Titanic

Data Description: The dataset contains information about Passengers Titanic. It includes details such as PassengerId, Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin and Embarked.

PassengerId: A unique identifier assigned to each passenger in the dataset.

Survived: This column indicates whether the passenger survived the Titanic disaster or not.

0: Passenger did not survive. 1: Passenger survived.

Pclass: This column represents the ticket class of the passenger.

1: First class 2: Second class 3: Third class

Name: The name of the passenger, including their title (Mr., Mrs., Miss., etc.).

Sex: The gender of the passenger.(Male and Female)

Age: The age of the passenger. It may contain missing values (NaN).

SibSp: This column indicates the number of siblings or spouses (SibSp) aboard the Titanic for each passenger.

Parch: This column indicates the number of parents or children (Parch) aboard the Titanic for each passenger.

Ticket: The ticket number of the passenger.

Fare: The fare paid by the passenger for the ticket.

Cabin: The cabin number of the passenger. It may contain missing values (NaN).

Embarked: The port of embarkation for the passenger.

C: Cherbourg Q: Queenstown S: Southampton

This dataset provides various details about each passenger, including their demographics, family relations aboard the ship, ticket information, and survival status

In [3]:
```
#Loading necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:
```
#Loading the dataset
df = pd.read_csv('Titanic-Dataset.csv')
```

In [7]:
```
df
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N
...	...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	N
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	E
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	N
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C1
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	N

891 rows × 12 columns

In [4]:

```
#Display the top 5 rows
df.head()
```

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

In [5]:

```
#Identifying the shape for the dataset
df.shape
```

Out[5]:

```
(891, 12)
```

In [6]:

```
#Finding the datatypes
df.dtypes
```

Out[6]:

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

In [7]:

```
#Overview of the Datastructure
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  
 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
```

In [8]:

```
#It returns the count of missing values in each column of the DataFrame
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 [9]:

```
#It replaces missing values in the 'Age' column with the median age of the non-null
print(df['Age'].fillna(df['Age'].median()))
```

```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       ... 
886    27.0
887    19.0
888    28.0
889    26.0
890    32.0
Name: Age, Length: 891, dtype: float64
```

In [10]:

```
# It replaces missing values in the 'Embarked' column with the most frequent port c
print(df['Embarked'].fillna(df['Embarked'].mode()[0]))
```

```
0      S
1      C
2      S
3      S
4      S
      ..
886    S
887    S
888    S
889    C
890    Q
Name: Embarked, Length: 891, dtype: object
```

In [11]:

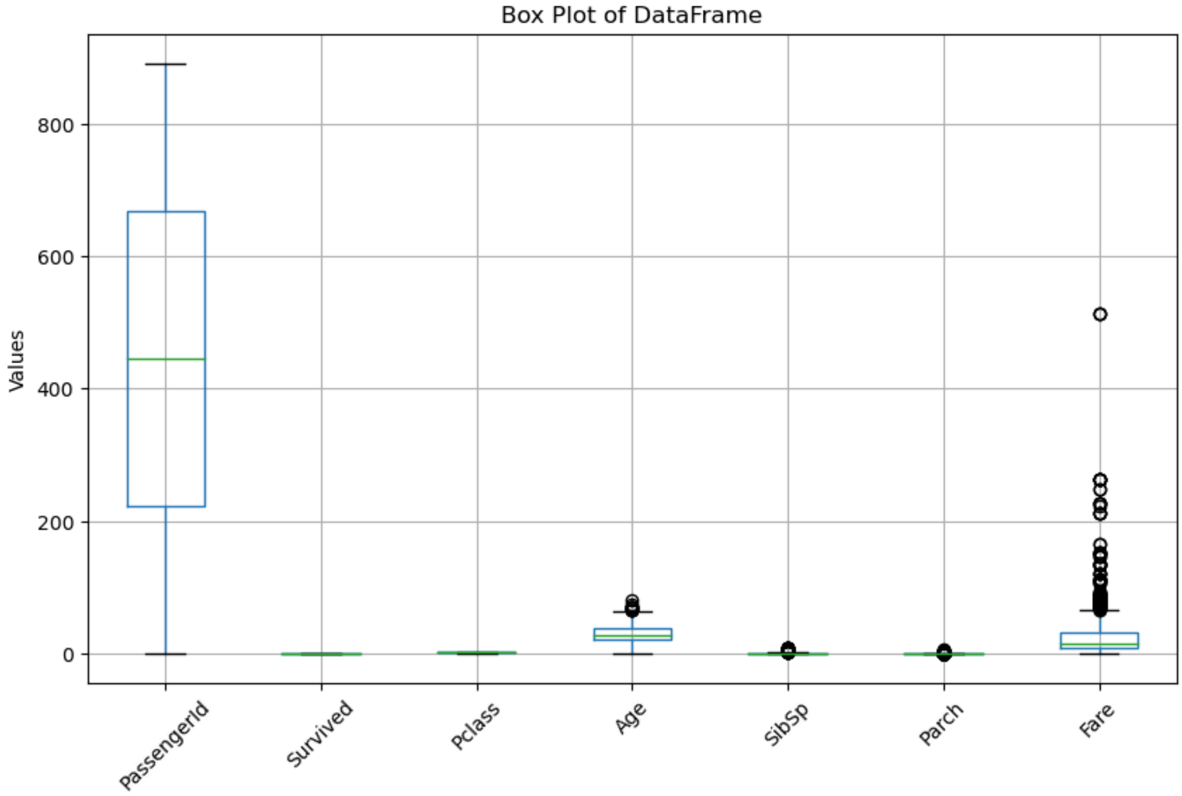
```python
#It provides statistical summary of numerical columns in the DataFrame.
df.describe()
```

Out[11]:

	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

In [6]:

```python
summary = df.describe().drop('count')

# Plotting using Matplotlib
plt.figure(figsize=(10, 6))
df.boxplot()
plt.title('Box Plot of DataFrame')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.show()
```



In [12]:
```
#It identifies duplicate rows in the DataFrame.
df.duplicated()
```

Out[12]:
```
0      False
1      False
2      False
3      False
4      False
       ...  
886    False
887    False
888    False
889    False
890    False
Length: 891, dtype: bool
```

In [13]:
```
#It calculates the count of non-null values in each column of the DataFrame.
df.count()
```

Out[13]:
```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          204
Embarked       889
dtype: int64
```

In [14]:
```
# It returns the count of unique values in each column of the DataFrame.
df.nunique()
```

Out[14]:
```
PassengerId    891
Survived         2
Pclass           3
Name           891
Sex              2
Age             88
SibSp            7
Parch            7
Ticket         681
Fare           248
Cabin          147
Embarked         3
dtype: int64
```

In [15]:
```
#It returns the index values of the DataFrame as an array.
df.index.values
```

Out[15]:

```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
       169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
       182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
       195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
       208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
       221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
       234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
       247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
       260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
       273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
       286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
       299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311,
       312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
       325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,
       338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350,
       351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363,
       364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376,
       377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389,
       390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402,
       403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415,
       416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428,
       429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441,
       442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454,
       455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467,
       468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
       481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493,
       494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506,
       507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
       520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532,
       533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545,
       546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558,
       559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571,
       572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584,
       585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597,
       598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610,
       611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623,
       624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636,
       637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649,
       650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662,
       663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675,
       676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688,
       689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701,
       702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714,
       715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727,
       728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740,
       741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753,
       754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766,
       767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779,
       780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792,
       793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805,
       806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818,
       819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831,
```

```
       832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844,
       845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857,
       858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870,
       871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883,
       884, 885, 886, 887, 888, 889, 890], dtype=int64)
```

In [16]:
```
#It Finds the length of the index value
len(df.index)
```

Out[16]:
```
891
```

In [17]:
```
#It returns the columns of the dataframe
df.columns
```

Out[17]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [13]:
```
#It encodes categorical variables 'Sex' and 'Embarked' into numerical representatic
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['Sex'] = encoder.fit_transform(df['Sex'])
df['Embarked'] = encoder.fit_transform(df['Embarked'])
print(df[['Sex']])
print(df[['Embarked']])
```

```
     Sex
0      1
1      0
2      0
3      0
4      1
..   ...
886    1
887    0
888    0
889    1
890    1

[891 rows x 1 columns]
     Embarked
0           2
1           0
2           2
3           2
4           2
..        ...
886         2
887         2
888         2
889         0
890         1

