Analyzing Titanic Passenger Survival with KNN Classification

By Meenakshi Sharad Sethi

Disclaimer: This project is conducted solely for academic purposes and as part of individual learning. The analysis and conclusions presented herein are based on personal exploration and may not be suitable for commercial or professional use without further validation and refinement. Any decisions made based on the findings of this project should be taken with caution, and additional verification may be necessary before applying the results in real-world scenarios.

Introduction:

The project aims to employ machine learning techniques to analyze the Titanic dataset, investigating factors influencing passenger survival during the historic maritime disaster.The objective is to glean insights from historical data and develop predictive models to understand survival outcomes in similar contexts. By leveraging K-nearest neighbors (KNN) classification, the project seeks to build a predictive model capable of determining whether a passenger survived or not based on various demographic and trip-related features.

Approach:

- **Data Exploration**: Initial exploration of the dataset involves understanding its structure, including features such as survival status, passenger demographics, ticket class, and fare. Descriptive statistics and visualizations are used to uncover patterns and relationships within the data, providing insights into factors potentially correlated with survival.

- **Data Cleaning and Preprocessing**: The dataset undergoes cleaning to address missing values, outliers and inconsistencies, ensuring data quality and reliability. Feature engineering may be performed to extract additional insights or enhance predictive performance, such as deriving new variables from existing ones or encoding categorical variables.

- **Model Building - KNN Classification**: KNN classification is chosen as the primary modeling approach due to its simplicity and interpretability, making it suitable for predictive analysis in this context. The dataset is split into training and testing sets to evaluate model performance effectively. The KNN algorithm is trained on the training data, utilizing features such as passenger attributes, ticket information, and embarkation port to predict survival outcomes.

- **Model Evaluation and Improvement**:Model performance is assessed using evaluation metrics such as accuracy, precision, recall, and F1-score, providing a comprehensive understanding of predictive capabilities. Hyperparameter tuning, including optimizing the value of K (number of neighbors), may be conducted using techniques like grid search to enhance model performance.

Importing necessary libraries

In [1]:
```python
import os
os.environ['OPENBLAS_NUM_THREADS'] = '5'
```

In [2]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```python
# Importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Explore the dataset

In [4]:
```python
# Load Titanic dataset

df = pd.read_csv('titanic_data.csv')
df
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN
...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN
887	888	1	1	Graham, Miss.	female	19.0	0	0	112053	30.0000	B42

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
				Margaret Edith							
888	889	0	3	Johnston, Miss. Catherine Helen \Carrie\""	female	NaN	1	2	W./C. 6607	23.4500	NaN
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN

891 rows × 12 columns

In [5]:

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

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN

In [6]:

```
# Display dataset information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Key points from the info summary:

- There are 891 rows in the dataset.
- The "Age" column has 714 non-null entries, indicating missing values.
- The "Cabin" column has only 204 non-null entries, indicating a significant number of missing values.
- The "Embarked" column has 889 non-null entries, suggesting a couple of missing values.

In [7]:

```
# Summary statistics

df.describe()
```

Out[7]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

interpretation of the summary statistics

- **PassengerId**: This column represents the unique identifier for each passenger.

- **Survived**: This column indicates whether the passenger survived or not, with 0 representing not survived and 1 representing survived. The mean indicates that approximately 38.4% of passengers survived.

- **Pclass**: This column represents the ticket class of the passenger, with values 1, 2, and 3 representing 1st, 2nd, and 3rd class, respectively. The mean indicates that on average, passengers were in the 2nd class.

- **Age**: This column represents the age of the passengers. There are 714 non-null entries, indicating missing values. The mean age is approximately 29.7 years, with a standard deviation of approximately 14.5 years. The minimum age is 0.42 years (approximately 5 months) and the maximum age is 80 years.

- **SibSp**: This column represents the number of siblings or spouses aboard the Titanic for each passenger. The mean indicates that on average, passengers had approximately 0.52 siblings or spouses aboard.

- **Parch**: This column represents the number of parents or children aboard the Titanic for each passenger. The mean indicates that on average, passengers had approximately 0.38 parents or children aboard.

- **Fare**: This column represents the passenger fare. The mean fare paid by passengers is approximately 32.20, with a wide range of values indicated by the standard deviation and the large difference between the 75th percentile and the maximum value.

