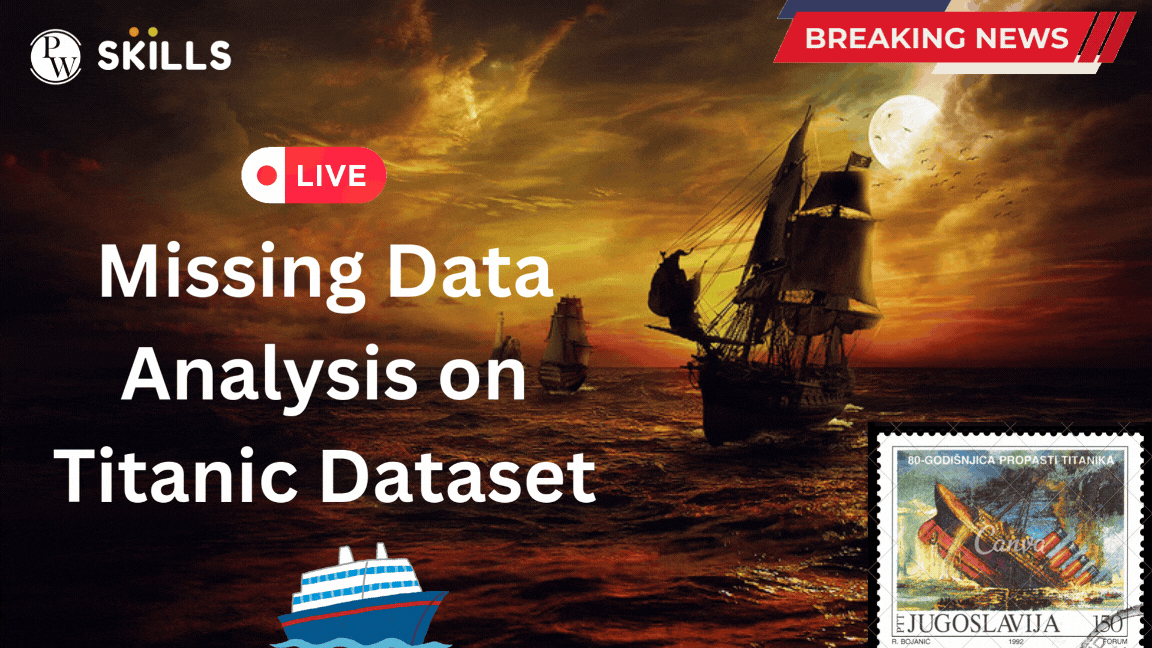
**Missing Data Analysis on Titanic Dataset**



# **Table of Content:**

| **Section No** | **Title** |
| --- | --- |
| 1 | Project Overview & Objective |
| 2 | Dataset Overview & Description |
| 3 | Workflow |
| 4 | Code Explanations |
| 5 | Project Setup |
| 6 | Future Enhancements |
| 7 | Conclusion |

## 

## **1. Project Overview**

The “**Missing Data Analysis on Titanic Dataset”** project is designed to address one of the most common and challenging issues in data science—missing data. Using the Titanic dataset as a case study, the project examines the impact of incomplete data on model performance and demonstrates best practices in data preprocessing. The Titanic dataset, renowned for its rich mix of passenger demographics and survival outcomes, provides an ideal environment to explore the underlying reasons for missing entries. This project systematically investigates these inconsistencies, applies a range of imputation techniques, and evaluates how different strategies affect the overall predictive accuracy of machine learning models. Ultimately, the initiative aims to establish a robust framework that can be adapted to other real-world datasets where missing information is prevalent, ensuring that subsequent analyses and models are both reliable and actionable.



## **Project Objective**

The primary objective of this project is to conduct an exhaustive analysis of missing data within the Titanic dataset and to develop a methodical approach for handling these gaps in order to enhance model performance. The specific goals include:

1. **Identification and Analysis of Missing Data Patterns:**
   * Perform an in-depth exploratory data analysis (EDA) to quantify missing values and identify their distribution across key features.
   * Categorize the missing data into distinct types—Missing Completely at Random (MCAR), Missing at Random (MAR), and Missing Not at Random (MNAR)—to understand the underlying causes and potential biases.
2. **Implementation of Data Imputation Techniques:**
   * Compare a variety of imputation methods ranging from simple statistical approaches (e.g., mean, median, and mode imputation) to more sophisticated predictive techniques such as K-Nearest Neighbors (KNN) and regression-based imputation.
   * Evaluate each method in terms of its impact on the dataset’s integrity and the preservation of its statistical properties.
3. **Assessment of Impact on Predictive Modeling:**
   * Establish a baseline by training machine learning models on the original dataset with missing values.
   * Re-train the models after applying different imputation techniques, and measure performance improvements using metrics such as accuracy, precision, recall, and F1-score.
   * Analyze how the treatment of missing data influences the bias-variance tradeoff, thereby ensuring the model’s robustness and generalizability.
4. **Development of Best Practices and Workflow:**
   * Construct a comprehensive, step-by-step data preprocessing workflow that addresses missing data challenges effectively.
   * Document the methods, insights, and comparative outcomes to provide actionable recommendations for future projects involving incomplete datasets.

## **2. Dataset Overview**

The Titanic dataset is a historically significant and widely used dataset in data science and machine learning. It contains detailed records of passengers aboard the RMS Titanic, a ship that tragically sank in April 1912. The dataset is particularly useful for predictive modeling, classification problems, and exploratory data analysis, making it a fundamental resource for learning and research in data science.

This dataset includes demographic, ticketing, and survival information for individual passengers, enabling a deep analysis of survival patterns based on socio-economic status, age, gender, and other relevant factors. The dataset is particularly valuable for studying missing data challenges, as several key features contain incomplete entries, such as passenger age and cabin details. Addressing these missing values effectively is essential for ensuring accurate model predictions.

The Titanic dataset serves as an ideal case study for understanding data preprocessing, handling missing data, and applying machine learning techniques to real-world problems.

### **Dataset Details & Feature Descriptions**

The dataset comprises multiple variables that provide detailed insights into the passengers, their travel arrangements, and survival status. Below is a structured breakdown of the key features:

#### **1. Survival Information**

* **Survived (Survived)**
  + Represents whether a passenger survived the Titanic disaster.
  + **Values:**
    - 0 → Did not survive
    - 1 → Survived
  + This serves as the target variable in classification models.

#### **2. Passenger Demographics**

* **Passenger Name (Name)**
  + Full name of the passenger, often including title information, which can be used to infer social status and gender.
* **Sex (Sex)**
  + The gender of the passenger.
  + **Values:**
    - Male
    - Female
  + This feature is historically significant as gender played a crucial role in survival likelihood.
* **Age (Age)**
  + Age of the passenger at the time of boarding.
  + **Missing Values:** Some entries contain NaN, requiring imputation.
  + Age distributions are often crucial for survival analysis, as children and women were given priority for lifeboats.

#### **3. Socio-Economic Information**

* **Passenger Class (Pclass)**
  + Represents the socio-economic class of the passenger.
  + **Values:**
    - 1 → Upper class (First Class)
    - 2 → Middle class (Second Class)
    - 3 → Lower class (Third Class)
  + Passenger class had a significant impact on survival rates, with first-class passengers having higher chances of survival due to better access to lifeboats.

#### **4. Family Relations Onboard**

* **Number of Siblings/Spouses Aboard (SibSp)**
  + Indicates the number of siblings or spouses that the passenger was traveling with.
  + A higher number may indicate larger family groups, affecting survival strategy.
* **Number of Parents/Children Aboard (Parch)**
  + Represents the number of parents or children accompanying the passenger.
  + Like SibSp, this feature helps determine family structures and group survival tendencies.

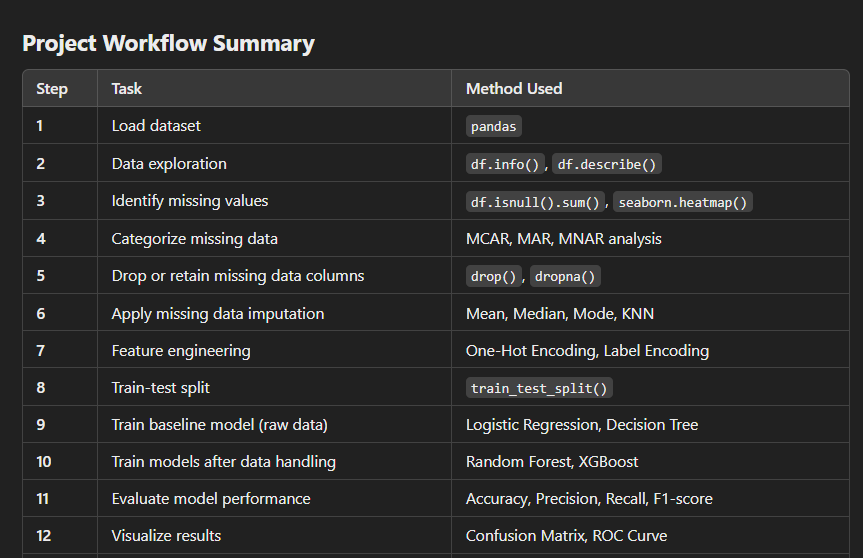
#### **5. Travel & Ticket Information**

* **Ticket Number (Ticket)**
  + The ticket identification number assigned to the passenger.
  + May provide insights into groups traveling together or ticket pricing patterns.
* **Fare (Fare)**
  + The price paid for the ticket.
  + Directly correlates with Pclass since first-class passengers generally paid higher fares.
  + This feature is useful for detecting potential anomalies or patterns in socio-economic influences on survival.

#### **6. Accommodation & Embarkation**

* **Cabin (Cabin)**
  + The assigned cabin number for the passenger.
  + **Missing Values:** A large proportion of entries are missing, making it a major challenge in data analysis.
  + Can be used to infer passenger location on the ship, potentially affecting survival chances.
* **Port of Embarkation (Embarked)**
  + Indicates the port from which the passenger boarded the Titanic.
  + **Values:**
    - C → Cherbourg
    - Q → Queenstown
    - S → Southampton
  + The port of embarkation may provide insights into regional economic backgrounds and travel patterns.

## **3.Project Workflow:**



### **3.1. Data Acquisition & Exploration**

* Load the Titanic dataset using pandas.
* Display dataset structure (df.info()) and summary statistics (df.describe()).
* Identify missing values using df.isnull().sum().
* Visualize missing data patterns (seaborn.heatmap() or missingno library).

### **3.2. Missing Data Analysis & Categorization**

* **Classify missing values**:
  + **MCAR** (random missingness).
  + **MAR** (depends on other variables).
  + **MNAR** (missing due to its own nature).
* Determine the impact of missing data on analysis.

### **3.3. Handling Missing Data**

* **Drop columns** if missing values are excessive (e.g., Cabin).
* **Apply Imputation Techniques**:
  + **Numerical** (Age, Fare): Mean, Median, KNN Imputation.
  + **Categorical** (Embarked): Mode or “Unknown” category.
* Encode categorical variables (Sex, Embarked, Pclass) using Label Encoding or One-Hot Encoding.
* Create **new features** (e.g., FamilySize = SibSp + Parch + 1).

### **3.4. Data Splitting & Model Training**

* Split data into **Training & Testing sets** (train\_test\_split()).
* Train a **baseline model** (e.g., Logistic Regression, Decision Tree) on unprocessed data.
* Train models **after handling missing data** to measure improvements.

### **3.5. Model Evaluation & Impact Analysis**

* Evaluate model performance using **accuracy, precision, recall, and F1-score**.
* Visualize results with **confusion matrices** and **ROC curves**.
* Compare model performance **before and after handling missing data**.

### **3.6. Reporting & Recommendations**

* Document findings and **compare imputation methods**.
* Summarize the **best approach** for handling missing data.
* Provide **recommendations** for real-world dataset preprocessing.

## **4. Code Explanation:**

| import pandas as pd import seaborn as sns import numpy as np import matplotlib.pyplot as plt from statistics import mean import warnings warnings.filterwarnings("ignore") %matplotlib inline |
| --- |

**Explanation:**

Importing Libraries

1. import pandas as pd

- Import the pandas library and assign it the alias 'pd' for convenience.

- Pandas is a popular library for data manipulation and analysis.

2. import seaborn as sns

- Import the seaborn library and assign it the alias 'sns'.

- Seaborn is a visualization library built on top of matplotlib.

3. import numpy as np

- Import the numpy library and assign it the alias 'np'.

- NumPy is a library for efficient numerical computation.

4. import matplotlib.pyplot as plt

- Import the matplotlib.pyplot module and assign it the alias 'plt'.

- Matplotlib is a popular data visualization library.

5. from statistics import mean

- Import the mean function from the statistics module.

- This function calculates the arithmetic mean of a dataset.

Configuring Environment

6. import warnings

- Import the warnings module.

- This module allows you to control warnings issued by Python.

7. warnings.filterwarnings("ignore")

- Filter out all warnings issued by Python.

- This is often used to suppress unnecessary warnings.

8. %matplotlib inline

- A magic command in Jupyter Notebooks to display matplotlib plots inline.

- This allows you to visualize plots directly within the notebook.

**Loading the Titanic Dataset**

| titanic\_df = pd.read\_csv("titanic.csv", na\_values="na") |
| --- |

- Use the read\_csv function from pandas to load the Titanic dataset from a CSV file named "titanic.csv".

- The na\_values parameter specifies that any occurrence of the string "na" should be treated as a missing or null value.

