WEEK 3 ASSIGNMENT REPORT

CYBER SHUJAA

ADM NO: CS-DA01-2507
COURSE: DATA AND AI
NAME: JEDIDAH WAVINYA MUSYOKA
DATE: 30TH MAY 2025
TASK: EXPLORATORY DATA ANALYSIS

INTRODUCTION

- Every dataset tells a story, but not all of them are immediately clear.
- Often, the most profound insights are hidden beneath layers of raw information, waiting to be discovered. This is where Exploratory Data Analysis (EDA) comes in.
- EDA is the process of analyzing datasets to summarize their main characteristics, often using visual methods.
- Think of EDA not just as a statistical exercise, but as a crucial first conversation with your data, a chance to really understand its nuances, identify patterns, and uncover anomalies before diving into deeper analysis or model building.
- This report documents the process and findings of an Exploratory Data Analysis (EDA) assignment, undertaken to better understand the structure, patterns, and potential anomalies within the given dataset.
- The link to the dataset is as follows: https://www.kaggle.com/code/mariyamalshatta/masterclass-1-a-comprehensive-guide-for-eda
- It involves visualizations, summary statistics, and thoughtful questioning to uncover the stories hidden in the data. It’s the moment I stopped and asked, “What’s really going on here?”
- This assignment provided an opportunity to practice these skills firsthand.
- From identifying missing values and outliers to uncovering trends and relationships between variables, each step revealed layers of complexity and insight that might otherwise be overlooked.
- The aim was not just to prepare the data for further analysis, but to develop an intuitive understanding of its behavior, an essential foundation for building accurate, meaningful models in the future.

The purpose of the assignment was to practice Exploratory Data Analysis steps:

1. Initial Data Exploration
2. Handling Missing Values and Outliers
3. Univariate Analysis
4. Bivariate Analysis
5. Multivariate Analysis
6. Target Variable Analysis

Starting point

Key Questions explored:

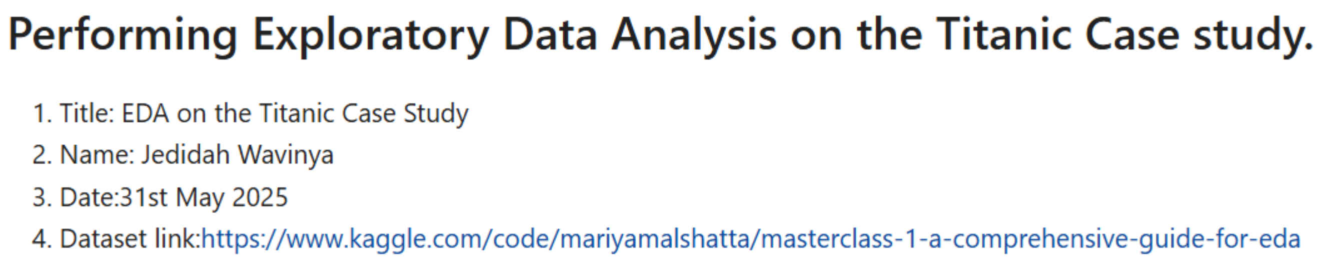
1) What was the overall survival rate on the Titanic?
2) How did survival rates differ based on gender?
3) How did survival rates differ based on passenger class (Pclass)?
4) What was the age distribution of passengers, and how did age relate to survival? (Were children or the elderly more likely to survive?)
5) Are there any interesting relationships between Pclass and Fare?
6) How does Age interact with Pclass and Gender in relation to survival?
7) Does the Cabin information (or lack thereof) provide any insights into survival?

8) Is there any relationship between Embarked and other features like Pclass or Fare that might influence survival?

- The aim was to personalize my Exploratory Data Analysis of the data set with the following link::

Link: https://www.kaggle.com/code/mariyamalshatta/masterclass-1-a-comprehensive-guide-for-eda

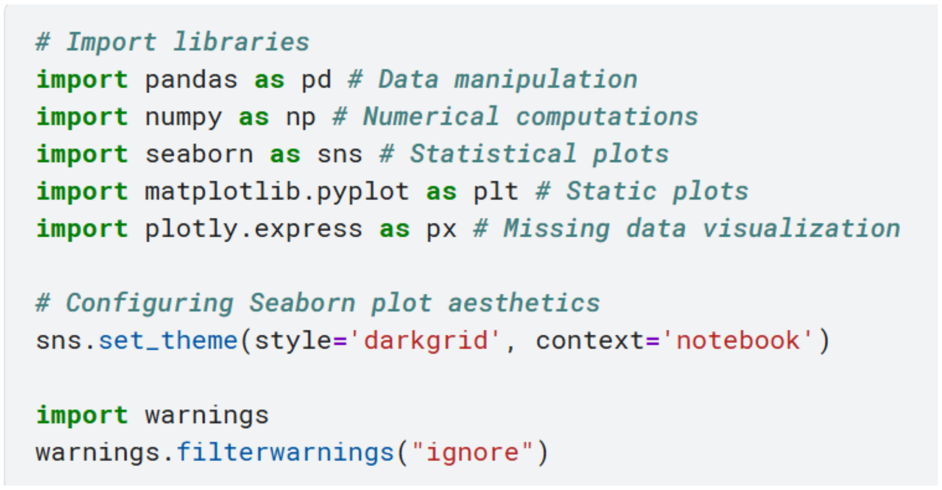
- Started off by giving a brief description of my EDA project for the Titanic Case study:
- See in the image below:



Performing Exploratory Data Analysis on the Titanic Case study.

1. Title: EDA on the Titanic Case Study
2. Name: Jedidah Wavinya
3. Date:31st May 2025
4. Dataset link:https://www.kaggle.com/code/mariyamalshatta/masterclass-1-a-comprehensive-guide-for-eda

- Imported the necessary libraries:
- See below:



```
# Import libraries
import pandas as pd # Data manipulation
import numpy as np # Numerical computations
import seaborn as sns # Statistical plots
import matplotlib.pyplot as plt # Static plots
import plotly.express as px # Missing data visualization

# Configuring Seaborn plot aesthetics
sns.set_theme(style='darkgrid', context='notebook')

import warnings
warnings.filterwarnings("ignore")
```

Tools and Libraries for EDA

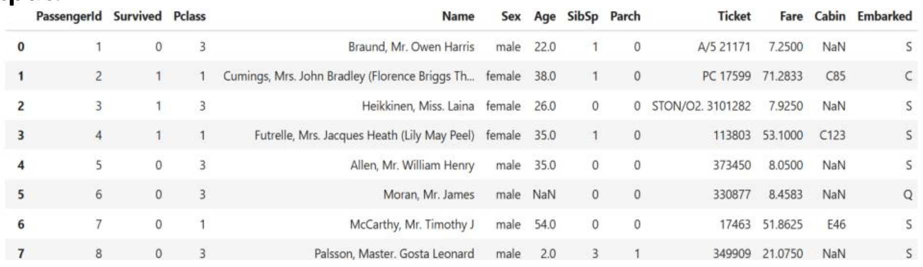
- To perform EDA efficiently, I used the several Python libraries:
  - **pandas**: For data manipulation and inspection.
  - **numpy**: For numerical computations.
  - **matplotlib**: For static visualizations (e.g., histograms, scatter plots).
  - **seaborn**: For statistical plots and heatmaps.
  - **missingno**: For visualizing missing data.
  - **plotly/altair**: For interactive visualizations.

STEP 1 INITIAL DATA EXPLORATION

- Have a preview of the dataset/load it
  - See on the image below:

```python
# Load dataset
train = pd.read_csv("/kaggle/input/titanic/train.csv")
test = pd.read_csv("/kaggle/input/titanic/test.csv")
gender = pd.read_csv("/kaggle/input/titanic/gender_submission.csv")
df = pd.read_csv("/kaggle/input/titanic/train.csv")
df.head()
# Preview the first 20 rows of the dataset
df.head(20)
```

- **Run the code!**
- **The output:**

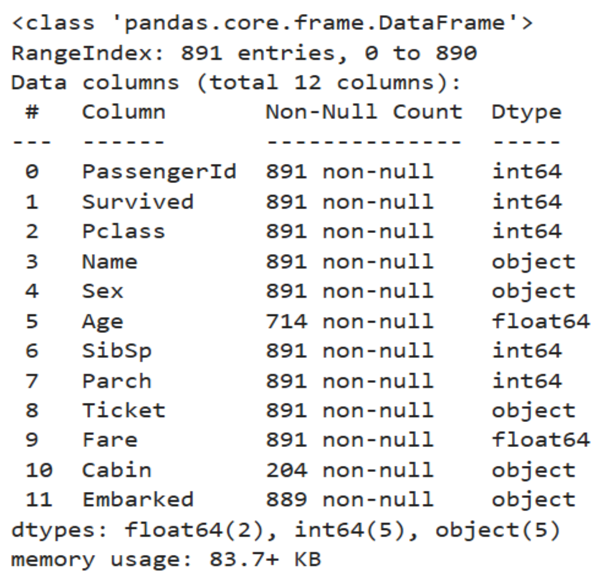


	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S

- Analyze the dataset’s data types, missing values, duplicates, errors, and plots you can explore per feature.
- Here are some essential pandas functions used for initial exploration:
  - df.head(): Displays the first few rows of the dataset to give you a quick preview.
  - df.shape: Returns the number of rows and columns in the dataset.
  - df.info(): Provides details about the columns, their data types, and the number of non-null (non-missing) values.
  - df.describe(): Provides summary statistics (mean, median, min, max, etc.) for numerical columns.
  - df.columns: Lists the names of all columns in the dataset.
  - df.nunique(): Returns the number of unique values in each column.
  - df.duplicated(): Checks for duplicate rows

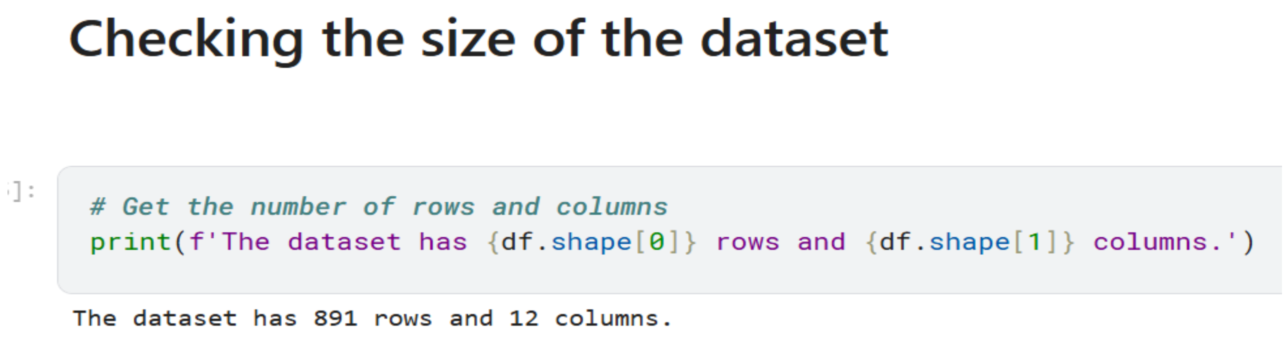
```python
df.shape #Returns the number of rows and columns in the dataset.
df.info() #Provides details about the columns, their data types, and the number of non-null (non-missing) values.
df.describe() # Provides summary statistics (mean, median, min, max, etc.) for numerical columns.
df.columns #Lists the names of all columns in the dataset.
df.nunique() #Returns the number of unique values in each column.
df.duplicated() #Checks for duplicate rows
```

- The output:



```
<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
```

- To check the size of the dataset:

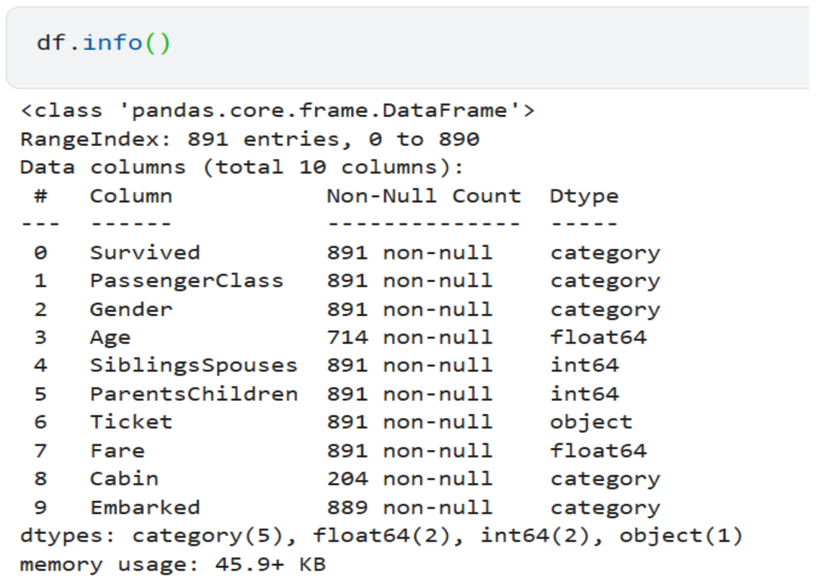


Checking the size of the dataset

```python
# Get the number of rows and columns
print(f'The dataset has {df.shape[0]} rows and {df.shape[1]} columns.')
```

```
The dataset has 891 rows and 12 columns.
```

