# BHARAT VIRTUAL INTERNSHIP

# PROJECT TWO

# (TASK NAME: TITANIC CLASSIFICATION)

# PROJECT OVERVIEW

- **Objective:** Build a Titanic Classification Model Using the Logistic Regression Model
- **Dataset:** Link to my dataset
- **Submission Date:** December 10, 2023
- **Author:** SUNMOLA M.A
- **Reference**: Youtube Resource

# STEP 1: DATA CLEANING & INITIAL EXPLORATION

1. Import necessary libraries
2. Load the dataset
3. Display a sample of the dataset
4. Explore the dataset's size and structure
5. Handle possible inconstencies with dataset structure accordingly
6. Check for data type consistency
7. Check for duplicates

## 1.1: Importing the necessary dependencies

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,recall_score, confusion_ma
```

## 1.2: Loading the Dataset

In [2]:
```
data = pd.read_csv("train.csv")
```

## 1.3: Previewing a sample of the dataset

In [3]:
```
data.sample(5)
```

Out[3]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Er |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **653** | 654 | 1 | 3 | O'Leary, Miss. Hanora "Norah" | female | NaN | 0 | 0 | 330919 | 7.8292 | NaN | |
| **647** | 648 | 1 | 1 | Simonius-Blumer, Col. Oberst Alfons | male | 56.0 | 0 | 0 | 13213 | 35.5000 | A26 | |
| **114** | 115 | 0 | 3 | Attalah, Miss. Malake | female | 17.0 | 0 | 0 | 2627 | 14.4583 | NaN | |
| **137** | 138 | 0 | 1 | Futrelle, Mr. Jacques Heath | male | 37.0 | 1 | 0 | 113803 | 53.1000 | C123 | |
| **125** | 126 | 1 | 3 | Nicola-Yarred, Master. Elias | male | 12.0 | 1 | 0 | 2651 | 11.2417 | NaN | |

## 1.4 Exploring the dataset's structure, size and contents

- Check the shape of the dataset
- Check general information about the structure of the dataset
- Check for missing values

In [4]:
```
# Checking the shape of the data
data.shape
```

Out[4]:
```
(891, 12)
```

In [5]:
```
# Displaying information about the data
data.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 [6]:
```
# Checking for null values
data.isnull().sum()
```

Out[6]:
```
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
```

# 1.4 INFERENCE

- The train dataset contains 12 features with 891 records
- The 'Age', 'Cabin', and 'Embarked' columns contains two or more missing values

# 1.5. Handling the found inconsistencies (i.e. null values) with the dataset structure

- Drop the **Cabin** column
- Fill the **Age** column with mean value
- Fill the **Embarked** column with mode value
- Check for Null Values After Handling

In [7]:
```
# Dropping the "Cabin" column
data.drop(columns="Cabin", inplace=True)
```

In [8]:
```python
# Handling missing values in "Age" by filling with mean
data['Age'].fillna(data['Age'].mean(), inplace = True)
```

In [9]:
```python
# Handling missing values in "Embarked" by filling with mode
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace = True)
```

In [10]:
```python
#Checking for Null Values After Handling
data.isnull().sum()
```

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

# 1.5 INFERENCE

- Now, we can be sure that our dataset no longer contains a null value

# 1.6. Checking for data type inconsistencies

In [11]:
```python
# Displaying Unique Data Types for Each Column

# First, create an empty dataframe to store the results
data_type= pd.DataFrame(columns=['COLUMN NAMES', 'UNIQUE DATA TYPES'])

# Secondly, iterate through columns and store unique data types in the new dataframe
for column_name in data.columns:
    unique_data_types = data[column_name].apply(type).unique()
    data_type = pd.concat([data_type, pd.DataFrame({'COLUMN NAMES': [column_name], 'UN

# Lastly, display the resulting table
print(data_type)
```

```
   COLUMN NAMES  UNIQUE DATA TYPES
0   PassengerId    [<class 'int'>]
1      Survived    [<class 'int'>]
2        Pclass    [<class 'int'>]
3          Name    [<class 'str'>]
4           Sex    [<class 'str'>]
5           Age  [<class 'float'>]
6         SibSp    [<class 'int'>]
7         Parch    [<class 'int'>]
8        Ticket    [<class 'str'>]
9          Fare  [<class 'float'>]
10     Embarked    [<class 'str'>]
```

## 1.6 INFERENCE

- It is satisfactory to know that each column contains unique data type
- Hence, there would not be need for **type casting**

## 1.7 Check for and drop duplicates entries

In [12]:
```
data[data.duplicated()]
```

Out[12]:

| PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|

## 1.7 INFERENCE

- Good to know! The dataset has no duplicate rows

# STEP 2: EXPLORATORY DATA ANALYSIS + DATA VISUALIZATION

1. Explore basic statistics of the data
2. Visually explore all of the dataset feature relationship
3. Visualize target distribution
4. Visualize Gender distribution
5. Visualize Social class distribution
6. Visualize Gender vs. Survival relationship
7. Visualize Social class vs. Survival relationship