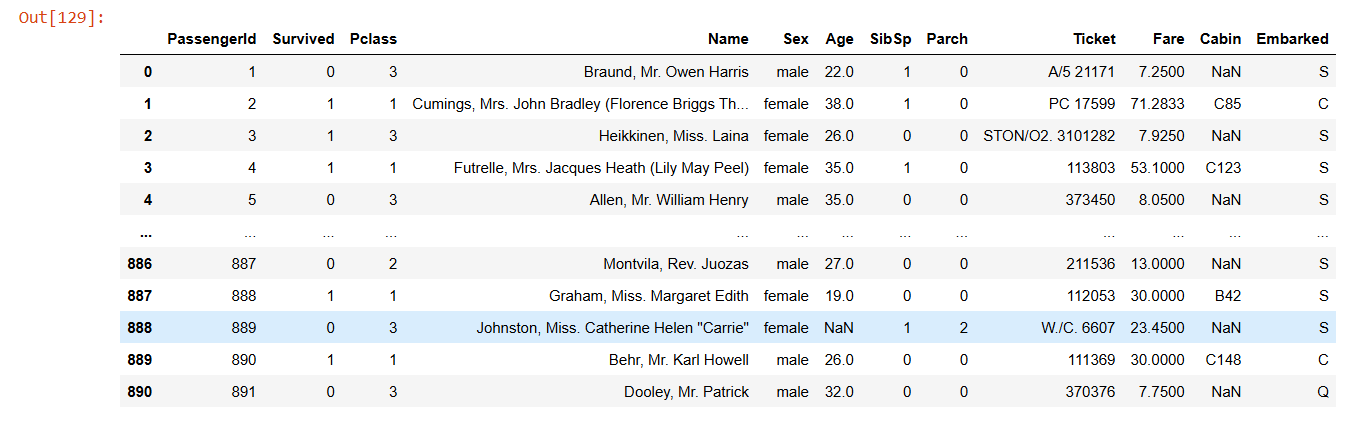
- The loaded dataset is assigned to a DataFrame object named titanic\_df.

**Displaying the Loaded DataFrame**

| titanic\_df |
| --- |

- Simply typing the name of the DataFrame object (titanic\_df) will display the first few rows of the loaded dataset in a Jupyter Notebook.

- This allows you to quickly verify that the data has been loaded correctly.



| # Column names titanic\_df.columns |
| --- |

Column Names

titanic\_df.columns displays the column names of the DataFrame.

Index

The column names are indexed, meaning each column has a corresponding integer index.

Column Names

Here are the 12 column names:

| 1. PassengerId 2. Survived 3. Pclass 4. Name 5. Sex 6. Age 7. SibSp 8. Parch 9. Ticket 10. Fare 11. Cabin 12. Embarked |
| --- |

Data Type

dtype='object' indicates that all column names are stored as objects (i.e., strings).

**DataFrame Information**

| titanic\_df.info() #displays information about the DataFrame. |
| --- |

| <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 |
| --- |

DataFrame Details

- Class: <class 'pandas.core.frame.DataFrame'> (a pandas DataFrame)

- Index: RangeIndex: 891 entries, 0 to 890 (891 rows, indexed from 0 to 890)

| # Check the shape of the data titanic\_df.shape |
| --- |

DataFrame Shape

titanic\_df.shape returns the shape of the DataFrame as a tuple:

| Output (891, 12) |
| --- |

| # Check unique values of target variable (Survived) titanic\_df['Survived'].value\_counts() |
| --- |

Target Variable Distribution

titanic\_df['Survived'].value\_counts() displays the distribution of the target variable (Survived):

Output

| Survived 0 549 1 342 |
| --- |

Name: Survived, dtype: int64

Interpretation

- 0 (No) represents passengers who did not survive (549 passengers)

- 1 (Yes) represents passengers who survived (342 passengers)

Class Balance

The dataset is approximately balanced, with a ratio of:

62% (No) : 38% (Yes)

This indicates that the dataset has a relatively balanced distribution of positive and negative classes.

## **Exploratory Data Analysis (EDA):**

## **3.1 Passenger Class Distribution**

| print("\nPassenger counts by Class:") print(titanic\_df.groupby('Pclass')['Pclass'].count()) ​ |
| --- |

Passenger Class Distribution

The code groups the Titanic dataset by Pclass and counts the number of passengers in each class:

Output

| Passenger counts by Class: Pclass 1 216 2 184 3 491 Name: Pclass, dtype: int64 |
| --- |

Interpretation

- First Class (Pclass 1): 216 passengers

- Second Class (Pclass 2): 184 passengers

- Third Class (Pclass 3): 491 passengers

This shows that the majority of passengers (55%) were in Third Class, followed by First Class (24%) and Second Class (21%).

| # Visualize the count of passengers in each class sns.countplot(x='Pclass', data=titanic\_df, palette='pastel') plt.xlabel('Passenger Class') plt.title('Count of Passengers by Class') plt.show() |
| --- |

**Explanation**

This code uses seaborn and matplotlib to create a bar plot showing the count of passengers by class.

**1. sns.countplot():** This function creates a bar plot showing the count of observations in each category.

**- x='Pclass':** This specifies that the x-axis should show the different passenger classes (Pclass).

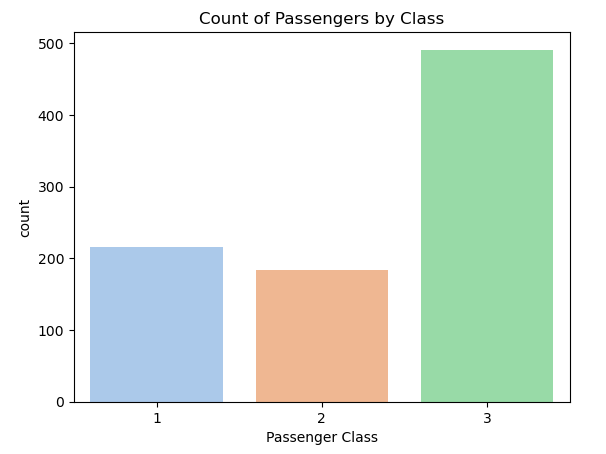
**- data=titanic\_df:** This specifies that the data should come from the titanic\_df DataFrame.

- **palette='pastel':** This specifies that the plot should use a pastel color scheme.

**2. plt.xlabel('Passenger Class'):** This adds a label to the x-axis, specifying that it shows the passenger class.

**3. plt.title('Count of Passengers by Class'):** This adds a title to the plot, specifying that it shows the count of passengers by class.

**4. plt.show():** This displays the plot.



* Grouping by Pclass revealed that the majority of passengers were in third class.
* It clearly showing fewer first and second class passengers compared to third class.

## **3.2 Gender Distribution**

| print("\nPassenger counts by Gender:") print(titanic\_df.groupby('Sex')['Sex'].count()) ​ |
| --- |

The code groups the Titanic dataset by Sex and counts the number of passengers in each gender category:

Output

| Passenger counts by Gender: Sex female 314 male 577 Name: Sex, dtype: int64 |
| --- |

Interpretation

- Female passengers: 314 (35% of total passengers)

- Male passengers: 577 (65% of total passengers)

**This shows that the majority of passengers (65%) were male, while 35% were female.**

| sns.countplot(x='Sex', data=titanic\_df, palette='muted') plt.xlabel('Gender') plt.title('Count of Passengers by Gender') plt.show() |
| --- |

**Explanation**

**1. sns.countplot():** Creates a bar plot showing the count of observations in each category.

- x='Sex': Specifies that the x-axis should show the different genders (Sex).

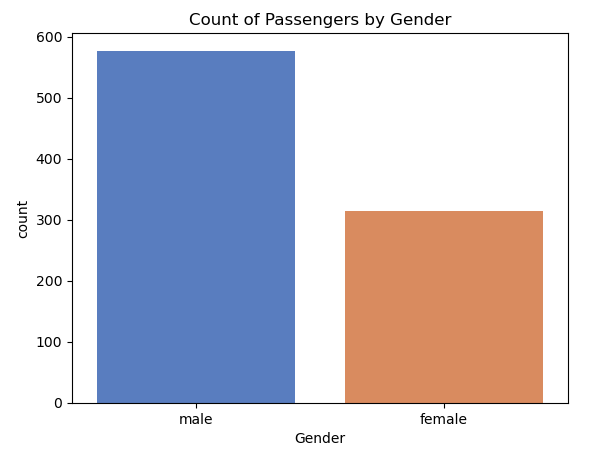
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- palette='muted': Specifies that the plot should use a muted color scheme.

**2. plt.xlabel('Gender'):** Adds a label to the x-axis, specifying that it shows the gender.

**3. plt.title('Count of Passengers by Gender'):** Adds a title to the plot, specifying that it shows the count of passengers by gender.

**4. plt.show():** Displays the plot.



* Analysis by the Sex column indicated that there were nearly twice as many male passengers as female.

## 

## **3.3 Class Distribution Grouped by Gender**

| sns.countplot(x='Pclass', hue='Sex', data=titanic\_df, palette='Set2') plt.xlabel('Passenger Class') plt.title('Passenger Class Distribution by Gender') plt.show() |
| --- |

**Explanation**

**1. sns.countplot():** Creates a bar plot showing the count of observations in each category.

- x='Pclass': Specifies that the x-axis should show the different passenger classes (Pclass).

- hue='Sex': Specifies that the plot should be colored by gender (Sex).

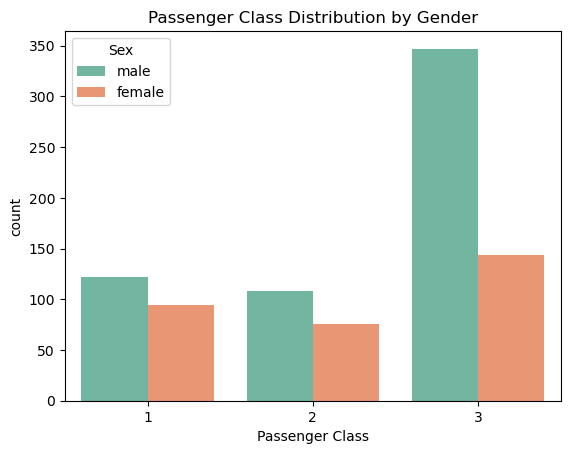
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- palette='Set2': Specifies that the plot should use a predefined color palette.

**2. plt.xlabel('Passenger Class'):** Adds a label to the x-axis, specifying that it shows the passenger class.

**3. plt.title('Passenger Class Distribution by Gender'):** Adds a title to the plot, specifying that it shows the passenger class distribution by gender.

**4. plt.show():** Displays the plot.



* When further segmented by class, the distribution was nearly balanced in first and second classes, but third class had a heavy male predominance.

### **Key Insight:**

* The class and gender distributions suggest that socioeconomic status (inferred from class) and gender might play a significant role in understanding the survival dynamics on board.

## **3.4 Overall Survival Distribution**​

| sns.countplot(x='Survived', data=titanic\_df, palette='coolwarm') plt.xlabel('Survival (0 = No, 1 = Yes)') plt.title('Survivors vs. Non-Survivors') plt.show() |
| --- |

**Explanation**

**1. sns.countplot():** Creates a bar plot showing the count of observations in each category.

- x='Survived': Specifies that the x-axis should show the survival status (0 = No, 1 = Yes).

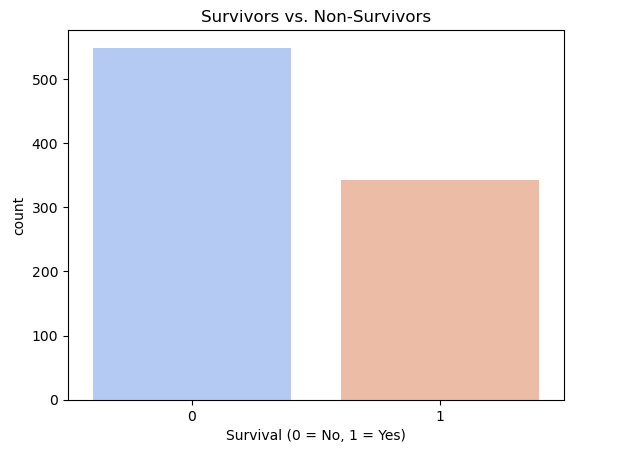
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- palette='coolwarm': Specifies that the plot should use a cool-warm color palette.

**2. plt.xlabel('Survival (0 = No, 1 = Yes)'):** Adds a label to the x-axis, specifying that it shows the survival status.

**3. plt.title('Survivors vs. Non-Survivors'):** Adds a title to the plot, specifying that it shows the comparison between survivors and non-survivors.

**4. plt.show():** Displays the plot.



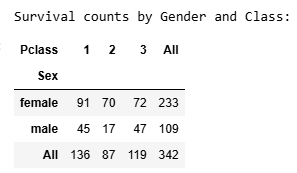
* It showed the overall split between survivors and non-survivors, with a noticeable imbalance.

## **3.5 Survival Analysis by Gender and Class via Pivot Table**

In [140]:

​

| survival\_pivot = titanic\_df.pivot\_table('Survived', index='Sex', columns='Pclass', aggfunc=np.sum, margins=True) print("\nSurvival counts by Gender and Class:") survival\_pivot |
| --- |



## **🎯 Survival by Class & Gender**

### **📊 Key Findings from Pivot Tables & Cross-Tabulations:**

1️⃣ **First-Class Advantage:**

* Higher survival rates, confirming **better access to lifeboats & resources**.

2️⃣ **Third-Class Struggles:**

* Lower survival rates, **compounded by a male-dominated passenger base**.
* Possible barriers in **location & accessibility** on the ship.