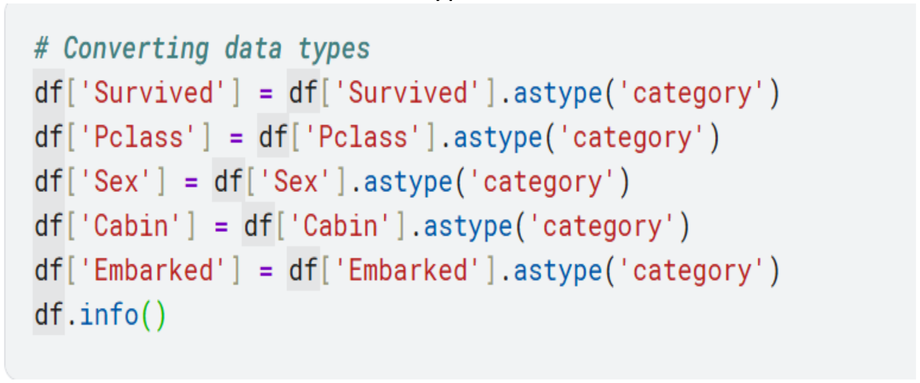
- To get the summary of the dataset, columns and data types:
- See the image below:
- Run the code!



```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype   
---  ------            --------------  -----   
 0   Survived          891 non-null    category
 1   PassengerClass    891 non-null    category
 2   Gender            891 non-null    category
 3   Age               714 non-null    float64 
 4   SiblingsSpouses   891 non-null    int64   
 5   ParentsChildren   891 non-null    int64   
 6   Ticket            891 non-null    object  
 7   Fare              891 non-null    float64 
 8   Cabin             204 non-null    category
 9   Embarked          889 non-null    category
dtypes: category(5), float64(2), int64(2), object(1)
memory usage: 45.9+ KB
```

- Converted the data types as shown below:

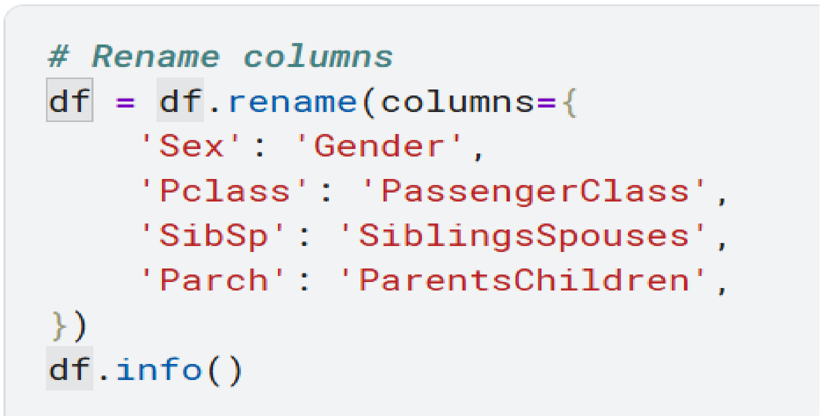


```python
# Converting data types
df['Survived'] = df['Survived'].astype('category')
df['Pclass'] = df['Pclass'].astype('category')
df['Sex'] = df['Sex'].astype('category')
df['Cabin'] = df['Cabin'].astype('category')
df['Embarked'] = df['Embarked'].astype('category')
df.info()
```

- The output is as shown below:

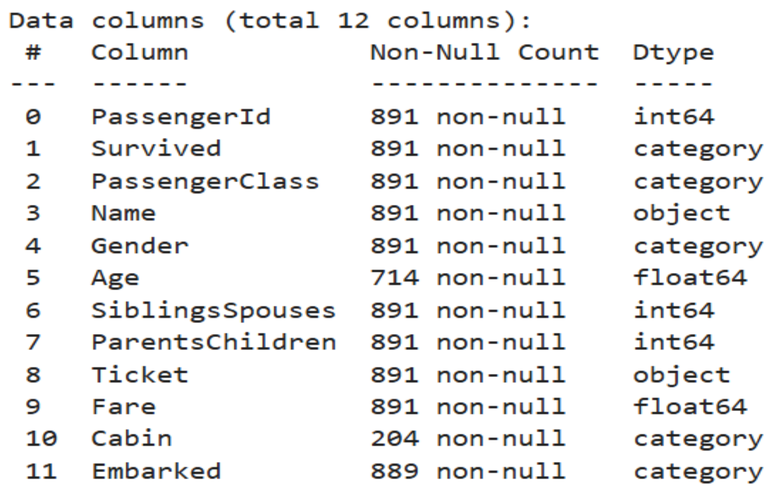
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    category
 2   Pclass       891 non-null    category
 3   Name         891 non-null    object
 4   Sex          891 non-null    category
 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    category
 11  Embarked     889 non-null    category
dtypes: category(5), float64(2), int64(3), object(2)
memory usage: 59.8+ KB
```

- I noticed that some of the columns are improperly named.
- The following renaming of the columns applied:



```python
# Rename columns
df = df.rename(columns={
    'Sex': 'Gender',
    'Pclass': 'PassengerClass',
    'SibSp': 'SiblingsSpouses',
    'Parch': 'ParentsChildren',
})
df.info()
```

- The output:



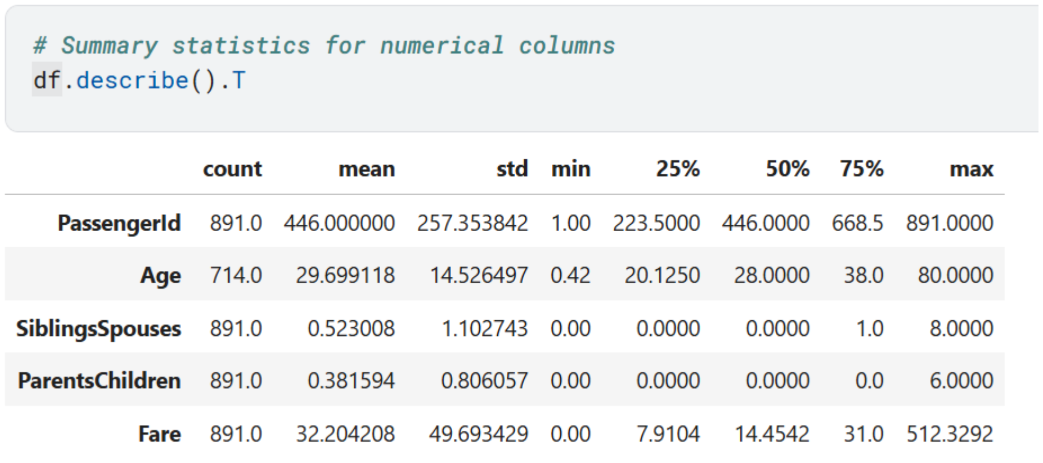
```
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   PassengerId      891 non-null    int64
 1   Survived         891 non-null    category
 2   PassengerClass   891 non-null    category
 3   Name             891 non-null    object
 4   Gender           891 non-null    category
 5   Age              714 non-null    float64
 6   SiblingsSpouses  891 non-null    int64
 7   ParentsChildren  891 non-null    int64
 8   Ticket           891 non-null    object
 9   Fare             891 non-null    float64
 10  Cabin            204 non-null    category
 11  Embarked         889 non-null    category
```

Statistical summary of numerical columns

- The .describe() function generates summary statistics for numerical columns:
- What is shows:
  - **Count:** Number of non-missing values.
  - **Mean:** Average value of the column.
  - **Std (Standard Deviation):** How much the values vary around the mean.
  - **Min/Max**: Minimum and maximum values.
  - **25%, 50%, 75%:** Percentiles (quartiles). The 50% value is the median.

To Use .describe():

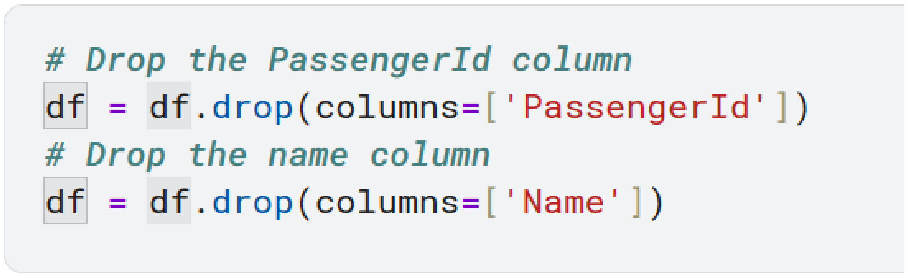
- **Check for unusual ranges:** For example, the Age column in Titanic should ideally range between 0 and 100—if you see negative ages or values over 150, something is off.
- **Detect skewness:** Large differences between the mean and median suggest a skewed distribution (common with variables like Fare).
- **Detect errors:** Look for outliers, such as extremely high or low values that don’t make sense.
- See in the image below:



```python
# Summary statistics for numerical columns
df.describe().T
```

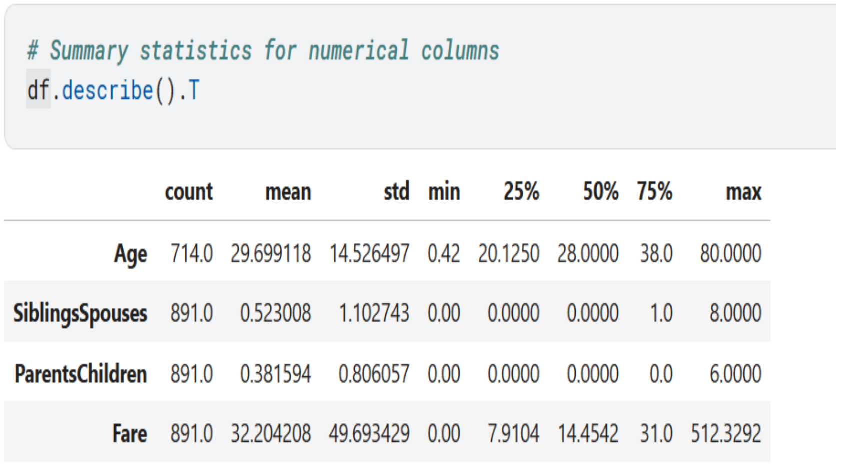
	count	mean	std	min	25%	50%	75%	max
PassengerId	891.0	446.000000	257.353842	1.00	223.5000	446.0000	668.5	891.0000
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SiblingsSpouses	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
ParentsChildren	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

- I noticed that there was no need for the passenger Id column since it was not going to be of any use in the analysis.
- So I opted to drop it.
- The same applied for the Name columns.
- See in the image below:



```python
# Drop the PassengerId column
df = df.drop(columns=['PassengerId'])
# Drop the name column
df = df.drop(columns=['Name'])
```

- To see the changes applied:



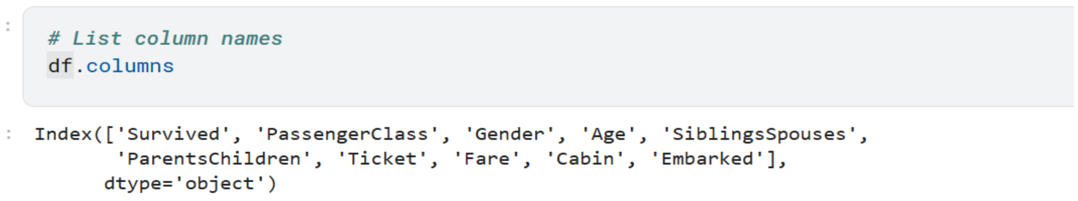
```python
# Summary statistics for numerical columns
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SiblingsSpouses	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
ParentsChildren	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

- To view the features:

Viewing Column Names

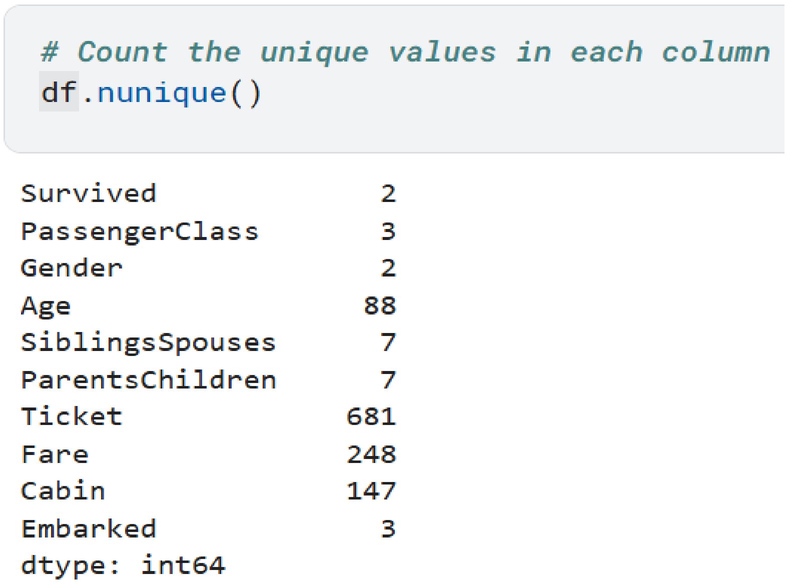
- Helps you quickly scan the names of features.
- Useful if you need to rename columns or drop irrelevant columns or investigate any inconsistency.



```
# List column names
df.columns
```

```
Index(['Survived', 'PassengerClass', 'Gender', 'Age', 'SiblingsSpouses',
       'ParentsChildren', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

- To check for unique Values in Each Column:

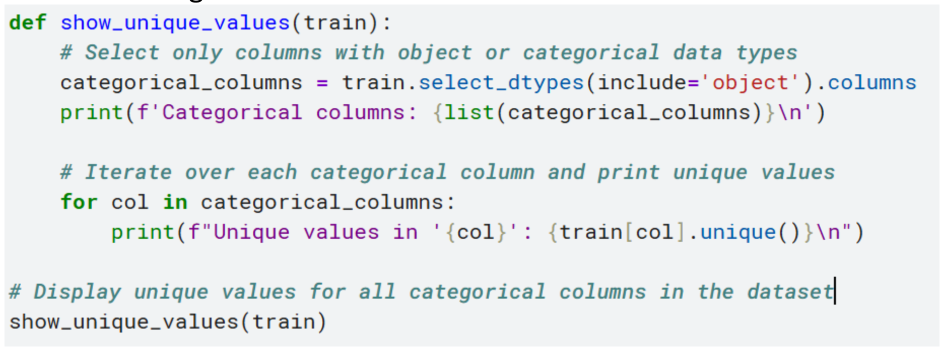