## 2.1 Displaying descriptive statistics of the data

In [13]:
```
data.describe()
```

Out[13]:

| | PassengerId | Survived | Pclass | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| **count** | 891.000000 | 891.000000 | 891.000000 | 891.000000 | 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 | 13.002015 | 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 | 22.000000 | 0.000000 | 0.000000 | 7.910400 |
| **50%** | 446.000000 | 0.000000 | 3.000000 | 29.699118 | 0.000000 | 0.000000 | 14.454200 |
| **75%** | 668.500000 | 1.000000 | 3.000000 | 35.000000 | 1.000000 | 0.000000 | 31.000000 |
| **max** | 891.000000 | 1.000000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

# 2.1 INFERENCE

- The mean age of passengers is approx 30yrs
- The mean fare amount is also 32
- The minimum and maximum vlaues for the 'PassengerId', 'survived', and 'Pclass' is a correct indication that the features contain unique values as contained in the dataset metadata

# 2.2 Visually explore all of the dataset feature relationship

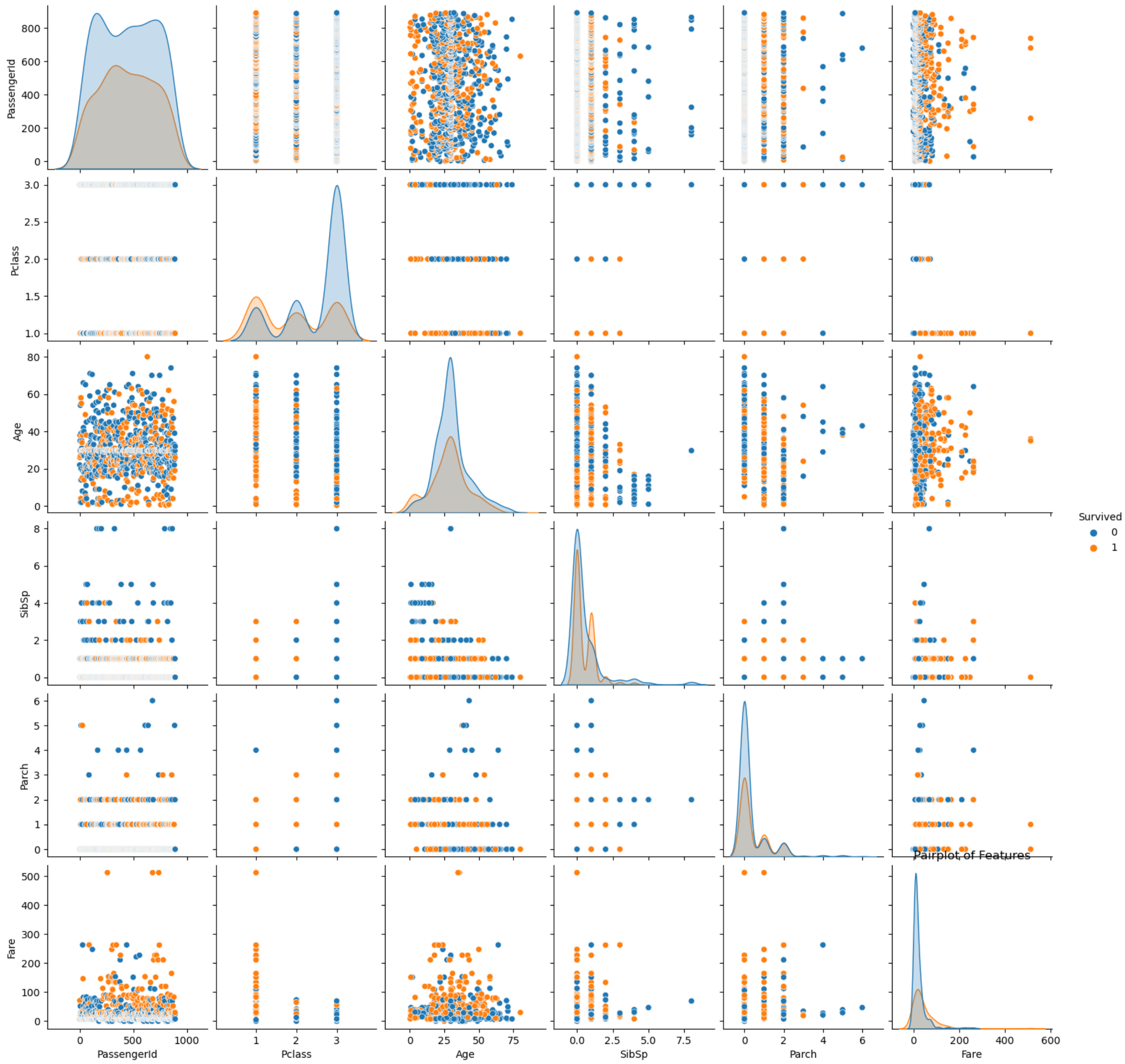
- With the aid of a pairplot
- With the aid of a heatmap