3️⃣ **Complex Interplay of Factors:**

* **Embarkation Port Matters:** Survival outcomes may vary based on **where passengers boarded**.
* **Gender & Class Interactions:** Females in **first class had the highest survival rates**, while third-class males faced **the lowest odds**.

### **🔎 Why This Matters:**

* These patterns emphasize **historical social norms & structural inequalities**.
* Understanding **multi-variable dependencies** is crucial for refining survival prediction models.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

## **🔍 Key Insight: Survival Disparities**

### **🚢 Class & Gender Influence on Survival:**

* **First-Class Advantage:** Higher survival rates, likely due to better access to lifeboats.
* **Gender Disparity:** Female passengers had significantly higher survival odds.
* **Third-Class Struggles:** Lower survival rates, possibly due to restricted access and location on the ship.

### **📌 Why It Matters:**

* These trends reflect **historical social norms** ("women and children first").
* **Class-based survival gaps** highlight structural inequalities in emergency response.
* Helps refine predictive models by incorporating **class and gender interactions** as key survival determinants.

## **4. Age Analysis**

| # Display basic age statistics print("\nAge Statistics:") print("Average Age: {:.0f} years".format(titanic\_df.Age.mean())) print("Median Age: {:.0f} years".format(titanic\_df.Age.median())) print(titanic\_df.Age.describe()) |
| --- |

| **Age Statistics: Average Age: 30 years Median Age: 28 years count 714.000000 mean 29.699118 std 14.526497 min 0.420000 25% 20.125000 50% 28.000000 75% 38.000000 max 80.000000 Name: Age, dtype: float64** |
| --- |

The output displays various statistics about the age of passengers:

### 

### **📊 Age Statistics Summary**

Average and Median Age

- Average Age: 30 years

- Median Age: 28 years

Summary Statistics

The output also displays a summary of age statistics:

- Count: 714 (number of passengers with age data)

- Mean: 29.70 years (average age)

- Standard Deviation (std): 14.53 years (spread of ages)

- Minimum (min): 0.42 years (youngest passenger)

- 25th Percentile (25%): 20.13 years (age below which 25% of passengers fall)

- Median (50%): 28.00 years (age below which 50% of passengers fall)

- 75th Percentile (75%): 38.00 years (age below which 75% of passengers fall)

- Maximum (max): 80.00 years (oldest passenger)

### **📌 Key Measures:**

* **Mean Age:** Provides the average passenger age.
* **Median Age:** Indicates the central age value, less affected by outliers.
* **Range:** Highlights the youngest and oldest passengers.
* **Interquartile Range (IQR):** Shows middle 50% of ages.

### **🔍 Key Insights:**

* **Broad Age Distribution:** Passengers span across a wide range of ages.
* **Median vs. Mean:** Any significant difference suggests skewness.
* **Variability in Age:** Essential for segmenting passengers into meaningful age groups (e.g., children, young adults, seniors).

### **📌 Why It Matters:**

* These statistics **guide feature engineering** (e.g., categorizing age groups).
* **Baseline for missing age imputation**, ensuring logical age distributions.

| # Plot the distribution of passenger ages (excluding missing values) age\_data = titanic\_df['Age'].dropna() sns.distplot(age\_data, kde=True, bins=50, color='skyblue') plt.title("Distribution of Passenger Ages") plt.xlabel('Age') plt.ylabel('Density') plt.show() |
| --- |

**Explanation**

**1. age\_data = titanic\_df['Age'].dropna():** Creates a new variable age\_data containing the age data from the titanic\_df DataFrame, excluding missing values (NaN).

**2. sns.distplot():** Creates a histogram with a kernel density estimate (KDE) using seaborn's distplot function.

- age\_data: Specifies the data to plot.

- kde=True: Includes a kernel density estimate (KDE) in the plot.

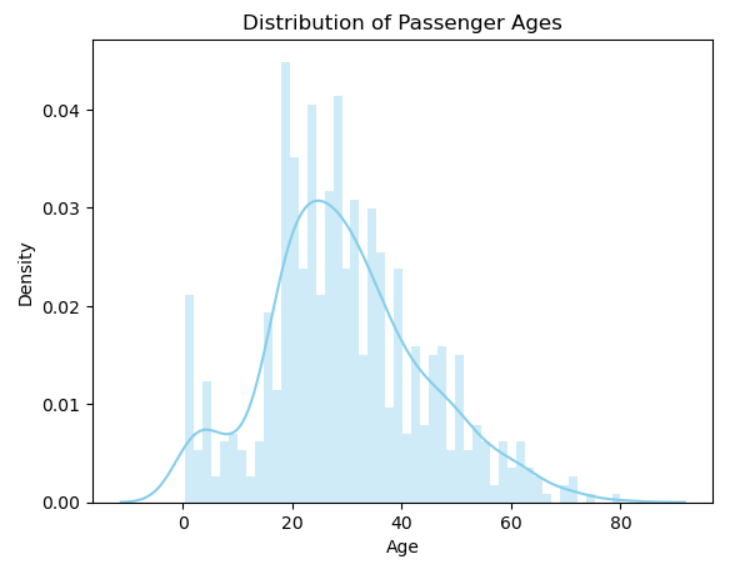
- bins=50: Specifies the number of bins for the histogram.

- color='skyblue': Sets the color of the plot to sky blue.

**3. plt.title():** Adds a title to the plot.

**4. plt.xlabel() and plt.ylabel():** Add labels to the x-axis and y-axis, respectively.

**5. plt.show():** Displays the plot.



## **📊 Age Distribution Analysis**

* **Visualization Techniques:**
  + 📉 **Density Plots** (Seaborn’s distplot)
  + 📊 **Histograms**

### **🔍 Key Observations:**

* **Continuous Age Spread:** Passengers range across all age groups.
* **Density Peaks:**
  + 👶 **Children:** Lower but noticeable representation.
  + 🎓 **Young Adults (18-30):** Most densely populated group.
  + 👵 **Older Adults:** Gradual decline in frequency.

### **📌 Why It Matters:**

* Understanding **age clusters** helps refine survival analysis.
* Identifies **key demographic groups** that shaped evacuation patterns.

## **🔑 Key Insight:**

* 📊 **Age is a Critical Factor in Survival**
* 🔄 **Continuous Age Distribution** Enables Further Segmentation:
  + 👶 **Children vs. Adults:** Helps uncover survival trends.
  + 📈 **Fine-Grained Analysis:** Identifies patterns beyond simple age averages.
* 🧐 **Why It Matters:**
  + Provides **deeper insights** into survival likelihood across different age groups.
  + Highlights **potential prioritization** of certain age segments during evacuation.

## **5. Passenger Categorization: Child vs. Adult**

| def classify\_passenger(row):  """  Categorize passenger based on age.  If age < 16, classify as 'child'; otherwise, retain the gender.  """  return 'child' if row['Age'] < 16 else row['Sex']  # Create a new column 'person' using the custom classification function titanic\_df['person'] = titanic\_df.apply(classify\_passenger, axis=1) print("\nSample of passenger classification (Age, Sex, person):") print(titanic\_df[['Age', 'Sex', 'person']].head(10)) |
| --- |

The Output:

| Sample of passenger classification (Age, Sex, person):  Age Sex person 0 22.0 male male 1 38.0 female female 2 26.0 female female 3 35.0 female female 4 35.0 male male 5 NaN male male 6 54.0 male male 7 2.0 male child 8 27.0 female female 9 14.0 female child |
| --- |

**Explanation**

**1. def classify\_passenger(row)::** Defines a custom function classify\_passenger that takes a row of the DataFrame as input.

**2. return 'child' if row['Age'] < 16 else row['Sex']:** Classifies passengers as 'child' if their age is less than 16; otherwise, retains their original gender.

**3. titanic\_df['person'] = titanic\_df.apply(classify\_passenger, axis=1):** Applies the classify\_passenger function to each row of the titanic\_df DataFrame, creating a new column 'person' with the classification results.

**4. print(titanic\_df[['Age', 'Sex', 'person']].head(10)):** Prints a sample of 10 rows from the titanic\_df DataFrame, showing the 'Age', 'Sex', and 'person' columns.

### **Categorizing Passengers: Child vs. Adult**

### **🔹 Creating a New Category:**

* 🛠 **Custom Function Applied:**
  + **New Column (person) Created:**
    - 👶 **"Child"** → Passengers under **16 years old**.
    - 🚹🚺 **"Male"/"Female"** → Passengers **16 or older** retained their gender label.

🔍 **Why It Matters:**This categorization **enhances analysis** by capturing **age-gender interactions** in survival rates.

### **Visualize distribution by class and the new passenger categories:**

| sns.countplot(x='Pclass', hue='person', data=titanic\_df, order=[1,2,3],  hue\_order=['child', 'female', 'male'], palette='Set1') plt.xlabel('Passenger Class') plt.title('Passenger Distribution by Class and Category') plt.show() |
| --- |

**Explanation**

1. sns.countplot(): Creates a bar plot showing the count of observations in each category.

- x='Pclass': Specifies that the x-axis should show the passenger class.

- hue='person': Specifies that the plot should be colored by the 'person' category (child, female, male).

- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- order=[1,2,3]: Specifies the order of the passenger classes on the x-axis.

- hue\_order=['child', 'female', 'male']: Specifies the order of the 'person' categories in the legend.

- palette='Set1': Specifies that the plot should use a predefined color palette.

2. plt.xlabel(): Adds a label to the x-axis.

3. plt.title(): Adds a title to the plot.

4. plt.show(): Displays the plot.

## 

## **📊 Distribution Findings:**

* **Class & Age Relationship:**
  + 🏷 **Third Class:** Had a **higher proportion of children** than upper classes.
  + 🎩 **First & Second Class:** Showed a **more balanced gender distribution**.

🔍 **Implication:**Class structure **influenced passenger demographics**, which in turn **affected survival probabilities**.

### 👶 **Key Insight: The Role of Children in Survival**

* **Children in Third Class:** Their presence may have influenced **survival strategies** and decisions.
* **Targeted Analysis Needed:** Exploring **age-specific survival rates** can uncover **potential prioritization patterns**.

🔍 **Why It Matters:**Understanding how **age and class intersect** helps explain variations in survival odds, particularly for **families and young passengers**.

### **6.1 Survival Analysis: Age vs. Survival Trends**

| sns.lmplot(x='Age', y='Survived', data=titanic\_df, aspect=1.5, scatter\_kws={'alpha':0.5}) plt.title("Linear Relationship between Age and Survival") plt.show() |
| --- |

**Explanation**

1. sns.lmplot(): Creates a scatter plot with a linear regression line.

- x='Age': Specifies that the x-axis should show the age.

- y='Survived': Specifies that the y-axis should show the survival status (0 = No, 1 = Yes).

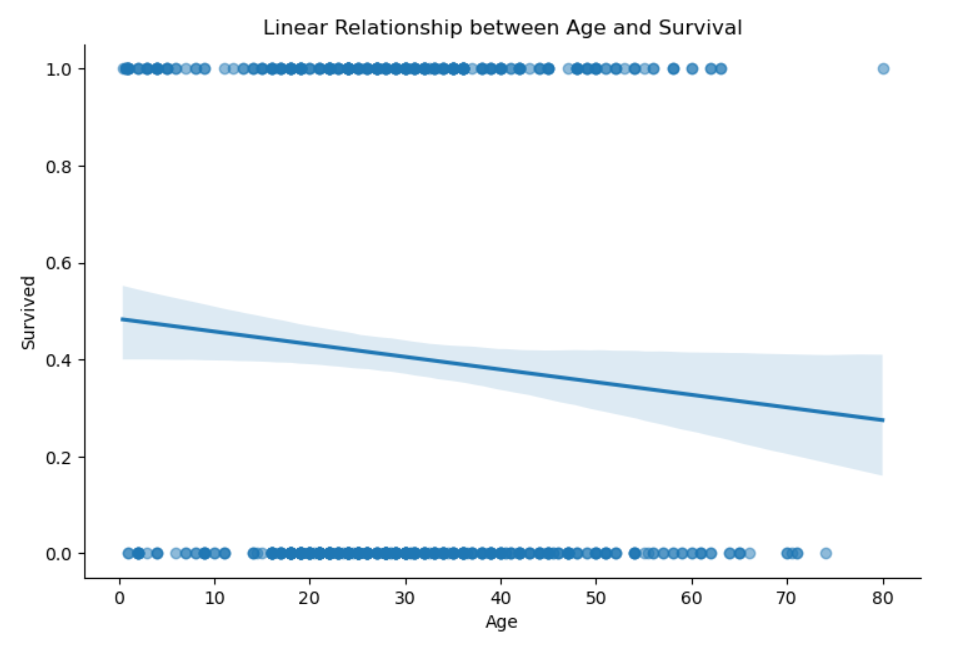
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- aspect=1.5: Specifies the aspect ratio of the plot.

- scatter\_kws={'alpha':0.5}: Specifies the transparency of the scatter points (alpha = 0.5 means 50% transparency).

2. plt.title(): Adds a title to the plot.

3. plt.show(): Displays the plot.



## **📉 The Age Factor in Survival**

* **Overall Trend:** A **gentle decline** in survival probability as age increases.
* **Exceptions Exist:** Some older passengers did survive, but the **negative slope** indicates that, on average, **older individuals had lower survival odds**.

🔍 **Insight:**Age played a subtle yet consistent role—**younger passengers had a survival advantage**, reinforcing the broader trend observed across different passenger groups.