[891 rows x 1 columns]
```

In [14]:
```
#It selects specified columns from the DataFrame df to create the feature matrix X
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = df[features]
y = df['Survived']
```

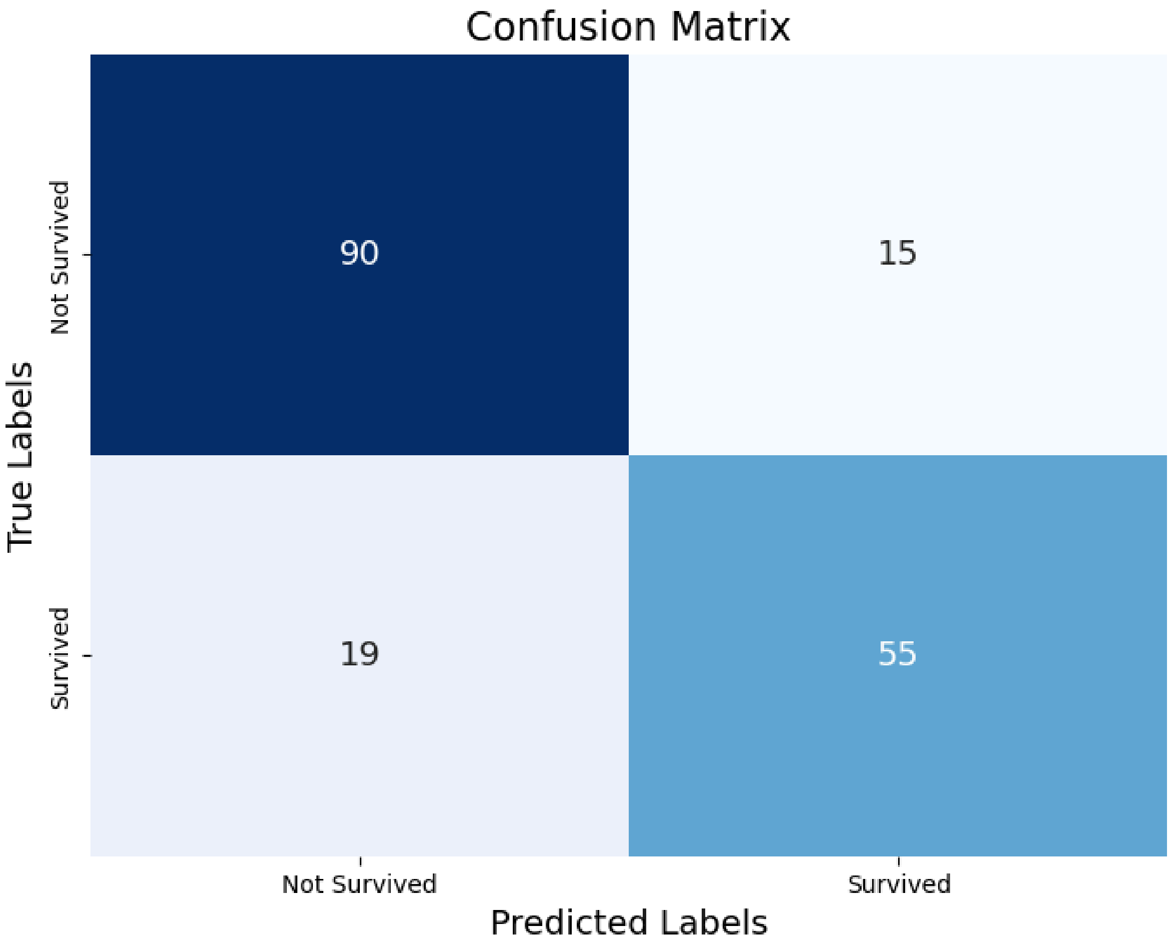
In [15]:
```
#: It splits the dataset into training and testing sets for features and target var
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [16]:
```
#It creates a logistic regression model object with specified parameters for maximu
from sklearn.linear_model import LogisticRegression
logistic_reg = LogisticRegression(max_iter=1000, random_state=42)
```

In [18]:
```
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='median')  # You can choose a different strategy i
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Now, train the logistic regression model using the imputed data
logistic_reg.fit(X_train_imputed, y_train)

# Predict on the test set
y_pred = logistic_reg.predict(X_test_imputed)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.8100558659217877
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.86      0.84       105
           1       0.79      0.74      0.76        74

    accuracy                           0.81       179
   macro avg       0.81      0.80      0.80       179
weighted avg       0.81      0.81      0.81       179

Confusion Matrix:
[[90 15]
 [19 55]]
```

In [14]:
```
#Plotting
import matplotlib.pyplot as plt
import seaborn as sns

# Plot Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={'fontsize': 14},
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.xlabel('Predicted Labels', fontsize=14)
plt.ylabel('True Labels', fontsize=14)
plt.title('Confusion Matrix', fontsize=16)
plt.show()
```



Observations from the output:

Accuracy: The accuracy of the model is approximately 81%, indicating that it correctly predicts the survival status of passengers around 81% of the time.

Precision: For the survival class (1), the precision is 79%, which means that out of all the instances predicted as survived, around 79% of them are correct. For the non-survival class (0), the precision is 83%, indicating that out of all the instances predicted as not survived, around 83% of them are correct.

Recall: For the survival class (1), the recall is 74%, which means that out of all the passengers who actually survived, the model correctly identifies around 74% of them. For the non-survival class (0), the recall is 86%, indicating that out of all the passengers who did not survive, the model correctly identifies around 86% of them.

F1-score: The F1-score, which is the harmonic mean of precision and recall, is approximately 0.76 for the survival class (1) and 0.84 for the non-survival class (0). It provides a balance between precision and recall.

Support: This represents the number of actual occurrences of each class in the test data. For class 0 (non-survival), the support is 105, and for class 1 (survival), the support is 74.

Confusion Matrix:

True Positive (TP): 55 (correctly predicted survived) True Negative (TN): 90 (correctly predicted not survived) False Positive (FP): 15 (incorrectly predicted survived) False Negative (FN): 19 (incorrectly predicted not survived)

**So, the total number of passengers who survived according to the model is 55. the total number of passengers who did not survive according to the model is 90.