In [14]:

```
#Pairplot

sns.pairplot(data, hue='Survived')
plt.title('Pairplot of Features')
```

Out[14]:

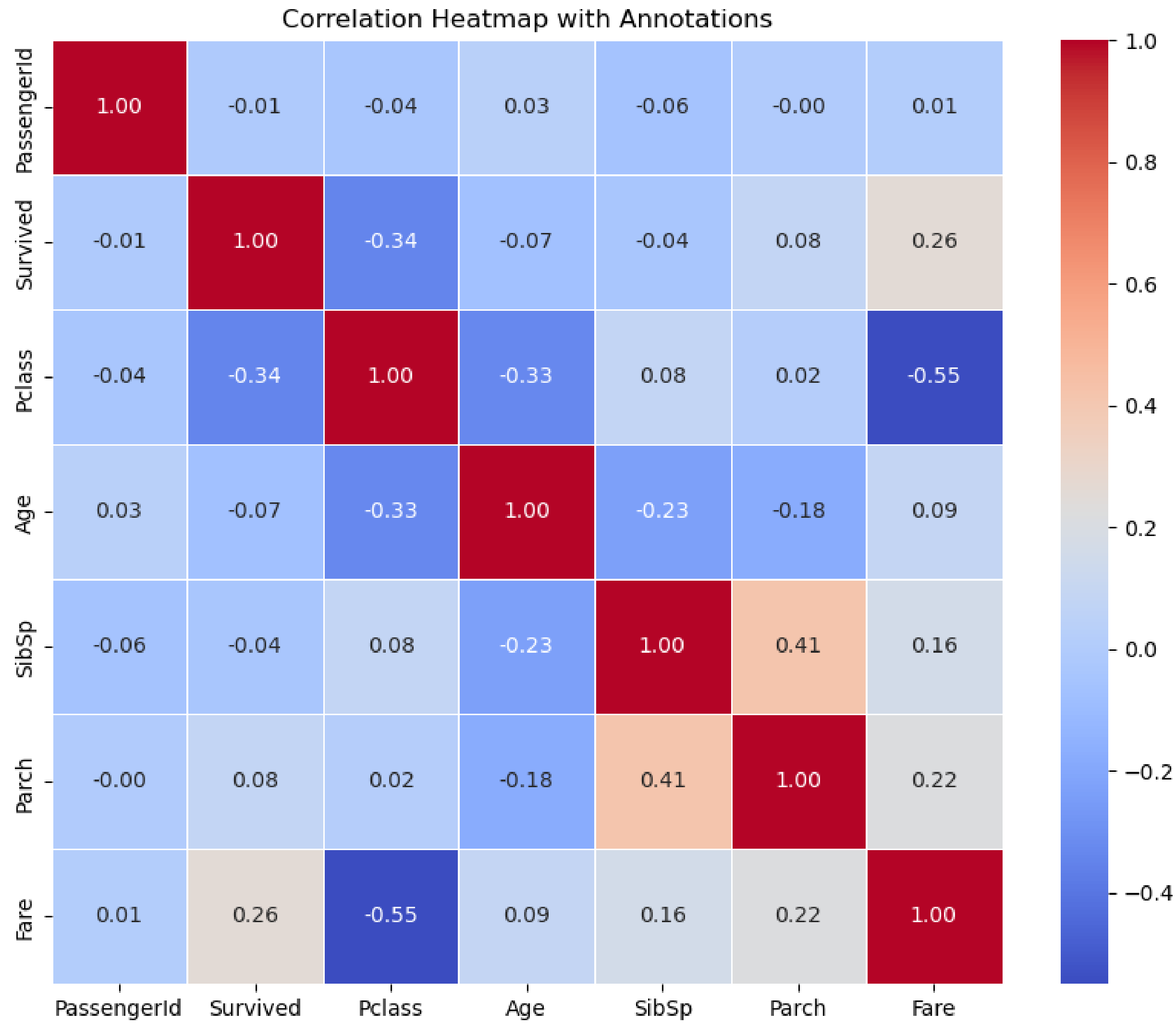
```
Text(0.5, 1.0, 'Pairplot of Features')
```




In [15]:

```python
#Heatmap

# Create a heatmap with annotated correlation values
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

# Set the title
plt.title('Correlation Heatmap with Annotations')

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