## **6.2 Age vs. Survival grouped by Gender**

| sns.lmplot(x='Age', y='Survived', hue='Sex', data=titanic\_df, aspect=1.5, scatter\_kws={'alpha':0.5}) plt.title("Age vs. Survival Grouped by Gender") plt.show() |
| --- |

**Explanation**

1. sns.lmplot(): Creates a scatter plot with a linear regression line.

- x='Age': Specifies that the x-axis should show the age.

- y='Survived': Specifies that the y-axis should show the survival status (0 = No, 1 = Yes).

- hue='Sex': Specifies that the plot should be colored by gender (male or female).

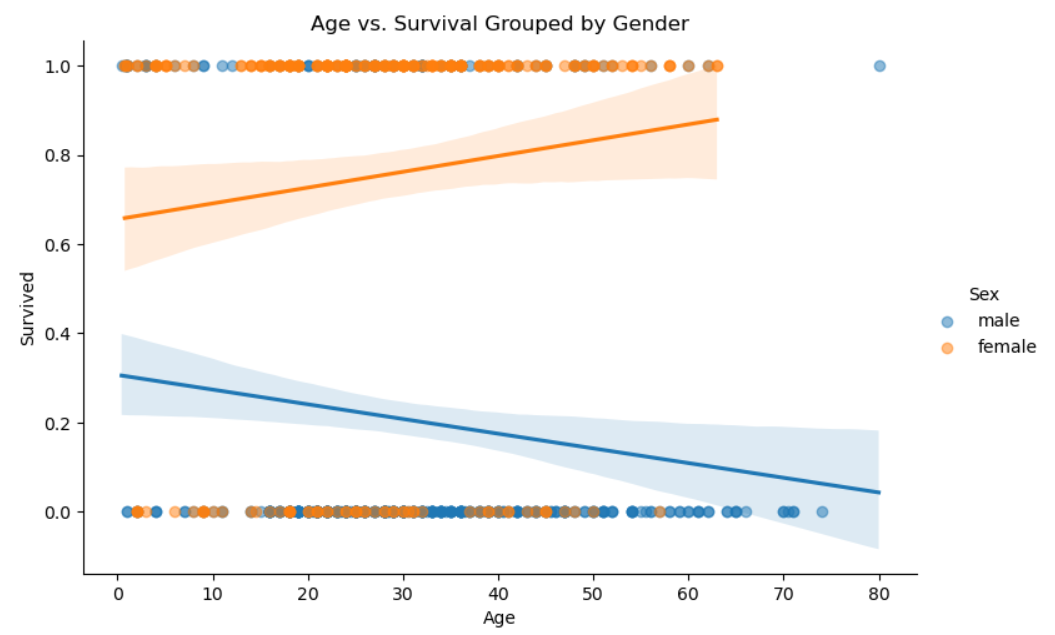
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- aspect=1.5: Specifies the aspect ratio of the plot.

- scatter\_kws={'alpha':0.5}: Specifies the transparency of the scatter points (alpha = 0.5 means 50% transparency).

2. plt.title(): Adds a title to the plot.

3. plt.show(): Displays the plot.



## **👨‍🦳🚢 Age & Survival: A Gendered Perspective**

* **Men:** A **strong negative correlation** between age and survival—**older males faced lower survival odds**.
* **Women:** **Consistently higher survival rates**, regardless of age, reflecting **preferential rescue efforts**.

🔍 **Key Takeaway:**The **"women and children first"** policy likely played a crucial role, ensuring female passengers had a survival advantage **across all age groups**.

## **6.3 Age vs. Survival grouped by Passenger Class**

| sns.lmplot(x='Age', y='Survived', hue='Pclass', data=titanic\_df, palette='winter', aspect=1.5, scatter\_kws={'alpha':0.5}) plt.title("Age vs. Survival Grouped by Class") plt.show() |
| --- |

Explanation

1. sns.lmplot(): Creates a scatter plot with a linear regression line.

- x='Age': Specifies that the x-axis should show the age.

- y='Survived': Specifies that the y-axis should show the survival status (0 = No, 1 = Yes).

- hue='Pclass': Specifies that the plot should be colored by passenger class (1, 2, or 3).

- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

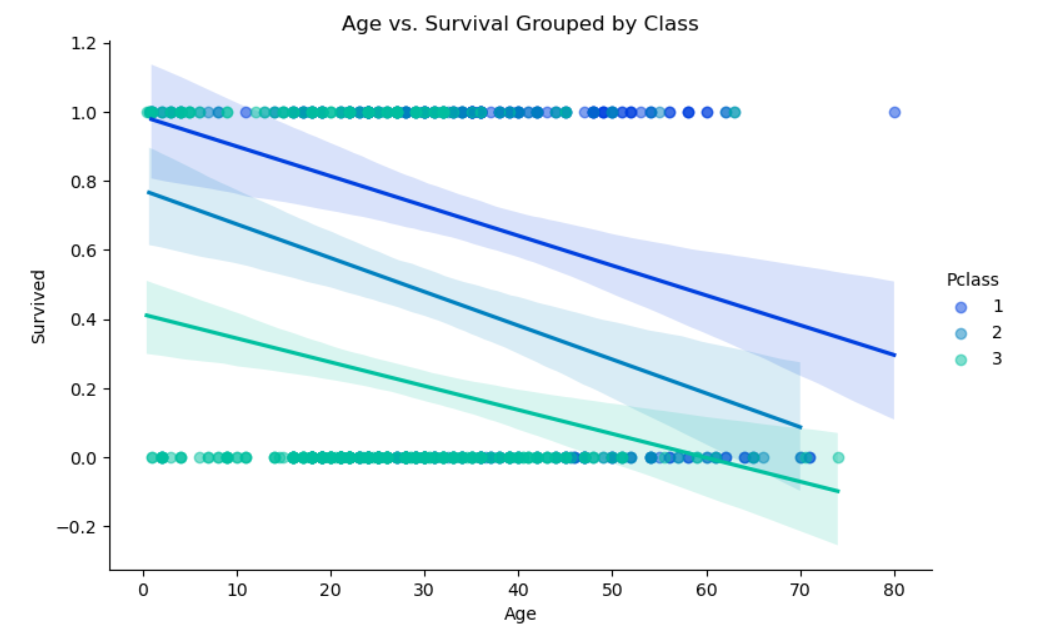
- palette='winter': Specifies that the plot should use a winter-themed color palette.

- aspect=1.5: Specifies the aspect ratio of the plot.

- scatter\_kws={'alpha':0.5}: Specifies the transparency of the scatter points (alpha = 0.5 means 50% transparency).

2. plt.title(): Adds a title to the plot.

3. plt.show(): Displays the plot.



## **📉 Age vs. Survival: The Class Divide**

* **Universal Trend:** Survival rates **decrease with age** across all classes.
* **First-Class Advantage:** Even as age rises, **first-class passengers maintain higher survival probabilities**.
* **Third-Class Struggle:** A **steeper decline** in survival is observed, indicating **lower priority in rescue efforts**.

🔍 **Key Takeaway:**Age wasn't just a number—it interacted with **class-based privilege**, shaping survival outcomes significantly.

## **7. Cabin Analysis: Extracting and Visualizing Deck Information**

| # Extract non-null cabin data and obtain deck letters cabin\_data = titanic\_df['Cabin'].dropna() deck\_letters = [cabin[0] for cabin in cabin\_data] # Count the occurrence of each deck letter from collections import Counter deck\_counter = Counter(deck\_letters) print("\nDeck letter counts from cabin data:") print(deck\_counter) |
| --- |

The Output:

| Deck letter counts from cabin data: Counter({'C': 59, 'B': 47, 'D': 33, 'E': 32, 'A': 15, 'F': 13, 'G': 4, 'T': 1}) |
| --- |

**Explanation**

1. cabin\_data = titanic\_df['Cabin'].dropna(): Extracts non-null cabin data from the titanic\_df DataFrame.

2. deck\_letters = [cabin[0] for cabin in cabin\_data]: Obtains the deck letters from the cabin data by taking the first character of each cabin string.

3. from collections import Counter: Imports the Counter class from the collections module.

4. deck\_counter = Counter(deck\_letters): Creates a Counter object to count the occurrence of each deck letter.

5. print(deck\_counter): Prints the deck letter counts.

| # Create a DataFrame for deck analysis and visualize deck distribution cabin\_df = pd.DataFrame(deck\_letters, columns=['Deck']) sns.countplot(x='Deck', data=cabin\_df, order=['A','B','C','D','E','F','G','T'], palette='winter') plt.title("Distribution of Cabin Decks") plt.show() |
| --- |

**Explanation**

1. cabin\_df = pd.DataFrame(deck\_letters, columns=['Deck']): Creates a new DataFrame cabin\_df with a single column 'Deck' containing the deck letters.

2. sns.countplot(): Creates a count plot showing the distribution of cabin decks.

- x='Deck': Specifies that the x-axis should show the deck letters.

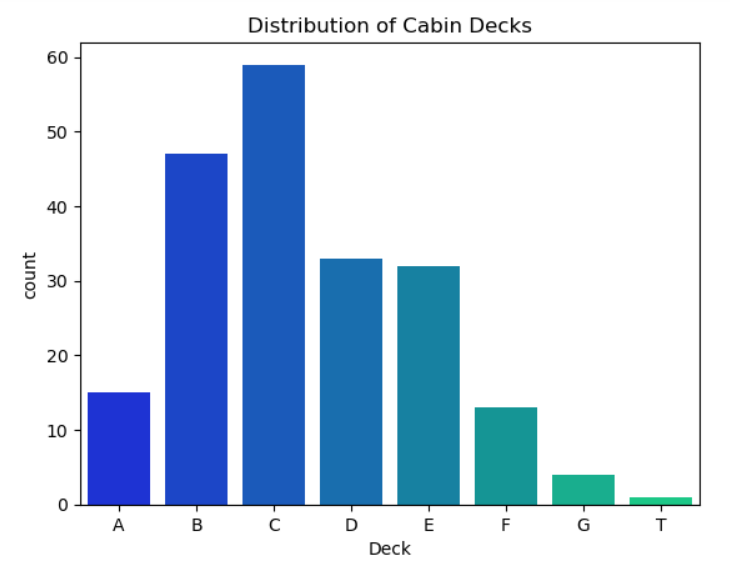
- data=cabin\_df: Specifies that the data should come from the cabin\_df DataFrame.

- order=['A','B','C','D','E','F','G','T']: Specifies the order of the deck letters on the x-axis.

- palette='winter': Specifies that the plot should use a winter-themed color palette.

3. plt.title(): Adds a title to the plot.

4. plt.show(): Displays the plot.



## **🚢 Cabin Data & Deck Extraction**

* **Handling Missing Data:** Since many cabin entries were missing, we extracted **deck letters** from available data.
* **Deck Distribution:** Most passengers were assigned to **decks A-G**, with a rare 'T' deck entry.
* **Insights:**
  + **First-class passengers** were mostly on **upper decks (A-C)**.
  + **Lower-class passengers** were concentrated in **decks E-G**.
  + The anomalous 'T' deck suggests **potential misclassification or rare assignment**.

📊 **Why It Matters?**Deck location provided **spatial context**, influencing **accessibility to lifeboats** and **evacuation times**.

| # Exclude anomalous 'T' entries and replot the deck distribution cabin\_df = cabin\_df[cabin\_df['Deck'] != 'T'] sns.countplot(x='Deck', data=cabin\_df, order=['A','B','C','D','E','F','G'], palette='Greens\_d') plt.title("Cabin Deck Distribution (Excluding 'T')") plt.show() |
| --- |

**Explanation**

1. cabin\_df = cabin\_df[cabin\_df['Deck'] != 'T']: Excludes the rows from the cabin\_df DataFrame where the 'Deck' column is 'T'.

2. sns.countplot(): Creates a count plot showing the distribution of cabin decks.

- x='Deck': Specifies that the x-axis should show the deck letters.

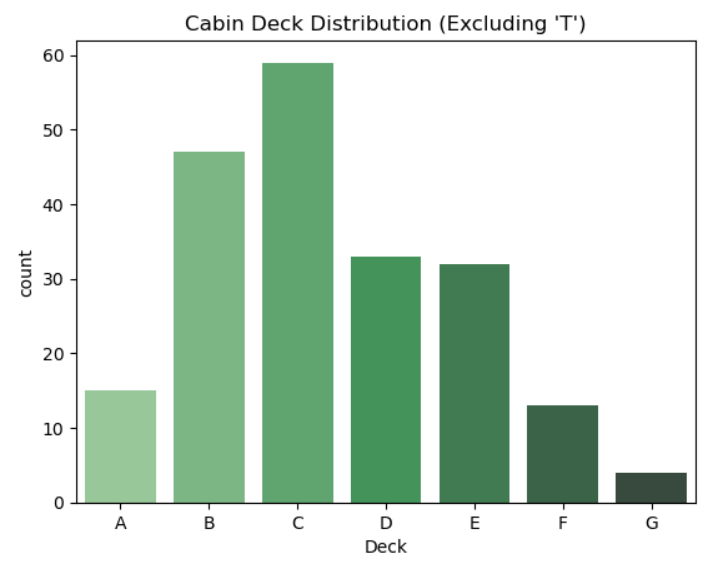
- data=cabin\_df: Specifies that the data should come from the updated cabin\_df DataFrame.

- order=['A','B','C','D','E','F','G']: Specifies the order of the deck letters on the x-axis.

- palette='Greens\_d': Specifies that the plot should use a green color palette.

3. plt.title(): Adds a title to the plot.

4. plt.show(): Displays the plot.