```
# Count the unique values in each column
df.nunique()
```

```
Survived             2
PassengerClass       3
Gender               2
Age                 88
SiblingsSpouses      7
ParentsChildren      7
Ticket             681
Fare               248
Cabin              147
Embarked             3
dtype: int64
```

- The following function I used to see the unique values inside each column.
- This helped to see inconsistency.
    - Function to display unique values for categorical variables:

- See In the image below:

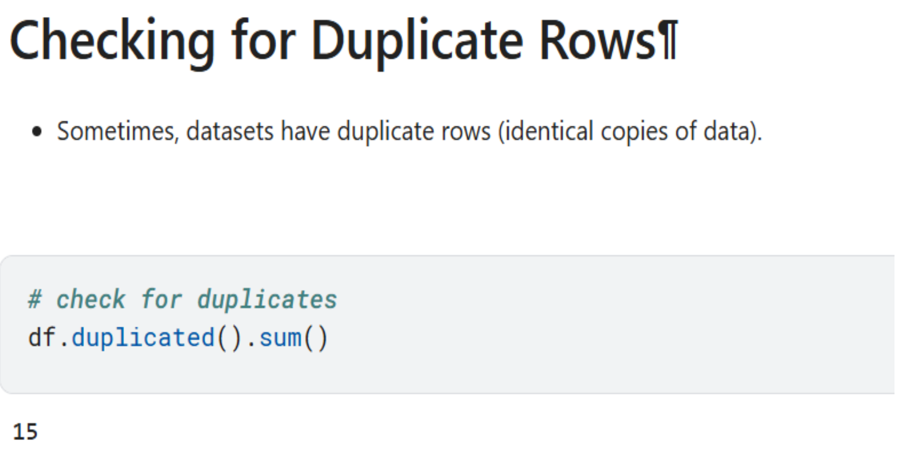


```
def show_unique_values(train):
    # Select only columns with object or categorical data types
    categorical_columns = train.select_dtypes(include='object').columns
    print(f'Categorical columns: {list(categorical_columns)}\n')

    # Iterate over each categorical column and print unique values
    for col in categorical_columns:
        print(f"Unique values in '{col}': {train[col].unique()}\n")

# Display unique values for all categorical columns in the dataset
show_unique_values(train)
```

- To check for duplicate rows:



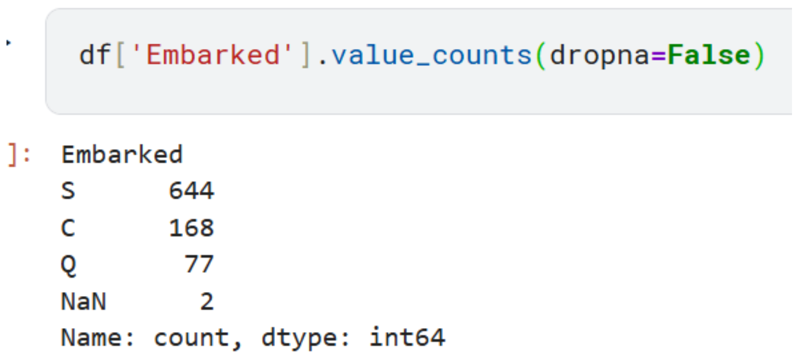
Checking for Duplicate Rows¶

- Sometimes, datasets have duplicate rows (identical copies of data).

```
# check for duplicates
df.duplicated().sum()
```

```
15
```

- The output showed a sum of 15 duplicate rows.
- In order to know the values for the Embarked feature:



```
df['Embarked'].value_counts(dropna=False)
```

```
]:  Embarked
    S      644
    C      168
    Q       77
    NaN      2
    Name: count, dtype: int64
```

- From the above initial data exploration, I have seen that:
  - Some columns have missing data (e.g., Cabin in Titanic).
  - Some columns contain unusually high or low values.
  - Columns like PassengerId may not contribute to the analysis which is why I dropped it as illustrated.
  - Columns with dates may be stored as strings, or numerical data may be stored as text.
  - There were duplicate rows and learned how to remove them if needed.

- **KEY FINDINGS FROM THE INITAIL DATA EXPLORATION STAGE**
  a) The dataset contains **891 rows** and **12 columns**.
  b) There were 577 male,314 female
  c) Fare values range widely, indicating a strong link with Pclass. There are significant outliers, especially in First Class, with some passengers paying more than £500.
  d) Embarked: Southampton (S): 644, Cherbourg (C): 168, Queenstown (Q): 77
  e) 0 = Did not survive: 549 passengers,1 = Survived: 342 passengers. This sets the baseline survival rate at 38.4%.

STEP 2: **HANDLING MISSING VALUES AND OUTLIERS**

- In the initial data exploration, I noticed that there were missing values.
- It was important to handle them appropriately.

- Why did I need them handled? Because ignoring them would lead to:
    - **Bias in analysis**: Missing data may skew averages or distributions.
    - **Errors in computations**: Some machine learning algorithms cannot handle missing values and may crash.
    - **Loss of valuable information**: Dropping rows or columns without thought can reduce the size of your dataset unnecessarily.
- The common methods of handling missing values:
    - Drop data
    - Impute data
    - Flag missing

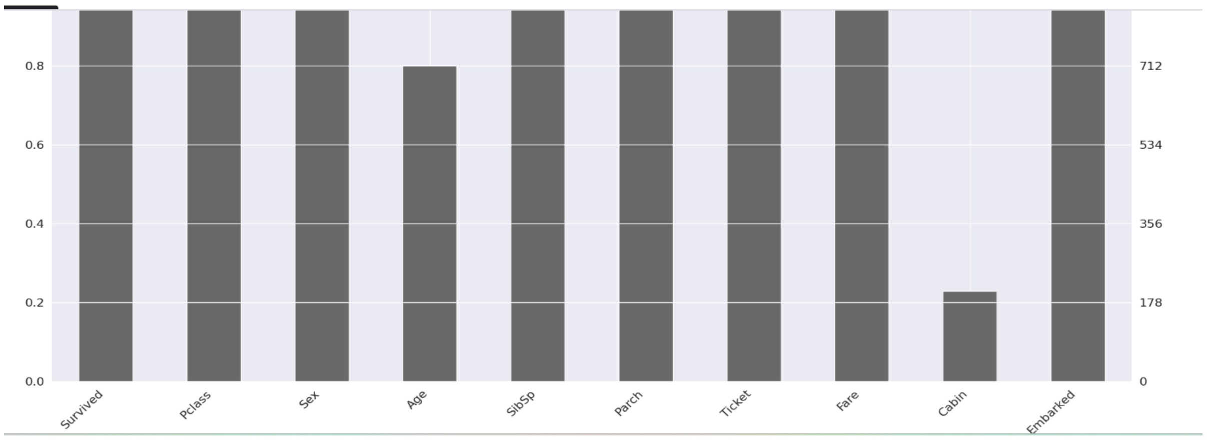
a) Visualizing Missing Data

~Before deciding how to handle missing values, it was helpful to visualize the missing data.

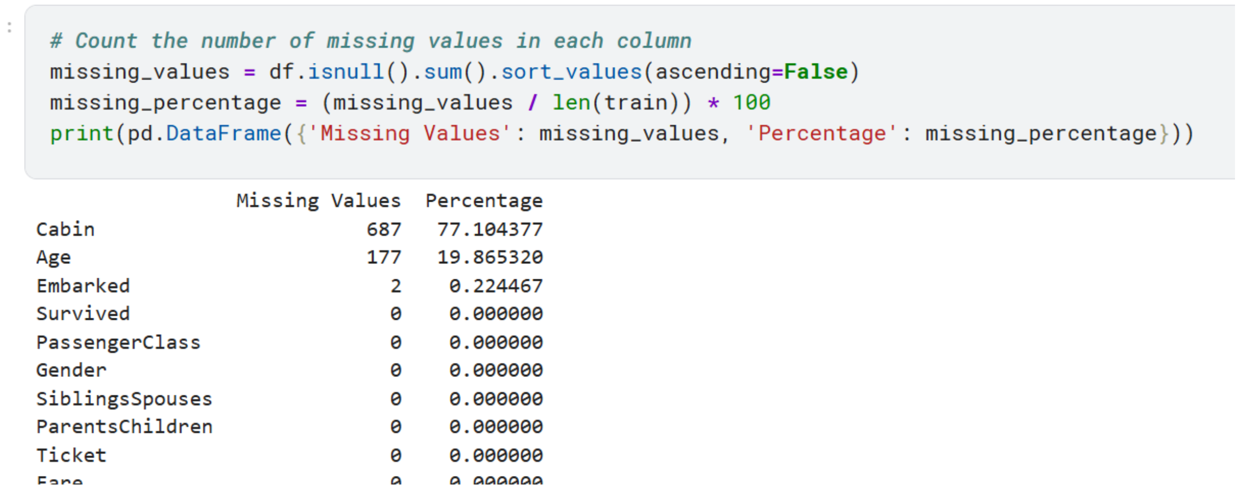
~I useD msno.bar(df) which shows the missing values of each column in a bar chart.

```
# Visualize missing data using missingno library
import missingno as msno
msno.bar(train)
```

- See the output:



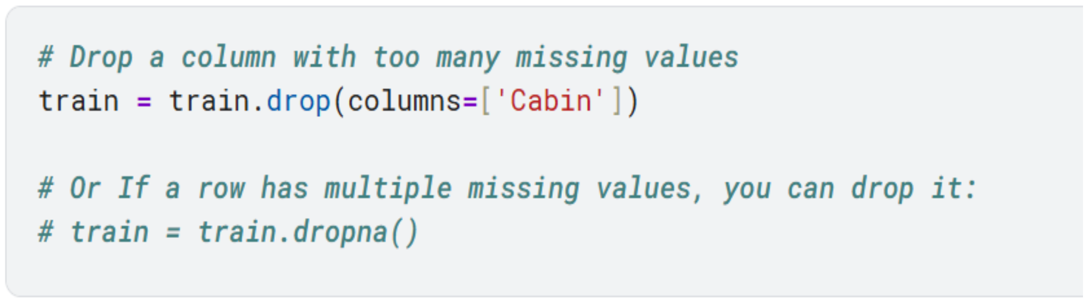
Detect missing values



```
# Count the number of missing values in each column
missing_values = df.isnull().sum().sort_values(ascending=False)
missing_percentage = (missing_values / len(train)) * 100
print(pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_percentage}))
```

```
                 Missing Values  Percentage
Cabin                       687   77.104377
Age                         177   19.865320
Embarked                      2    0.224467
Survived                      0    0.000000
PassengerClass                0    0.000000
Gender                        0    0.000000
SiblingsSpouses               0    0.000000
ParentsChildren               0    0.000000
Ticket                        0    0.000000
Fare                          0    0.000000
```

Drop missing values

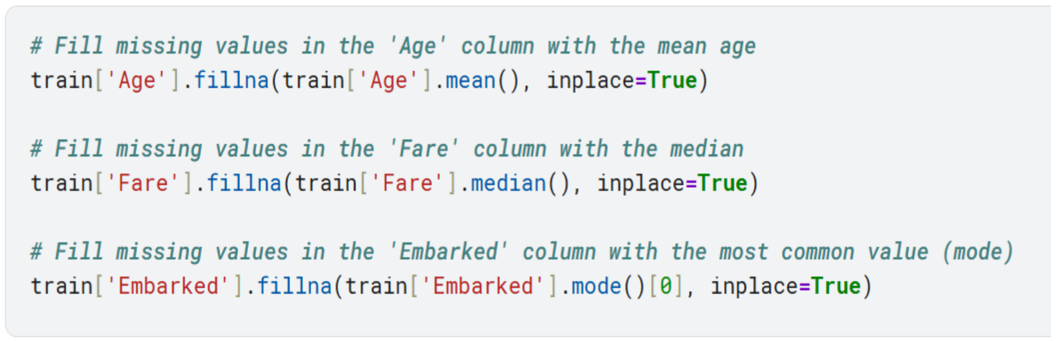