In [8]:

```
# Check for missing values

print(df.isnull().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In analyzing Titanic passenger survival using KNN classification, we typically focus on features that are likely to have predictive power.

Relevant columns for analyzing passenger survival are:

- **1. Survived**: This is the target variable we're trying to predict, so it's obviously relevant.
- **2. Pclass**: The ticket class could be relevant as it might correlate with socio-economic status, which could impact survival chances. It should be included.
- **3. Sex**: Gender could be a significant factor in survival, as there was a "women and children first" policy during the evacuation. It should be included.
- **4.Age**: Age could also be a significant factor, as older individuals or infants might have had different survival rates. Despite missing values, it should be included.
- **5. SibSp**: The number of siblings/spouses could be relevant, as individuals with family members onboard might have had different survival strategies. It should be included.
- **6. Parch**: Similarly to SibSp, the number of parents/children could impact survival and should be included.
- **7. Fare**: Fare might be correlated with socio-economic status and, therefore, survival chances. It should be included.
- **8. Embarked**: The port of embarkation might have some correlation with survival due to potential differences in passenger demographics. It should be included.

Irrelevant columns for analyzing passenger survival are:

- **1. PassengerId**: This column is just an identifier and does not provide any meaningful information for predicting survival. It can be excluded from the analysis.
- **2. Name**: The name of the passenger is unlikely to have a direct impact on survival and can be excluded.
- **3. Ticket**: The ticket number is unlikely to have predictive power and can be excluded.
- **4. Cabin**: With a significant number of missing values (687 out of 891), the cabin number is not likely to be useful for analysis and can be excluded.

Given that the columns we've identified as irrelevant lack direct relevance to the survival outcome, we have opted to remove them from our analysis.

In [9]:
```
# Dropping irrelevant columns
df_new = df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'])

df_new.head()
```

Out[9]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

Splitting the dataset

In [10]:
```
# Split the dataset into features (x) and target variable (y)
x = df_new.drop('Survived', axis=1)  # Features
y = df_new['Survived']  # Target variable

# Split the dataset into training and testing sets (70% train, 30% test)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_stat
```

Data cleaning on the training dataset

In [11]:
```
# Step 1

# Check for missing values in the training dataset
missing_values_train = x_train.isnull().sum()
print("Count of missing values in the training dataset:")
print(missing_values_train)
```

```
Count of missing values in the training dataset:
Pclass        0
Sex           0
Age         124
SibSp         0
Parch         0
Fare          0
Embarked      1
dtype: int64
```

In [12]:

```python
#Step 2

# Display unique values for all columns in the train dataset
cols = x_train.columns

# for each column
for col in cols:
    print(col)

    # get a list of unique values
    unique = x_train[col].unique()  # Accessing unique values from x_train
    print(unique, '\n====================================\n\n')
```

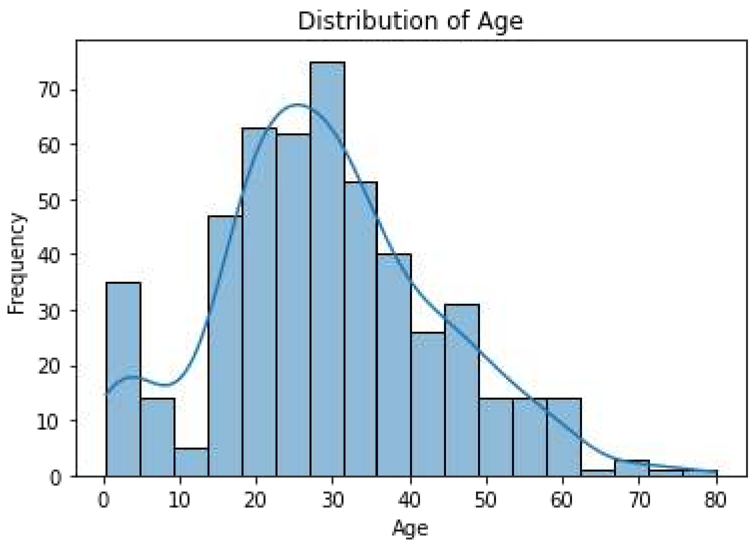
```
Pclass
[1 3 2]
====================================


Sex
['male' 'female']
====================================


Age
[ 4.     nan  1.   36.   43.   38.   31.   29.   18.   39.   26.   20.
 49.   23.    3.   19.   40.5  21.   54.   25.   22.   24.   16.   47.
 60.   27.   44.   45.    8.   32.   50.   15.   28.   41.   33.   52.
  9.   17.   37.   62.   46.   56.   59.   58.   30.   28.5   0.75 35.
 55.   51.    2.   14.   45.5  40.   12.   11.    7.   70.5  34.   70.
 42.   48.   80.   55.5  14.5  10.   53.   32.5  74.   64.    6.    5.
 24.5   0.42 61.    0.67 13.    0.83]
====================================