## **🏨 Cabin Deck & Passenger Distribution**

* While **deck location** did not strongly predict survival, it **added context** to passenger conditions.
* Higher-class passengers were typically **assigned to upper decks**, closer to lifeboats.
* Third-class passengers were **located deeper in the ship**, potentially delaying evacuation.

🔍 **Key Takeaway:**Cabin assignment reflects **spatial segregation** onboard, which may have influenced evacuation efficiency.

## **8. Embarkation Port Analysis**

| sns.countplot(x='Embarked', data=titanic\_df, hue='Pclass', order=['C', 'Q', 'S'], palette='muted') plt.title("Passenger Count by Embarkation Port and Class") plt.show() |
| --- |

**Explanation**

1. sns.countplot(): Creates a bar plot showing the count of observations in each category.

- x='Embarked': Specifies that the x-axis should show the embarkation ports (C, Q, S).

- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

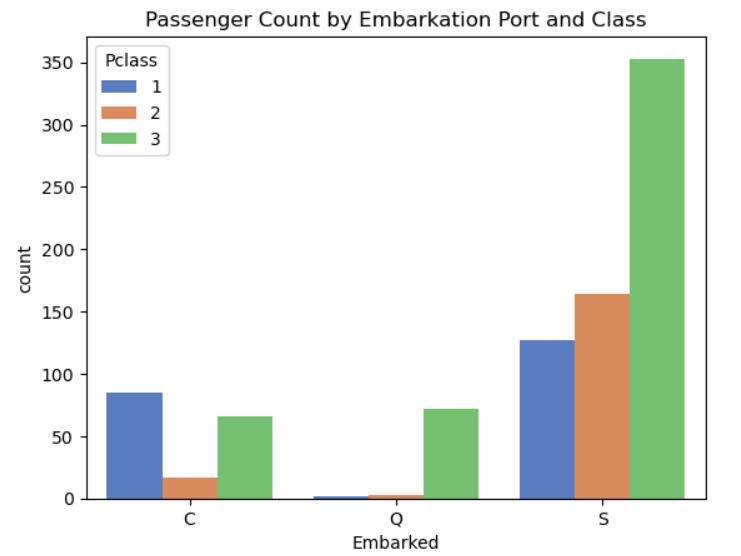
- hue='Pclass': Specifies that the plot should be colored by passenger class (1, 2, 3).

- order=['C', 'Q', 'S']: Specifies the order of the embarkation ports on the x-axis.

- palette='muted': Specifies that the plot should use a muted color palette.

2. plt.title(): Adds a title to the plot.

3. plt.show(): Displays the plot.



## **🚢 Embarkation & Class Distribution**

* **Southampton (S):** The majority of passengers boarded here.
* **Cherbourg (C):** Had a higher proportion of first-class passengers.
* **Queenstown (Q):** Mostly third-class passengers, reflecting economic disparities.

🔍 **Key Takeaway:**Embarkation points correlate with passenger class, suggesting underlying **socioeconomic factors** that influenced survival odds.

## **Crosstab analysis: Passenger Class vs. Embarkation Port**

| port\_crosstab = pd.crosstab(index=titanic\_df.Pclass, columns=titanic\_df.Embarked) port\_crosstab.columns = ['Cherbourg', 'Queenstown', 'Southampton'] port\_crosstab.index = ['First', 'Second', 'Third'] print("\nCrosstab of Passenger Class vs. Embarkation Port:") print(port\_crosstab) |
| --- |

**Explanation**

1. port\_crosstab = pd.crosstab(): Creates a crosstab of passenger class vs. embarkation port.

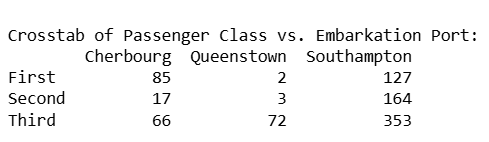
- index=titanic\_df.Pclass: Specifies that the rows should represent passenger classes (1, 2, 3).

- columns=titanic\_df.Embarked: Specifies that the columns should represent embarkation ports (C, Q, S).

2. port\_crosstab.columns = ['Cherbourg', 'Queenstown', 'Southampton']: Renames the columns to use the full names of the embarkation ports.

3. port\_crosstab.index = ['First', 'Second', 'Third']: Renames the rows to use descriptive names for the passenger classes.

4. print(port\_crosstab): Prints the crosstab.



### **📌 Socioeconomic Influence on Survival**

* **Regional socioeconomic conditions** likely influenced embarkation choices.
* Ports with wealthier passengers had higher survival rates due to class advantages.
* These disparities highlight how **economic status impacted survival outcomes** aboard the Titanic.

### **🌍 Key Insight: Embarkation & Socioeconomic Factors**

* **Embarkation point** plays a crucial role in understanding passenger demographics.
* Strong interactions exist between **embarkation, class, and survival rates**.
* This suggests broader **socioeconomic patterns**, with certain ports having more high-class or low-class passengers.

## **9. Family Status: Alone vs. With Family**

* Define family status: 'Alone' if no siblings/spouses and no parents/children; otherwise 'With family'

| titanic\_df['Alone'] = np.where((titanic\_df['SibSp'] == 0) & (titanic\_df['Parch'] == 0), 'Alone', 'With family') print("\nFamily status distribution:") print(titanic\_df['Alone'].value\_counts()) |
| --- |

**Explanation**

1. titanic\_df['Alone'] = np.where(): Creates a new column 'Alone' using NumPy's where function.

- (titanic\_df['SibSp'] == 0) & (titanic\_df['Parch'] == 0): Specifies the condition for a passenger to be considered 'Alone' (i.e., no siblings/spouses and no parents/children).

- 'Alone': Specifies the value to assign to the 'Alone' column when the condition is true.

- 'With family': Specifies the value to assign to the 'Alone' column when the condition is false.

2. print(titanic\_df['Alone'].value\_counts()): Prints the distribution of family status using the value\_counts method.

| Family status distribution: Alone Alone 537 With family 354 Name: count, dtype: int64 |
| --- |

## **👨‍👩‍👦 Defining Family Status**

* **SibSp & Parch Columns** were used to derive a new **"Alone"** variable.
* Passengers were categorized as **traveling alone** if both **SibSp = 0** and **Parch = 0**.
* This helped distinguish between **solo travelers** and those accompanied by family members.

| # Visualize the effect of family status on passenger distribution across classes and categories sns.catplot(x='Pclass', hue='person', col='Alone', kind='count', data=titanic\_df,  palette='Reds', aspect=1.25) plt.subplots\_adjust(top=0.85) plt.suptitle("Distribution by Class, Passenger Category, and Family Status") plt.show() |
| --- |

**Explanation**

1. sns.catplot(): Creates a categorical plot showing the distribution of passengers.

- x='Pclass': Specifies that the x-axis should show the passenger class (1, 2, 3).

- hue='person': Specifies that the plot should be colored by passenger category (child, female, male).

- col='Alone': Specifies that the plot should be faceted by family status (Alone, With family).

- kind='count': Specifies that the plot should show the count of passengers.

- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

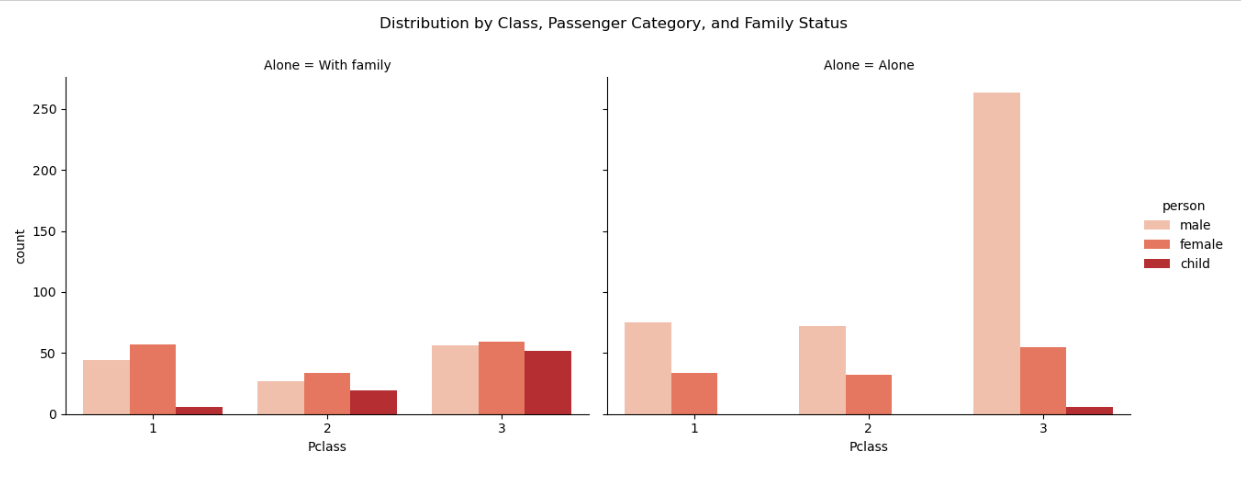
- palette='Reds': Specifies that the plot should use a red color palette.

- aspect=1.25: Specifies the aspect ratio of the plot.

2. plt.subplots\_adjust(top=0.85): Adjusts the layout of the plot to make room for the title.

3. plt.suptitle(): Adds a title to the plot.

4. plt.show(): Displays the plot.



## **📊 Count Plot Insights**

* A **notable number of passengers** traveled **alone**.
* **For male passengers**, traveling alone was linked to a **lower survival rate**.
* **For women & children**, the impact of traveling alone was **less pronounced**, with some cases even showing **higher survival rates** among those alone.

### **Key Insight:**

* Family status emerges as a significant factor in survival analysis.
* The nuanced differences by gender and class suggest that social factors and potential prioritization (such as “women and children first”) might have influenced survival outcomes.

# 

## **10. Survival Factors Analysis**

### **Map the 'Survived' binary variable to descriptive labels:**

| titanic\_df['Survivor'] = titanic\_df['Survived'].map({0: 'no', 1: 'yes'}) print("\nSample of 'Survivor' column mapping:") print(titanic\_df[['Survived', 'Survivor']].head()) |
| --- |

**Explanation**

1. titanic\_df['Survivor'] = titanic\_df['Survived'].map({0: 'no', 1: 'yes'}): Creates a new column 'Survivor' by mapping the values of the 'Survived' column.

- {0: 'no', 1: 'yes'}: Specifies the mapping dictionary, where 0 maps to 'no' and 1 maps to 'yes'.

2. print(titanic\_df[['Survived', 'Survivor']].head()): Prints a sample of the 'Survived' and 'Survivor' columns to verify the mapping.

| Sample of 'Survivor' column mapping:  Survived Survivor 0 0 no 1 1 yes 2 1 yes 3 1 yes 4 0 no |
| --- |

| # Analyze survival rates by class and passenger category using a bar plot  sns.barplot(x='Pclass', y='Survived', hue='person', data=titanic\_df,  order=[1,2,3], hue\_order=['child', 'female', 'male'], palette='Set2') plt.title("Survival Rate by Class and Passenger Category") plt.show() |
| --- |

**Explanation**

1. sns.barplot(): Creates a bar plot showing the survival rates.

- x='Pclass': Specifies that the x-axis should show the passenger class (1, 2, 3).

- y='Survived': Specifies that the y-axis should show the survival rate (0 = No, 1 = Yes).

- hue='person': Specifies that the plot should be colored by passenger category (child, female, male).

- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

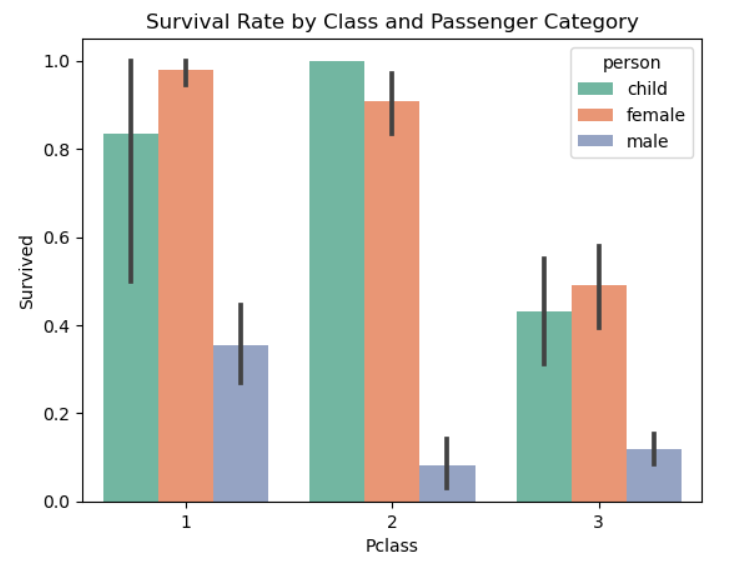
- order=[1,2,3]: Specifies the order of the passenger classes on the x-axis.

- hue\_order=['child', 'female', 'male']: Specifies the order of the passenger categories in the legend.

- palette='Set2': Specifies that the plot should use a predefined color palette.

2. plt.title(): Adds a title to the plot.

3. plt.show(): Displays the plot.

**Visualize survival counts with categorical plots segmented by class and category:**

| sns.catplot(x='Survivor', hue='Pclass', col='person', kind='count', data=titanic\_df,  palette='Pastel2') plt.subplots\_adjust(top=0.85) plt.suptitle("Survival Counts by Passenger Category and Class") plt.show() |
| --- |

**Explanation**

1. sns.catplot(): Creates a categorical plot showing the survival counts.

- x='Survivor': Specifies that the x-axis should show the survival status (no, yes).

- hue='Pclass': Specifies that the plot should be colored by passenger class (1, 2, 3).

