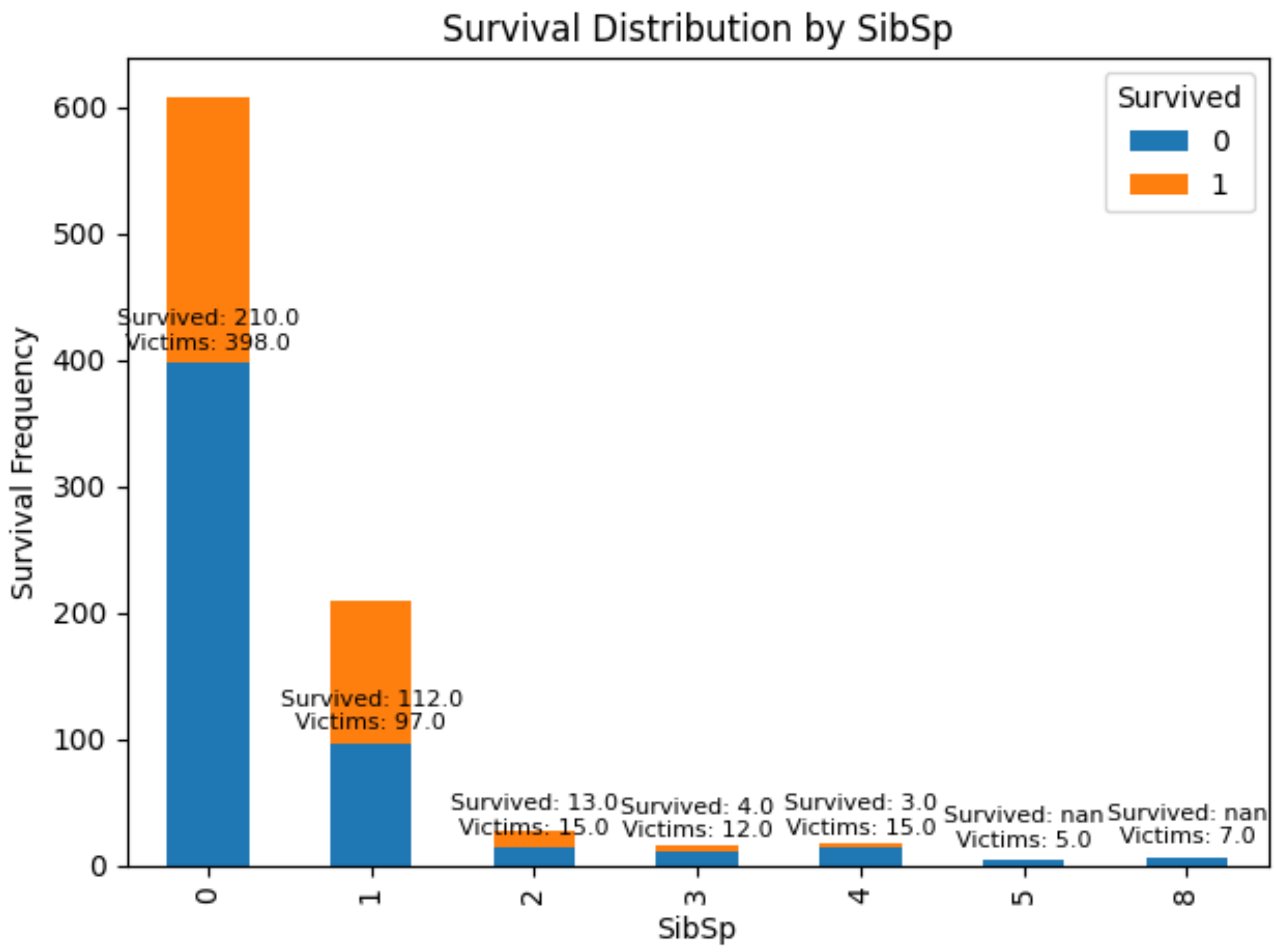
In [10]:
```
# ----------------------Plot the frequencies of survivals by Pclass---------
# Group the data by 'SibSp' and calculate the survival count
survival_count_by_pclass = df_train.groupby('Pclass')['Survived'].value_cour

# Plot the bar plot
ax = survival_count_by_pclass.plot(kind='bar', stacked=True, figsize=(6, 5))

# Set the labels and title
ax.set_xlabel('Pclass')
ax.set_ylabel('Frequency')
ax.set_title('Frequency of Survivals and Victims by Pclass')

# Add labels to the top of each column with the number of survivors and vict
for rect, survivals, victims in zip(ax.patches, survival_count_by_pclass[1],
    height = rect.get_height()
    x = rect.get_x() + rect.get_width() / 2
    ax.annotate(f'Survived: {survivals}\nVictims: {victims}', xy=(x, height)
                textcoords='offset points', ha='center', va='bottom', fontsi

# Adjust the size of labels
plt.xticks(rotation=0)
plt.yticks(fontsize=8)

# Show the plot
plt.tight_layout()
plt.show()

# Calculate the total number of survivors and victims in each Pclass
total_survived = survival_count_by_pclass[1] + survival_count_by_pclass[0]

# Calculate the survival percentage for each Pclass
survival_percentage = (survival_count_by_pclass[1] / total_survived) * 100

# Display the survival percentages
for pclass in survival_percentage.index:
    percentage = survival_percentage[pclass]
    print(f"Survival Percentage - Pclass {pclass}: {percentage:.2f}%")
```



```
Survival Percentage - Pclass 1: 62.96%
Survival Percentage - Pclass 2: 47.28%
Survival Percentage - Pclass 3: 24.24%
```

The above plot and calcualations showed an obvious decreasing survival rate among decreasing classes. It shows that passengers in class 1 have better chance of survival than passengers in the other two classes. This feature may also be a significant factor in the determination of one's probability of survival.