```
C:\Users\msunmola\AppData\Local\Temp\ipykernel_32172\2997496438.py:5: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of numeri
c_only to silence this warning.
  sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
```




## 2.2 INFERENCE

- **'PassengerID', 'Name', 'Ticket', and 'Cabin'** seem not to have any causal relationship/effect with/to the chances of survival

## 2.3 Visualize target (i.e. Survival) distribution

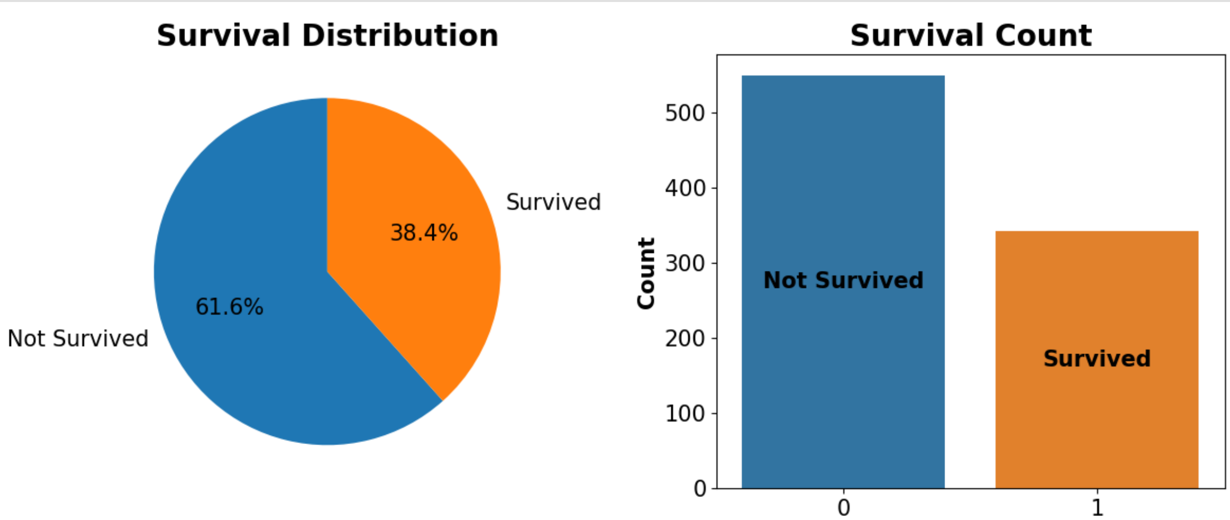
- Display unique values for the 'Survival' column
- Using Pie Chart and Countplot

In [16]:
```
data['Survived'].value_counts()
```

Out[16]:
```
0    549
1    342
Name: Survived, dtype: int64
```

In [17]:
```
# Create a 1x2 subplot grid
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
label_mapping = {data['Survived'].value_counts()[0]: 'Not Survived', data['Survived'].
data['Survived_Label'] = data['Survived'].map(label_mapping)
```

```
# Pie chart
labels = ['Not Survived', 'Survived']
sizes = data['Survived'].value_counts()
axes[0].pie(sizes, labels=labels, autopct=lambda p: '{:.1f}%'.format(p), startangle=90
         labeldistance=1.1, textprops={'fontsize': 15})
axes[0].set_title('Survival Distribution', fontweight='bold', fontsize=20)

# Countplot
sns.countplot(x='Survived', data=data, ax=axes[1])
axes[1].set_title('Survival Count')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('')
plt.ylabel('Count', fontsize=15, fontweight='bold')


# Annotate the bars with labels
for p in axes[1].patches:
    label = label_mapping[p.get_height()]
    axes[1].text(p.get_x() + p.get_width() / 2., p.get_height() / 2, label,
                 ha='center', va='center', fontweight='bold', color='black', fontsize=

data.drop(columns='Survived_Label', inplace=True)
axes[1].set_title('Survival Count', fontweight='bold', fontsize=20)