```
# Drop a column with too many missing values
train = train.drop(columns=['Cabin'])

# Or If a row has multiple missing values, you can drop it:
# train = train.dropna()
```

Imputing Missing Values

- If I wanted to fill in missing values instead of dropping them, I could use the following methods:
  - Fill with Mean: Useful for numerical columns that follow a normal distribution (bell curve).
  - Fill with Median: Better for numerical columns with outliers or skewed data.
  - Fill with Mode: Best for categorical columns.
- See in the image below:



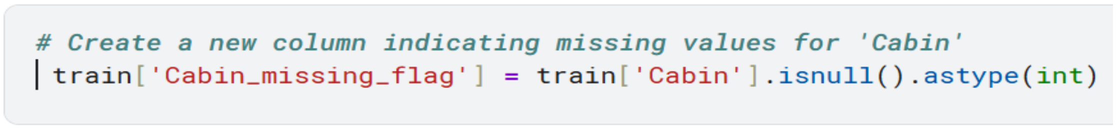
```
# Fill missing values in the 'Age' column with the mean age
train['Age'].fillna(train['Age'].mean(), inplace=True)

# Fill missing values in the 'Fare' column with the median
train['Fare'].fillna(train['Fare'].median(), inplace=True)

# Fill missing values in the 'Embarked' column with the most common value (mode)
train['Embarked'].fillna(train['Embarked'].mode()[0], inplace=True)
```

- To flag the missing

Flagging missing values



```python
# Create a new column indicating missing values for 'Cabin'
train['Cabin_missing_flag'] = train['Cabin'].isnull().astype(int)
```

KEY FINDINGS FROM THIS STAGE:

- **Missing values** :
  - Age:177 missing entries
  - Cabin :687 missing entries
  - Embarked: 2 missing entries
- **Outliers Detected:** Some passengers paid very high fares (e.g., > £500).
- Outliers in Fare and Age are retained as they reflect real-world variation rather than data issues.
- Minimal missing data in Embarked implies low risk to analysis.

STEP 3: **UNIVARIATE ANALYSIS**

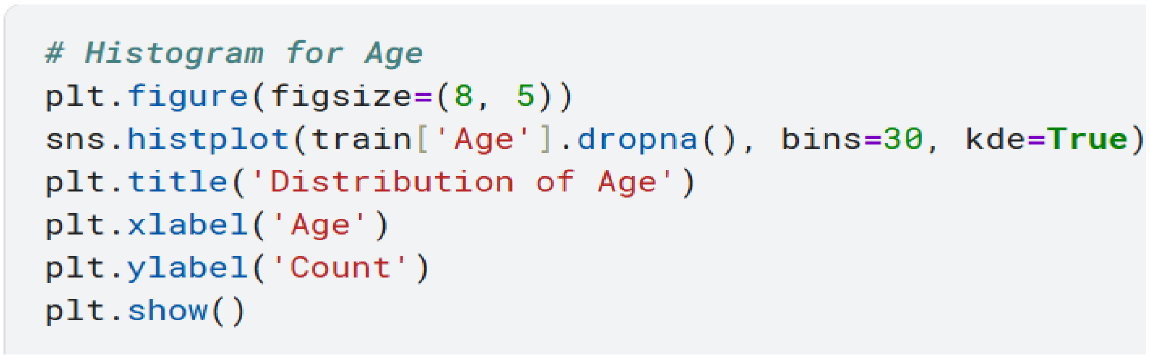
- This focuses on one variable at a time.
- This helps understand its distribution and seek to answer questions like:
  1. What is the age distribution of passengers?
  2. How many passengers embarked from each location?
  3. Are ticket prices evenly distributed, or are they skewed?

Univariate Analysis for Numerical Columns

1. Histogram: Shows the frequency distribution of the values.
2. KDE (Kernel Density Estimate) Plot: A smoothed version of the histogram to visualize the probability density.
3. Boxplot: Displays the distribution and highlights outliers.

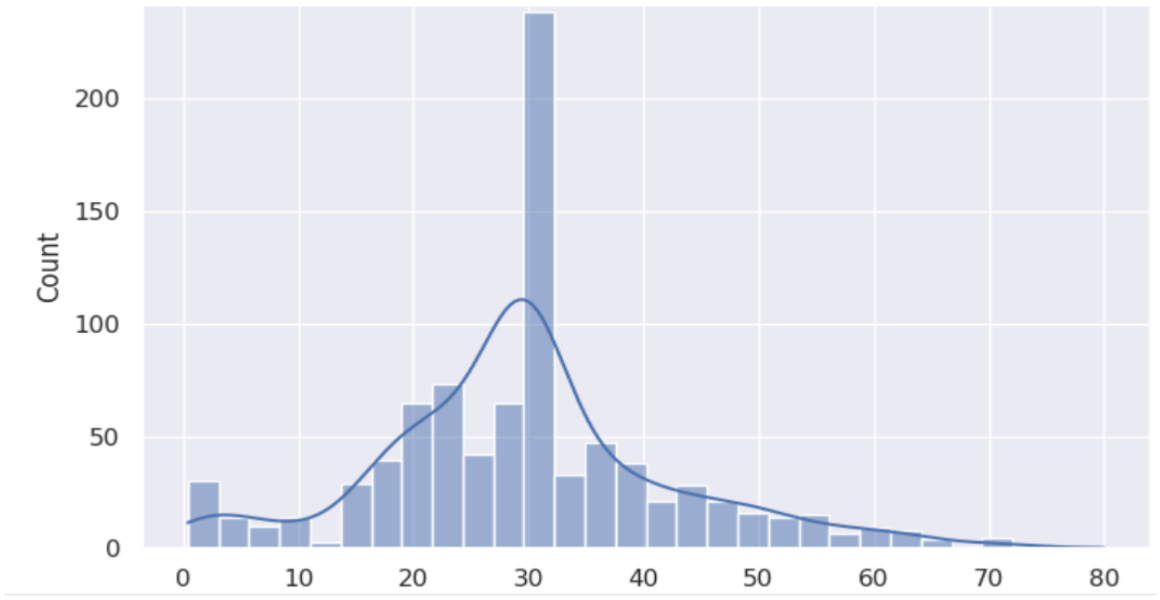
HISTOGRAM

- The following code applied in displaying a Histogram for the Age feature.



```python
# Histogram for Age
plt.figure(figsize=(8, 5))
sns.histplot(train['Age'].dropna(), bins=30, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

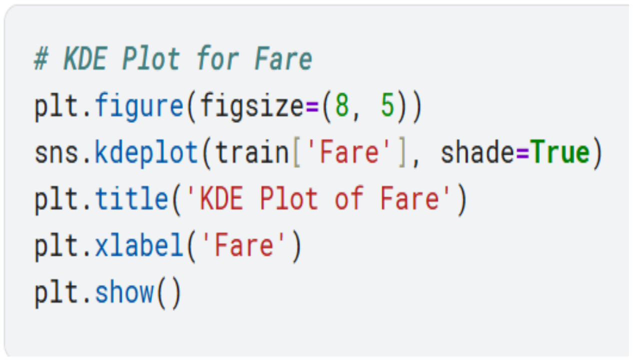
- Run the code.
- See the image below for the output



- The peak shows then common age group is around 30.

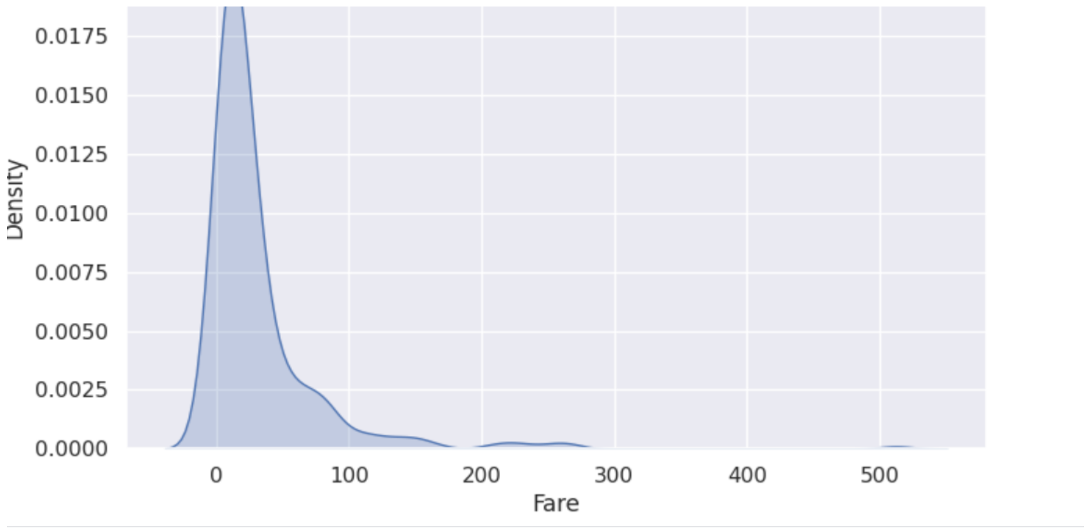
KDE PLOT

- The KDE plot shows the probability density function of a numerical column:



```
# KDE Plot for Fare
plt.figure(figsize=(8, 5))
sns.kdeplot(train['Fare'], shade=True)
plt.title('KDE Plot of Fare')
plt.xlabel('Fare')
plt.show()
```

- The output is as follows:



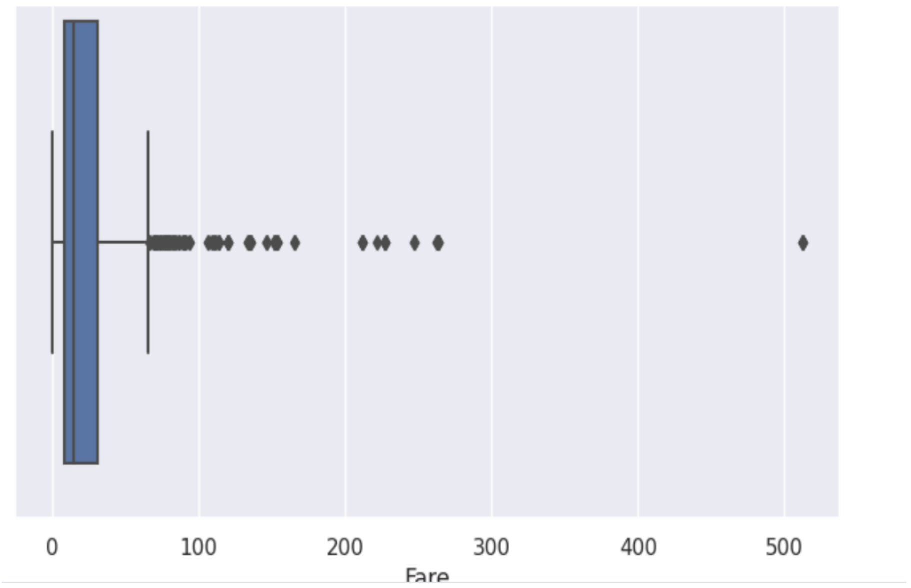
BOXPLOT

- Boxplots visualize the minimum, lower quartile (25%), median, upper quartile (75%), and maximum values:
  Interpretation:
  - **Line in the middle:** Median (50% value).
  - **Box edges:** 25th percentile (Q1) and 75th percentile (Q3).
  - **Whiskers:** Minimum and maximum values (excluding outliers).
  - **Dots outside the whiskers:** Outliers.