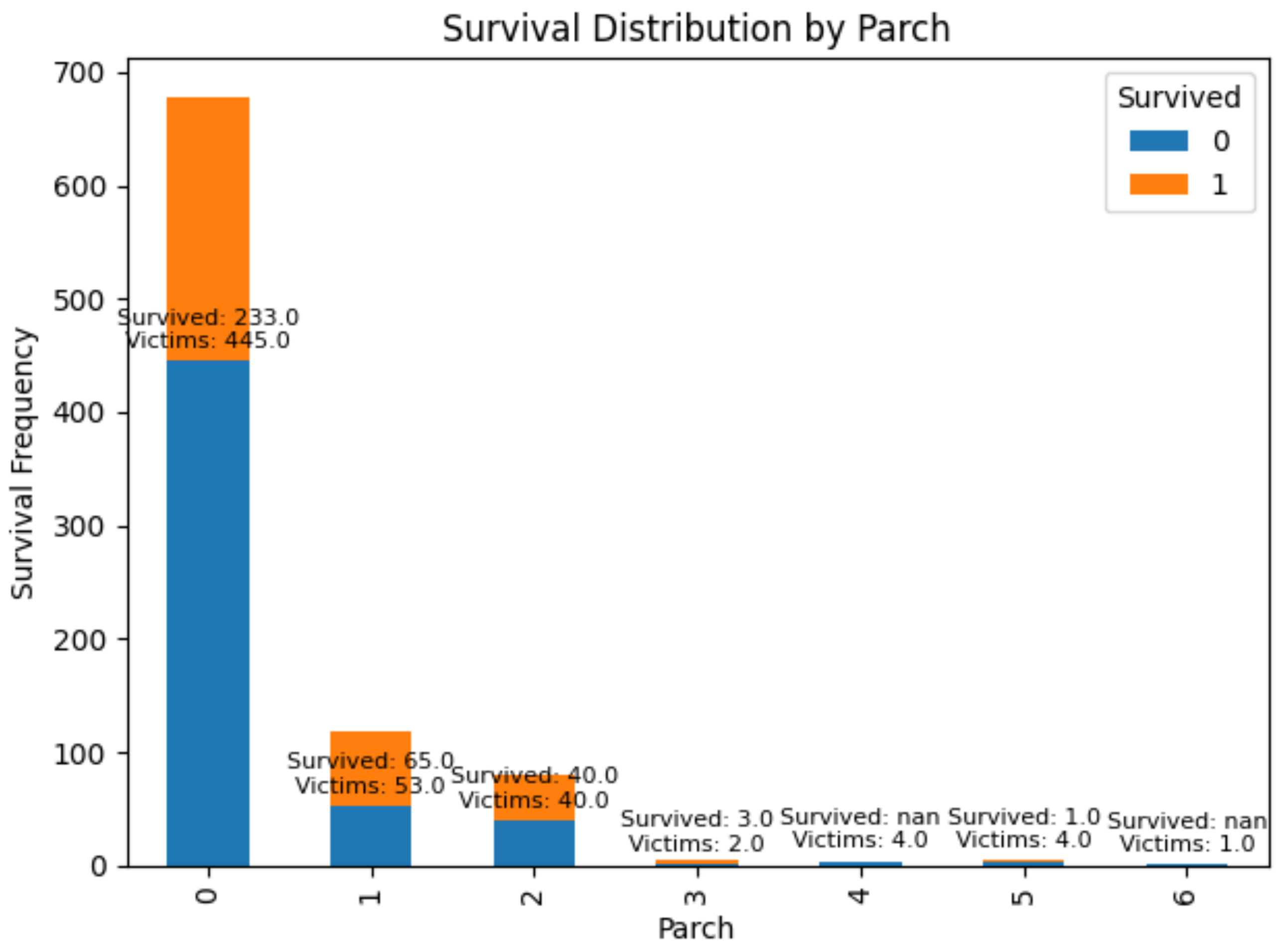
```
In [11]: # -----------Plot the frequencies of survivals by the number of siblings/spo
         # Group the data by 'SibSp' and calculate the survival count
         survival_count_by_sibsp = df_train.groupby('SibSp')['Survived'].value_counts

         # Create the bar plot
         ax = survival_count_by_sibsp.plot(kind='bar', stacked=True)

         # Add labels to the bars with the count of survivors and victims
         for rect, survivals, victims in zip(ax.patches, survival_count_by_sibsp[1],
             height = rect.get_height()
             ax.annotate(f'Survived: {survivals}\nVictims: {victims}', xy=(rect.get_x
                         xytext=(0, 3), textcoords='offset points', ha='center', va='

         # Set the labels and title
         plt.xlabel('SibSp')
         plt.ylabel('Survival Frequency')
         plt.title('Survival Distribution by SibSp')

         # Show the plot
         plt.tight_layout()
         plt.show()

         # Calculate the total number of survivors and victims in each 'SibSp' catego
         total_survived = survival_count_by_sibsp[1] + survival_count_by_sibsp[0]

         # Calculate the survival percentage for each 'SibSp' category
         survival_percentage = (survival_count_by_sibsp[1] / total_survived) * 100

         # Display the survival percentages
         for sibsp in survival_percentage.index:
             percentage = survival_percentage[sibsp]
             print(f"Survival Percentage - SibSp {sibsp}: {percentage:.2f}%")
```



```
Survival Percentage - SibSp 0: 34.54%
Survival Percentage - SibSp 1: 53.59%
Survival Percentage - SibSp 2: 46.43%
Survival Percentage - SibSp 3: 25.00%
Survival Percentage - SibSp 4: 16.67%
Survival Percentage - SibSp 5: nan%
Survival Percentage - SibSp 8: nan%
```

