Titanicprediction

September 25, 2024

In this article, we will learn to predict the survival chances of the Titanic passengers using the given information about their sex, age, etc. As this is a classification task we will be using random forest.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier,␣
↪GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from xgboost import XGBRegressor, XGBClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, roc_auc_score,␣
↪r2_score,mean_absolute_error
from sklearn.compose import ColumnTransformer
from sklearn.svm import SVR, SVC
from tabulate import tabulate

import joblib
```

[2]:
```
# We have those 2 columns that are about the train and the test, and we want to␣
↪predict ig=f the passenger in the test column will survive or no
```

[3]:
```
df=pd.read_csv('train.csv')
df

# SibSp is a data that define the familial relation
```

[3]:
```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           …       …     …
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                             Allen, Mr. William Henry    male  35.0      0
..                                                 …     …   …    …
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                              Behr, Mr. Karl Howell    male  26.0      0
890                                Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
2        0  STON/O2. 3101282   7.9250   NaN        S
3        0            113803  53.1000  C123        S
4        0            373450   8.0500   NaN        S
..     …             …     …   …      …
886      0            211536  13.0000   NaN        S
887      0            112053  30.0000   B42        S
888      2        W./C. 6607  23.4500   NaN        S
889      0            111369  30.0000  C148        C
890      0            370376   7.7500   NaN        Q

[891 rows x 12 columns]
```

[4]:
```python
df.describe(include='all')
```

[4]:
```
        PassengerId    Survived      Pclass                     Name   Sex  \
count    891.000000  891.000000  891.000000                      891   891
unique          NaN         NaN         NaN                      891     2
top             NaN         NaN         NaN  Braund, Mr. Owen Harris  male
```

```
freq            NaN         NaN         NaN                            1    577
mean     446.000000    0.383838    2.308642                          NaN    NaN
std      257.353842    0.486592    0.836071                          NaN    NaN
min        1.000000    0.000000    1.000000                          NaN    NaN
25%      223.500000    0.000000    2.000000                          NaN    NaN
50%      446.000000    0.000000    3.000000                          NaN    NaN
75%      668.500000    1.000000    3.000000                          NaN    NaN
max      891.000000    1.000000    3.000000                          NaN    NaN

              Age       SibSp       Parch  Ticket        Fare    Cabin  \
count  714.000000  891.000000  891.000000     891  891.000000      204
unique        NaN         NaN         NaN     681         NaN      147
top           NaN         NaN         NaN  347082         NaN  B96 B98
freq          NaN         NaN         NaN       7         NaN        4
mean    29.699118    0.523008    0.381594     NaN   32.204208      NaN
std     14.526497    1.102743    0.806057     NaN   49.693429      NaN
min      0.420000    0.000000    0.000000     NaN    0.000000      NaN
25%     20.125000    0.000000    0.000000     NaN    7.910400      NaN
50%     28.000000    0.000000    0.000000     NaN   14.454200      NaN
75%     38.000000    1.000000    0.000000     NaN   31.000000      NaN
max     80.000000    8.000000    6.000000     NaN  512.329200      NaN

       Embarked
count       889
unique        3
top           S
freq        644
mean        NaN
std         NaN
min         NaN
25%         NaN
50%         NaN
75%         NaN
max         NaN
```

[5]:

```python
# Checking for the data
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
```

[6]:
```
# Checking if there is a duplicate

df.duplicated().sum()
```

[6]:
```
0
```

[7]:
```
df.shape
```

[7]:
```
(891, 12)
```

[8]:
```
# checking of the null value
df.isnull().sum()
```

[8]:
```
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
```

[9]:
```
# Dealing with the missing value such as age and cabin
# For numerical columns: Impute missing values using the mean of the column.
# For categorical columns: Impute missing values using the most frequent␣
↪category (mode).







# Assuming `train` is your DataFrame
```

```python
# 1. Impute missing values for 'Age' using the median
age_imputer = SimpleImputer(strategy='median')
df['Age'] = age_imputer.fit_transform(df[['Age']])

# 2. Create 'CabinBool' column for 'Cabin' missing values
df['CabinBool'] = df['Cabin'].notnull().astype(int)

# Fill missing 'Cabin' values with 'Unknown'
df['Cabin'].fillna('Unknown', inplace=True)

# 3. Impute missing values for 'Embarked' using the most frequent value (mode)
embarked_imputer = SimpleImputer(strategy='most_frequent')
df['Embarked'] = embarked_imputer.fit_transform(df[['Embarked']]).flatten()
```

```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\1984183918.py:19:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Cabin'].fillna('Unknown', inplace=True)
```

[10]:
```python
# checking for the missing value

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

[10]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
CabinBool      0
dtype: int64
```

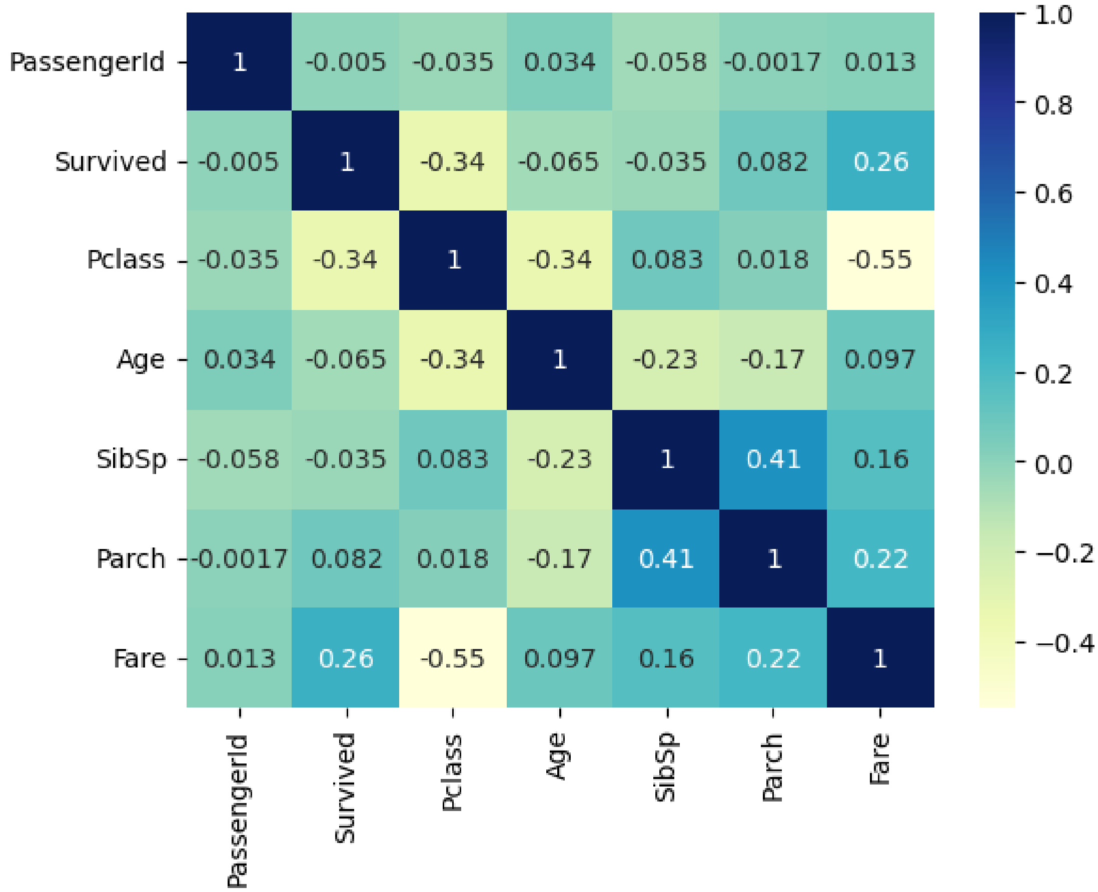
[11]:
```python
# Delete the column 'Cabin' from test and train dataset
df = df.drop(['Cabin'], axis=1)
```

[12]:
```python
df.describe(include='all')
```

[12]:
```
        PassengerId    Survived      Pclass                     Name   Sex  \
count    891.000000  891.000000  891.000000                      891   891
unique          NaN         NaN         NaN                      891     2
top             NaN         NaN         NaN  Braund, Mr. Owen Harris  male
freq            NaN         NaN         NaN                        1   577
mean     446.000000    0.383838    2.308642                      NaN   NaN
std      257.353842    0.486592    0.836071                      NaN   NaN
min        1.000000    0.000000    1.000000                      NaN   NaN
25%      223.500000    0.000000    2.000000                      NaN   NaN
50%      446.000000    0.000000    3.000000                      NaN   NaN
75%      668.500000    1.000000    3.000000                      NaN   NaN
max      891.000000    1.000000    3.000000                      NaN   NaN