# Adjust layout
plt.tight_layout()


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



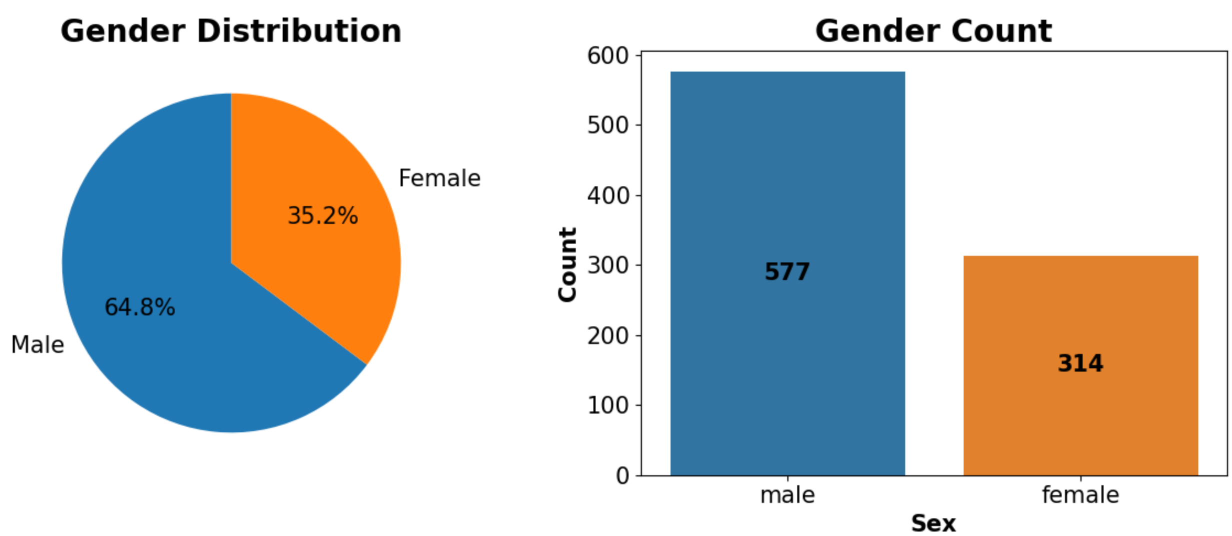

## 2.4 Visualize Gender distribution

- Display unique values for the 'Sex' column
- Using Pie Chart and Countplot

In [18]:
```
data['Sex'].value_counts()
```

```
Out[18]: male      577
         female    314
         Name: Sex, dtype: int64
```

```
In [19]: # Map numeric values to labels for the "Sex" column
label_mapping_sex = {'male': 'Male', 'female': 'Female'}
data['Sex_Label'] = data['Sex'].map(label_mapping_sex)

# Create a 1x2 subplot grid
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Pie chart for "Sex" column
labels_sex = ['Male', 'Female']
sizes_sex = data['Sex'].value_counts()

axes[0].pie(sizes_sex, labels=labels_sex, autopct=lambda p: '{:.1f}%'.format(p), start
axes[0].set_title('Gender Distribution', fontweight='bold', fontsize=20)


# Countplot for "Sex" column with labels at the center of the bars
sns.countplot(x='Sex', data=data, ax=axes[1])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Sex', fontsize=15, fontweight='bold')
plt.ylabel('Count', fontsize=15, fontweight='bold')

# Annotate the bars with count labels at the center
for p in axes[1].patches:
    label = f"{int(p.get_height())}"  # Get the count as an integer
    axes[1].text(p.get_x() + p.get_width() / 2., p.get_height() / 2, label,
                 ha='center', va='center', fontweight='bold', color='black', fontsize=

data.drop(columns='Sex_Label',inplace=True)
axes[1].set_title('Gender Count', fontweight='bold', fontsize=20)

# Adjust layout
plt.tight_layout()

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



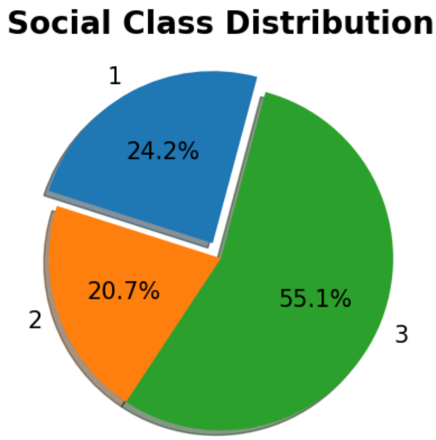
# 2.5 Visualize Social class distribution

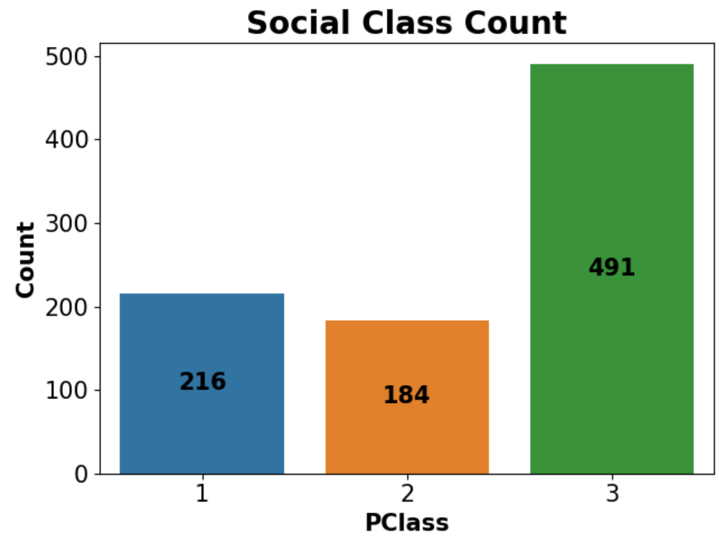
- Display unique values for the 'Pclass' column
- Using Pie Chart and Countplot

In [20]:
```
data['Pclass'].value_counts()
```

Out[20]:
```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