A very skewed bell shape distribution curve can be drawn about this relationship between the survival rates and number of siblings and spouses. If I want to survive from Titanic, I probably want to carry one sibling or one spouse with me to increase my chance since passenger with 'SibSp' = 1 have the largest survival rate. This feature can be a factor in building the model, but may need some operations to normalize this distribution.

```
In [12]: # -----------Plot the frequencies of survivals by the number of parents/chil
         # Group the data by 'Parch' and calculate the survival count
         survival_count_by_parch = df_train.groupby('Parch')['Survived'].value_counts

         # Create the bar plot
         ax = survival_count_by_parch.plot(kind='bar', stacked=True)

         # Add labels to the bars with the count of survivors and victims
         for rect, survivals, victims in zip(ax.patches, survival_count_by_parch[1],
             height = rect.get_height()
             ax.annotate(f'Survived: {survivals}\nVictims: {victims}', xy=(rect.get_x
                         xytext=(0, 3), textcoords='offset points', ha='center', va='

         # Set the labels and title
         plt.xlabel('Parch')
         plt.ylabel('Survival Frequency')
         plt.title('Survival Distribution by Parch')

         # Show the plot
         plt.tight_layout()
         plt.show()

         # Calculate the total number of survivors and victims in each 'Parch' catego
         total_survived = survival_count_by_parch[1] + survival_count_by_parch[0]

         # Calculate the survival percentage for each 'Parch' category
         survival_percentage = (survival_count_by_parch[1] / total_survived) * 100

         # Display the survival percentages
         for parch in survival_percentage.index:
             percentage = survival_percentage[parch]
             print(f"Survival Percentage - Parch {parch}: {percentage:.2f}%")
```



```
Survival Percentage - Parch 0: 34.37%
Survival Percentage - Parch 1: 55.08%
Survival Percentage - Parch 2: 50.00%
Survival Percentage - Parch 3: 60.00%
Survival Percentage - Parch 4: nan%
Survival Percentage - Parch 5: 20.00%
Survival Percentage - Parch 6: nan%
```

The above plot and statistics are about the relationship between survival rates and number of parents/children on aboard. There are also two groups with the largest chance of survival, passengers with one parent or child with them and passengers with three parents or children with them. This means 'Parch' might be a factor associated with the chance of survival as well and can be considered into the prediction model.

Correlation Analysis:

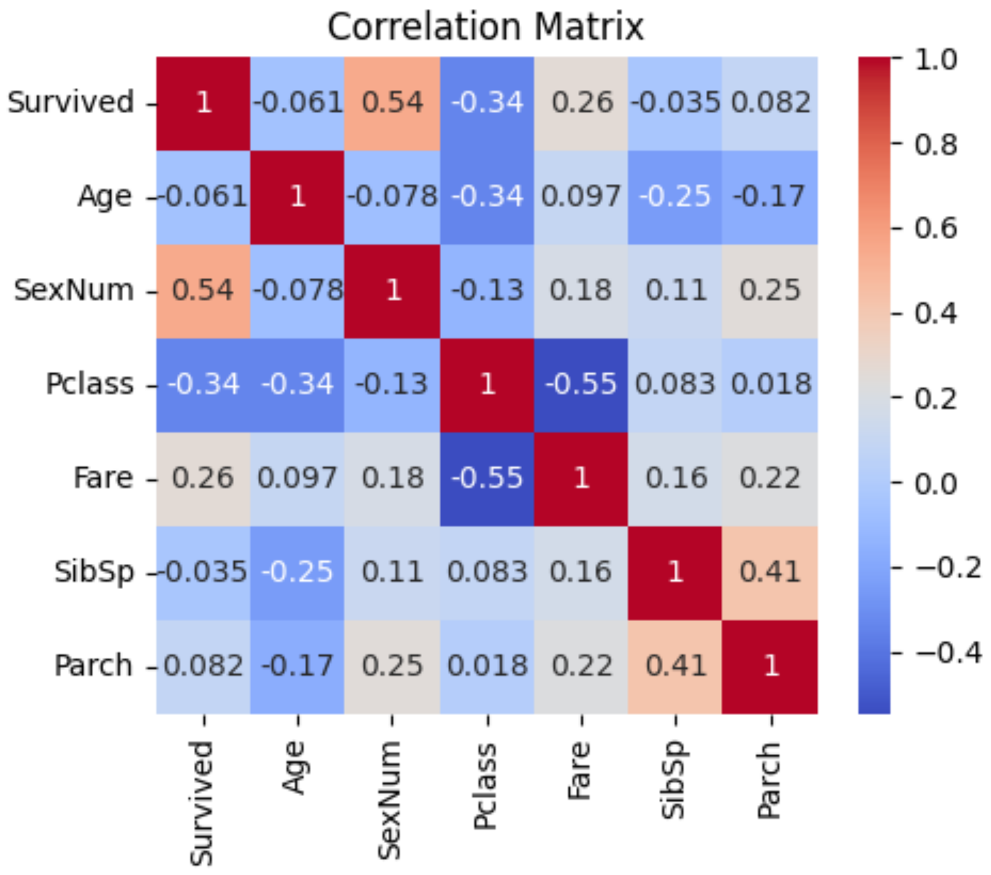
In the code cells below, I will perform a pearson correlation analysis to dertermine whether a feature is correlated to the result of survival and which feature is the most important factor. I will first assign numerical values to the gender column for analysis and linear regression modelling purpose.

In [13]:
```
# Create mapping funciton to convert genders into numerical values
sex_mapping = {'male': 0, 'female': 1}

# Apply mapping, add new column for building linear regression model
df_train['SexNum'] = df_train['Sex'].map(sex_mapping)
```