               Age       SibSp       Parch  Ticket        Fare Embarked  \
count   891.000000  891.000000  891.000000     891  891.000000      891
unique         NaN         NaN         NaN     681         NaN        3
top            NaN         NaN         NaN  347082         NaN        S
freq           NaN         NaN         NaN       7         NaN      646
mean     29.361582    0.523008    0.381594     NaN   32.204208      NaN
std      13.019697    1.102743    0.806057     NaN   49.693429      NaN
min       0.420000    0.000000    0.000000     NaN    0.000000      NaN
25%      22.000000    0.000000    0.000000     NaN    7.910400      NaN
50%      28.000000    0.000000    0.000000     NaN   14.454200      NaN
75%      35.000000    1.000000    0.000000     NaN   31.000000      NaN
max      80.000000    8.000000    6.000000     NaN  512.329200      NaN

         CabinBool
count   891.000000
unique         NaN
top            NaN
freq           NaN
mean      0.228956
std       0.420397
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       1.000000
```

[13]:
```python
# Select only the numerical columns from the dataset
numeric_data = df.select_dtypes(include=['float64', 'int64'])
```

```
# View the correlation matrix as a heatmap
sns.heatmap(numeric_data.corr(), cmap='YlGnBu', annot=True)
plt.show()
```



The lower number in pclass , the higher the survive is

[14]:
```
# 1. Extract Titles from the 'Name' column
df['Title'] = df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

```
<>:2: SyntaxWarning: invalid escape sequence '\.'
<>:2: SyntaxWarning: invalid escape sequence '\.'
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\4061229650.py:2:
SyntaxWarning: invalid escape sequence '\.'
  df['Title'] = df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

[15]:
```
# 2. Drop unnecessary columns
df = df.drop(['PassengerId', 'Ticket'], axis=1)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Survived   891 non-null    int64
 1   Pclass     891 non-null    int64
 2   Name       891 non-null    object
 3   Sex        891 non-null    object
 4   Age        891 non-null    float64
 5   SibSp      891 non-null    int64
 6   Parch      891 non-null    int64
 7   Fare       891 non-null    float64
 8   Embarked   891 non-null    object
 9   CabinBool  891 non-null    int32
 10  Title      891 non-null    object
dtypes: float64(2), int32(1), int64(4), object(4)
memory usage: 73.2+ KB
```

[16]:

```python
# Define a function to extract the title from a name string
def get_title(name):
    # Check if the name contains a period (.)
    if "." in name:
        # Split the name string by a comma, take the second part, and then␣
 ↪split by a period
        # The first part after splitting by the period is the title (e.g., Mr,␣
 ↪Mrs, Miss)
        return name.split(",")[1].split(".")[0].strip()
    else:
        # If no period is found, return "Unknown" as the title
        return "Unknown"

# Define a function to map the extracted title to a numerical value
def title_map(title):
    # Map the title "Mr" to 1
    if title in ["Mr"]:
        return 1
    # Map the title "Master" to 3
    elif title in ["Master"]:
        return 3
    # Map the titles "Ms", "Mlle", "Miss" to 4
    elif title in ["Ms", "Mlle", "Miss"]:
        return 4
    # Map the titles "Mme", "Mrs" to 5
    elif title in ["Mme", "Mrs"]:
        return 5
    # If the title doesn't match any of the above, map it to 2 (which could be␣
 ↪"Unknown" or less common titles)
```

```
    else:
        return 2

# Apply the get_title function to the "Name" column to extract titles
# Ensure that 'Name' column exists in the DataFrame before this line
df["Title"] = df["Name"].apply(get_title).apply(title_map)

# Display the updated DataFrame
df.head()
```

[16]:

```
   Survived  Pclass                                               Name  \
0         0       3                            Braund, Mr. Owen Harris
1         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th…
2         1       3                             Heikkinen, Miss. Laina
3         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)
4         0       3                           Allen, Mr. William Henry

      Sex   Age  SibSp  Parch     Fare Embarked  CabinBool  Title
0    male  22.0      1      0   7.2500        S          0      1
1  female  38.0      1      0  71.2833        C          1      5
2  female  26.0      0      0   7.9250        S          0      4
3  female  35.0      1      0  53.1000        S          1      5
4    male  35.0      0      0   8.0500        S          0      1
```

[17]:

```
df = df.drop(['Name'],axis='columns')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Survived   891 non-null    int64
 1   Pclass     891 non-null    int64
 2   Sex        891 non-null    object
 3   Age        891 non-null    float64
 4   SibSp      891 non-null    int64
 5   Parch      891 non-null    int64
 6   Fare       891 non-null    float64
 7   Embarked   891 non-null    object
 8   CabinBool  891 non-null    int32
 9   Title      891 non-null    int64
dtypes: float64(2), int32(1), int64(5), object(2)
memory usage: 66.3+ KB
```

[18]:
```
# 3. Create 'FamilySize' feature
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1  # +1 to include the person␣
 ↪themselves
```

[19]:
```
# View the updated dataset
df.head(10)
```

[19]:
```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked  CabinBool  \
0         0       3    male  22.0      1      0   7.2500        S          0
1         1       1  female  38.0      1      0  71.2833        C          1
2         1       3  female  26.0      0      0   7.9250        S          0
3         1       1  female  35.0      1      0  53.1000        S          1
4         0       3    male  35.0      0      0   8.0500        S          0
5         0       3    male  28.0      0      0   8.4583        Q          0
6         0       1    male  54.0      0      0  51.8625        S          1
7         0       3    male   2.0      3      1  21.0750        S          0
8         1       3  female  27.0      0      2  11.1333        S          0
9         1       2  female  14.0      1      0  30.0708        C          0

   Title  FamilySize
0      1           2
1      5           2
2      4           1
3      5           2
4      1           1
5      1           1
6      1           1
7      3           5
8      5           3
9      5           2
```

- Mr.: Adult male
- Mrs.: Married female
- Miss.: Unmarried female
- Master.: Young male

[20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    object
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
```

```
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    object
 8   CabinBool   891 non-null    int32
 9   Title       891 non-null    int64
 10  FamilySize  891 non-null    int64
dtypes: float64(2), int32(1), int64(6), object(2)
memory usage: 73.2+ KB
```

[21]:
```python
# Convert specified columns to categorical type
df['Sex'] = df['Sex'].astype('category')

df['Embarked'] = df['Embarked'].astype('category')
df['Title'] = df['Title'].astype('category')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    category
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    category
 8   CabinBool   891 non-null    int32
 9   Title       891 non-null    category
 10  FamilySize  891 non-null    int64
dtypes: category(3), float64(2), int32(1), int64(5)
memory usage: 55.4 KB
```