The following code applied for the Boxplot for the Fare feature/column:

```python
# Boxplot for Fare
plt.figure(figsize=(8, 5))
sns.boxplot(x=train['Fare'])
plt.title('Boxplot of Fare')
plt.show()
```

The output is as follows:

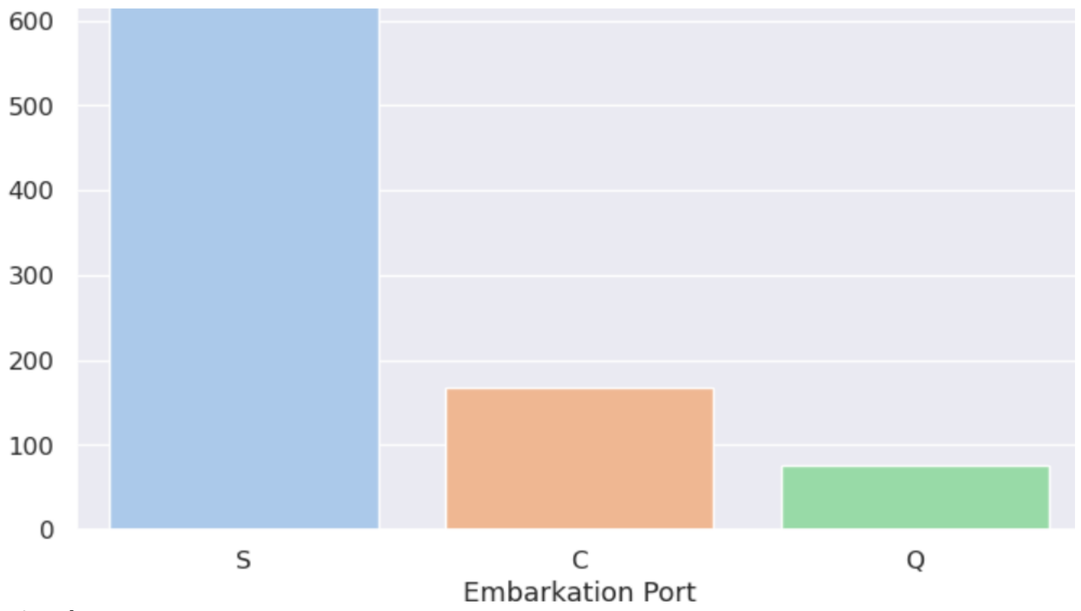


Univariate Analysis for Categorical Columns

- **Countplot:** Shows the frequency count of each category.
- **Pie Chart:** Shows proportions of categories in a pie format (less commonly used in data science).

```python
# Countplot for Embarked
plt.figure(figsize=(8, 5))
sns.countplot(x='Embarked', data=train, palette='pastel')
plt.title('Countplot of Embarked')
plt.xlabel('Embarkation Port')
plt.ylabel('Count')
plt.show()
```

The output is as follows:



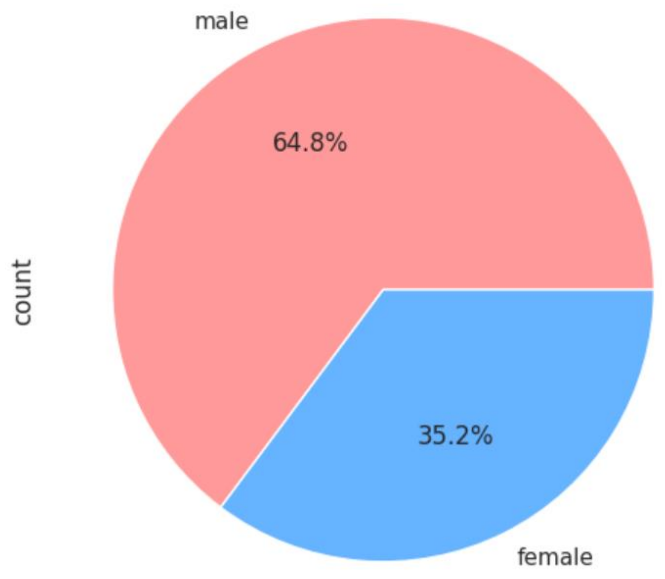
Pie Chart

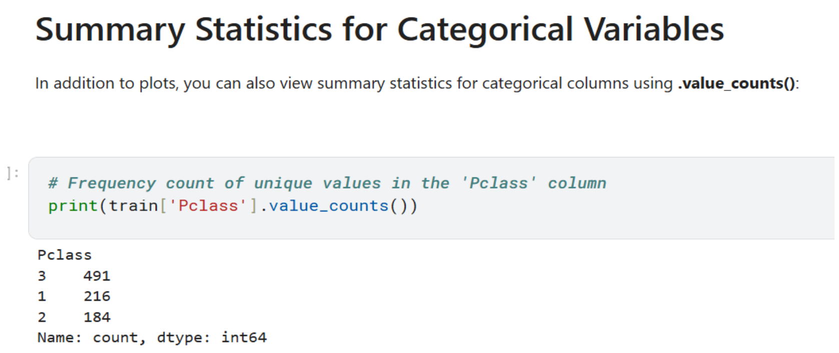
While pie charts are visually appealing, they're generally less informative than bar charts for categorical data.

Why Use a Pie Chart?

- Useful for displaying proportions (e.g., percentage of males vs. females).
- Avoid using them when you have more than 3-4 categories.

```
# Pie chart for Sex distribution
train['Sex'].value_counts().plot.pie(autopct='%1.1f%%', figsize=(6, 6), colors=['#ff9999', '#66b3ff'])
plt.title('Sex Distribution')
plt.show()
```





Summary Statistics for Categorical Variables

In addition to plots, you can also view summary statistics for categorical columns using **.value_counts()**:

```
# Frequency count of unique values in the 'Pclass' column
print(train['Pclass'].value_counts())
```

```
Pclass
3    491
1    216
2    184
Name: count, dtype: int64
```

From the Univariate analysis:

- ✓ Used histograms, KDE plots, and boxplots for numerical columns to understand distributions and outliers.
- ✓ Used countplots and pie charts to explore categorical columns.
- ✓ Used .value_counts() to summarize the frequency of categories.

KEY FINDINGS FROM THE UNIVARIATE ANALYISIS

- Survived
  - Around **38%** of passengers survived.
  - Visualization: Countplot shows fewer survivors than non-survivors.
- Pclass (Passenger Class)
  - Most passengers were in Third Class (Pclass = 3).
  - First Class had the fewest passengers.
  - Suggests socioeconomic diversity and will likely influence survival.
- Sex
  - There were more males (577) than females (314).
  - Gender distribution is skewed toward males, which will be significant in survival analysis.
- Age
  - Distribution is right-skewed, with most passengers aged 20–40 years.
  - Presence of infants and elderly passengers, with some outliers above age 70.
  - A small number of missing values were observed.
- Fare
  - Distribution is heavily right-skewed with extreme outliers (some fares > £500).
  - Most fares are under £100, with a high concentration around £10–30.
  - Reflects class-related pricing, suggesting a potential relationship with Pclass.
- SibSp (Siblings/Spouses aboard)
  - Majority of passengers had 0 siblings/spouses aboard.
  - Few passengers had large family groups (SibSp > 3).
  - Could suggest many traveled alone or in small family units.
- Parch (Parents/Children aboard)
  - Most passengers had no parents or children aboard.
  - A small portion had 1 or 2 dependents, with very few having more.
- Embarked
  - Southampton (S) was the most common port of embarkation.
  - Fewer passengers boarded at Cherbourg (C) and Queenstown (Q).
  - Only 2 entries were missing.

STEP 4: BIVARIATE ANALYSIS

- **Bivariate Analysis goal**: Examine pairs of features of interest. Justify which features you would like to pair in the analysis and seek to answer questions like?
    i. Does the Fare change depending on the Pclass?
    ii. Are younger passengers more likely to survive on the Titanic?
    iii. Does the Embarked location affect survival rate?
- By comparing two variables, one can detect **correlations, trends, and group-level differences**.

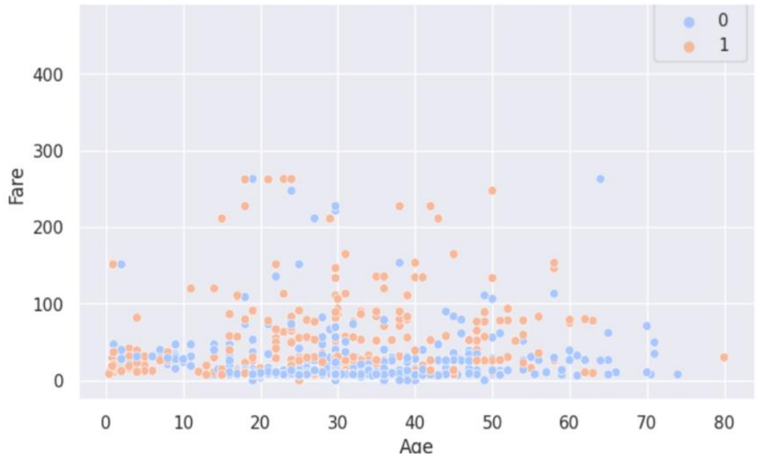
i. Numerical vs Numerical Analysis

1. Scatter Plot

- Scatter plots are used to visualize the **relationship between two numerical variables**
- The following code applied for Age vs Fare:

```python
# Scatter plot for Age vs Fare
plt.figure(figsize=(8, 5))
sns.scatterplot(x='Age', y='Fare', data=train, hue='Survived', palette='coolwarm')
plt.title('Scatter Plot of Age vs Fare (Colored by Survived)')
plt.show()
```

- The output:



2. Correlation Heatmap

A correlation heatmap shows the strength and direction of relationships between numerical variables:

What the Heatmap Shows:

- **Positive correlations** (closer to 1): Variables increase together (e.g., age and fare).
- **Negative correlations** (closer to -1): One variable decreases as the other increases.
- **Values near 0**: No clear relationship between variables.

- The following code applied for correlation heatmap for numerical columns:

```python
# Correlation heatmap for numerical columns only
plt.figure(figsize=(8, 6))
numerical_columns = train.select_dtypes(include=['int64', 'float64']).columns  # Select only numerical columns
sns.heatmap(train[numerical_columns].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

Numerical vs Categorical Analysis

1. Boxplot

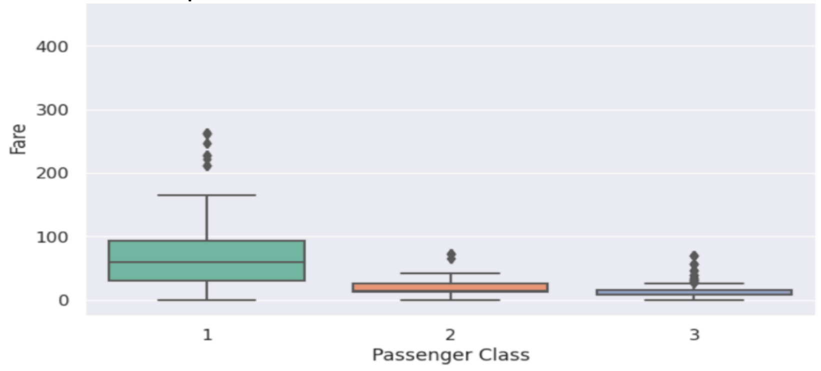
Boxplots are great for visualizing the distribution of **numerical values grouped by categories**:

Interpretation:

- **Length of the box**: Shows the interquartile range (IQR).
- **Line inside the box**: Median fare for each Pclass.
- **Outliers** may indicate passengers who paid abnormally high fares (luxury tickets).

```python
# Boxplot of Fare grouped by Pclass
plt.figure(figsize=(8, 5))
sns.boxplot(x='Pclass', y='Fare', data=train, palette='Set2')
plt.title('Boxplot of Fare by Pclass')
plt.xlabel('Passenger Class')
plt.ylabel('Fare')
plt.show()
```

- The output:



2. Violin Plot

A violin plot is similar to a boxplot but also shows the **density of the data**:

Interpretation:

- Wider parts of the plot show where data points are concentrated.
- Use violin plots when you want to see both distribution and summary statistics.

Categorical vs Categorical Analysis

1. Grouped Bar Plot

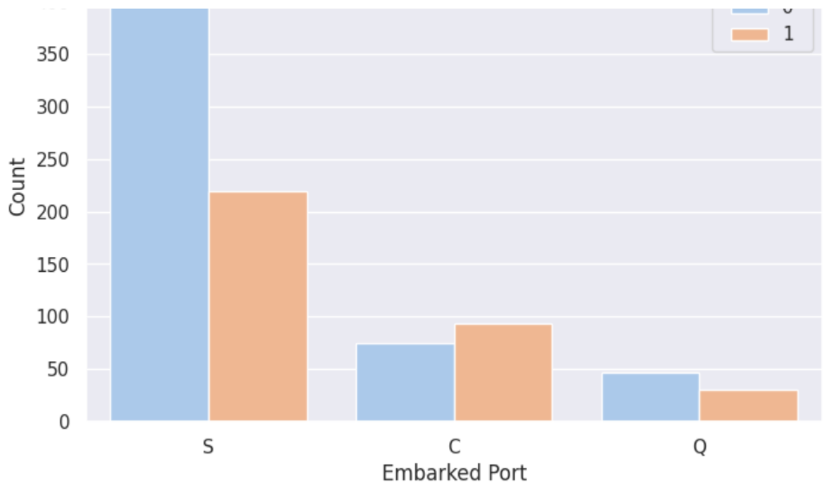
Grouped bar plots show the count or **proportion of one category for each level of another category**:

Interpretation:

- This plot shows how survival rates differ across embarkation ports (Embarked).
- Large differences between the bar heights suggest that the embarkation location influenced survival chances.

```python
# Grouped bar plot of Survived vs Embarked
plt.figure(figsize=(8, 5))
sns.countplot(x='Embarked', hue='Survived', data=train, palette='pastel')
plt.title('Survival Counts by Embarked Port')
plt.xlabel('Embarked Port')
plt.ylabel('Count')
plt.show()
```

- The following was the output:



2. Mosaic Plot

Mosaic plots show the **proportion of categories across different groups**:

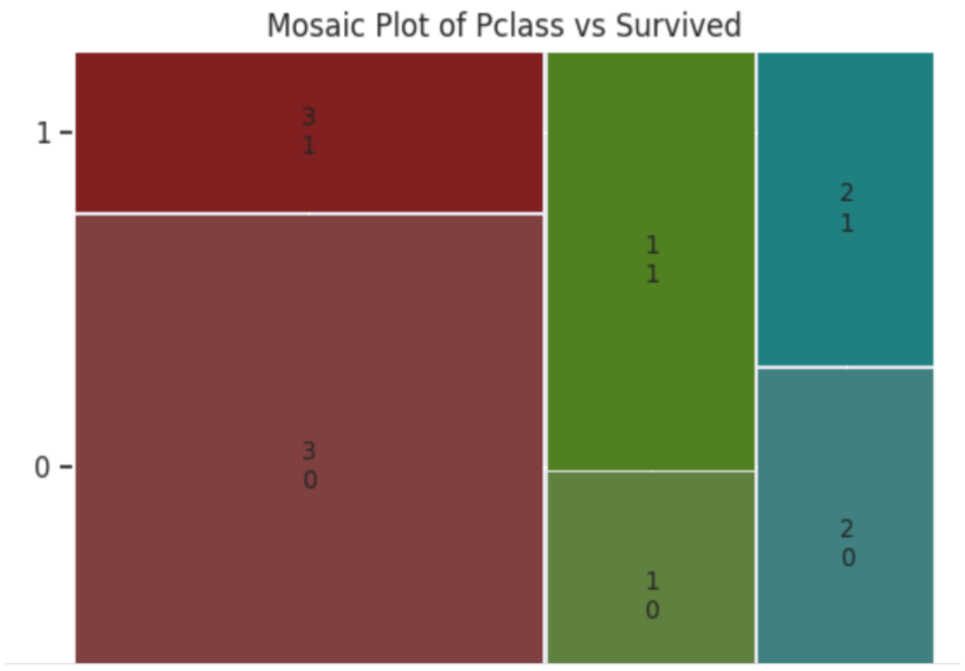
Interpretation:

- Larger blocks indicate more frequent category combinations (e.g., "Pclass 3 + Did Not Survive" may have a large block).
- Helps detect imbalances between groups.