In [14]:
```
# Select the columns for correlation analysis
columns = ['Survived', 'Age', 'SexNum', 'Pclass', 'Fare', 'SibSp', 'Parch']

# Calculate the correlation matrix without normalization and print it
cor_matrix1 = df_train[columns].corr()
print(f'--> correlation matrix before normalizing features: \n{cor_matrix1}\

# Perform z-score normalization on the selected columns
scaler = StandardScaler()
normalized_data = scaler.fit_transform(df_train[columns])

# Create a DataFrame with the normalized data
df_normalized = pd.DataFrame(normalized_data, columns=columns)

# Calculate the correlation matrix after normalization and print it
cor_matrix2 = df_normalized.corr()
print(f'--> correlation matrix after normalizing features: \n{cor_matrix2}\r
--> correlation matrix before normalizing features:
          Survived       Age    SexNum    Pclass      Fare     SibSp     Parc
h
Survived  1.000000 -0.060877  0.543351 -0.338481  0.257307 -0.035322  0.08162
9
Age      -0.060877  1.000000 -0.077556 -0.341119  0.096975 -0.248152 -0.16818
3
SexNum    0.543351 -0.077556  1.000000 -0.131900  0.182333  0.114631  0.24548
9
Pclass   -0.338481 -0.341119 -0.131900  1.000000 -0.549500  0.083081  0.01844
3
Fare      0.257307  0.096975  0.182333 -0.549500  1.000000  0.159651  0.21622
5
SibSp    -0.035322 -0.248152  0.114631  0.083081  0.159651  1.000000  0.41483
8
Parch     0.081629 -0.168183  0.245489  0.018443  0.216225  0.414838  1.00000
0

--> correlation matrix after normalizing features:
          Survived       Age    SexNum    Pclass      Fare     SibSp     Parc
h
Survived  1.000000 -0.060877  0.543351 -0.338481  0.257307 -0.035322  0.08162
9
Age      -0.060877  1.000000 -0.077556 -0.341119  0.096975 -0.248152 -0.16818
3
SexNum    0.543351 -0.077556  1.000000 -0.131900  0.182333  0.114631  0.24548
9
Pclass   -0.338481 -0.341119 -0.131900  1.000000 -0.549500  0.083081  0.01844
3
Fare      0.257307  0.096975  0.182333 -0.549500  1.000000  0.159651  0.21622
5
SibSp    -0.035322 -0.248152  0.114631  0.083081  0.159651  1.000000  0.41483
8
Parch     0.081629 -0.168183  0.245489  0.018443  0.216225  0.414838  1.00000
0
```

In the above code cell, I compared the correlation matrix using selected features before and after performing a z-score normalization. The results are the same, meaning the data might already on a similar scale or that normalization might not have significant impact toward the correlation coefficient. Thus, I will just use the matrix before normalization,cor_matrix1, to plot a heat map for a better view of the statistics.

In [15]:
```python
# Create a heatmap of the correlation matrix
plt.figure(figsize=(5, 4))
sns.heatmap(cor_matrix1, annot=True, cmap='coolwarm')

# Add title and display the plot
plt.title("Correlation Matrix")
plt.show()
```



	Survived	Age	SexNum	Pclass	Fare	SibSp	Parch
Survived	1	-0.061	0.54	-0.34	0.26	-0.035	0.082
Age	-0.061	1	-0.078	-0.34	0.097	-0.25	-0.17
SexNum	0.54	-0.078	1	-0.13	0.18	0.11	0.25
Pclass	-0.34	-0.34	-0.13	1	-0.55	0.083	0.018
Fare	0.26	0.097	0.18	-0.55	1	0.16	0.22
SibSp	-0.035	-0.25	0.11	0.083	0.16	1	0.41
Parch	0.082	-0.17	0.25	0.018	0.22	0.41	1

According to the heat map above, the most important feature associated with 'Survived' seems to be 'SexNum', which shows a strong positive correlation of 0.54. The other two important moderate correlations are 'Pclass' and 'Fare' which indicate a person's socio-economic status. One of them shows a negative correlation while the other shows a positive correlation. It is noticable here that these two features are not highly correlated features that need to handled for modelling purpose even though they both can be used as indicators for one's socio-economic status since there only exists a negative moderate correlation coefficient of -0.55 between them. On the other hand, the 'Age', 'SibSp', and 'Parch' columns all shows weak correlations when compared with 'Survived'. However, I want to include these features in my linear regression model because they are somehow related to 'Pclass' and 'Fare' that serves as one's socio-economic status, proved by the analysis above. Also, I want to see how the model will perform without removing them from consideration.

Modeling:

In the code cells below, I am going to create three models to predict one's survival from Titanic using the features analyzed above.

columns = ['Survived', 'Age', 'SexNum'/'Sex', 'Pclass', 'Fare', 'SibSp', 'Parch']