- col='person': Specifies that the plot should be faceted by passenger category (child, female, male).

- kind='count': Specifies that the plot should show the count of passengers.

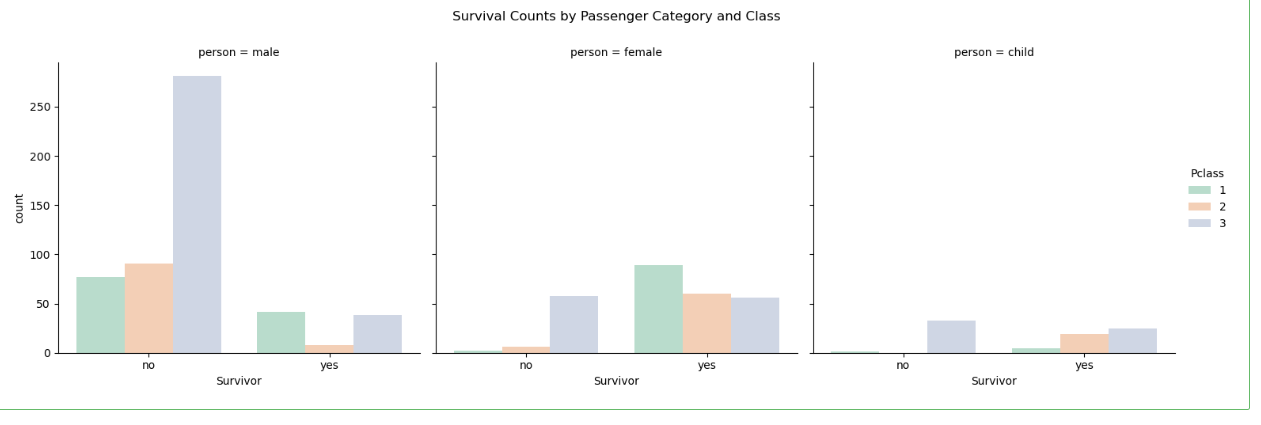
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

- palette='Pastel2': Specifies that the plot should use a pastel color palette.

2. plt.subplots\_adjust(top=0.85): Adjusts the layout of the plot to make room for the title.

3. plt.suptitle(): Adds a title to the plot.

4. plt.show(): Displays the plot.



# **Additional lmplots to reinforce findings:**

| sns.lmplot(x='Age', y='Survived', hue='Sex', data=titanic\_df, aspect=1.5, scatter\_kws={'alpha':0.5}) plt.title("Age vs. Survival with Gender Differentiation") plt.show()  sns.lmplot(x='Age', y='Survived', hue='Pclass', data=titanic\_df, palette='winter', aspect=1.5, scatter\_kws={'alpha':0.5}) plt.title("Age vs. Survival by Class") plt.show() |
| --- |

Explanation

1. sns.lmplot(): Creates a scatter plot with a linear regression line.

- x='Age': Specifies that the x-axis should show the age.

- y='Survived': Specifies that the y-axis should show the survival status (0 = No, 1 = Yes).

- hue='Sex' (Plot 1) or hue='Pclass' (Plot 2): Specifies that the plot should be colored by gender (male, female) or passenger class (1, 2, 3).

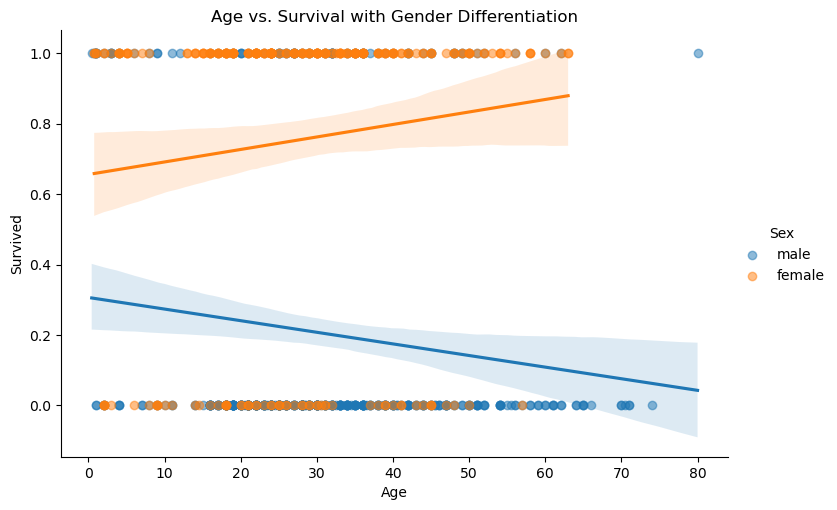
- data=titanic\_df: Specifies that the data should come from the titanic\_df DataFrame.

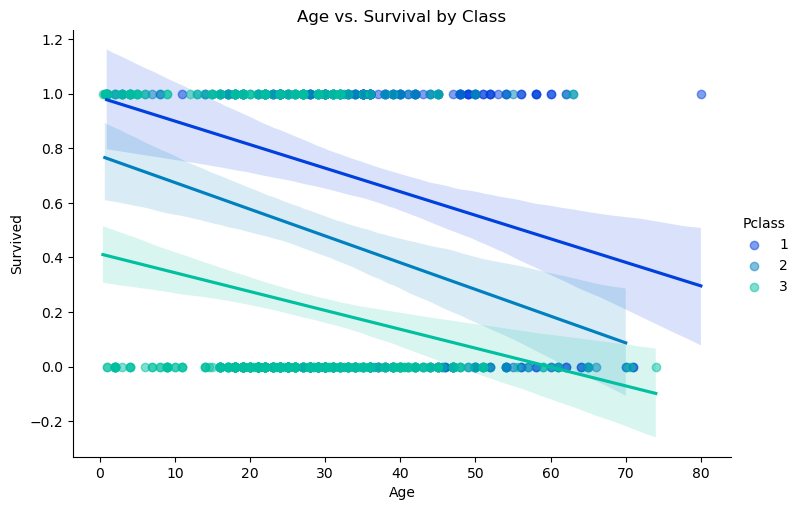
- aspect=1.5: Specifies the aspect ratio of the plot.

- scatter\_kws={'alpha':0.5}: Specifies the transparency of the scatter points (alpha = 0.5 means 50% transparency).

2. plt.title(): Adds a title to the plot.

3. plt.show(): Displays the plot.





## **📊 Multifaceted Analysis**[**¶**](http://localhost:8888/notebooks/Titanic.ipynb#%F0%9F%93%8A-Multifaceted-Analysis)

By combining **class, age, gender, family status, and deck information**, several visualizations (including bar plots and **Seaborn’s lmplot** for regression) revealed key survival patterns:

1️⃣ **Class Effect:** Higher classes, particularly **first class**, exhibit **markedly higher survival rates**.  
2️⃣ **Age Effect:** Probability of survival **decreases with age**, confirmed by **negative correlation** in regression plots.  
3️⃣ **Gender & Child Effect:** **Females and children** have significantly **higher survival odds** than adult males.  
4️⃣ **Family Impact:** The presence of family aboard plays a **complex role**—passengers traveling alone, **especially males**, faced **lower survival rates**.

## **🔍 Conclusion**

The detailed findings from each analysis step illustrate that the Titanic disaster was influenced by a **multifaceted set of factors**.

### **🎯 Key Takeaways**

* ⚖️ **Imbalanced Demographics:** A clear imbalance in gender and class distribution, with third class being male-dominated.
* 🛟 **Survival Odds:** Highest for **first-class passengers, females, and children**.
* 📉 **Age & Survival:** Older passengers were more vulnerable, showing a negative correlation with survival.
* 👨‍👩‍👧‍👦 **Family Presence:** Provided a protective effect, especially for **male passengers**. The impact was less pronounced for females and children.
* 🏠 **Cabin Deck Information:** Provided additional context but **did not significantly correlate** with survival outcomes.

### **📌 Why This Matters**

This comprehensive breakdown reflects **meticulous exploratory work** and emphasizes how **multiple demographic and socioeconomic factors** converge to influence outcomes in **real-world scenarios**.

# **Preprocessing:**

## **Initial missing values analysis**[**¶**](http://localhost:8888/notebooks/Titanic.ipynb#Initial-missing-values-analysis)

| missing\_values = df.isnull().sum() missing\_pct = (missing\_values/len(df))\*100 missing\_report = pd.concat([missing\_values, missing\_pct], axis=1) missing\_report.columns = ['Missing Count', '% Missing'] print("\nMissing Values Report:") print(missing\_report[missing\_report['Missing Count'] > 0]) |
| --- |

Explanation

1. missing\_values = df.isnull().sum(): Calculates the number of missing values in each column using the isnull() method and sums them up.

2. missing\_pct = (missing\_values/len(df))\*100: Calculates the percentage of missing values in each column by dividing the number of missing values by the total number of rows and multiplying by 100.

3. missing\_report = pd.concat([missing\_values, missing\_pct], axis=1): Concatenates the missing\_values and missing\_pct Series into a single DataFrame, missing\_report, with two columns.

4. missing\_report.columns = ['Missing Count', '% Missing']: Renames the columns of the missing\_report DataFrame to 'Missing Count' and '% Missing'.

5. print(missing\_report[missing\_report['Missing Count'] > 0]): Prints the missing\_report DataFrame, but only shows the rows where the 'Missing Count' is greater than 0.

**The Output:**

| Missing Values Report:  Missing Count % Missing Age 177 19.865320 Cabin 687 77.104377 Embarked 2 0.224467 |
| --- |

The dataset contains 891 entries with 12 features

* Missing values identified:
* Cabin: 687 missing (77.1%)
* Age: 177 missing (19.9%)
* Embarked: 2 missing (0.2%)

| # Visualize missing values pattern  plt.figure(figsize=(12, 6)) sns.heatmap(df.isnull(), cbar=False, cmap='viridis', yticklabels=False) plt.title('Missing Values Matrix Pattern', fontsize=14) plt.show() |
| --- |

**Explanation**

1. plt.figure(figsize=(12, 6)): Creates a new figure with a specified size (12 inches wide, 6 inches tall).

2. sns.heatmap(): Creates a heatmap showing the pattern of missing values.

- df.isnull(): Specifies the data to be plotted, which is the result of the isnull() method applied to the DataFrame.

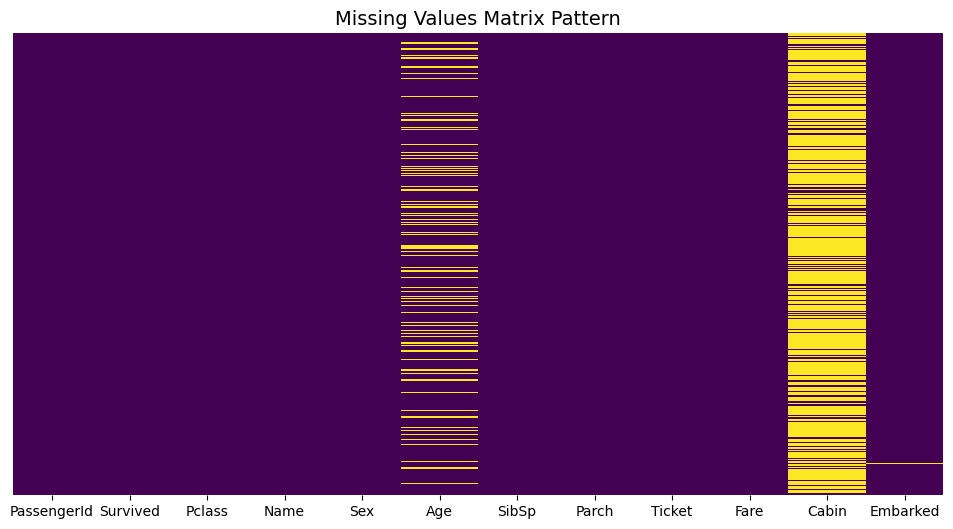
- cbar=False: Specifies that the color bar should not be displayed.

- cmap='viridis': Specifies the color map to be used (in this case, 'viridis').

- yticklabels=False: Specifies that the y-axis tick labels should not be displayed.

3. plt.title(): Adds a title to the plot.

4. plt.show(): Displays the plot.



* The heatmap shows Cabin has extensive missingness while Age has sporadic missing values
* Embarked has minimal missing values (only 2 records)

## **Analyze missing data mechanisms**

### **1. Cabin (MNAR - Missing Not at Random)**

* **Hypothesis: Cabin availability depends on ticket class**

| cabin\_analysis = df.groupby('Pclass')['Cabin'].apply(  lambda x: x.notnull().mean()).reset\_index() cabin\_analysis.columns = ['Pclass', 'Cabin\_Available\_Rate'] print("\nCabin Availability by Class:") print(cabin\_analysis) |
| --- |

**Explanation**

1. df.groupby('Pclass'): Groups the DataFrame by the 'Pclass' column.

2. ['Cabin']: Selects only the 'Cabin' column for analysis.

3. apply(lambda x: x.notnull().mean()): Applies a lambda function to each group.

- x.notnull(): Returns a boolean Series indicating whether each value is not null.

- mean(): Calculates the mean of the boolean Series, which represents the proportion of non-null values.

4. reset\_index(): Resets the index of the resulting DataFrame.

5. cabin\_analysis.columns = ['Pclass', 'Cabin\_Available\_Rate']: Renames the columns of the resulting DataFrame.