```python
# Install required library for mosaic plot
# !pip install statsmodels
from statsmodels.graphics.mosaicplot import mosaic
from itertools import product

# Mosaic plot of Pclass vs Survived
plt.figure(figsize=(10, 10))
mosaic(train, ['Pclass', 'Survived'], title='Mosaic Plot of Pclass vs Survived')
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



KEY FINDINGS FROM BIVARIATE ANALYSIS

- Survival vs Sex
  - Females had a significantly higher survival rate compared to males.
  - Strong relationship between gender and survival supports the "women and children first" evacuation practice.
  - Visualized using a countplot: taller bars for females in the "Survived" category.
- Survival vs Pclass
  - First-Class passengers had the highest survival rate, followed by Second Class and Third Class.
  - Clear inverse relationship: the higher the class, the higher the survival rate.
- Survival vs Age
  - Children (<16) had a higher survival rate than adults.
  - Survival drops with age, especially for older males in lower classes.
  - Visualizations (like boxplots or histograms split by survival) show the difference in age distributions between survivors and non-survivors.
- Survival vs Fare
  - Survivors generally paid higher fares on average.
  - This again aligns with the class effect those who paid more (i.e., were in higher class cabins) had better survival chances.
  - Boxplots or violin plots highlight this difference visually.

STEP5 MULTIVARIATE ANALYSIS

- Explore more complex relationships between three or more variables simultaneously.
- Detect interactions, combined effects, and hidden patterns that may not be visible in bivariate analysis.
- This can help answer complex questions, such as:

  1. How do Pclass, Age, and Fare jointly affect survival?
  2. Are survival rates different for Embarked locations when considering Pclass?

Pair Plot

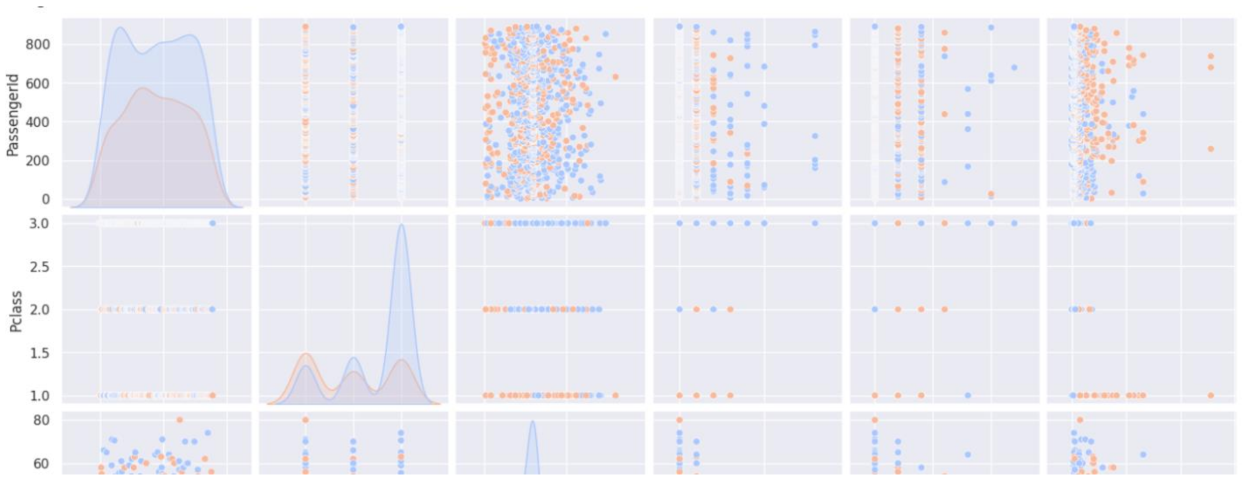
A pair plot shows **scatter plots and histograms** for all numerical variable pairs:

Interpretation:

- The **diagonal plots** are KDE plots showing the distribution of individual variables.
- The **off-diagonal scatter** plots show pairwise relationships (e.g., Age vs Fare).
- **Hue (color)** differentiates categories (e.g., Survived vs Did Not Survive).

```
# Pair plot for numerical columns
plt.figure(figsize=(10, 10))
sns.pairplot(train, hue='Survived', diag_kind='kde', palette='coolwarm')
plt.show()
```

- The following output is displayed. Check out the link to the Notebook shared before the conclusion for visible details.



FacetGrid (Subplots for Subgroups)

FacetGrid creates multiple subplots for **different subsets of data based on categorical variables**:

Interpretation:

- This plot shows how the age distribution differs based on Survived (columns) and Pclass (rows).
- You can see patterns like younger passengers in Pclass 1 having higher survival rates.

💡 Tip: FacetGrid is useful for detecting interactions between multiple variables.

```
# FacetGrid for Age distribution by Survived and Pclass
g = sns.FacetGrid(train, col='Survived', row='Pclass', height=4, aspect=1.5)
g.map(sns.histplot, 'Age', kde=True)
plt.show()
```

Correlation Heatmap for Numerical Columns

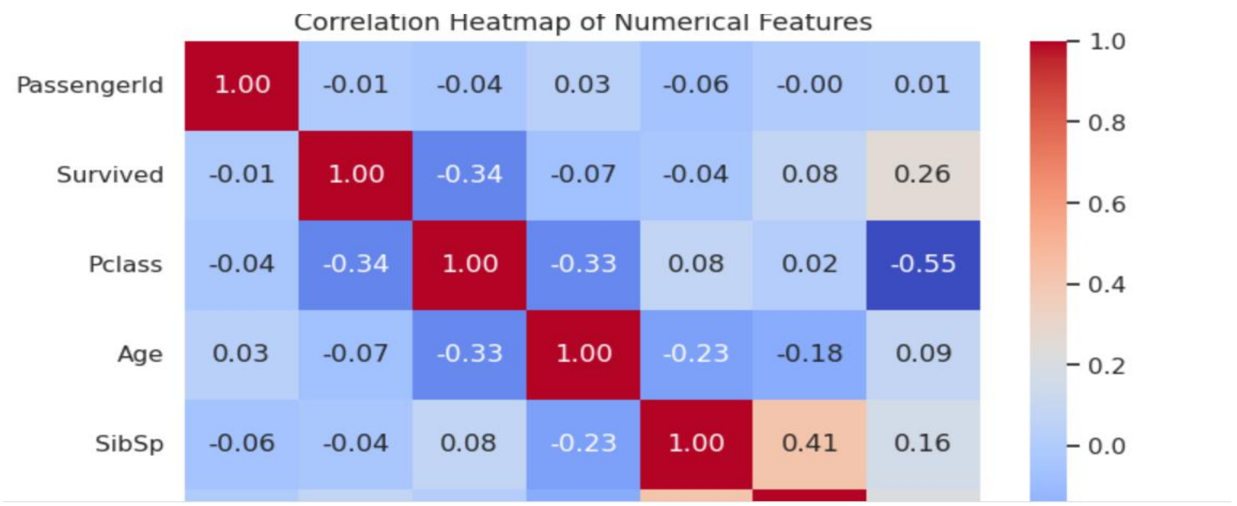
A heatmap shows the **correlation between multiple numerical variables**:

What the Heatmap Shows:

- **Strong positive correlations** (closer to +1) indicate that variables increase together.
- **Strong negative correlations** (closer to -1) indicate that as one variable increases, the other decreases.
- Correlation values **close to zero** indicate no strong relationship.

```python
# Heatmap of numerical features only
plt.figure(figsize=(8, 6))
numerical_columns = train.select_dtypes(include=['int64', 'float64']).columns  # Select only numerical columns
sns.heatmap(train[numerical_columns].corr(), annot=True, cmap='Blues', fmt='.2f')
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

- The following output is displayed. Check out the link to the Notebook shared before the conclusion for visible details.



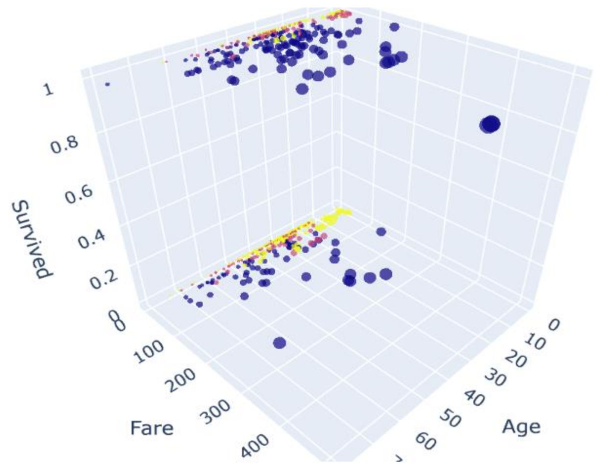
3D SCATTER PLOT

- The following code applied:

```python
# 3D scatter plot for Age, Fare, and Survived
import plotly.express as px

fig = px.scatter_3d(train, x='Age', y='Fare', z='Survived', color='Pclass', size='Fare', opacity=0.7)
fig.update_traces(marker=dict(line=dict(width=0)))
fig.update_layout(title='3D Scatter Plot: Age vs Fare vs Survived')
fig.show()
```

- The output:

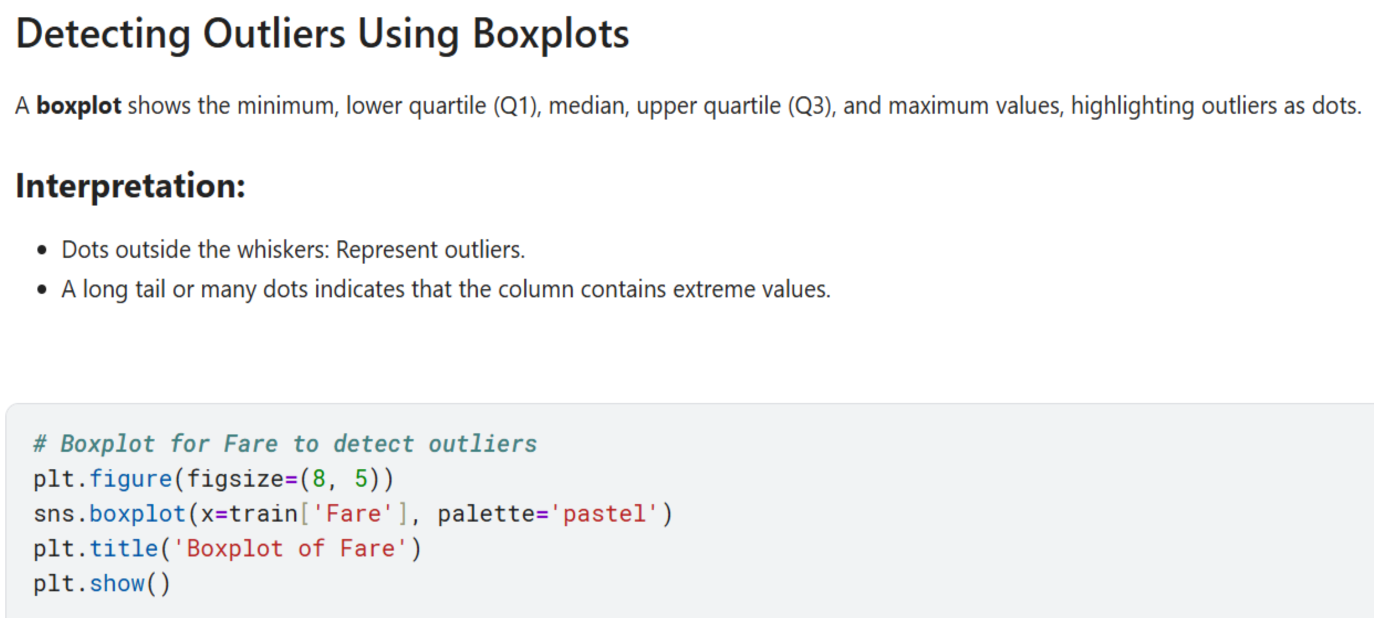


KEY FINDINGS FROM THE MULTIVARIATE ANALYSIS

- ✓ Male passengers in Third Class had the lowest survival rate, often below 20%.
- ✓ Older passengers in **lower classes** were the most likely to **not survive**.
- ✓ **Children in First and Second Class** were more likely to survive than those in Third Class.
- ✓ Cabin data is sparse but seems to correlate with **class and wealth**, indirectly affecting survival.
- ✓ Most Third-Class passengers boarded from **Southampton (S)** and had the lowest survival rate.
- ✓ **Combinations of age, class, sex, fare, and embarkation port** produce much stronger predictive patterns than individual variables.

STEP 6: OUTLIER DETECTION AND HANDLING

- There are different ways to handle outliers, which include removing, capping, imputing, or leaving them as is.
- Removing outliers in Fare may help for predictive models, but could hide important insights for understanding passenger wealth.



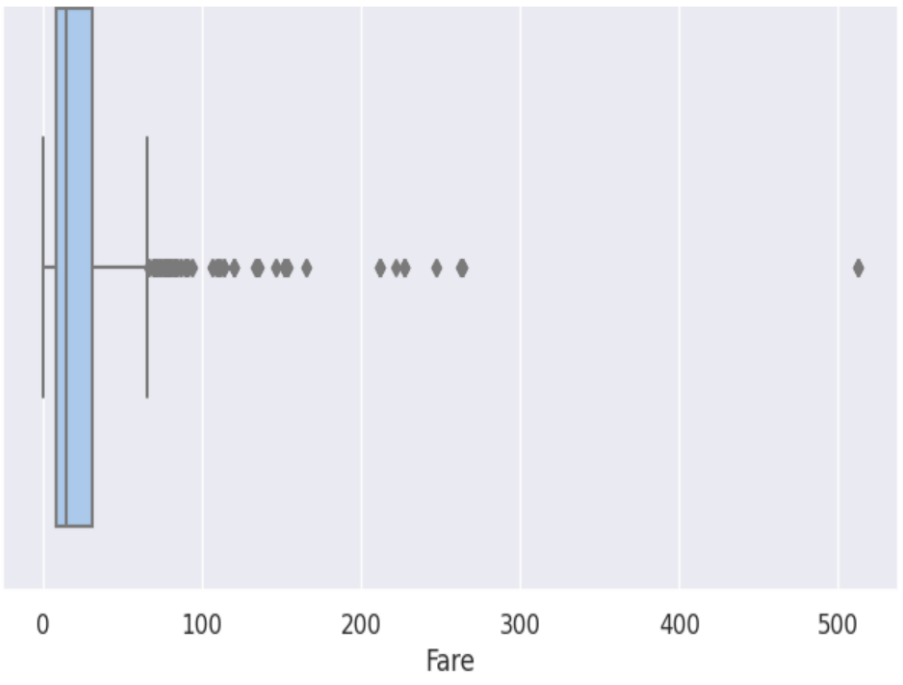
Detecting Outliers Using Boxplots

A **boxplot** shows the minimum, lower quartile (Q1), median, upper quartile (Q3), and maximum values, highlighting outliers as dots.

Interpretation:

- Dots outside the whiskers: Represent outliers.
- A long tail or many dots indicates that the column contains extreme values.