SibSp
[0 1 2 4 3 8 5]
====================================


Parch
[2 0 1 6 4 3 5]
====================================


Fare
[ 81.8583   7.8958  11.1333  27.75     26.25    153.4625   8.05      8.3
  15.05    110.8833  13.        8.6625   7.05    133.65      0.       15.0458
  39.6875   7.8792  23.45     26.        7.65      7.75     15.7417  15.2458
   7.925   51.8625  15.5      41.5792  14.4542  10.5167  20.525   89.1042
  36.75    10.5      55.4417  24.15     14.5     26.55     50.       21.
  13.8625  16.7      13.5      21.075    35.      55.9       7.8       7.8542
  34.375    7.225     7.2292  18.       47.1      80.       19.5      20.25
  31.3875   8.1125    7.8292  59.4      79.2      56.4958  57.9792  25.4667
  46.9      52.5542  29.125     9.825   14.4583  61.175    15.1      66.6
  83.1583  37.0042    7.25    16.1      27.9     211.3375 106.425     7.7958
  40.125    28.7125  19.2583  49.5042  65.       52.       86.5      16.
  53.1       6.8583  19.9667  13.7917   7.7333 113.275    69.55     30.0708
   8.0292  55.       39.6      24.       20.575    17.4      22.025    29.7
   7.0542   6.95     25.5875 263.       11.5      11.2417  18.7875  13.4167
  73.5     164.8667  79.65     71.       69.3     108.9      14.4       6.4958
  12.2875 146.5208    7.775   13.8583  10.1708   77.2875    7.7417  17.8
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js  18.75   227.525   90.   30.
  42.4       7.3125  23.        7.1417  12.35      9.5     512.3292  23.25
```

```
  7.55     33.       21.6792  25.9292  34.0208   9.35     27.7208   9.5875
 78.85     39.        8.6542  15.      31.275   51.4792 151.55    12.525
 15.9     135.6333    6.75     8.4042   8.85     6.975    7.8875   8.1375
 78.2667   30.5       5.      14.      77.9583   8.5167   7.125   76.7292
 33.5       7.4958   57.      27.       8.6833   7.7375   7.6292  83.475
  4.0125    9.2167   35.5     22.3583  50.4958  93.5       9.475  120.
134.5      10.4625   15.75    12.      71.2833   9.      82.1708  22.525
 75.25      6.45     91.0792   9.8375   8.3625  31.      14.1083]
====================================


Embarked
['S' 'C' 'Q' nan]
====================================
```

Treatment of the 'Age' Column

In [13]:

```
# Step 3: Treatment of Age column

# Step 3.a

# Plotting the distribution of Age
sns.histplot(x_train['Age'].dropna(), kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



Visual inspection of the age distribution revealed a non-normal pattern. This observation influenced subsequent decisions on imputation methods.

In [14]:

```
# Step 3.b

# Calculate the median age
median_age = x_train['Age'].median()

# Replace missing values with the median age
x_train['Age'].fillna(median_age, inplace=True)
```

Missing age values were imputed using the median age, given the non-normal distribution and

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [15]:

```
#Step 3.c

# Count missing values in the 'Age' column
missing_values_age = x_train['Age'].isnull().sum()
print("Number of missing values in the 'Age' column:", missing_values_age)
```

```
Number of missing values in the 'Age' column: 0
```

In [16]:

```
# Step 3.d

# Round values in the 'Age' column to the nearest integer
x_train['Age'] = x_train['Age'].round()
```

In [17]:

```
# Step 3.e

# Display unique values for the 'Age' column
unique_age_values = x_train['Age'].unique()
print(unique_age_values)
```

```
[ 4. 28.  1. 36. 43. 38. 31. 29. 18. 39. 26. 20. 49. 23.  3. 19. 40. 21.
 54. 25. 22. 24. 16. 47. 60. 27. 44. 45.  8. 32. 50. 15. 41. 33. 52.  9.
 17. 37. 62. 46. 56. 59. 58. 30. 35. 55. 51.  2. 14. 12. 11.  7. 70. 34.
 42. 48. 80. 10. 53. 74. 64.  6.  5.  0. 61. 13.]
```

In [18]:

```
# Step 3.f

# Convert 'Age' column to integers
x_train['Age'] = x_train['Age'].astype(int)
```

Treatment of the 'Embarked' Column

In [19]:

```
# Step 4 Treatment of the 'Embarked' Column

# Step 4.a

# Count missing values in the 'Embarked' column
missing_values_embarked_train = x_train['Embarked'].isnull().sum()
print("Number of missing values in the 'Embarked' column:", missing_values_embarked_
```

```
Number of missing values in the 'Embarked' column: 1
```

Handling missing data in a categorical column like 'Embarked' can be done by imputing the missing values with the mode (most frequent value) of the column. The mode is a suitable choice for imputing categorical data because it represents the most common category.

In [20]:

```
# Step 4.b

# Calculate the mode value of the 'Embarked' column
mode_embarked = x_train['Embarked'].mode()[0]

# Fill missing values with the mode value
x_train['Embarked'].fillna(mode_embarked, inplace=True)
```

In [21]:

```
# Step 5: Verifying the details of train dataset after cleaning
```



```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 623 entries, 445 to 102
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Pclass    623 non-null    int64  
 1   Sex       623 non-null    object 
 2   Age       623 non-null    int32  
 3   SibSp     623 non-null    int64  
 4   Parch     623 non-null    int64  
 5   Fare      623 non-null    float64
 6   Embarked  623 non-null    object 
dtypes: float64(1), int32(1), int64(3), object(2)
memory usage: 36.5+ KB
```