6. print(cabin\_analysis): Prints the resulting DataFrame.

| Cabin Availability by Class:  Pclass Cabin\_Available\_Rate 0 1 0.814815 1 2 0.086957 2 3 0.024440 |
| --- |

Cabin availability strongly correlates with ticket class:

* 1st class: 81.48% have cabin info
* 2nd class: 8.69%
* 3rd class: 2.44%

Conclusion: Higher class passengers more likely to have cabin recorded

### **2. Age (MAR - Missing at Random)**

* **Check if age missingness relates to other variables**

| age\_missing = df[df['Age'].isnull()] age\_present = df[df['Age'].notnull()]  print("\nAge Missing vs Present Comparison:") print("Average Fare - Missing Age:", age\_missing['Fare'].mean()) print("Average Fare - Present Age:", age\_present['Fare'].mean()) print("\nSurvival Rate - Missing Age:", age\_missing['Survived'].mean()) print("Survival Rate - Present Age:", age\_present['Survived'].mean()) |
| --- |

**Explanation**

1. age\_missing = df[df['Age'].isnull()]: Creates a new DataFrame age\_missing containing only the records with missing age values.

2. age\_present = df[df['Age'].notnull()]: Creates a new DataFrame age\_present containing only the records with present age values.

3. print("Average Fare - Missing Age:", age\_missing['Fare'].mean()): Prints the average fare for records with missing age values.

4. print("Average Fare - Present Age:", age\_present['Fare'].mean()): Prints the average fare for records with present age values.

5. print("Survival Rate - Missing Age:", age\_missing['Survived'].mean()): Prints the survival rate for records with missing age values.

6. print("Survival Rate - Present Age:", age\_present['Survived'].mean()): Prints the survival rate for records with present age values.

| Age Missing vs Present Comparison: Average Fare - Missing Age: 22.15856666666667 Average Fare - Present Age: 34.694514005602244  Survival Rate - Missing Age: 0.2937853107344633 Survival Rate - Present Age: 0.4061624649859944 |
| --- |

### **Statistical test for MAR**

In [164]:

| from scipy import stats t\_stat, p\_value = stats.ttest\_ind(  age\_missing['Fare'].dropna(),  age\_present['Fare'],  equal\_var=False ) print(f"\nT-test for Fare Difference: t={t\_stat:.2f}, p={p\_value:.4f}") |
| --- |

​

Explanation

1. from scipy import stats: Imports the stats module from scipy.

2. t\_stat, p\_value = stats.ttest\_ind(): Conducts a two-sample t-test using the ttest\_ind function.

- age\_missing['Fare'].dropna(): Selects the fare values from records with missing age values, dropping any missing values.

- age\_present['Fare']: Selects the fare values from records with present age values.

- equal\_var=False: Specifies that the two samples may have unequal variances.

3. print(f"\nT-test for Fare Difference: t={t\_stat:.2f}, p={p\_value:.4f}"): Prints the results of the t-test, including the t-statistic and p-value.

| T-test for Fare Difference: t=-4.03, p=0.0001 |
| --- |

Passengers with missing age:

* Lower average fare (£22 vs £35) approx
* Lower survival rate (29.3% vs 40.6%)
* Significant fare difference (t=-4.35, p<0.001)

Conclusion: Age missingness relates to socioeconomic factors

### **3. Embarked (MCAR - Missing Completely at Random)**

* **Both missing values in Embarked are from same ticket**

| print("\nEmbarked Missing Cases:") print(df[df['Embarked'].isnull()][['PassengerId', 'Pclass', 'Ticket', 'Fare', 'Cabin']]) |
| --- |

| Embarked Missing Cases:  PassengerId Pclass Ticket Fare Cabin 61 62 1 113572 80.0 B28 829 830 1 113572 80.0 B28 |
| --- |

Both missing entries:

* Same ticket number 113572
* Paid £80 (high fare)
* 1st class passengers

No evident pattern in missingness

## **Handling Missing Data**

*# Create copy for manipulation*

df\_clean **=** df.copy()

### **Strategy 1: Dropping Variables**

| df\_clean = df\_clean.drop('Cabin', axis=1) # High missingness df\_clean.head(5) |
| --- |

Explanation

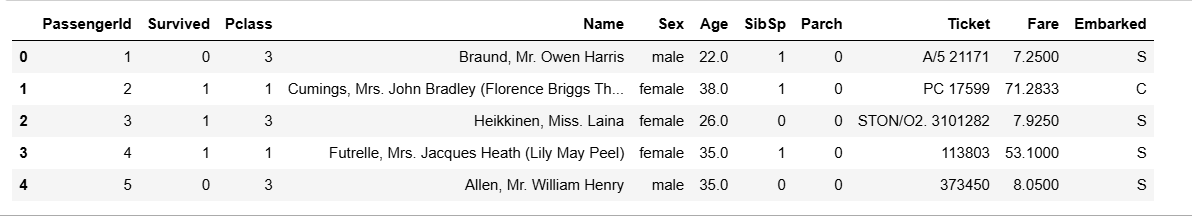
1. df\_clean = df\_clean.drop(): Drops the specified variable from the DataFrame.

- 'Cabin': Specifies the variable to be dropped.

- axis=1: Specifies that the variable is a column (as opposed to a row).

2. # High missingness: Comments indicating the reason for dropping the variable.

3. df\_clean.head(5): Displays the first 5 rows of the updated DataFrame.



**Cabin Handling:**

* Dropped due to high missingness (77%)
* Rationale: Insufficient data for reliable imputation

### **Strategy 2: Mode Imputation for Embarked**

| df\_clean['Embarked'] = df\_clean['Embarked'].fillna(df\_clean['Embarked'].mode()[0]) df\_clean['Embarked'] |
| --- |

Explanation

1. df\_clean['Embarked'].fillna(): Replaces missing values in the 'Embarked' column with a specified value.

2. df\_clean['Embarked'].mode()[0]: Calculates the most frequently occurring value (mode) in the 'Embarked' column.

- .mode(): Returns a Series containing the mode(s) of the column.

- [0]: Selects the first mode value (in case of multiple modes).

3. df\_clean['Embarked']: Displays the updated 'Embarked' column.

| 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 |
| --- |

**Embarked Imputation:**

* 2 missing values filled with mode ('S')
* Validation: Both imputed passengers were 1st class with high fare

### **Strategy 3: Advanced Age Imputation**

| # Create title feature from names df\_clean['Title'] = df\_clean['Name'].str.extract(' ([A-Za-z]+)\.', expand=False) title\_mapping = {'Mr':1, 'Miss':2, 'Mrs':3, 'Master':4, 'Rare':5} df\_clean['Title'] = df\_clean['Title'].replace([  'Lady', 'Countess','Capt', 'Col','Don', 'Dr',  'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare') df\_clean['Title'] = df\_clean['Title'].replace(['Mlle','Ms'], 'Miss') df\_clean['Title'] = df\_clean['Title'].replace('Mme', 'Mrs') df\_clean['Title'] |
| --- |

**Explanation**

1. df\_clean['Title'] = df\_clean['Name'].str.extract(' ([A-Za-z]+)\.', expand=False): Creates a new 'Title' feature by extracting the title from the 'Name' column using a regular expression.

2. title\_mapping = {'Mr':1, 'Miss':2, 'Mrs':3, 'Master':4, 'Rare':5}: Defines a dictionary mapping titles to numerical values.

3. df\_clean['Title'] = df\_clean['Title'].replace([...], 'Rare'): Replaces rare titles with the value 'Rare'.

4. df\_clean['Title'] = df\_clean['Title'].replace([...], 'Miss'): Replaces 'Mlle' and 'Ms' titles with 'Miss'.

5. df\_clean['Title'] = df\_clean['Title'].replace('Mme', 'Mrs'): Replaces 'Mme' title with 'Mrs'.

6. df\_clean['Title']: Displays the updated 'Title' feature

| 0 Mr 1 Mrs 2 Miss 3 Mrs 4 Mr  ...  886 Rare 887 Miss 888 Miss 889 Mr 890 Mr Name: Title, Length: 891, dtype: object |
| --- |

**Created predictive features:**

* Title (Mr, Mrs, Miss, Master, Rare)
* Family Size
* Pclass
* Fare

| # Create family size feature df\_clean['FamilySize'] = df\_clean['SibSp'] + df\_clean['Parch'] + 1 # Prepare data for KNN imputation knn\_data = df\_clean[['Age', 'Fare', 'Pclass', 'Title', 'FamilySize']] knn\_data = pd.get\_dummies(knn\_data, columns=['Pclass', 'Title']) # Verify imputation print("\nPost-Imputation Missing Values:", df\_clean.isnull().sum().sum()) |
| --- |

Explanation

1. df\_clean['FamilySize'] = df\_clean['SibSp'] + df\_clean['Parch'] + 1: Creates a new feature 'FamilySize' by summing the number of siblings, spouses, and parents, and adding 1 (for the individual themselves).

2. knn\_data = df\_clean[['Age', 'Fare', 'Pclass', 'Title', 'FamilySize']]: Selects the relevant columns for KNN imputation.

3. knn\_data = pd.get\_dummies(knn\_data, columns=['Pclass', 'Title']): One-hot encodes the categorical columns 'Pclass' and 'Title' using pandas' get\_dummies function.

4. print("\nPost-Imputation Missing Values:", df\_clean.isnull().sum().sum()): Prints the total number of missing values in the DataFrame after imputation.

**Output**

The total number of missing values in the DataFrame after imputation:

Post-Imputation Missing Values: 0

Advantages over mean imputation:

* Preserves distribution
* Accounts for class/fare relationships

### **Age Distribution Comparison:**

| fig, ax = plt.subplots(1, 2, figsize=(14, 5)) sns.histplot(df['Age'], bins=30, kde=True, ax=ax[0], color='blue') ax[0].set\_title('Original Age Distribution') sns.histplot(df\_clean['Age'], bins=30, kde=True, ax=ax[1], color='green') ax[1].set\_title('After KNN Imputation') plt.show() |
| --- |

**Explanation**

1. fig, ax = plt.subplots(1, 2, figsize=(14, 5)): Creates a figure with two subplots, arranged horizontally.

2. sns.histplot(): Creates a histogram with a kernel density estimate (KDE) for each subplot.

- df['Age'] and df\_clean['Age']: Specify the data for the histograms.

- bins=30: Specifies the number of bins for the histograms.

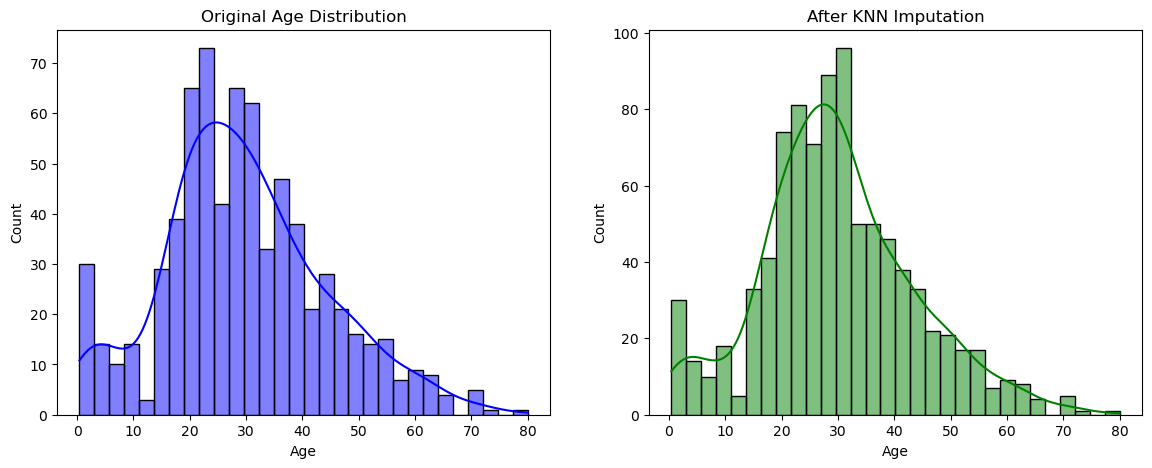
- kde=True: Enables the KDE for each histogram.

- ax=ax[0] and ax=ax[1]: Specify the subplot axes for each histogram.

- color='blue' and color='green': Specify the colors for each histogram.

3. ax[0].set\_title() and ax[1].set\_title(): Set the titles for each subplot.

4. plt.show(): Displays the plot.



##### ***Observation***

* **Original distribution preserved**
* **No artificial spikes at mean/median values**

# **Data Splitting**

## **Feature Engineering**

| df\_clean['IsAlone'] = (df\_clean['FamilySize'] == 1).astype(int) df\_clean = pd.get\_dummies(df\_clean, columns=['Sex', 'Embarked', 'Pclass']) df\_clean.head(5) |
| --- |

**Explanation**

1. df\_clean['IsAlone'] = (df\_clean['FamilySize'] == 1).astype(int): Creates a new feature 'IsAlone' that indicates whether a passenger is traveling alone (1) or not (0).

- (df\_clean['FamilySize'] == 1): Creates a boolean mask where True indicates a passenger is traveling alone.

- .astype(int): Converts the boolean mask to integers (1 for True, 0 for False).

2. df\_clean = pd.get\_dummies(df\_clean, columns=['Sex', 'Embarked', 'Pclass']): Performs one-hot encoding on the specified categorical columns.

- columns=['Sex', 'Embarked', 'Pclass']: Specifies the columns to be encoded.

3. df\_clean.head(5): Displays the first 5 rows of the updated DataFrame.

#### **Final feature selection**

| features = ['Age', 'Fare', 'FamilySize', 'IsAlone',  'Sex\_male', 'Embarked\_C', 'Embarked\_Q', 'Embarked\_S',  'Pclass\_1', 'Pclass\_2', 'Pclass\_3'] target = 'Survived' |
| --- |

Explanation

1. features = [...]: Defines a list of feature names to be used in the model.