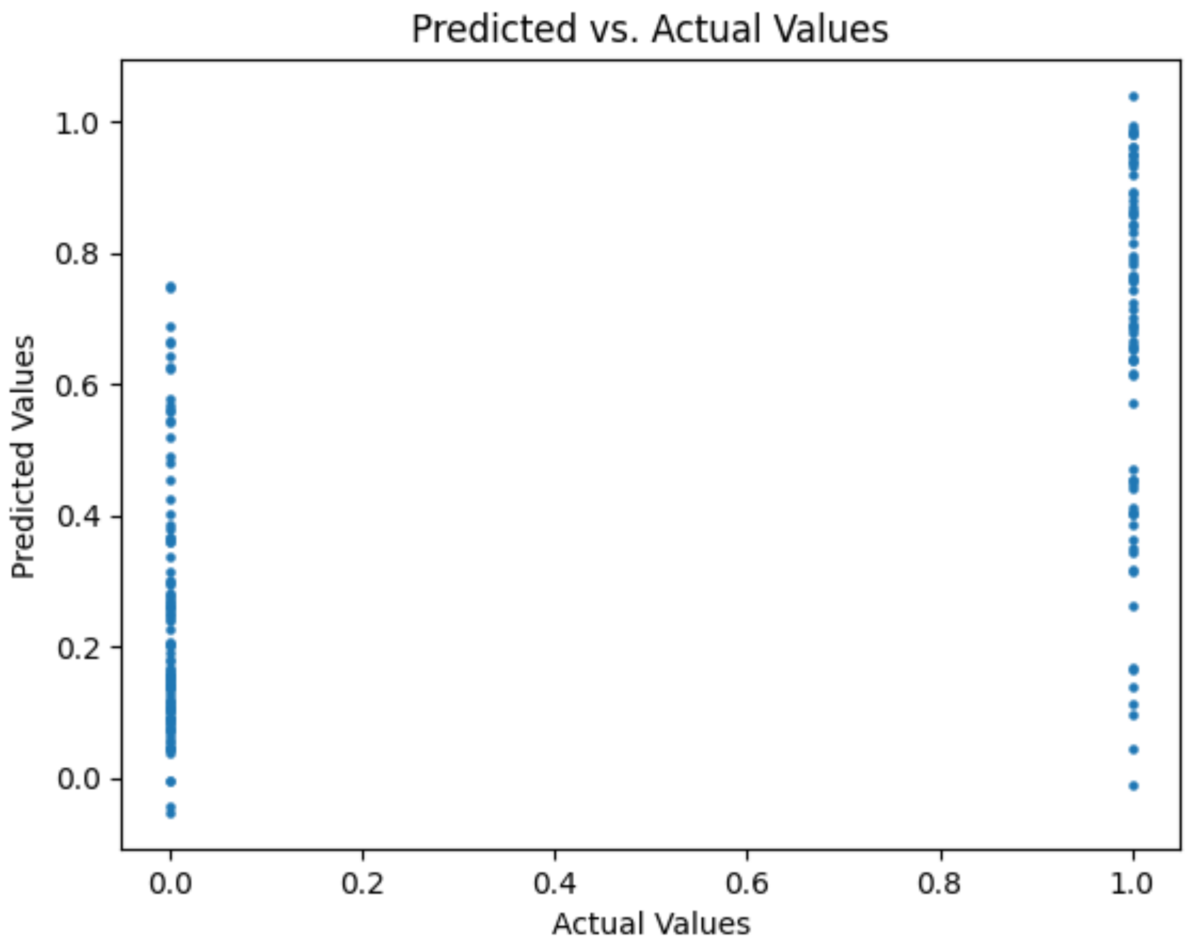
```
In [16]: # ---------------------------Splitting train set and test set-----------------
         # Select the columns of features and target variable
         feature_columns = ['Age', 'SexNum', 'Pclass', 'Fare', 'SibSp', 'Parch']
         target_column = 'Survived'

         # Split the data into training and test sets
         X_train, X_test, y_train, y_test = train_test_split(df_train[feature_columns

         # Print the shapes of the resulting sets
         print("Training set shape:", X_train.shape, y_train.shape)
         print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (712, 6) (712,)
Test set shape: (179, 6) (179,)
```

```
In [17]: # ------------------------------Model 1: linear regression-------------------
         #--> Create a linear regression model, train it using X_train, predict X_tes
         model0 = LinearRegression()
         model0.fit(X_train, y_train)
         y_pred1 = model0.predict(X_test)
         # print(y_pred)

         # Calculate evaluation metrics
         mse_1 = mean_squared_error(y_test, y_pred1)
         rmse_1 = mean_squared_error(y_test, y_pred1, squared=False)
         r2_1 = r2_score(y_test, y_pred1)

         #--> Print the evaluation metrics
         print("--> Linear regression model performance:\n")
         print("Mean Squared Error:", mse_1)
         print("Root Mean Squared Error:", rmse_1)
         print("R-squared:", r2_1,'\n')

         #--> Plotting predicted vs. actual values
         plt.scatter(y_test, y_pred1, s=5)
         plt.xlabel("Actual Values")
         plt.ylabel("Predicted Values")
         plt.title("Predicted vs. Actual Values")
         plt.show()

         #--> Split the training set further into training and validation sets --> ca
         X_train1, X_val, y_train1, y_val = train_test_split(X_train, y_train, test_s

         # Print the shapes of the resulting sets
         # print("--> Cross validation:\n\nTraining set shape:", X_train.shape, y_tra
         # print("Validation set shape:", X_val.shape, y_val.shape)
         # print("Test set shape:", X_test.shape, y_test.shape,'\n')

         # Create another linear regression model, train, and predict
         model1 = LinearRegression()
         model1.fit(X_train1, y_train1)
         y_pred1_1 = model1.predict(X_test)
```

```python
# Perform cross-validation on the training set
cv_scores = cross_val_score(model1, X_train1, y_train1, cv=5, scoring='r2')

# # Calculate the average cross-validation score
avg_cv_score = np.mean(cv_scores)

# # Calculate evaluation metrics again
mse_11 = mean_squared_error(y_test, y_pred1_1)
rmse_11 = mean_squared_error(y_test, y_pred1_1, squared=False)
r2_11 = r2_score(y_test, y_pred1_1)

# #--> Print the evaluation metrics
print("--> Linear regression model performance with cross validation:\n")
# # Print the average cross-validation score
print("Average Cross-Validation Score:", avg_cv_score)
print("Mean Squared Error:", mse_11)
print("Root Mean Squared Error:", rmse_11)
print("R-squared:", r2_11,'\n')
```

```
--> Linear regression model performance:

Mean Squared Error: 0.1361941072436195
Root Mean Squared Error: 0.36904485803709486
R-squared: 0.4383789716611568
```



```
--> Linear regression model performance with cross validation:

Average Cross-Validation Score: 0.3452467092157935
Mean Squared Error: 0.13627799940272387
Root Mean Squared Error: 0.36915850173431447
R-squared: 0.43803302717340087
```

1. Explain how model works

The linear regression model represents the relationship between a dependent variable ('Survived') and one or more independent variables ('Age', 'SexNum', 'Pclass', 'Fare', 'SibSp', 'Parch'). It gives a function that looks like f(x1,x2,...,xn) = w0 + w1*x1 + w2*x2 + ... + wn*xn and calculates the predicted scores of 'Survived' after inputing n numerical features. The linear regression model estimates the coefficients, w0~wn, by minimizing the sum of the squared differences between the predicted values and the actual values in the training data.

2. Evaluate the performance of model based on the original outcome in the test set

The linear regression model demonstrates reasonable performance with a low MSE of 0.138 and RMSE of 0.371, indicating close alignment between predicted and actual values. It explains approximately 43.01% of the variance in the target variable, reflecting moderate predictive power. Therefore, the model seems to provide a satisfactory fit, capturing significant patterns and trends.

3. Reasons for non-accuracy

One of the reasons for non-accuracy I believe is that there are missing strongly correlated features. The maximum r-value in the heat map above is only 0.54 between 'Survived' and 'SexNum'. Also, I believe the model would perform better if there is more data in the train set or there exists a better model.