In [24]:

```
# Display unique values for all columns in the x_train
for col in x_train.columns:
    print(col)

    # Get a list of unique values
    unique = x_train[col].unique()

    # If the number of unique values is less than 30, print the values. Otherwise, p
    if len(unique) < 30:
        print(unique, '\n====================================\n\n')
    else:
        print(str(len(unique)) + ' unique values', '\n==============================
```

```
Pclass
[1 3 2]
====================================


Sex
['male' 'female']
====================================


Age
66 unique values
====================================


SibSp
[0 1 2 4 3 8 5]
====================================


Parch
[2 0 1 6 4 3 5]
====================================


Fare
207 unique values
====================================


Embarked
['S' 'C' 'Q']
====================================
```

In [26]:

```python
# Display unique values for all columns in the train dataset
cols = x_train.columns

# for each column
for col in cols:
    print(col)

    # get a list of unique values
    unique = x_train[col].unique()  # Accessing unique values from x_train
    print(unique, '\n====================================\n\n')
```

```
Pclass
[1 3 2]
====================================


Sex
[1 0]
====================================


Age
[ 4 28  1 36 43 38 31 29 18 39 26 20 49 23  3 19 40 21 54 25 22 24 16 47
 60 27 44 45  8 32 50 15 41 33 52  9 17 37 62 46 56 59 58 30 35 55 51  2
 14 12 11  7 70 34 42 48 80 10 53 74 64  6  5  0 61 13]
====================================
```

```
SibSp
[0 1 2 4 3 8 5]
====================================


Parch
[2 0 1 6 4 3 5]
====================================


Fare
[ 81.8583   7.8958  11.1333  27.75     26.25    153.4625   8.05      8.3
  15.05    110.8833  13.        8.6625   7.05    133.65      0.       15.0458
  39.6875   7.8792  23.45     26.        7.65      7.75     15.7417  15.2458
   7.925   51.8625  15.5      41.5792  14.4542  10.5167  20.525   89.1042
  36.75     10.5     55.4417  24.15     14.5     26.55     50.      21.
  13.8625  16.7     13.5     21.075    35.      55.9       7.8      7.8542
  34.375    7.225    7.2292  18.      47.1     80.      19.5     20.25
  31.3875   8.1125   7.8292  59.4     79.2     56.4958  57.9792  25.4667
  46.9     52.5542  29.125    9.825   14.4583  61.175   15.1     66.6
  83.1583  37.0042   7.25    16.1     27.9    211.3375 106.425    7.7958
  40.125   28.7125  19.2583  49.5042  65.      52.      86.5     16.
  53.1      6.8583  19.9667  13.7917   7.7333 113.275   69.55    30.0708
   8.0292  55.      39.6     24.      20.575   17.4     22.025   29.7
   7.0542   6.95    25.5875 263.      11.5     11.2417  18.7875  13.4167
  73.5    164.8667  79.65    71.      69.3    108.9     14.4      6.4958
  12.2875 146.5208   7.775   13.8583  10.1708  77.2875   7.7417  17.8
  12.475   15.85    12.65    26.2875  18.75   227.525   90.      30.
  42.4      7.3125  23.       7.1417  12.35     9.5    512.3292  23.25
   7.55    33.      21.6792  25.9292  34.0208   9.35    27.7208   9.5875
  78.85    39.       8.6542  15.      31.275   51.4792 151.55    12.525
  15.9    135.6333   6.75     8.4042   8.85     6.975    7.8875   8.1375
  78.2667  30.5      5.      14.      77.9583   8.5167   7.125   76.7292
  33.5      7.4958  57.      27.       8.6833   7.7375   7.6292  83.475
   4.0125   9.2167  35.5     22.3583  50.4958  93.5      9.475  120.
 134.5     10.4625  15.75    12.      71.2833   9.      82.1708  22.525
  75.25     6.45    91.0792   9.8375   8.3625  31.      14.1083]
====================================


Embarked
[2 0 1]
====================================
```

In [22]:

```python
# Step 6: Verifying the null of train dataset after cleaning

# Step 6.a Checking for null values

# Check for null values in the train dataset
null_values_train = x_train.isnull().sum()
print(null_values_train)
```

```
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [27]:
```python
# Step 6.b Checking for nan values

# Count NaN values in x_train
nan_count = x_train.isna().sum()

# Print the count of NaN values
print("Count of NaN values in x_train:")
print(nan_count)
```

```
Count of NaN values in x_train:
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked    0
dtype: int64
```

After cleaning the train dataset, there are no missing values or null values remaining in any of the columns. The dataset is now ready for further analysis and model training.

Data cleaning on the testing dataset

In [34]:
```python
# Step 1: Check for missing values in the test dataset

missing_values_test = x_test.isnull().sum()
print("Count of missing values in the test dataset:")
print(missing_values_test)
```