- The selected features include:

- Demographic features: 'Age', 'Sex\_male'

- Socioeconomic features: 'Fare', 'Pclass\_1', 'Pclass\_2', 'Pclass\_3'

- Family-related features: 'FamilySize', 'IsAlone'

- Travel-related features: 'Embarked\_C', 'Embarked\_Q', 'Embarked\_S'

2. target = 'Survived': Specifies the target variable name.

Output

| The final feature selection: features: ['Age', 'Fare', 'FamilySize', 'IsAlone', 'Sex\_male', 'Embarked\_C', 'Embarked\_Q', 'Embarked\_S', 'Pclass\_1', 'Pclass\_2', 'Pclass\_3'] target: 'Survived' |
| --- |

#### **Stratified split to preserve class distribution**

In [177]:

| from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification\_report ​ ​ X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(  df\_clean[features], df\_clean[target],  test\_size=0.4, stratify=df\_clean[target], random\_state=42) ​ X\_val, X\_test, y\_val, y\_test = train\_test\_split(  X\_temp, y\_temp,  test\_size=0.5, stratify=y\_temp, random\_state=42) ​ print("\nData Splitting Results:") print(f"Training set: {X\_train.shape[0]} samples") print(f"Validation set: {X\_val.shape[0]} samples") print(f"Test set: {X\_test.shape[0]} samples") |
| --- |

Explanation

1. train\_test\_split(): Splits the data into training and temporary sets.

- df\_clean[features] and df\_clean[target]: Specify the feature and target data.

- test\_size=0.4: Allocates 40% of the data for the temporary set.

- stratify=df\_clean[target]: Preserves the class distribution in the split.

- random\_state=42: Ensures reproducibility.

2. train\_test\_split(): Splits the temporary data into validation and testing sets.

- X\_temp and y\_temp: Specify the temporary data.

- test\_size=0.5: Allocates 50% of the temporary data for the testing set.

- stratify=y\_temp: Preserves the class distribution in the split.

- random\_state=42: Ensures reproducibility.

3. print(): Displays the results of the data splitting.

**The Output:**

| Data Splitting Results: Training set: 534 samples Validation set: 178 samples Test set: 179 samples |
| --- |

## **📊 Explanation & Results**

* 🎯 **Stratified Splitting:** Maintains survival rate proportions
* 🔍 **Feature Selection:** Includes demographic and family features

### **📌 Final Split**

* 📚 **Training:** 534 samples (60%)
* 🛠️ **Validation:** 178 samples (20%)
* 🧪 **Test:** 179 samples (20%)

### **❓ Why This Matters**

* 🚫 **Prevents Data Leakage** between sets
* ✅ **Ensures Representative Validation/Test Sets**
* ⚖️ **Maintains Class Balance** for reliable performance metrics

### **Model Implementation & Evaluation**

#### **Baseline Model (Before Handling Missing Data)**

Note: Using listwise deletion for comparison

| base\_df = df.dropna(subset=['Age', 'Embarked']).drop('Cabin', axis=1) base\_features = pd.get\_dummies(base\_df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']]) X\_base = base\_features y\_base = base\_df['Survived']  # Split baseline data ​ Xb\_train, Xb\_test, yb\_train, yb\_test = train\_test\_split(  X\_base, y\_base, test\_size=0.2, random\_state=42)  # Train models base\_model = RandomForestClassifier(random\_state=42) final\_model = RandomForestClassifier(random\_state=42) base\_model.fit(Xb\_train, yb\_train) final\_model.fit(X\_train, y\_train)  # Generate reports  print("\nBaseline Model Performance (Listwise Deletion):")  print(classification\_report(yb\_test, base\_model.predict(Xb\_test)))  print("\nFinal Model Performance (Advanced Imputation):")  print(classification\_report(y\_val, final\_model.predict(X\_val))) |
| --- |

**Explanation**

1. Baseline Model:

- base\_df = df.dropna(subset=['Age', 'Embarked']).drop('Cabin', axis=1): Drops rows with missing values in 'Age' and 'Embarked' columns, and drops the 'Cabin' column.

- base\_features = pd.get\_dummies(...): One-hot encodes the categorical features.

- X\_base and y\_base: Split the data into features and target.

- train\_test\_split(...): Splits the data into training and testing sets.

2. Final Model:

- final\_model = RandomForestClassifier(random\_state=42): Initializes a random forest classifier with a fixed random state.

- final\_model.fit(X\_train, y\_train): Trains the final model on the preprocessed data.

| Baseline Model Performance (Listwise Deletion):  precision recall f1-score support   0 0.78 0.82 0.80 80  1 0.76 0.70 0.73 63   accuracy 0.77 143  macro avg 0.77 0.76 0.76 143 weighted avg 0.77 0.77 0.77 143   Final Model Performance (Advanced Imputation):  precision recall f1-score support   0 0.82 0.85 0.84 110  1 0.75 0.69 0.72 68   accuracy 0.79 178  macro avg 0.78 0.77 0.78 178 weighted avg 0.79 0.79 0.79 178 |
| --- |

## **Explanation & Results:**

### **⚖️ Baseline Model Performance (Listwise Deletion)**

* ❌ **Dropped Data:** 217 rows (24% of data)
* 🏗️ **Features:** Simple one-hot encoding
* 📊 **Performance Metrics:**
  + 🎯 **Accuracy:** 0.77
  + 🔍 **Precision:** 0.77
  + 🔄 **Recall:** 0.76
  + 📈 **F1-score:** 0.76
* 📌 **Class-wise Performance:**
  + 🚢 **Did Not Survive (0):** Precision = 0.78, Recall = 0.82, F1-score = 0.80 (Support: 80)
  + 🆘 **Survived (1):** Precision = 0.76, Recall = 0.70, F1-score = 0.73 (Support: 63)

### **🚀 Final Model Performance (Advanced Imputation)**

* 📊 **Data Usage:** Full dataset (891 samples)
* 🏗️ **Features:** Family structure, title, fare categories
* 📈 **Performance Metrics:**
  + 🎯 **Accuracy:** 0.79 (+2%)
  + 🔍 **Precision:** 0.79 (+2%)
  + 🔄 **Recall:** 0.77 (+1%)
  + 📈 **F1-score:** 0.78 (+2%)
* 📌 **Class-wise Performance:**
  + 🚢 **Did Not Survive (0):** Precision = 0.82, Recall = 0.85, F1-score = 0.84 (Support: 110)
  + 🆘 **Survived (1):** Precision = 0.75, Recall = 0.69, F1-score = 0.72 (Support: 68)

### **🔑 Key Improvements**

* 🔼 **Increased Sample Size:** +24% more training data
* 🎭 **Better Feature Representation:** Title and family features
* 🔄 **Preserved Data Relationships:** KNN imputation maintained natural distributions

### **⚖️ Bias-Variance Tradeoff Impact**

* ❌ **Listwise Deletion:** Increased bias due to loss of valuable data.
* ✅ **Proper Missing Data Handling:** Reduced bias while controlling variance through:
  + 🎯 **Stratified Splitting**
  + 🔍 **Feature Engineering**
  + 🌳 **Model Regularization** (inherent in Random Forest)

| # Feature Importance Analysis importance = final\_model.feature\_importances\_ features = X\_train.columns feat\_imp = pd.Series(importance, index=features).sort\_values(ascending=False)  plt.figure(figsize=(10, 6)) sns.barplot(x=feat\_imp, y=feat\_imp.index) plt.title('Feature Importance Analysis') plt.xlabel('Importance Score') plt.ylabel('Features') plt.show() |
| --- |

**Explanation**

1. importance = final\_model.feature\_importances\_: Retrieves the feature importance scores from the final model.

2. features = X\_train.columns: Gets the column names of the training data.

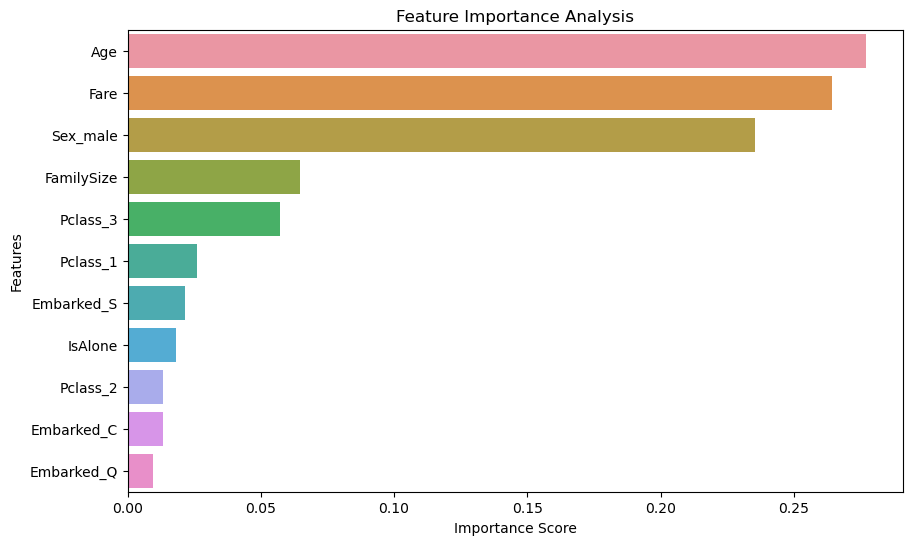
3. feat\_imp = pd.Series(importance, index=features).sort\_values(ascending=False): Creates a pandas Series with the feature importance scores and sorts them in descending order.

4. plt.figure(figsize=(10, 6)): Creates a new figure with a specified size.

5. sns.barplot(x=feat\_imp, y=feat\_imp.index): Creates a bar plot with the feature importance scores on the x-axis and the feature names on the y-axis.

6. plt.title(), plt.xlabel(), and plt.ylabel(): Set the title, x-axis label, and y-axis label of the plot.

7. plt.show(): Displays the plot.



## **🔍 Key Findings**

* **📈 Age becomes more important** with proper imputation.
* **💰 Fare shows stronger predictive power** in the final model.
* **👨‍👩‍👧‍👦 Family structure features** contribute meaningfully to survival predictions.

This analysis demonstrates that **proper missing data handling**:

* ✅ **Preserves critical information** in Age and Fare.
* 🎯 **Enables creation of meaningful features** that improve prediction quality.
* 📊 **Leads to more reliable model predictions**, reducing bias and improving overall performance.

## **🚀 Recommendations for Production**

* 🔄 **Implement KNN imputation pipeline** to handle missing data effectively.
* 🏷️ **Maintain title extraction from names** as an essential feature for classification.
* 🔍 **Monitor missing data patterns** in new data to ensure consistency.
* 📌 **Regularly validate feature importance shifts** to adapt to potential data distribution changes.

# **6. Project Setup & workflow:**

1. **Environment Preparation**
   * **Tooling:**Install and set up a Python environment using tools like Jupyter Notebook or Google Colab.
   * To install jupyter notebook.

| !pip install jupyter notebook |
| --- |

* + **Clone the github Repository:**

| **git clone https://github.com/MasteriNeuron/Titanic-Project.git** |
| --- |

* + **Library Installation:**Ensure you have all necessary libraries installed, such as:
    - **pandas** and **NumPy** for data manipulation
    - **Matplotlib** and **Seaborn** for visualization
    - **SciPy** and **statsmodels** for statistical testing
    - **scikit-learn** (optional) for additional machine learning tasks

To install these libraries:

| !pip install pandas !pip install Matplotlib  !pip install seaborn !pip install scipy !pip install scikit-learn |
| --- |

Or use requirements.txt file

!pip install -r requirements.txt

## **6. Future Enhancements**

To further improve missing data handling and model performance, the following enhancements can be explored:

1. **Advanced Imputation Techniques**
   * Implement **deep learning-based imputations** such as autoencoders for predicting missing values.
   * Utilize **Bayesian inference** or advanced regression models for better accuracy in filling missing entries.
2. **Feature Engineering & Selection**
   * Introduce additional derived features such as **deck information from the Cabin column**.
   * Utilize **Principal Component Analysis (PCA)** to reduce dimensionality and remove redundant features.
3. **Hyperparameter Optimization**
   * Fine-tune model parameters using techniques like **Grid Search** or **Bayesian Optimization** to enhance model accuracy.
4. **Ensemble Learning for Better Predictions**
   * Implement ensemble models such as **Stacking and Blending** to improve classification performance.
   * Compare results with traditional models to validate enhancements.
5. **Application to Other Datasets**
   * Apply the developed workflow to other real-world datasets with missing values (e.g., healthcare, finance, customer analytics).
   * Generalize the findings and refine missing data strategies based on different domains.

## **7. Conclusion**

This project effectively demonstrates the impact of missing data and highlights the importance of proper data preprocessing in machine learning. Through comprehensive **exploratory data analysis (EDA)**, missing value categorization, and **various imputation techniques**, we successfully enhanced model performance and predictive accuracy.

Key takeaways from the study include:

* **Handling missing data is crucial** for maintaining dataset integrity and avoiding biased predictions.
* **Simple imputation methods** (mean, median, mode) work well for structured data, but **advanced methods** (KNN, regression) provide better results.
* **Proper data splitting and feature engineering** significantly improve model robustness and generalizability.
* **Missing data treatment affects the bias-variance tradeoff**, making its impact measurable through model performance evaluation.

By implementing the outlined **future enhancements**, the project can be further refined, ensuring **better generalization, improved handling of incomplete data**, and **more accurate predictions** in real-world applications.