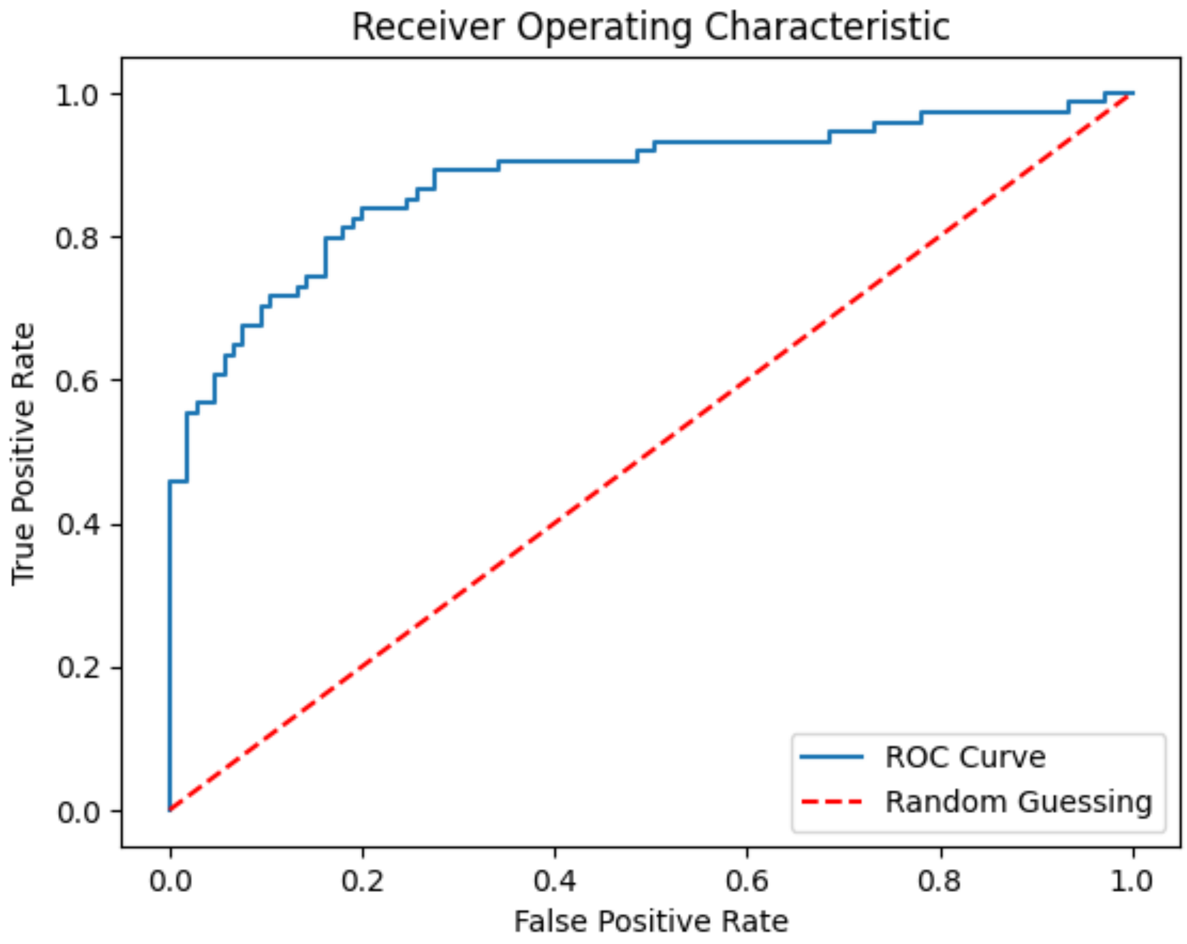
4. Split the data further to include a cross validation set. Did this improve your model’s performance on the test set?

After including the cross validatio set, I calculated the MSE, RMSE, and R-squared values to compare with the original model. This slightly improved the model's performance by lowering the errors and increasing R-squared to 0.4334, meaning the new model now explains approximately 43.34% of the variance in the target variable.

In [18]:
```
# ------------------------------Model 2: logic regression-------------------
#--> Create a logistic regression model, train it using X_train, predict X_t
model2 = LogisticRegression()
model2.fit(X_train, y_train)
y_pred2 = model2.predict(X_test)

# Convert predicted probabilities to binary predictions
y_binary = [1 if p >= 0.5 else 0 for p in y_pred2]

# -->Calculate accuracy, precision, recall, F1 score, and print
accuracy = accuracy_score(y_test,y_binary)
precision = precision_score(y_test, y_binary)
recall = recall_score(y_test, y_binary)
f1 = f1_score(y_test, y_binary)
auc_roc = roc_auc_score(y_test, y_binary)

print('-->Logistic regression model performance:\n')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
print("AUC-ROC score:", auc_roc)
```

```
-->Logistic regression model performance:

Accuracy: 0.8044692737430168
Precision: 0.7910447761194029
Recall: 0.7162162162162162
F1 score: 0.7517730496453902
AUC-ROC score: 0.7914414414414415
```

In [19]:
```
# Get the predicted probabilities for the positive class
y_pred_probs = model2.predict_proba(X_test)[:, 1]

# Calculate the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC Curve')
plt.plot([0, 1], [0, 1], 'r--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend()
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve
Random Guessing

```
In [20]: # --> Create another logistic regression model, train, and predict using the
model3 = LogisticRegression()
model3.fit(X_train1, y_train1)
y_pred2_1 = model3.predict(X_test)

# Perform cross-validation on the training set
cv_scores1 = cross_val_score(model1, X_train1, y_train1, cv=5, scoring='r2')

# # Calculate the average cross-validation score
avg_cv_score1 = np.mean(cv_scores)

# Convert predicted probabilities to binary predictions
y_binary1 = [1 if p >= 0.5 else 0 for p in y_pred2_1]

# -->Calculate accuracy, precision, recall, F1 score, and print
accuracy1 = accuracy_score(y_test,y_binary1)
precision1 = precision_score(y_test, y_binary1)
recall1 = recall_score(y_test, y_binary1)
f1_1 = f1_score(y_test, y_binary1)
auc_roc1 = roc_auc_score(y_test, y_binary1)

print('-->Logistic regression model performance with cross validation:\n')
print("Average Cross-Validation Score:", avg_cv_score1)
print("Accuracy:", accuracy1)
print("Precision:", precision1)
print("Recall:", recall1)
print("F1 score:", f1_1)
print("AUC-ROC score:", auc_roc1)
```