```
Count of missing values in the test dataset:
Pclass       0
Sex          0
Age         53
SibSp        0
Parch        0
Fare         0
Embarked     1
dtype: int64
```

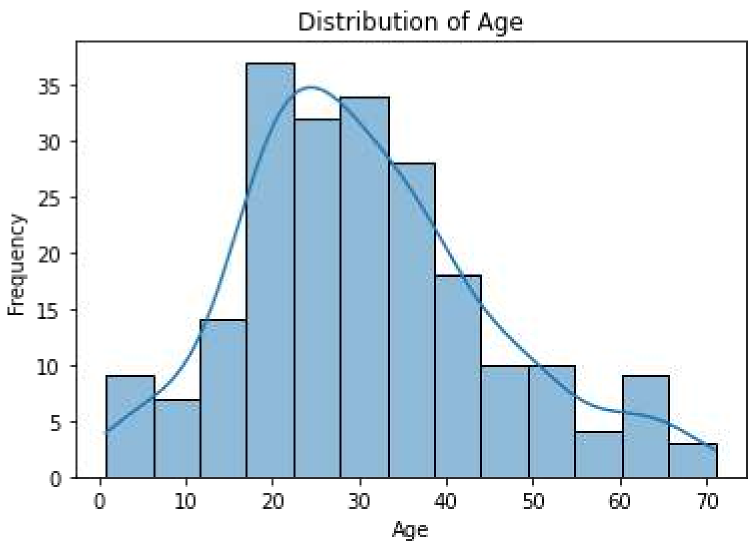
In [35]:
```python
# Step 2: Display unique values for all columns in the test dataset
cols = x_test.columns

# for each column
for col in cols:
    print(col)

    # get a list of unique values
    unique = x_test[col].unique()  # Accessing unique values from x_test
    print(unique, '\n====================================\n\n')
```

```
Pclass
[3 2 1]
====================================


Sex
[1 0]
====================================
```

```
Age
[  nan 31.   20.    6.   14.   26.   16.   19.   37.   44.   30.   36.
 42.   27.   47.   24.   34.   10.   40.    4.   22.   18.   28.   21.
 29.   45.   23.   58.    5.   52.   11.   65.   32.   50.   35.   13.
 57.   17.   39.   30.5  38.   41.   56.   71.    9.   61.   48.   64.
 25.   20.5  63.    0.83 49.   15.   66.   43.   23.5  45.5  33.    7.
  2.   51.    0.92 62.   34.5  36.5 ]
====================================


SibSp
[1 0 2 3 4]
====================================


Parch
[1 0 2 3 4 5]
====================================


Fare
[ 15.2458  10.5      7.925   33.      11.2417  78.85     7.75    18.
  26.2833  53.1      8.05    25.4667   7.225   13.      39.4     52.5542
   7.8292  52.       7.8958  26.55     7.8542   9.225   14.5     27.9
  27.7208  30.6958   7.55    14.4542  35.5     31.      73.5      7.0458
  34.375    8.1583   7.05   113.275   19.2583  93.5    120.      79.65
 247.5208  21.       7.8792   7.775   19.5     26.      25.925    7.875
   7.2292  78.2667 262.375   30.      83.1583  15.55    49.5     20.2125
  39.6875 134.5      7.7375  49.5042  56.9292   0.       7.25    20.525
  31.275   32.3208   7.7292   7.125   91.0792  39.      38.5      9.5
  89.1042  26.2875 221.7792   9.8458  34.6542  12.875   29.      11.5
  76.2917  77.9583  25.9292  79.2     14.4583  12.475  110.8833  17.8
   8.4333  29.125   32.5    146.5208   8.7125   7.4958   7.7958  24.15
  56.4958   7.7875  15.5     82.1708   8.4583  61.9792  51.8625  63.3583
  26.3875  31.3875  16.7     28.5     12.275    7.0542  90.     227.525
  57.       6.2375   8.6625  26.25     9.5875  22.3583   9.4833 211.5
   7.725   21.075   61.3792  30.5      7.5208 151.55    80.       9.8417
  12.35     6.4375 133.65     6.975  106.425 ]
====================================


Embarked
['C' 'S' 'Q' nan]
====================================
```

In [36]:

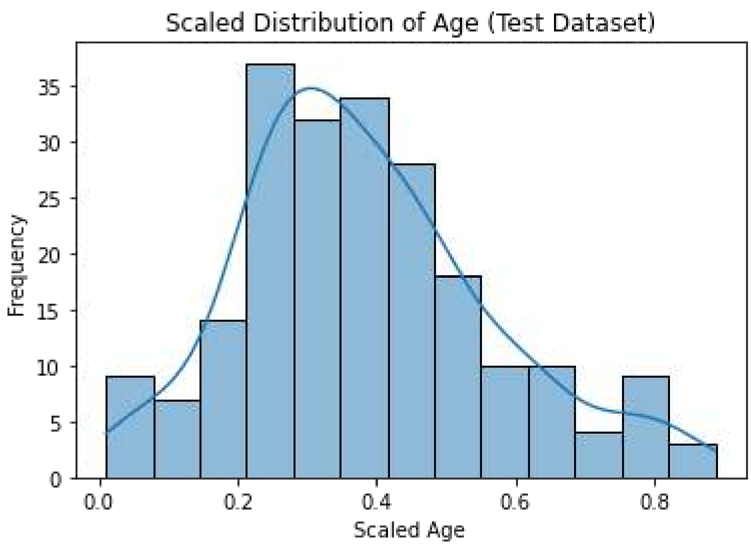
```python
# Step 3: Treatment of Age column