[22]:
```python
# Replace 'male' and 'female' with 0 and 1, then convert to int
df["Sex"] = df["Sex"].replace(["male", "female"], [0, 1]).astype(int)

# Check the data types
df.info()
```

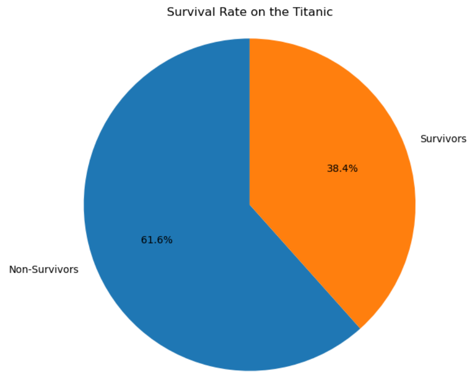
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    int32
```

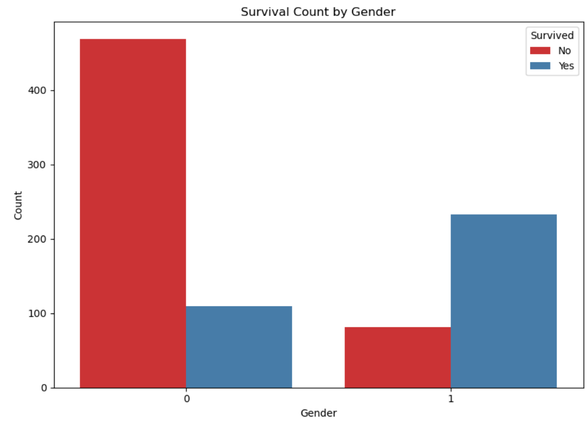
```
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    category
 8   CabinBool   891 non-null    int32
 9   Title       891 non-null    category
 10  FamilySize  891 non-null    int64
dtypes: category(2), float64(2), int32(2), int64(5)
memory usage: 57.9 KB
```

```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\436737572.py:2:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  df["Sex"] = df["Sex"].replace(["male", "female"], [0, 1]).astype(int)
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\436737572.py:2:
FutureWarning: The behavior of Series.replace (and DataFrame.replace) with
CategoricalDtype is deprecated. In a future version, replace will only be used
for cases that preserve the categories. To change the categories, use
ser.cat.rename_categories instead.
  df["Sex"] = df["Sex"].replace(["male", "female"], [0, 1]).astype(int)
```

EDA

[23]:
```
# Create a figure with 1 row and 2 columns
plt.figure(figsize=(16, 6))

# 1. Pie chart for survival rate
plt.subplot(1, 2, 1)
survival_counts = df['Survived'].value_counts()
plt.pie(survival_counts, labels=['Non-Survivors', 'Survivors'], autopct='%1.
 ↪1f%%', startangle=90)
plt.title('Survival Rate on the Titanic')
plt.axis('equal')

# 2. Bar plot for survival count by gender
plt.subplot(1, 2, 2)
sns.countplot(data=df, x='Sex', hue='Survived', palette='Set1')
plt.title('Survival Count by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])

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



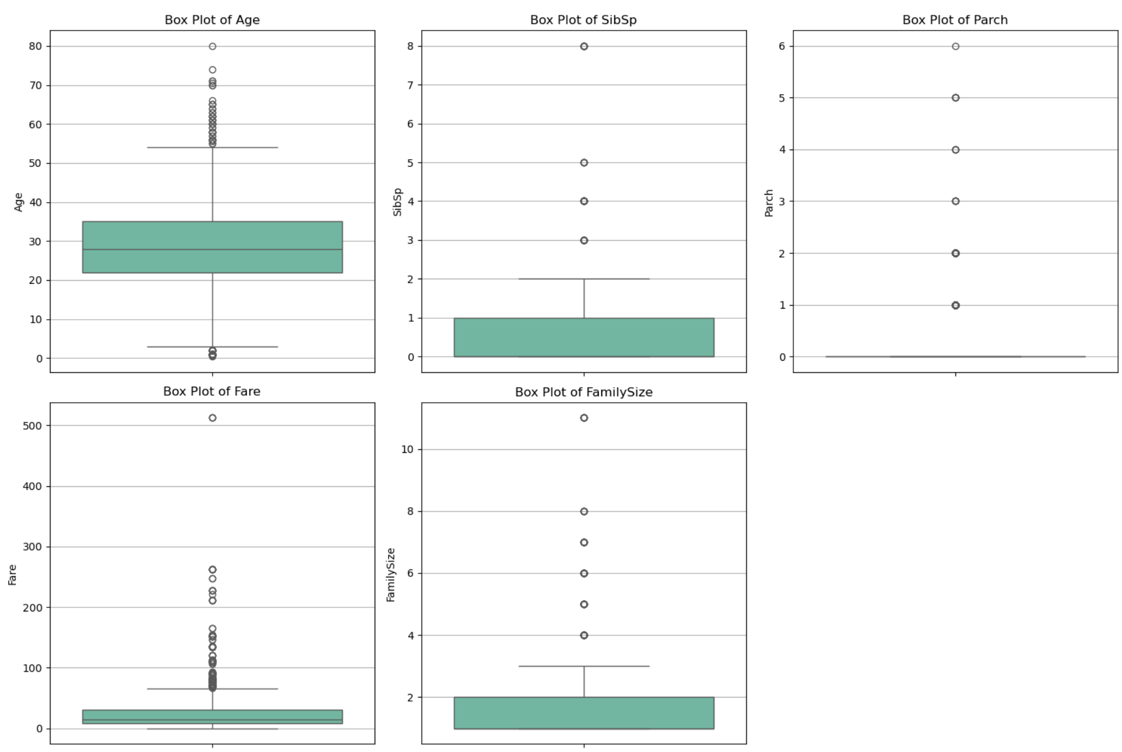


[24]:

```
# Set the size for the plots
plt.figure(figsize=(15, 10))

# List of numerical columns to plot
numerical_columns = ['Age', 'SibSp', 'Parch', 'Fare', 'FamilySize']

# Create box plots for each numerical column
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 3, i)  # Adjust the grid as needed
    sns.boxplot(data=df, y=column, palette='Set2')
    plt.title(f'Box Plot of {column}')
    plt.grid(axis='y')

plt.tight_layout()
plt.show()
```