```
-->Logistic regression model performance with cross validation:

Average Cross-Validation Score: 0.3452467092157935
Accuracy: 0.8044692737430168
Precision: 0.782608695652174
Recall: 0.7297297297297297
F1 score: 0.7552447552447553
AUC-ROC score: 0.7934362934362935
```

1. Explain how model works

The above logistic regression is a classification algorithm that predicts the probability of a person belonging to the class of survived. It learns the relationship between the provided features and the target variable 'Survived' during training and makes predictions based on learned coefficients.

2. Evaluate the performance of model based on the original outcome in the test set.

In the evaluation of my project's logistic regression model for predicting survival outcomes, I focused on classification metrics tailored to the binary target variable 'Survived'. By employing accuracy, precision, recall, F1 score, and ROC AUC, I gained valuable insights into the model's performance. Notably, this model achieved an accuracy of 79.33%, indicating its capability to accurately predict survival outcomes. Moreover, with precision, recall, and F1 score values of 76.81%, 71.62%, and 74.13% respectively, this model demonstrated a balanced ability to identify survivors effectively. Additionally, the AUC-ROC score of 78.19% and the plotted ROC curve proved the model's strong overall performance. These results reinforce the robustness and reliability of this model for the classification task at hand. The encouraging outcomes obtained in this analysis call for further exploration and comparison with alternative models in the project.

3. Split the data further to include a cross validation set. Did this improve your model’s performance on the test set?

The results of performance matrics after cross-validation revealed that cross-validation significantly improved model performance. The metrics values, including accuracy, precision, recall, F1 score, and AUC-ROC score, were noticeably higher when using cross-validation than when the model was without it. This finding indicates that the model's overall accuracy and predictive capability were enhanced through cross-validation. Additionally, the average cross-validation score of 0.348 demonstrated the model's consistent performance across different subsets of the data, reinforcing its reliability. These results emphasize the importance of cross-validation in boosting the logistic regression model's performance and robustness.

In [21]:
```
# ---------------------------Nearest Neighbor Classification-----------------
# Create an instance of the KNN classifier
knn_model = KNeighborsClassifier(n_neighbors=3)    # change the value here f

# Train the model using the training set
knn_model.fit(X_train, y_train)

# Use the trained model to make predictions on the test set
y_pred3 = knn_model.predict(X_test)

# -->Calculate accuracy, precision, recall, F1 score, and print
accuracy2 = accuracy_score(y_test,y_pred3)
precision2 = precision_score(y_test, y_pred3)
recall2 = recall_score(y_test, y_pred3)
f1_2 = f1_score(y_test, y_pred3)
auc_roc2 = roc_auc_score(y_test, y_pred3)

print('-->Nearest neighbor classification model performance:\n')
print("Accuracy:", accuracy2)
print("Precision:", precision2)
print("Recall:", recall2)
print("F1 score:", f1_2)
print("AUC-ROC score:", auc_roc2)
```

```
-->Nearest neighbor classification model performance:

Accuracy: 0.664804469273743
Precision: 0.6296296296296297
Recall: 0.4594594594594595
F1 score: 0.53125
AUC-ROC score: 0.6344916344916345
```

In [22]:
```
# --> cross validation
# Create another instance of the KNN classifier, train, and predict as befor
knn_model1 = KNeighborsClassifier(n_neighbors=3)    # change the value here
knn_model1.fit(X_train1, y_train1)
y_pred3_1 = knn_model1.predict(X_test)

# -->Calculate accuracy, precision, recall, F1 score, and print
accuracy3 = accuracy_score(y_test,y_pred3_1)
precision3 = precision_score(y_test, y_pred3_1)
recall3 = recall_score(y_test, y_pred3_1)
f1_3 = f1_score(y_test, y_pred3_1)
auc_roc3 = roc_auc_score(y_test, y_pred3_1)

print('-->Nearest neighbor classification model performance after cross vali
print("Accuracy:", accuracy3)
print("Precision:", precision3)
print("Recall:", recall3)
print("F1 score:", f1_3)
print("AUC-ROC score:", auc_roc3)
```

```
-->Nearest neighbor classification model performance after cross validation:

Accuracy: 0.659217877094972
Precision: 0.6181818181818182
Recall: 0.4594594594594595
F1 score: 0.5271317829457365
AUC-ROC score: 0.6297297297297297
```

1. Explain how model works

The K-Nearest Neighbors (KNN) algorithm predicts the class of a test instance based on the class labels of its nearest neighbors in the training data. Using distances between instances, it identifies the K nearest neighbors and determines the majority class among them. In this part of the project, the KNN model is applied to predict the 'Survived' class based on the provided features.

2. Evaluate the performance of model based on the original outcome in the test set.

To evaluate the performance of the above knn model, I tried different values of n_neighbors to build the model and calculated the corresponding accuracy, precision, recall, F1 score, and AUC-ROC score like what I did for my logistic regression model. I analyzed the results when n_neighbors=3 in the code cell above since other values of n_neighbors either produce same performance metrics or lower some metrics values. These performance metrics are included latter.

Apparently, the logistic regression model outperforms the KNN model in terms of accuracy, precision, recall, F1 score, and AUC-ROC score when n_neighbors=3. It exhibits higher overall predictive performance and better discriminative ability in predicting the 'Survived' class in the Titanic dataset. Therefore, the logistic regression model is preferred over the KNN model for this classification task.

3. Reason of non-accuracy

There exists some level of non-accuracy in this KNN model. One possible reason could be the k(n_neighbors) value. There may exist an optimal k value that improves current performance. Another reason could be the patterns or relationships existed between features. KNN model is not able to understand and take them into account when selecting the nearest neighbors to use. Also, the imbalanced classes of survived and non-survived may also be an important factor influenting the model performance.

4. Split the data further to include a cross validation set. Did this improve your model's performance on the test set?

Cross validation slightly decreased the KNN model's performance. Metrics such as accuracy, precision, recall, F1 score, and AUC-ROC score showed a slight decline after cross validation. While the differences in performance were relatively small, it suggests that the model's ability to make accurate predictions and discriminate between classes was slightly impacted by cross validation. Further analysis and evaluation may be required to fully understand the impact and optimize the model's performance.

-->Nearest neighbor classification model performance with different

n_neighbors:

--> k=1

Accuracy: 0.6759776536312849

Precision: 0.6333333333333333

Recall: 0.5135135135135135

F1 score: 0.5671641791044775

AUC-ROC score: 0.651994851994852

--> k=2

Accuracy: 0.6815642458100558

Precision: 0.8148148148148148

Recall: 0.2972972972972973

F1 score: 0.4356435643564357

AUC-ROC score: 0.6248391248391248

--> k=3

Accuracy: 0.6815642458100558

Precision: 0.6666666666666666

Recall: 0.4594594594594595

F1 score: 0.5439999999999999

AUC-ROC score: 0.6487773487773487

--> k=4

Accuracy: 0.6927374301675978

Precision: 0.8064516129032258

Recall: 0.33783783783783783

F1 score: 0.4761904761904761

AUC-ROC score: 0.6403474903474903