# Step 3.a: Plotting the distribution of Age
sns.histplot(x_test['Age'].dropna(), kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



Given that the 'Age' distribution is rightly skewed, indicating a non-normal pattern, a transformation method is applied to mitigate its effect. we're adjusting the 'Age' values using min-max scaling. This helps make sure all ages are on a similar scale, making it easier for our model to understand and work with them. This will helps the model perform better overall.

In [38]:

```
from sklearn.preprocessing import MinMaxScaler

# Step 3.b: Apply min-max scaling to 'Age' column
x_test['Age_scaled'] = scaler.transform(x_test[['Age']])

#Plot the scaled distribution of Age
sns.histplot(x_test['Age_scaled'], kde=True)
plt.title('Scaled Distribution of Age (Test Dataset)')
plt.xlabel('Scaled Age')
plt.ylabel('Frequency')
plt.show()
```



In [45]:

```
# Step 3.c: Calculate the median age
median_age_test = x_test['Age'].median()

# Replace missing values with the median age
x_test['Age'].fillna(median_age_test, inplace=True)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [47]:
```
# Step 3.d: Count missing values in the 'Age' column
missing_values_age_test = x_test['Age'].isnull().sum()
print("Number of missing values in the 'Age' column:", missing_values_age_test)
```

```
Number of missing values in the 'Age' column: 0
```

In [48]:
```
# Step 3.e: Round values in the 'Age' column to the nearest integer
x_test['Age'] = x_test['Age'].round()
```

In [49]:
```
# Step 3.f: Display unique values for the 'Age' column
unique_age_values_test = x_test['Age'].unique()
print(unique_age_values_test)
```

```
[29. 31. 20.  6. 14. 26. 16. 19. 37. 44. 30. 36. 42. 27. 47. 24. 34. 10.
 40.  4. 22. 18. 28. 21. 45. 23. 58.  5. 52. 11. 65. 32. 50. 35. 13. 57.
 17. 39. 38. 41. 56. 71.  9. 61. 48. 64. 25. 63.  1. 49. 15. 66. 43. 46.
 33.  7.  2. 51. 62.]
```

In [50]:
```
# Step 3.g: Convert 'Age' column to integers
x_test['Age'] = x_test['Age'].astype(int)
```

In [51]:
```
# Step 4: Treatment of the 'Embarked' Column

# Step 4.a: Count missing values in the 'Embarked' column
missing_values_embarked_test = x_test['Embarked'].isnull().sum()
print("Number of missing values in the 'Embarked' column:", missing_values_embarked_
```

```
Number of missing values in the 'Embarked' column: 1
```

In [52]:
```
# Step 4.b: Calculate the mode value of the 'Embarked' column
mode_embarked_test = x_test['Embarked'].mode()[0]

# Fill missing values with the mode value
x_test['Embarked'].fillna(mode_embarked_test, inplace=True)
```

In [53]:
```
# Step 5: Verifying the details of test dataset after cleaning
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 268 entries, 709 to 430
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Pclass      268 non-null    int64  
 1   Sex         268 non-null    int32  
 2   Age         268 non-null    int32  
 3   SibSp       268 non-null    int64  
 4   Parch       268 non-null    int64  
 5   Fare        268 non-null    float64
 6   Embarked    268 non-null    object 
 7   Age_scaled  215 non-null    float64
 8   Age_log     215 non-null    float64
 9   Age_sqrt    215 non-null    float64
 10  Age_cbrt    215 non-null    float64
 11  Age_boxcox  215 non-null    float64
dtypes: float64(6), int32(2), int64(3), object(1)
memory usage: 25.1+ KB
```

The embarked feature contains categorical data, which needs to be converted into a numerical format for machine learning models like KNN. One-hot encoding does this by transforming each category into binary columns, enabling the algorithm to understand and use the information without assuming any hierarchy or order among the categories.

In [54]:
```
# Step 6