```python
# Boxplot for Fare to detect outliers
plt.figure(figsize=(8, 5))
sns.boxplot(x=train['Fare'], palette='pastel')
plt.title('Boxplot of Fare')
plt.show()
```



Detecting Outliers Using Z-Score

A **boxplot** shows the minimum, lower quartile (Q1), median, upper quartile (Q3), and maximum values, highlighting outliers as dots.

Interpretation:

- Data points with a Z-score greater than the threshold (typically 3) are considered outliers.
- A higher threshold (e.g., 4) detects fewer outliers, while a lower threshold (e.g., 2.5) detects more.

```python
# Function to detect outliers using Z-score
from scipy.stats import zscore

def detect_outliers_zscore(data, threshold=3):
    z_scores = zscore(data.dropna())  # Drop NaN to avoid errors
    outliers = data[(abs(z_scores) > threshold)]
    return outliers

# Detect outliers in the 'Age' column
outliers_age = detect_outliers_zscore(train['Age'])
print(f'Number of outliers in Age: {len(outliers_age)}')
```

The output:

```
Number of outliers in Age: 7
```

Detecting Outliers Using IQR (Interquartile Range)**

The IQR method identifies outliers as data points that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR:

Interpretation:

- Lower bound: Q1 - 1.5 * IQR (minimum expected value).
- Upper bound: Q3 + 1.5 * IQR (maximum expected value).
- Data points outside this range are considered outliers.

```
# Function to detect outliers using IQR
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data[(data < lower_bound) | (data > upper_bound)]
    return outliers

# Detect outliers in the 'Fare' column using IQR
outliers_fare = detect_outliers_iqr(train['Fare'])
print(f'Number of outliers in Fare: {len(outliers_fare)}')
```

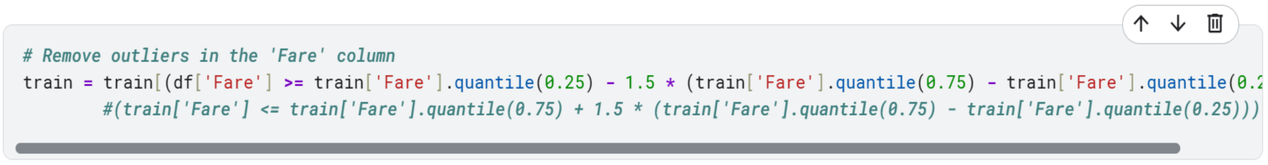
The following output applies:

```
Number of outliers in Fare: 116
```

Handling Outliers

Once outliers are detected, you can handle them using the following approaches:

- **Remove Outliers:** Remove rows containing outliers
- **Cap Outliers:** Cap values at the upper and lower bounds
- **Impute Outliers:** Replace outliers with mean or median values
- **Leave Outliers:** In some cases (e.g., fraud detection, rare event analysis), outliers contain meaningful information and should be kept.



```
# Remove outliers in the 'Fare' column
train = train[(df['Fare'] >= train['Fare'].quantile(0.25) - 1.5 * (train['Fare'].quantile(0.75) - train['Fare'].quantile(0.2
        #(train['Fare'] <= train['Fare'].quantile(0.75) + 1.5 * (train['Fare'].quantile(0.75) - train['Fare'].quantile(0.25)))
```

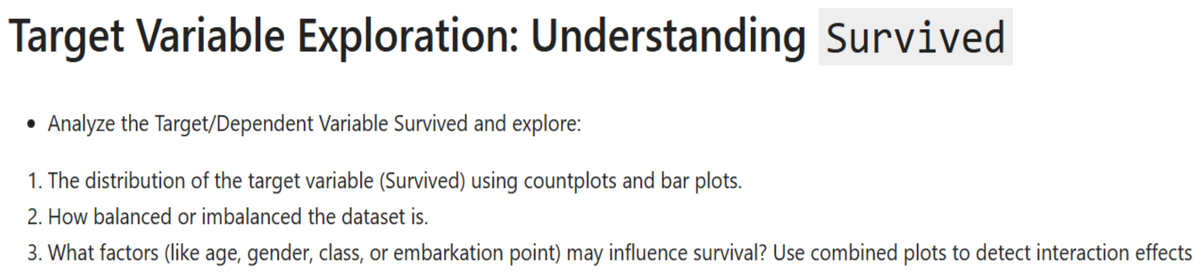
KEY FINDINGS FROM THE OUTLIER DETECTION AND HANDLING STAGE

- ✓ Several extreme fare values above £250, with the maximum around £512
- ✓ A few passengers were over 70 years old, but these were plausible ages (e.g., elderly travelers).
- ✓ Some passengers had unusually high numbers (e.g., SibSp = 8, Parch = 6), suggesting large family groups.
- ✓ Many missing values, but no numeric outliers to handle.

✓ Outliers were treated as potentially meaningful rather than errors.

STEP7: TARGET VARIABLE EXPLORATION

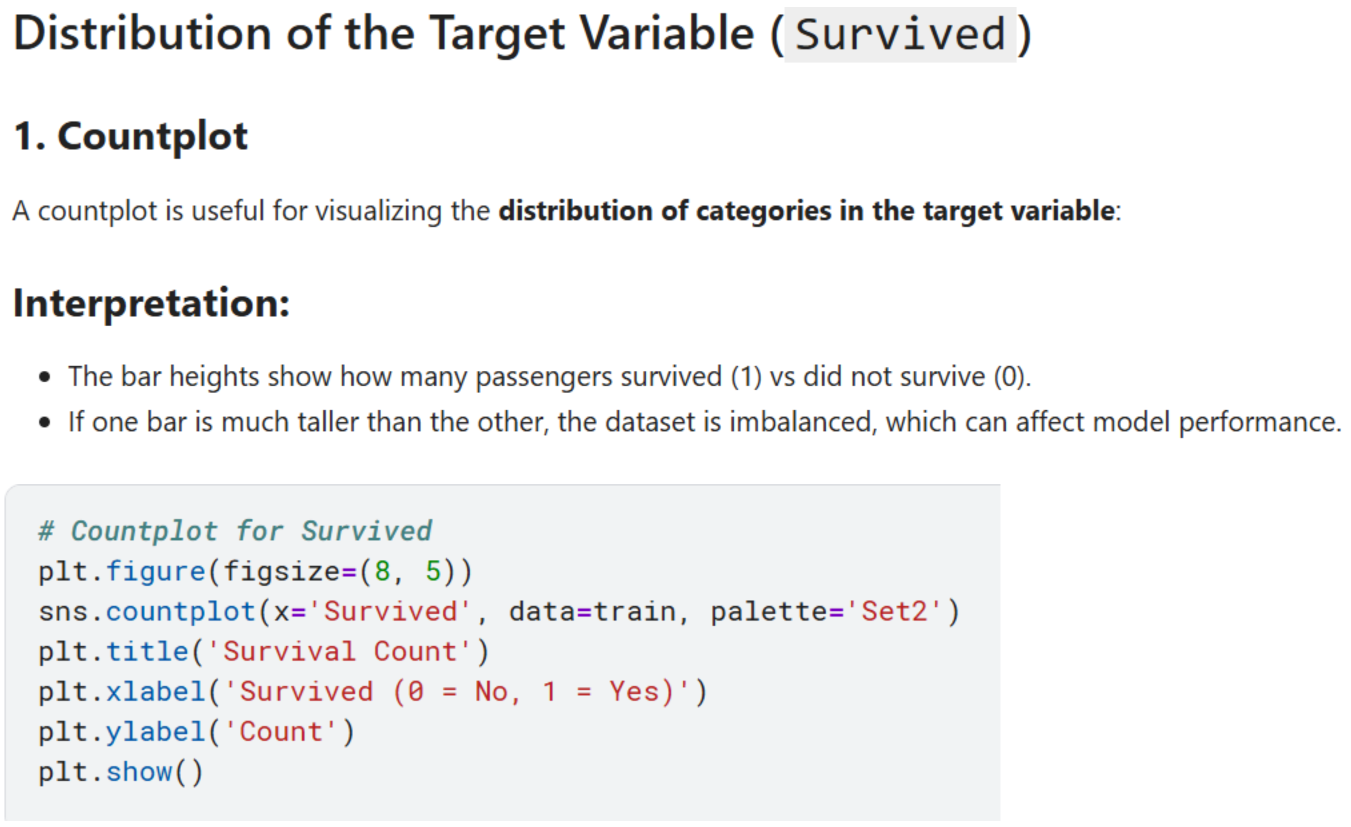
- Analyze the Target/Dependent Variable Survived and explore:
  1. The distribution of the target variable (Survived) using countplots and bar plots.
  2. How balanced or imbalanced the dataset is.
  3. What factors (like age, gender, class, or embarkation point) may influence survival?
  4. Use combined plots to detect interaction effects



Target Variable Exploration: Understanding `Survived`

- Analyze the Target/Dependent Variable Survived and explore:

1. The distribution of the target variable (Survived) using countplots and bar plots.
2. How balanced or imbalanced the dataset is.
3. What factors (like age, gender, class, or embarkation point) may influence survival? Use combined plots to detect interaction effects



Distribution of the Target Variable (`Survived`)

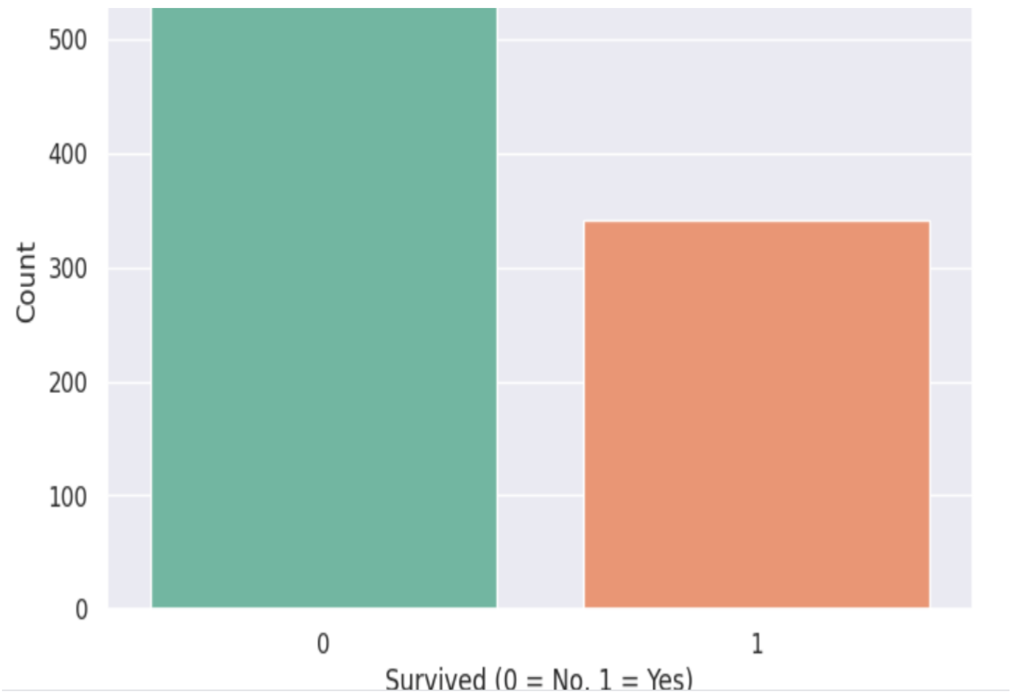
1. Countplot

A countplot is useful for visualizing the **distribution of categories in the target variable**:

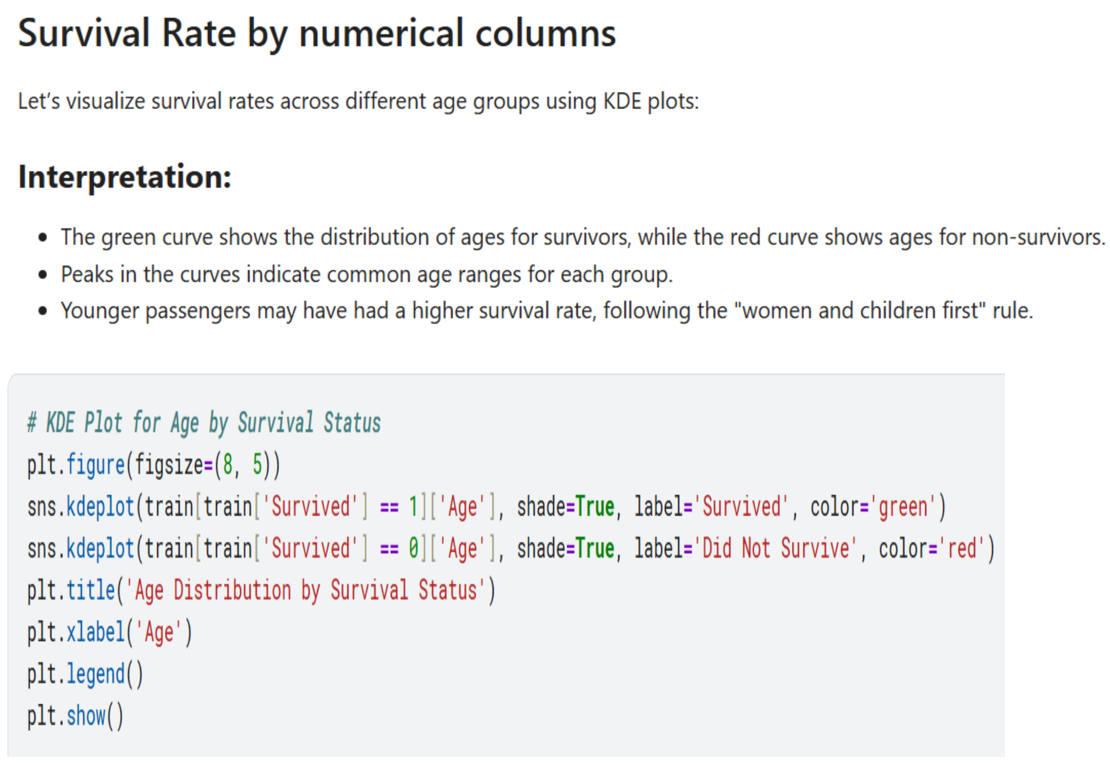
Interpretation:

- The bar heights show how many passengers survived (1) vs did not survive (0).
- If one bar is much taller than the other, the dataset is imbalanced, which can affect model performance.