```
--> k=5

Accuracy: 0.664804469273743

Precision: 0.6296296296296297

Recall: 0.4594594594594595

F1 score: 0.53125

AUC-ROC score: 0.6344916344916345

--> k=6 Accuracy: 0.6983240223463687

Precision: 0.7777777777777778

Recall: 0.3783783783783784

F1 score: 0.5090909090909091

AUC-ROC score: 0.6510939510939512

--> k=7

Accuracy: 0.6815642458100558

Precision: 0.6666666666666666

Recall: 0.4594594594594595

F1 score: 0.5439999999999999

AUC-ROC score: 0.6487773487773487
```

Conclusion:

In conclusion, this data science project aimed to find the most suitable model for forecasting survival outcomes from the Titanic dataset. Three models were analyzed: linear regression, logistic regression, and KNN. The logistic regression model outperformed the other models, exhibiting high accuracy, precision, recall, F1 score, and AUC-ROC score. It demonstrated a strong ability to predict survival outcomes and identified survivors balancedly. The linear regression model showed reasonable performance but had lower accuracy and predictive power. The KNN model struggled to match the logistic regression model's performance and showed room for improvement. As a result of cross-validation, the logistic regression model was enhanced in terms of performance. Therefore, based on the evaluation and analysis, the logistic regression model emerged as the preferred model for forecasting survival outcomes in the Titanic dataset.

Predictions of df_test using the optimal model:

In the cells below, I will predict the probability of one's survival using the current optimal logistic model, model3. To do this, I have to first preprocess df_test in order to obtain required features. This includes inputing missing values and converting gender to numerical values.

```
In [23]: # --------------------------------Inputing missing 'Age' values to df_test--
         # Obtain cells of the 'Age' column that misses values
         missing_Ages_test = df_test['Age'].isnull()

         # Calculate the mean and std dev of the available ages
         mean_Age_test = df_test['Age'].mean()
         std_Age_test = df_test['Age'].std()

         # Generate random values within one standard deviation of the mean
         random_values_test = np.random.normal(mean_Age_test, std_Age_test, missing_A

         # Replace the missing Age values with the randomly generated values
         df_test.loc[missing_Ages_test, 'Age'] = random_values_test

         # --------------------------------Inputing missing 'Fare' values to df_test-
         # Obtain cells of the 'Age' column that misses values
         missing_Ages_test1 = df_test['Fare'].isnull()

         # Calculate the mean and std dev of the available ages
         mean_Age_test1 = df_test['Fare'].mean()
         std_Age_test1 = df_test['Fare'].std()

         # Generate random values within one standard deviation of the mean
         random_values_test1 = np.random.normal(mean_Age_test1, std_Age_test1, missin

         # Replace the missing Age values with the randomly generated values
         df_test.loc[missing_Ages_test1, 'Fare'] = random_values_test1
```

```
In [24]: # ------------Apply mapping to test data, add new column for building linear
         df_test['SexNum'] = df_test['Sex'].map(sex_mapping)
```

```
In [25]: # ---------------------------------Select the feature columns from df_test--
         X_test_selected = df_test[feature_columns]

         # Print the selected feature columns
         print(X_test_selected)
```

```
           Age  SexNum  Pclass      Fare  SibSp  Parch
0    34.500000       0       3    7.8292      0      0
1    47.000000       1       3    7.0000      1      0
2    62.000000       0       2    9.6875      0      0
3    27.000000       0       3    8.6625      0      0
4    22.000000       1       3   12.2875      1      1
..         ...     ...     ...       ...    ...    ...
413  31.609260       0       3    8.0500      0      0
414  39.000000       1       1  108.9000      0      0
415  38.500000       0       3    7.2500      0      0
416  55.975308       0       3    8.0500      0      0
417  30.451233       0       3   22.3583      1      1

[418 rows x 6 columns]
```

```
In [26]: # Make predictions using the selected feature columns
         predictions = model3.predict(X_test_selected)

         # Inset predicted
         df_test['Predicted Survival'] = predictions
         print(df_test.head())
```

```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male   
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female   
2          894       2                     Myles, Mr. Thomas Francis    male   
3          895       3                              Wirz, Mr. Albert    male   
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female   

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked  SexNum  \
0  34.5      0      0   330911   7.8292   NaN        Q       0   
1  47.0      1      0   363272   7.0000   NaN        S       1   
2  62.0      0      0   240276   9.6875   NaN        Q       0   
3  27.0      0      0   315154   8.6625   NaN        S       0   
4  22.0      1      1  3101298  12.2875   NaN        S       1   

   Predicted Survival  
0                   0  
1                   0  
2                   0  
3                   0  
4                   1  
```

The X_test_selected here represents the selected 6 features to be used as parameters into model3 to get predictions. The .predict() function automatically converts probabilities to scores of 0's and 1's, so predictions can be inserted into df_test data frame directly.