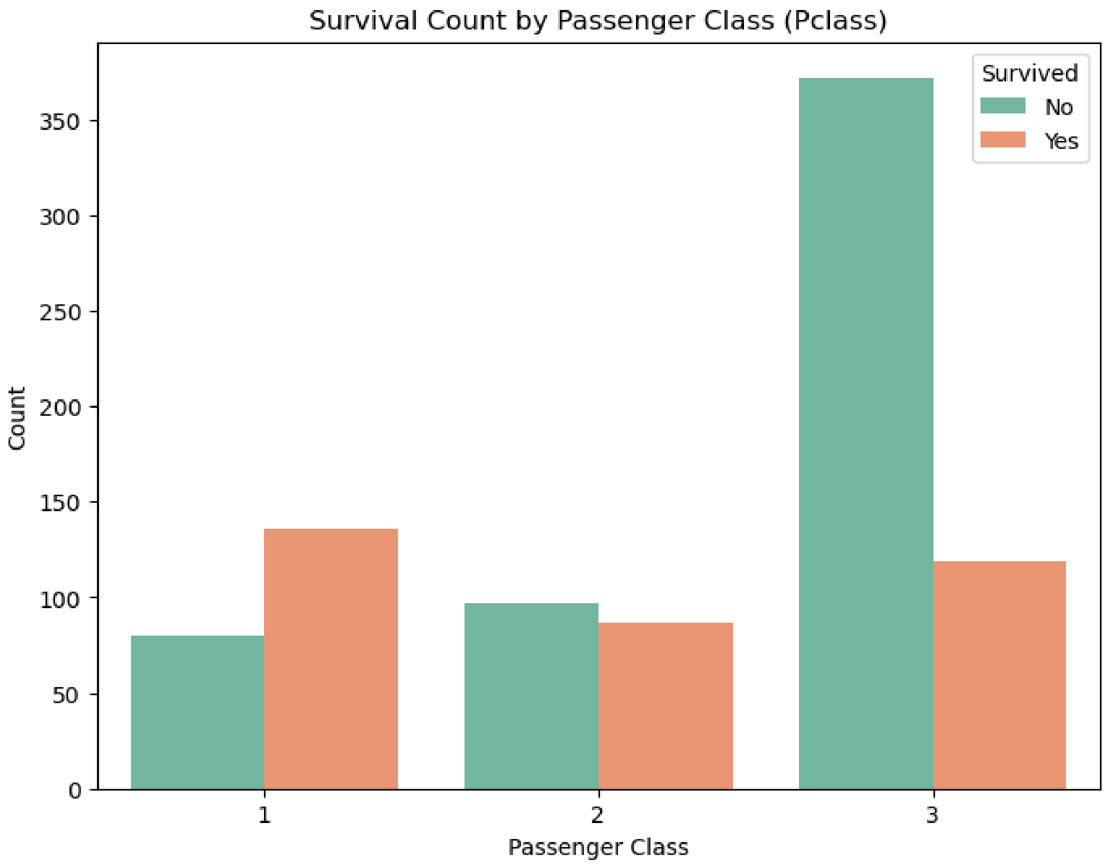
```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\406157803.py:10:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=column, palette='Set2')
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\406157803.py:10:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=column, palette='Set2')
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\406157803.py:10:
```

```
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=column, palette='Set2')
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\406157803.py:10:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=column, palette='Set2')
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\406157803.py:10:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, y=column, palette='Set2')
```



The number of males that has been dead is higher then the number of womens and the same thing is for who lived

[25]:
```
# Create a bar plot for survival count by passenger class
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Pclass', hue='Survived', palette='Set2')
plt.title('Survival Count by Passenger Class (Pclass)')
plt.xlabel('Passenger Class')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])
plt.show()
```

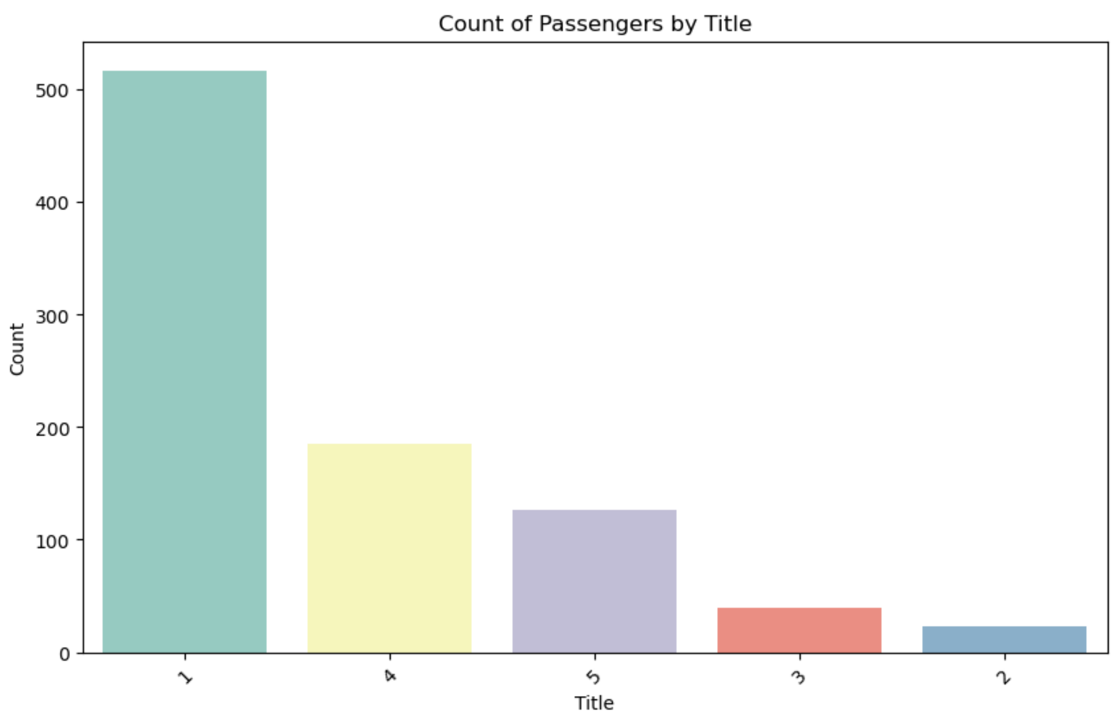


The passenger of the 3rd class have the best chance to survive

[26]:
```
# Create a bar plot for the count of passengers by title
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Title', palette='Set3', order=df['Title'].
 ↪value_counts().index)
plt.title('Count of Passengers by Title')
plt.xlabel('Title')
```

```
plt.ylabel('Count')
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.show()
```

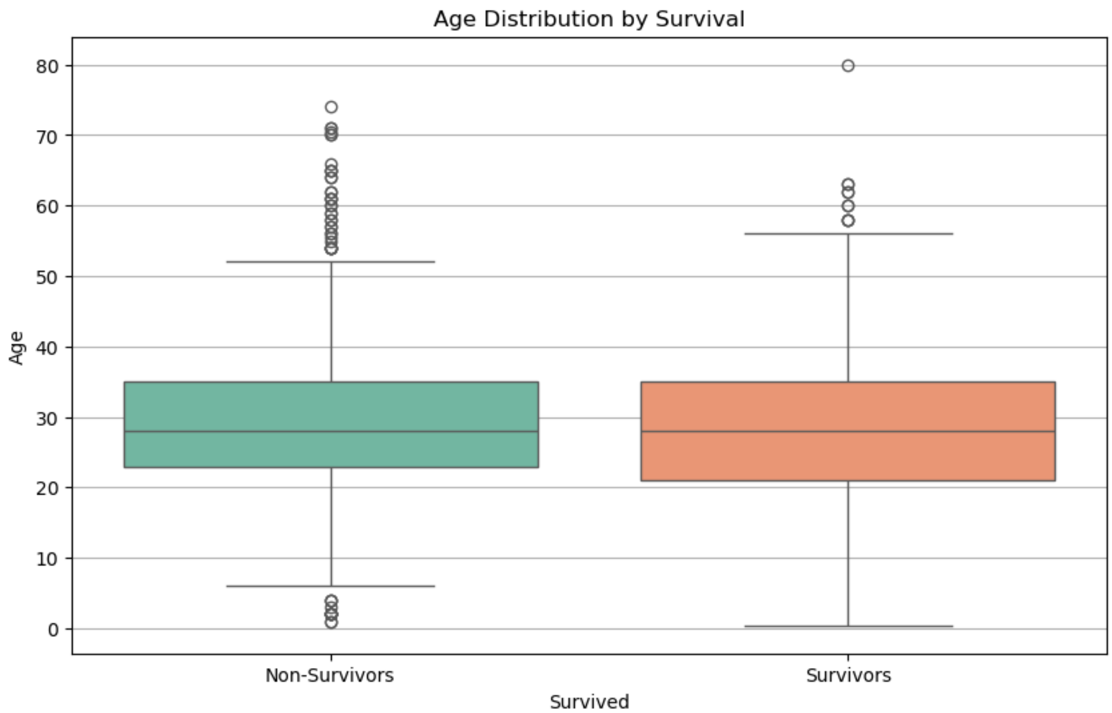
```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\3183109734.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=df, x='Title', palette='Set3',
order=df['Title'].value_counts().index)
```