# Perform one-hot encoding on the 'Embarked' column
x_test = pd.get_dummies(x_test, columns=['Embarked'], drop_first=True)
```

In [55]:
```
# Step 7: Verifying the details of test dataset after cleaning
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 268 entries, 709 to 430
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Pclass      268 non-null    int64  
 1   Sex         268 non-null    int32  
 2   Age         268 non-null    int32  
 3   SibSp       268 non-null    int64  
 4   Parch       268 non-null    int64  
 5   Fare        268 non-null    float64
 6   Age_scaled  215 non-null    float64
 7   Age_log     215 non-null    float64
 8   Age_sqrt    215 non-null    float64
 9   Age_cbrt    215 non-null    float64
 10  Age_boxcox  215 non-null    float64
 11  Embarked_Q  268 non-null    uint8  
 12  Embarked_S  268 non-null    uint8  
dtypes: float64(6), int32(2), int64(3), uint8(2)
memory usage: 23.6 KB
```

In [58]:
```
# Drop the specified columns from the DataFrame if they exist
columns_to_drop = ['Age_scaled', 'Age_log', 'Age_sqrt', 'Age_cbrt', 'Age_boxcox']
x_test = x_test.drop(columns_to_drop, axis=1, errors='ignore')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 268 entries, 709 to 430
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Pclass      268 non-null    int64  
 1   Sex         268 non-null    int32  
 2   Age         268 non-null    int32  
 3   SibSp       268 non-null    int64  
 4   Parch       268 non-null    int64  
 5   Fare        268 non-null    float64
 6   Embarked_Q  268 non-null    uint8  
 7   Embarked_S  268 non-null    uint8  
dtypes: float64(1), int32(2), int64(3), uint8(2)
memory usage: 13.1 KB
```

Creating feature

Creating features, also known as feature engineering. Feature engineering enhances machine learning models by creating new informative features from existing data, improving performance and interpretability.

For Train Dataset

In [29]:

```
# Creating a new feature 'FamilySize' based on 'SibSp' (number of siblings/spouses a
x_train['FamilySize'] = x_train['SibSp'] + x_train['Parch']
```

The "FamilySize" feature combines the counts of siblings, spouses, parents, and children aboard the Titanic for each passenger. It provides insight into the passenger's family composition and potential impact on survival, reflecting their social connections and support network onboard.

In [31]:

```
# Creating a new feature 'IsAlone'
x_train['IsAlone'] = (x_train['FamilySize'] == 0).astype(int)
```

The "IsAlone" feature indicates whether a passenger is traveling alone or with family. It assigns a binary value of 1 if the passenger is alone and 0 otherwise. This feature helps in understanding whether being accompanied by family members affects the passenger's chances of survival.

In [32]:

```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 623 entries, 445 to 102
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Pclass      623 non-null    int64  
 1   Sex         623 non-null    int32  
 2   Age         623 non-null    int32  
 3   SibSp       623 non-null    int64  
 4   Parch       623 non-null    int64  
 5   Fare        623 non-null    float64
 6   Embarked    623 non-null    int32  
 7   FamilySize  623 non-null    int64  
 8   IsAlone     623 non-null    int32  
dtypes: float64(1), int32(4), int64(4)
memory usage: 38.9 KB
```


For Test Dataset

In [60]:

```
# Creating a new feature 'FamilySize' based on 'SibSp' (number of siblings/spouses a
x_test['FamilySize'] = x_test['SibSp'] + x_test['Parch']

# Creating a new feature 'IsAlone'
x_test['IsAlone'] = (x_test['FamilySize'] == 0).astype(int)


x_test.info()
```

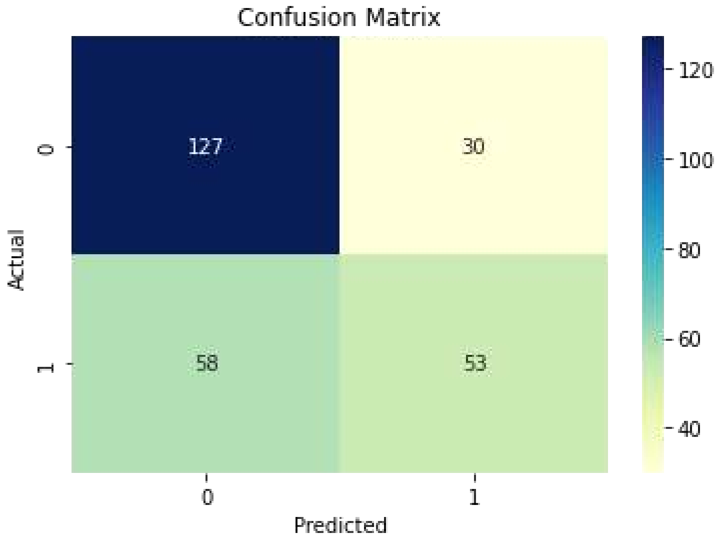
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 268 entries, 709 to 430
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Pclass      268 non-null    int64  
 1   Sex         268 non-null    int32  
 2   Age         268 non-null    int32  
 3   SibSp       268 non-null    int64  
 4   Parch       268 non-null    int64  
 5   Fare        268 non-null    float64
 6   Embarked_Q  268 non-null    uint8  
 7   Embarked_S  268 non-null    uint8  
 8   FamilySize  268 non-null    int64  
 9   IsAlone     268 non-null    int32  
dtypes: float64(1), int32(3), int64(4), uint8(2)
memory usage: 16.2 KB
```

Build a KNN model to predict whether a passenger survives or not.