In [21]:
```
# Map numeric values to labels for the "Pclass" column
label_mapping_pclass = {1: 'Class 1', 2: 'Class 2', 3: 'Class 3'}
data['Pclass_Label'] = data['Pclass'].map(label_mapping_pclass)

# Create a 1x2 subplot grid
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Pie chart for 'Pclass'
sizes_pclass = data['Pclass'].value_counts().sort_index().values
explode_pclass = (0.1, 0, 0)
axes[0].pie(sizes_pclass, explode=explode_pclass, labels=label_mapping_pclass, autopct
axes[0].set_title('Social Class Distribution', fontweight='bold', fontsize=20)


# Countplot for 'Pclass'
sns.countplot(x='Pclass', data=data, ax=axes[1])
axes[1].set_title('Social Class Count', fontweight='bold', fontsize=20)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('PClass', fontsize=15, fontweight='bold')
plt.ylabel('Count', fontsize=15, fontweight='bold')

# Annotate the bars with count labels at the center
for p in axes[1].patches:
    label = f"{int(p.get_height())}"  # Get the count as an integer
    axes[1].text(p.get_x() + p.get_width() / 2., p.get_height() / 2, label,
                 ha='center', va='center', fontweight='bold', color='black', fontsize=

data.drop(columns='Pclass_Label', inplace=True)
# Adjust layout
plt.tight_layout()

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



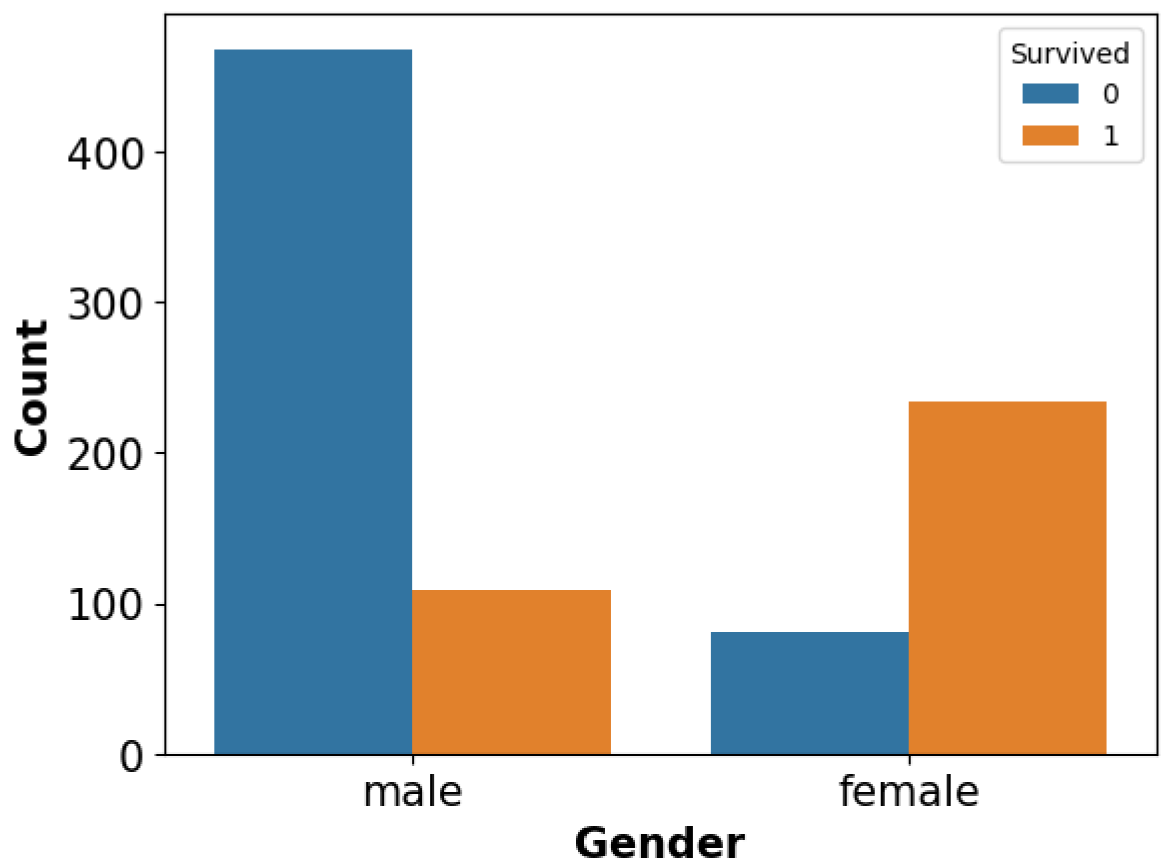

## 2.6 Visualize Gender Vs. Survival relationship

In [22]:

```
#Gender Vs. Survival relationship
sns.countplot(x='Sex', hue='Survived', data=data)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('Gender', fontsize=15, fontweight='bold')
plt.ylabel('Count', fontsize=15, fontweight='bold')
```

Out[22]:

```
Text(0, 0.5, 'Count')
```



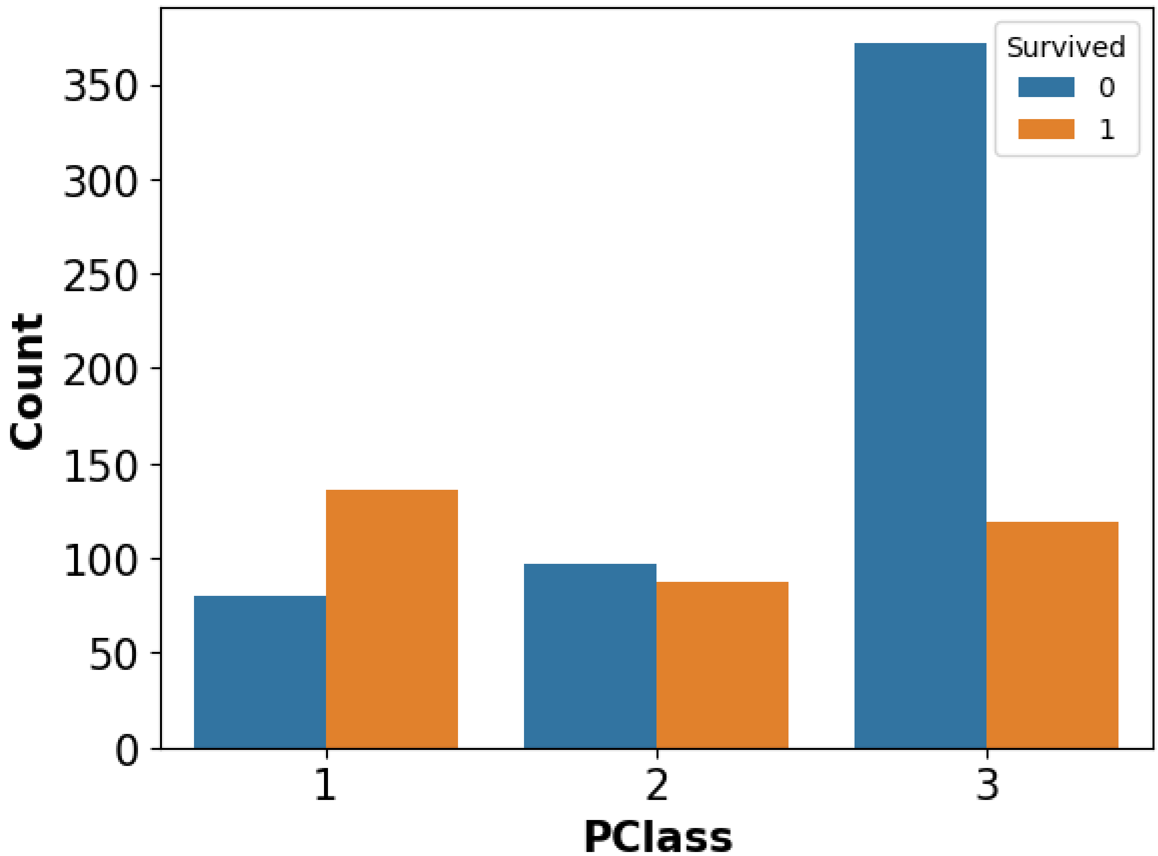
## 2.6 INFERENCE

- Very few percentage of male who boarded the ship survived
- More female passagers tends to have survived the wreck.

## 2.7 Visualize Social Class Vs. Survival relationship

In [23]:
```
sns.countplot(x='Pclass', hue='Survived', data=data)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.xlabel('PClass', fontsize=15, fontweight='bold')
plt.ylabel('Count', fontsize=15, fontweight='bold')
```

Out[23]:
```
Text(0, 0.5, 'Count')
```




## 2.7 INFERENCE

- Passenger's social class status tends to have a direct correlation with their chances of survival
- A greater percentage of lower social class percentage did not make it alive from the ship wreck

# STEP 3: DATA PREPROCESSING

1. Preview few samples of the data
2. Feature Engineering
3. Feature selection
4. Train Test Split

## 3.1 Preview Data

In [24]:
```
data.sample(5)
```

Out[24]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Embarl |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **174** | 175 | 0 | 1 | Smith, Mr. James Clinch | male | 56.000000 | 0 | 0 | 17764 | 30.6958 | |
| **29** | 30 | 0 | 3 | Todoroff, Mr. Lalio | male | 29.699118 | 0 | 0 | 349216 | 7.8958 | |
| **788** | 789 | 1 | 3 | Dean, Master. Bertram Vere | male | 1.000000 | 1 | 2 | C.A. 2315 | 20.5750 | |
| **631** | 632 | 0 | 3 | Lundahl, Mr. Johan Svensson | male | 51.000000 | 0 | 0 | 347743 | 7.0542 | |
| **209** | 210 | 1 | 1 | Blank, Mr. Henry | male | 40.000000 | 0 | 0 | 112277 | 31.0000 | |

## 3.2 Feature Engineering

- Handling Categorical Variables

In [25]:
```
#Encode male and female values in the 'Sex' column as 0 and 1 respectively
#Similarly, encode C, Q, & S values in the 'Embarked' column as 0, 1, and 2 respective