[27]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Survived', y='Age', palette='Set2')
plt.title('Age Distribution by Survival')
plt.xlabel('Survived')
plt.ylabel('Age')
plt.xticks([0, 1], ['Non-Survivors', 'Survivors'])
plt.grid(axis='y')
plt.show()
```

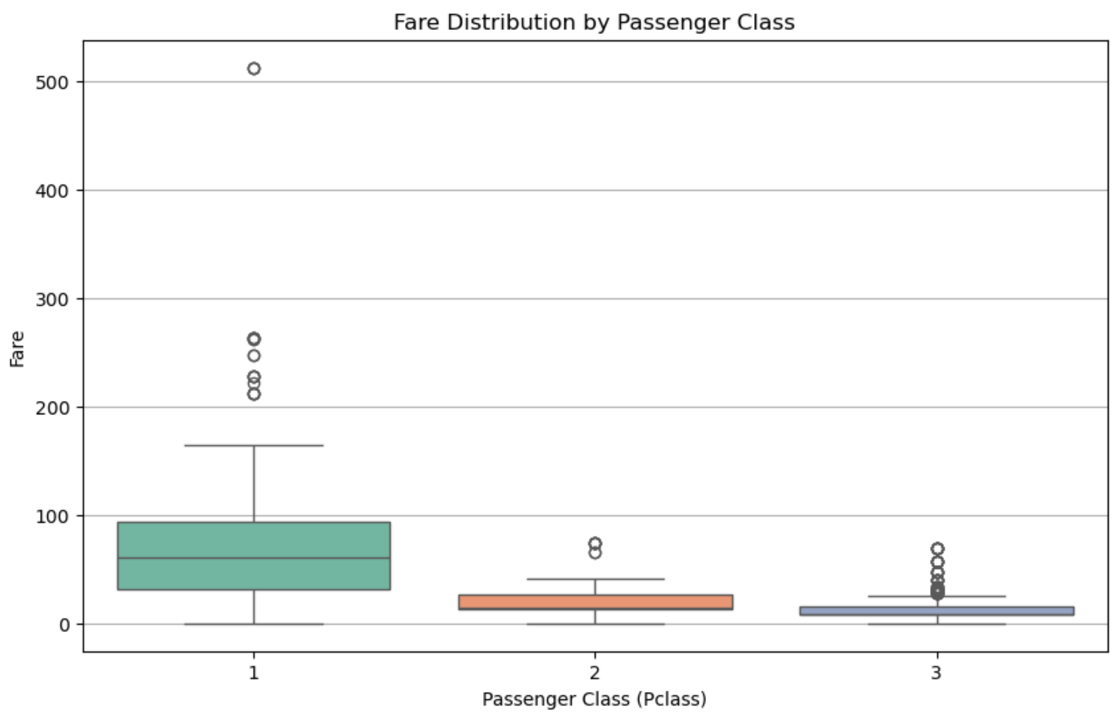
```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\1134825523.py:2:
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, x='Survived', y='Age', palette='Set2')
```



[28]:
```python
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Pclass', y='Fare', palette='Set2')
plt.title('Fare Distribution by Passenger Class')
plt.xlabel('Passenger Class (Pclass)')
plt.ylabel('Fare')
plt.grid(axis='y')
plt.show()
```

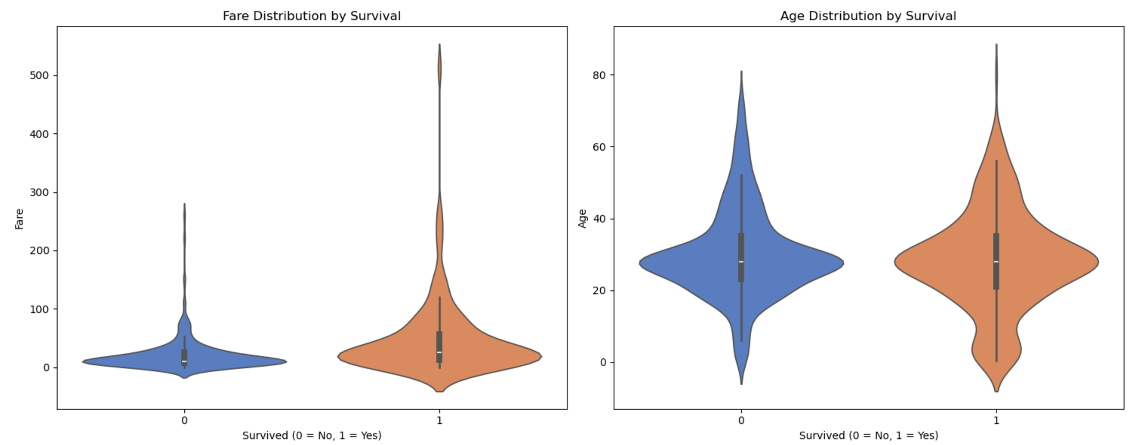
```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\2498514627.py:2:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(data=df, x='Pclass', y='Fare', palette='Set2')
```



[29]:
```
# Set up the figure size
plt.figure(figsize=(15, 6))

# Violin plot for Fare by Survival
plt.subplot(1, 2, 1)
sns.violinplot(data=df, x='Survived', y='Fare', palette='muted')
plt.title('Fare Distribution by Survival')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Fare')

# Violin plot for Age by Survival
plt.subplot(1, 2, 2)
sns.violinplot(data=df, x='Survived', y='Age', palette='muted')
plt.title('Age Distribution by Survival')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Age')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\3004225523.py:6:
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.violinplot(data=df, x='Survived', y='Fare', palette='muted')
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\3004225523.py:13:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.violinplot(data=df, x='Survived', y='Age', palette='muted')
```

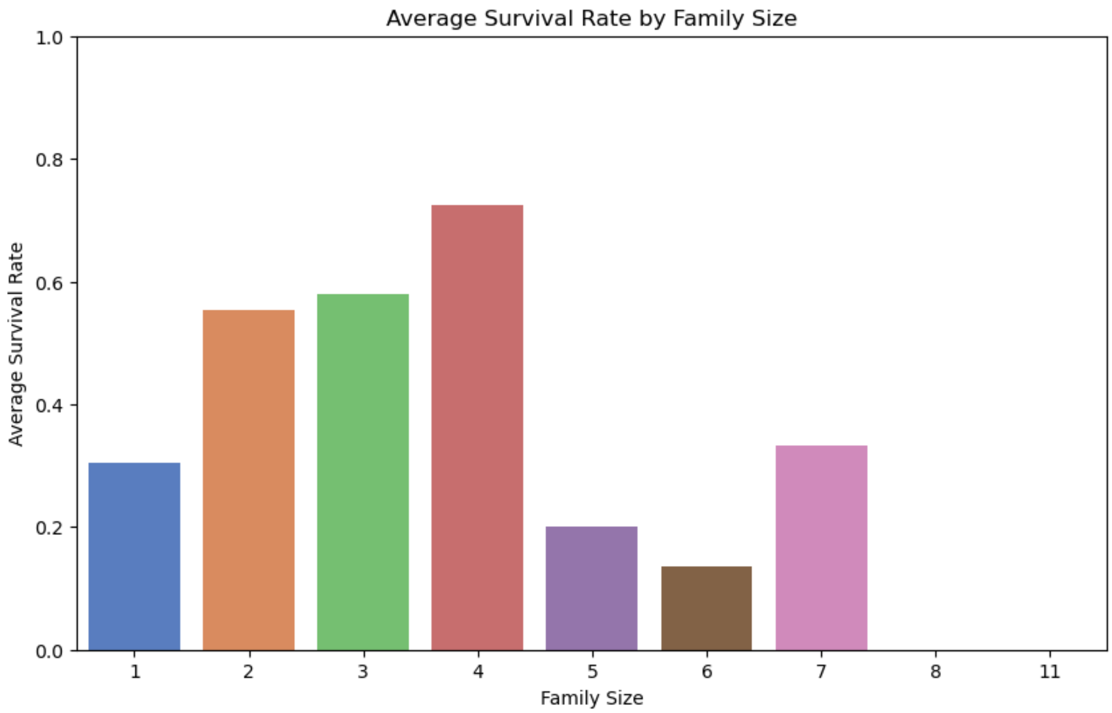


Fare ya3ni ujra - The most who survived is below 40

[30]:
```python
# Set the figure size
plt.figure(figsize=(10, 6))