In [61]:

```
# Step 1: Train the Model
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)

# Step 2: Evaluate the Model
y_pred = knn.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Step 3: Visualize Model Performance
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, cmap="YlGnBu", fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.6716417910447762
```



We have built and evaluated a KNN (K-Nearest Neighbors) model for predicting passenger survival in a dataset. It trains the model on training data, evaluates its accuracy on test data, and visualizes its performance using a confusion matrix.

The model achieves an accuracy of approximately 67%, indicating its ability to predict survival outcomes, although further optimization may enhance its performance.

See if the model can be improved using grid search.

In [62]:

```
# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # Number of neighbors
    'weights': ['uniform', 'distance'],  # Weighting strategy
    'metric': ['euclidean', 'manhattan']  # Distance metric
}

# Initialize KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='accu

# Fit the grid search to the training data
grid_search.fit(x_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Get the best model
best_knn_model = grid_search.best_estimator_

# Evaluate the best model
y_pred_best = best_knn_model.predict(x_test)
accuracy_best = accuracy_score(y_test, y_pred_best)
print("Accuracy with Best Model:", accuracy_best)
```

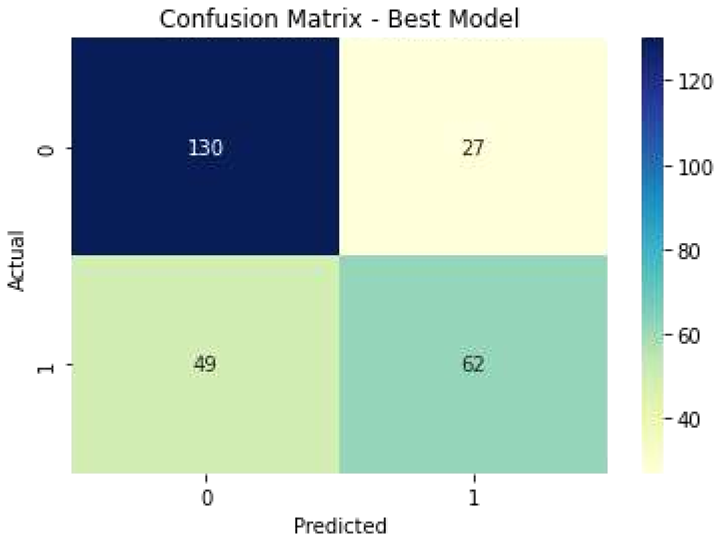
```
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'}
Accuracy with Best Model: 0.7164179104477612
```

Based on grid search:

- The best combination of hyperparameters is {'metric': 'manhattan', 'n_neighbors': 5, 'weights': 'uniform'}, which resulted in an accuracy of approximately 71.64%.
- The mean test scores show the performance of the model across various combinations of hyperparameters, providing insight into how different configurations affect the model's accuracy.

In [67]:
```
# To visualize the performance of the best model, we can plot a confusion matrix

# Train the model with the best parameters
best_knn = KNeighborsClassifier(metric='manhattan', n_neighbors=5, weights='uniform'
best_knn.fit(x_train, y_train)

# Predict on the test set
y_pred_best = best_knn.predict(x_test)

# Evaluate accuracy
accuracy_best = accuracy_score(y_test, y_pred_best)
print("Accuracy with Best Model:", accuracy_best)

# Plot confusion matrix
conf_matrix_best = confusion_matrix(y_test, y_pred_best)
sns.heatmap(conf_matrix_best, annot=True, cmap="YlGnBu", fmt="d")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Best Model")
plt.show()
```

```
Accuracy with Best Model: 0.7164179104477612
```



Summary of the key steps and findings:

A. Data Cleaning and Preprocessing:

- ***Treatment of Missing Values***: Missing values in the 'Embarked' column are imputed with the mode value, 'S'. Missing values in the 'Age' column are handled by imputing the median age and rounding the values to the nearest integer.

- ***Feature Engineering***: A new feature 'FamilySize' is created by summing 'SibSp' and 'Parch' columns, indicating the total number of family members onboard. Another feature 'IsAlone' is created to identify whether a passenger is traveling alone or with family.

- ***One-Hot Encoding***: The categorical feature 'Embarked' is one-hot encoded to convert it into numerical format for model training.

B. Model Training and Evaluation:

- ***KNN Model Training***: A KNN classifier is trained on the training data.
- ***Model Evaluation***: The accuracy of the model is evaluated on the test data, achieving approximately 67%.

C. Grid Search for Hyperparameter Tuning:

- ***Grid Search***: Grid search is performed to find the best combination of hyperparameters for the KNN classifier. Hyperparameters include the number of neighbors, weighting strategy, and distance metric.

- ***Best Model Evaluation***: The best model obtained from grid search is evaluated on the test data, achieving an accuracy of approximately 71.64%.

- ***Confusion Matrix Visualization***: The performance of the best model is visualized using a confusion matrix.

Overall, the analysis demonstrates the process of building and optimizing a KNN classifier for predicting Titanic passenger survival, highlighting the importance of data preprocessing and hyperparameter tuning in improving model performance.

** END ***