data.replace({'Sex':{'male':0,'female':1}, 'Embarked':{'C': 0, 'Q':1, 'S':2}}, inplace
data.sample(2)
```

Out[25]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **724** | 725 | 1 | 1 | Chambers, Mr. Norman Campbell | 0 | 27.0 | 1 | 0 | 113806 | 53.1 | 2 |
| **432** | 433 | 1 | 2 | Louch, Mrs. Charles Alexander (Alice Adelaide ... | 1 | 42.0 | 1 | 0 | SC/AH 3085 | 26.0 | 2 |

## 3.3 Feature Selection

In [26]:
```
#Split the dataset into target and features for model training purposes

X=data.drop(columns=['PassengerId','Survived', 'Name', 'Ticket'], axis=1) #feature
y=data['Survived'] #target
```

## 3.4 Train Test Split

In [27]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=5)
```

# STEP 4: MODEL BUILDING & EVALUATION

1. Develop a Logistic Regression Model
2. Train the model on the 'training' dataset
3. Evalaute the model performance on the training dataset
4. Train the model on the 'test' dataset
5. Evalaute the model performance on the testing dataset
6. Finally, submit your predictions on the Kaggle dataset to evaluate model overall performance

## 4.1 Build the Model

In [28]:
```
LR_model = LogisticRegression()
LR_model.fit(X_train, y_train)
```

```
C:\Users\msunmola\AppData\Local\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[28]:

```
▾ LogisticRegression
LogisticRegression()
```

## 4.2 Training the model on the Train dataset

In [29]:

```
y_hat = LR_model.predict(X_test)
y_hat
```

Out[29]:

```
array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0], dtype=int64)
```

## 4.3 Evaluating the model on the Train dataset

In [30]:

```
LRmodel_accuracy = accuracy_score(y_test, y_hat)
LRmodel_precision = precision_score(y_test, y_hat)
LRmodel_ConfMatrix = confusion_matrix(y_test, y_hat)

Model_Evaluation=pd.DataFrame({'TrainingSet Accuracy':[LRmodel_accuracy], 'TrainingSet

Model_Evaluation = Model_Evaluation.style.format({
    'TrainingSet Accuracy': '{:.2%}',
    'TrainingSet Precision': '{:.2%}'
})


Model_Evaluation
```

Out[30]:

| | TrainingSet Accuracy | TrainingSet Precision |
|---|---|---|
| **Linear Regression Model** | 82.12% | 82.12% |

## 4.4 & 4.5 Train & Evaluate model performance on the test dataset

In [31]:

```
# Load the test data
test_data = pd.read_csv("test.csv")

# Drop unnecessary columns
test_data.drop(columns="Cabin", inplace=True)

# Fill missing values
test_data['Age'].fillna(test_data['Age'].mean(), inplace=True)
test_data['Fare'].fillna(test_data['Fare'].mean(), inplace=True)

# Map categorical values to numeric
```

```
test_data.replace({'Sex': {'male': 0, 'female': 1}, 'Embarked': {'C': 0, 'Q': 1, 'S':

# Extract features for prediction
X_test_data = test_data.drop(columns=['PassengerId', 'Name', 'Ticket'], axis=1)


# Use the trained model to make predictions
test_predictions = LR_model.predict(X_test_data)
```

## 4.6 Evaluate model overall performance via Kaggle metric score

- Create a prediction based on the test dataset
- Create a submission file in csv
- Submit your predictions on Kaggle competition

In [32]:

```
# Create a DataFrame with PassengerId and corresponding predictions
submission_df = pd.DataFrame({'PassengerId': test_data['PassengerId'], 'Survived': tes

# Save the predictions to a CSV file
submission_df.to_csv('submission.csv', index=False)

submission_df.sample(7)
```

Out[32]:

| | PassengerId | Survived |
|---|---|---|
| **127** | 1019 | 0 |
| **53** | 945 | 1 |
| **43** | 935 | 1 |
| **275** | 1167 | 1 |
| **143** | 1035 | 0 |
| **69** | 961 | 1 |
| **391** | 1283 | 1 |

## 4.6 INFERENCE

- Upon submitting my **submission.csv** file on Kaggle competion, I got a metric score of 0.76555.
- This implies that my model got approximately **76.56%** predictions correctly
- THAT's A BIT IMPRESSING