# Calculate the mean survival rate for each family size
family_survival = df.groupby('FamilySize')['Survived'].mean().reset_index()

# Create a bar plot
sns.barplot(data=family_survival, x='FamilySize', y='Survived', palette='muted')

# Add titles and labels
plt.title('Average Survival Rate by Family Size')
plt.xlabel('Family Size')
plt.ylabel('Average Survival Rate')
plt.ylim(0, 1)  # Set y-axis limit to show percentage

# Show the plot
```

```
plt.show()
```

```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\4245167758.py:8:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=family_survival, x='FamilySize', y='Survived',
palette='muted')
```



[31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    int32
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
```

```
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    category
 8   CabinBool   891 non-null    int32
 9   Title       891 non-null    category
 10  FamilySize  891 non-null    int64
dtypes: category(2), float64(2), int32(2), int64(5)
memory usage: 57.9 KB
```

[32]:
```
# Create dummy variables for Embarked
df = pd.get_dummies(df, columns=["Embarked"])
df.head()
```

[32]:
```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  CabinBool  Title  \
0         0       3    0  22.0      1      0   7.2500          0      1
1         1       1    1  38.0      1      0  71.2833          1      5
2         1       3    1  26.0      0      0   7.9250          0      4
3         1       1    1  35.0      1      0  53.1000          1      5
4         0       3    0  35.0      0      0   8.0500          0      1

   FamilySize  Embarked_C  Embarked_Q  Embarked_S
0           2       False       False        True
1           2        True       False       False
2           1       False       False        True
3           2       False       False        True
4           1       False       False        True
```

[33]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    int32
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   CabinBool   891 non-null    int32
 8   Title       891 non-null    category
 9   FamilySize  891 non-null    int64
 10  Embarked_C  891 non-null    bool
 11  Embarked_Q  891 non-null    bool
 12  Embarked_S  891 non-null    bool
dtypes: bool(3), category(1), float64(2), int32(2), int64(5)
memory usage: 59.5 KB
```

Model building

[34]:
```python
# Define dependent and independent variables
X = df.drop(columns=['Survived'])
y = df['Survived']
```

[35]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,random_state=42)
```

[36]:
```python
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier,␣
 ↪GradientBoostingClassifier
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.svm import SVR, SVC
from xgboost import XGBRegressor, XGBClassifier
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    mean_squared_error,
    r2_score,
    roc_auc_score,
)
from tabulate import tabulate

# Define regression pipelines
regression_pipelines = {
    "LinearRegression": Pipeline([("scaler", StandardScaler()), ("lr",␣
 ↪LinearRegression())]),
    "Lasso": Pipeline([("scaler", StandardScaler()), ("lasso", Lasso())]),
    "Ridge": Pipeline([("scaler", StandardScaler()), ("ridge", Ridge())]),
    "DecisionTreeRegressor": Pipeline([("scaler", StandardScaler()), ("dt",␣
 ↪DecisionTreeRegressor())]),
    "RandomForestRegressor": Pipeline([("scaler", StandardScaler()), ("rf",␣
 ↪RandomForestRegressor())]),
    "KNeighborsRegressor": Pipeline([("scaler", StandardScaler()), ("kn",␣
 ↪KNeighborsRegressor())]),
    "SVR": Pipeline([("scaler", StandardScaler()), ("svr", SVR())]),
    "XGBRegressor": Pipeline([("scaler", StandardScaler()), ("xgb",␣
 ↪XGBRegressor())]),
    "GradientBoostingRegressor": Pipeline([("scaler", StandardScaler()),␣
 ↪("gbr", GradientBoostingRegressor())]),
}
```

```python
# Define classification pipelines
classification_pipelines = {
    "DecisionTreeClassifier": Pipeline([("scaler", StandardScaler()), ("dt", DecisionTreeClassifier())]),
    "RandomForestClassifier": Pipeline([("scaler", StandardScaler()), ("rf", RandomForestClassifier())]),
    "KNeighborsClassifier": Pipeline([("scaler", StandardScaler()), ("kn", KNeighborsClassifier())]),
    "SVC": Pipeline([("scaler", StandardScaler()), ("svc", SVC(probability=True))]),
    "XGBClassifier": Pipeline([("scaler", StandardScaler()), ("xgbc", XGBClassifier())]),
    "GradientBoostingClassifier": Pipeline([("scaler", StandardScaler()), ("gbc", GradientBoostingClassifier())]),
}

def evaluate_regression_models(pipelines, X_train, y_train, X_test, y_test):
    results = []
    for name, model in pipelines.items():
        model.fit(X_train, y_train)
        train_pred = model.predict(X_train)
        test_pred = model.predict(X_test)

        r2_train = r2_score(y_train, train_pred)
        r2_test = r2_score(y_test, test_pred)
        rmse_train = np.sqrt(mean_squared_error(y_train, train_pred))
        rmse_test = np.sqrt(mean_squared_error(y_test, test_pred))

        overfit = r2_train - r2_test
        results.append([
            name,
            f"{r2_train:.4f}",
            f"{r2_test:.4f}",
            f"{rmse_train:.4f}",
            f"{rmse_test:.4f}",
            f"{overfit:.4f}"
        ])

    headers = ["Model", "R² (Train)", "R² (Test)", "RMSE (Train)", "RMSE (Test)", "Overfitting (R² Train - Test)"]
    return results, headers

def evaluate_classification_models(pipelines, X_train, y_train, X_test, y_test):
    results = []
    for name, model in pipelines.items():
        model.fit(X_train, y_train)
```

```
        train_pred = model.predict(X_train)
        test_pred = model.predict(X_test)

        acc_train = accuracy_score(y_train, train_pred)
        acc_test = accuracy_score(y_test, test_pred)

        if hasattr(model, "predict_proba"):
            train_proba = model.predict_proba(X_train)[:, 1]
            test_proba = model.predict_proba(X_test)[:, 1]
            auc_train = roc_auc_score(y_train, train_proba)
            auc_test = roc_auc_score(y_test, test_proba)
            overfit_acc = acc_train - acc_test
            overfit_auc = auc_train - auc_test

            results.append([
                name,
                f"{acc_train:.4f}",
                f"{acc_test:.4f}",
                f"{auc_train:.4f}",
                f"{auc_test:.4f}",
                f"{overfit_acc:.4f}",
                f"{overfit_auc:.4f}"
            ])
        else:
            results.append([
                name,
                f"{acc_train:.4f}",
                f"{acc_test:.4f}",
                "N/A",
                "N/A",
                f"{acc_train - acc_test:.4f}",
                "N/A"
            ])

    headers = ["Model", "Accuracy (Train)", "Accuracy (Test)", "AUC-ROC␣
↪(Train)", "AUC-ROC (Test)", "Overfitting (Acc Train - Test)", "Overfitting␣
↪(AUC Train - Test)"]
    return results, headers

def find_best_model(results, is_regression=True):
    if is_regression:
        # For regression, maximize R² (Test) and minimize overfitting
        sorted_results = sorted(results, key=lambda x: (float(x[2]),␣
↪-float(x[5])), reverse=True)
    else:
        # For classification, maximize Accuracy (Test) and minimize overfitting
```

```python
        sorted_results = sorted(results, key=lambda x: (float(x[2]),␣
↪-float(x[5])), reverse=True)

    return sorted_results[0]

# Main execution logic
if np.issubdtype(y.dtype, np.number) and len(np.unique(y)) > 10:
    print("Performing regression analysis")
    results, headers = evaluate_regression_models(regression_pipelines,␣
↪X_train, y_train, X_test, y_test)
    print("\nRegression Models Evaluation:")
    is_regression = True
else:
    print("Performing classification analysis")
    results, headers = evaluate_classification_models(classification_pipelines,␣
↪X_train, y_train, X_test, y_test)
    print("\nClassification Models Evaluation:")
    is_regression = False

print(tabulate(results, headers=headers, tablefmt="grid"))
```

```
Performing classification analysis

Classification Models Evaluation:
+-----------------------------+--------------------+-------------------+---------
----------+------------------+-----------------------------------+---------------
--------------------+
| Model                       |   Accuracy (Train) |   Accuracy (Test) |   AUC-
ROC (Train) |   AUC-ROC (Test) |   Overfitting (Acc Train - Test) |
Overfitting (AUC Train - Test) |
+=============================+====================+===================+=========
==========+==================+===================================+===============
====================+
| DecisionTreeClassifier      |             0.9846 |            0.7709 |
0.9994 |           0.7772 |                            0.2136 |
0.2222 |
+-----------------------------+--------------------+-------------------+---------
----------+------------------+-----------------------------------+---------------
--------------------+
| RandomForestClassifier      |             0.9846 |            0.8212 |
0.9975 |           0.9004 |                            0.1633 |
0.0971 |
+-----------------------------+--------------------+-------------------+---------
----------+------------------+-----------------------------------+---------------
--------------------+
| KNeighborsClassifier        |             0.8624 |            0.8045 |
0.9336 |           0.8663 |                            0.0579 |
```

```
0.0673 |
+----------------------------+--------------------+------------------+---------
----------+------------------+----------------------------------+---------------
-------------------+
| SVC                        |             0.8511 |           0.8156 |
0.8771 |           0.8498 |                            0.0355 |
0.0273 |
+----------------------------+--------------------+------------------+---------
----------+------------------+----------------------------------+---------------
-------------------+
| XGBClassifier              |             0.9691 |           0.7989 |
0.996  |           0.8838 |                            0.1702 |
0.1122 |
+----------------------------+--------------------+------------------+---------
----------+------------------+----------------------------------+---------------
-------------------+
| GradientBoostingClassifier |             0.9087 |           0.838  |
0.9502 |           0.8902 |                            0.0707 |
0.06   |
+----------------------------+--------------------+------------------+---------
----------+------------------+----------------------------------+---------------
-------------------+
```

- AUC-ROC (Train): This is the Area Under the Receiver Operating Characteristic curve for the training set. It measures the model's ability to distinguish between classes (e.g., survived vs. not survived). A higher value indicates better performance on the training data.
- AUC-ROC (Test): Similar to the training AUC-ROC, but this measures the performance of the model on the unseen test data. It reflects how well the model generalizes to new data.
- Overfitting (Acc Train - Test): This is the difference in accuracy between the training and test sets. A high difference indicates that the model performs well on the training data but poorly on the test data, suggesting overfitting (the model memorizes the training data instead of learning general patterns).
- Overfitting (AUC Train - Test): This measures the difference in AUC-ROC between the training and test sets. Like the accuracy difference, a high value indicates overfitting.
- In summary, these metrics help you assess not only how well your model learns from the training data but also how well it performs on new, unseen data, which is crucial for effective predictive modeling

[37]:
```
# After running the evaluation and obtaining the results

# Find the best model
best_model = find_best_model(results, is_regression)

# Print the best model details
print("\nBest Model:")
print(best_model)
```

```
Best Model:
['GradientBoostingClassifier', '0.9087', '0.8380', '0.9502', '0.8902', '0.0707',
'0.0600']
```

[ ]:

[38]:
```python
gbc_pipeline = Pipeline([
    ('scaler', StandardScaler()),  # Scaling the data
    ('gbc', GradientBoostingClassifier())  # Gradient Boosting Classifier
])
```

[39]:
```python
gbc_pipeline.fit(X_train, y_train)
```

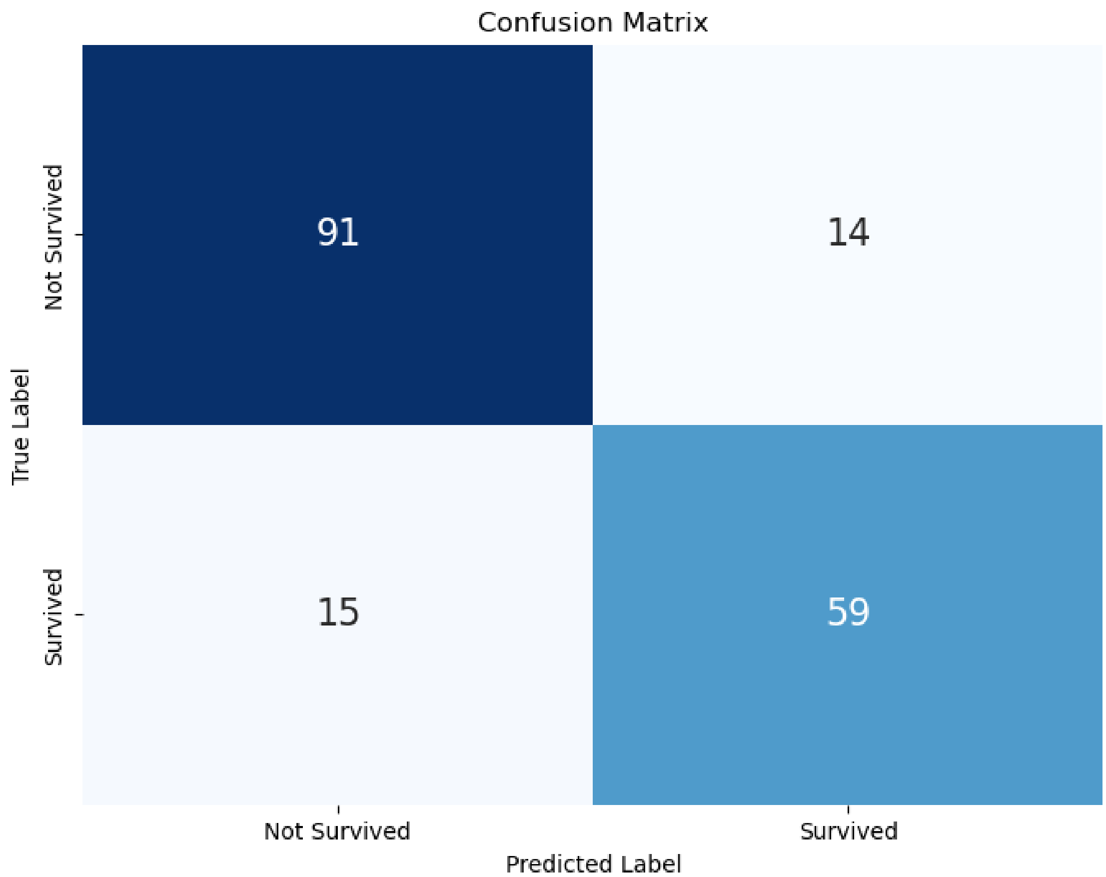
[39]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('gbc', GradientBoostingClassifier())])
```

[40]:
```python
y_pred = gbc_pipeline.predict(X_test)
```

[41]:
```python
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[91 14]
 [15 59]]
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       105
           1       0.81      0.80      0.80        74

    accuracy                           0.84       179
   macro avg       0.83      0.83      0.83       179
weighted avg       0.84      0.84      0.84       179
```

[42]:
```python
 # Create a DataFrame for better labeling
cm_df = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['Not␣
↪Survived','Survived'], columns=['Not Survived', 'Survived'])
plt.figure(figsize=(8, 6))
sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues',␣
↪cbar=False,annot_kws={"size": 16})
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



[43]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f"Accuracy of the Gradient Boosting model: {accuracy:.4f}")
```

```
Accuracy of the Gradient Boosting model: 0.8380
```

[44]:
```
# Save the model to a file
model_filename = 'gradient_boosting_model.pkl'
joblib.dump(gbc_pipeline, model_filename)
print(f"Model saved to {model_filename}")
```

```
Model saved to gradient_boosting_model.pkl
```

[45]:
```
# Predict using the test set
y_pred = gbc_pipeline.predict(X_test)
```

[46]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```python
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8380
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.87      0.86       105
           1       0.81      0.80      0.80        74

    accuracy                           0.84       179
   macro avg       0.83      0.83      0.83       179
weighted avg       0.84      0.84      0.84       179
```

[47]:
```python
# --- In future use cases, you can load and reuse the model ---

# Load the model from file
loaded_model = joblib.load(model_filename)
```

[48]:
```python
# Use loaded model for prediction on test data (or new data)
loaded_model_pred = loaded_model.predict(X_test)
```

[49]:
```python
# Evaluate the loaded model
loaded_accuracy = accuracy_score(y_test, loaded_model_pred)
print(f"Accuracy of the loaded model: {loaded_accuracy:.4f}")
```

```
Accuracy of the loaded model: 0.8380
```

- Saving the model: The trained model (gbc_pipeline) is saved to a file named 'gradient_boosting_model.pkl' using joblib.dump().
- Predicting: The model makes predictions on the test set (X_test), and the results (y_pred) are compared to the actual values (y_test) to calculate accuracy and display a classification report.
- Loading the model: Later, the saved model is reloaded using joblib.load() to use again without retraining.
- Evaluating the loaded model: The reloaded model is used to make predictions again and its accuracy is calculated, ensuring it performs the same as before being saved.

[50]:
```python
test_df=pd.read_csv('test.csv')
```

[51]:
```python
# 2. Preprocess the test data
# Apply the same steps as in the training dataset

# Impute missing values for 'Age' using the median
age_imputer = SimpleImputer(strategy='median')
```

```
test_df['Age'] = age_imputer.fit_transform(test_df[['Age']])
```

[52]:
```
# Create 'CabinBool' column for 'Cabin' missing values
test_df['CabinBool'] = test_df['Cabin'].notnull().astype(int)
test_df['Cabin'].fillna('Unknown', inplace=True)
```

```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\2042715492.py:3:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  test_df['Cabin'].fillna('Unknown', inplace=True)
```

[53]:
```
# Impute missing values for 'Embarked' using the most frequent value
embarked_imputer = SimpleImputer(strategy='most_frequent')
test_df['Embarked'] = embarked_imputer.fit_transform(test_df[['Embarked']]).
 ↪flatten()
```

[54]:
```
# Drop 'Cabin' and other unnecessary columns
test_df = test_df.drop(['Cabin', 'Ticket', 'PassengerId'], axis=1)

# Extract titles and map them
test_df['Title'] = test_df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

```
<>:5: SyntaxWarning: invalid escape sequence '\.'
<>:5: SyntaxWarning: invalid escape sequence '\.'
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\774836579.py:5:
SyntaxWarning: invalid escape sequence '\.'
  test_df['Title'] = test_df['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

[55]:
```
# Define the same title map function
test_df['Title'] = test_df['Name'].apply(get_title).apply(title_map)

# Create 'FamilySize' feature
test_df['FamilySize'] = test_df['SibSp'] + test_df['Parch'] + 1
```

[56]:
```
# Drop 'Name' column
test_df = test_df.drop(['Name'], axis=1)

# Convert 'Sex' and 'Embarked' to categorical
test_df['Sex'] = test_df['Sex'].astype('category')
```

```
test_df['Embarked'] = test_df['Embarked'].astype('category')
test_df['Title'] = test_df['Title'].astype('category')
```

[57]:
```
# Replace 'male' and 'female' with 0 and 1
test_df["Sex"] = test_df["Sex"].replace(["male", "female"], [0, 1]).astype(int)

# One-hot encode 'Embarked'
test_df = pd.get_dummies(test_df, columns=["Embarked"])

# --- End of preprocessing ---
```

```
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\3756072260.py:2:
FutureWarning: Downcasting behavior in `replace` is deprecated and will be
removed in a future version. To retain the old behavior, explicitly call
`result.infer_objects(copy=False)`. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  test_df["Sex"] = test_df["Sex"].replace(["male", "female"], [0,
1]).astype(int)
C:\Users\Khoder Asmar\AppData\Local\Temp\ipykernel_5760\3756072260.py:2:
FutureWarning: The behavior of Series.replace (and DataFrame.replace) with
CategoricalDtype is deprecated. In a future version, replace will only be used
for cases that preserve the categories. To change the categories, use
ser.cat.rename_categories instead.
  test_df["Sex"] = test_df["Sex"].replace(["male", "female"], [0,
1]).astype(int)
```

[58]:
```
# Impute missing values for 'Fare' (if any) using the median
if 'Fare' in test_df.columns:
    fare_imputer = SimpleImputer(strategy='median')
    test_df['Fare'] = fare_imputer.fit_transform(test_df[['Fare']])
```

[59]:
```
test_df.isnull().sum()
```

[59]:
```
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
CabinBool      0
Title          0
FamilySize     0
Embarked_C     0
Embarked_Q     0
Embarked_S     0
dtype: int64
```

[60]:
```python
# 3. Load the saved model
model_filename = 'gradient_boosting_model.pkl'
loaded_model = joblib.load(model_filename)

# 4. Predict using the test data
# Make sure that the test data has the same columns as the training data
predictions = loaded_model.predict(test_df)

# 5. Save predictions to CSV
output_df = pd.DataFrame({'PassengerId': test_df.index, 'Survived':␣
 ↪predictions})
output_df.to_csv('test_predictions.csv', index=False)

print("Predictions saved to 'test_predictions.csv'.")
```

```
Predictions saved to 'test_predictions.csv'.
```

[61]:
```python
# knowing the number of survived and the number od ddead



# Load the predictions
predictions_df = pd.read_csv('test_predictions.csv')

# Count the number of survivors and non-survivors
survivor_count = predictions_df['Survived'].value_counts()
print(f"Number of Survivors: {survivor_count.get(1, 0)}")
print(f"Number of Deceased: {survivor_count.get(0, 0)}")
```

```
Number of Survivors: 152
Number of Deceased: 266
```

[62]:
```python
# Count the number of survivors and non-survivors
survivor_count = predictions_df['Survived'].value_counts()
total_count = survivor_count.sum()

# Calculate percentages
survived_percentage = (survivor_count.get(1, 0) / total_count) * 100
not_survived_percentage = (survivor_count.get(0, 0) / total_count) * 100

# Print results
print(f"Number of Survivors: {survivor_count.get(1, 0)} ({survived_percentage:.
 ↪2f}%)")
print(f"Number of Deceased: {survivor_count.get(0, 0)}␣
 ↪({not_survived_percentage:.2f}%)")
```

```
Number of Survivors: 152 (36.36%)
Number of Deceased: 266 (63.64%)
```

```
[ ]:
```