```python
# Countplot for Survived
plt.figure(figsize=(8, 5))
sns.countplot(x='Survived', data=train, palette='Set2')
plt.title('Survival Count')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```



- For clearer view of the output, see the Notebook to this shared at the end of the page before the conclusion.



Survival Rate by numerical columns

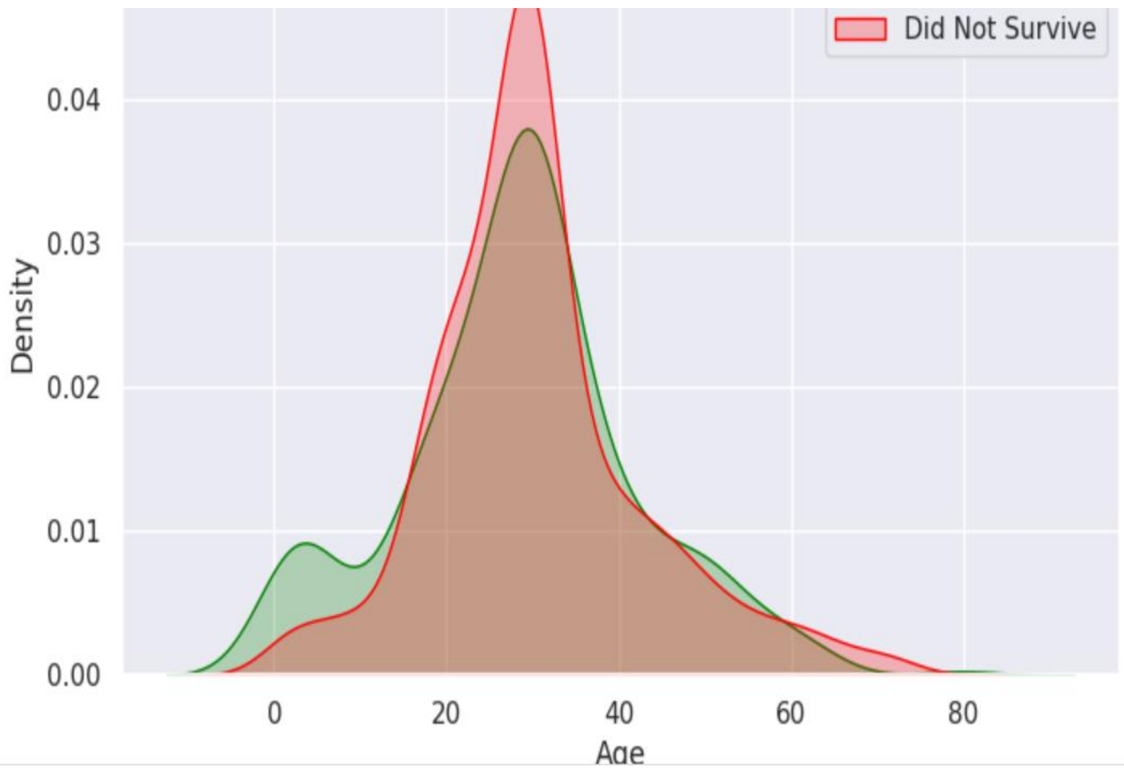
Let's visualize survival rates across different age groups using KDE plots:

Interpretation:

- The green curve shows the distribution of ages for survivors, while the red curve shows ages for non-survivors.
- Peaks in the curves indicate common age ranges for each group.
- Younger passengers may have had a higher survival rate, following the "women and children first" rule.

```python
# KDE Plot for Age by Survival Status
plt.figure(figsize=(8, 5))
sns.kdeplot(train[train['Survived'] == 1]['Age'], shade=True, label='Survived', color='green')
sns.kdeplot(train[train['Survived'] == 0]['Age'], shade=True, label='Did Not Survive', color='red')
plt.title('Age Distribution by Survival Status')
plt.xlabel('Age')
plt.legend()
plt.show()
```

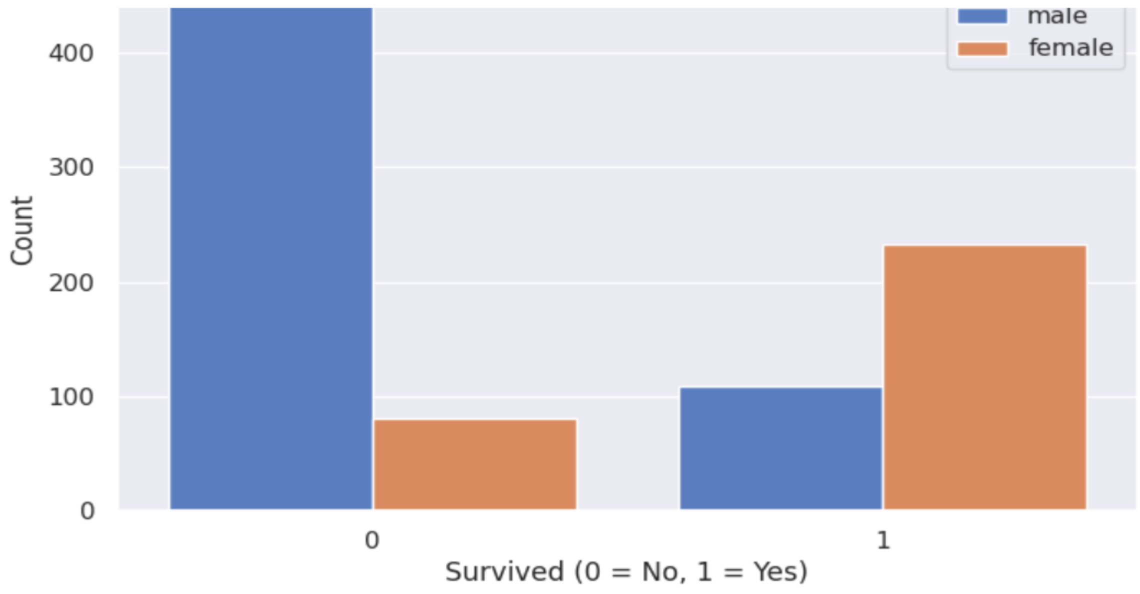
The output:



SURVIVAL RATE BY CATEGORICAL COLUMNS

```
# Countplot for Survived grouped by Gender
plt.figure(figsize=(8, 5))
sns.countplot(x='Survived', hue='Sex', data=train, palette='muted')
plt.title('Survival Rate by Gender')
plt.xlabel('Survived (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```

- The following output is displayed. Check out the link to the Notebook shared before the conclusion for visible details.



```
# Calculate survival rate by gender
gender_survival_rate = train.groupby('Sex')['Survived'].apply(lambda x: (x == 1).mean() * 100)
print(gender_survival_rate)
```

The output:

```
Sex
female    74.203822
male      18.890815
Name: Survived, dtype: float64
```

Combined Analysis (Gender, Class, and Survival)

To explore survival rates based on multiple variables at once (e.g., gender and passenger class):

Interpretation:

This plot shows survival counts broken down by gender and passenger class. Look for patterns like:

- High survival counts for first-class females.
- Low survival counts for third-class males.

```
# Grouped bar plot for survival by Gender and Class
plt.figure(figsize=(10, 6))
sns.countplot(x='Pclass', hue='Sex', data=train[train['Survived'] == 1], palette='Set1')
plt.title('Survivors by Gender and Class')
plt.xlabel('Passenger Class')
plt.ylabel('Survivor Count')
plt.show()
```

Common Insights During Target Variable Exploration

- Class Imbalance:

ii. There may be a large difference in the number of survivors vs non-survivors, indicating an imbalanced dataset. Feature Importance:

iii. Features like Pclass, Sex, and Age may have a strong influence on survival. Some features (like Ticket) may not show clear trends. Combined Effects:

iv. The survival rate for first-class females may be much higher than for third-class males due to priority in lifeboats.

KEY FINDINGS FROM THE ABOVE EDA OF THE TITANIC CASE STUDY

✓ Out of 891 passengers in the dataset, 342 survived, resulting in an overall survival rate of approximately 38.4%

✓ Females: 233 out of 314 survived, yielding a survival rate of approximately 74.2%.

✓ Males: 109 out of 577 survived, resulting in a survival rate of approximately 18.9%.

- ✓ First Class: 136 out of 216 passengers survived 63.0%.
- ✓ Second Class: 87 out of 184 passengers survived 47.3%.
- ✓ Third Class: 119 out of 491 passengers survived 24.2%.
- ✓ Higher-class passengers had better access to lifeboats and were prioritized during evacuation, leading to higher survival rates.
- ✓ Adults (Age ≥ 16): 290 out of 708 survived (40.9%).
- ✓ Children had a higher survival rate, suggesting the evacuation priority given to "women and children."
- ✓ Clear correlation between passenger class and fare:
    - o First Class: Average fare of approximately £84.15.
    - o Second Class: Average fare of approximately £20.66.
    - o Third Class: Average fare of approximately £13.68.
- ✓ Higher-class passengers paid significantly more, reflecting the luxurious accommodations and services provided.
- ✓ Analyzing the interplay between age, class, and gender reveals:
    - o Young females in First Class had the highest survival rates.
    - o Older males in Third Class had the lowest survival rates.
- ✓ This suggests that being young, female, and in a higher class significantly increased the likelihood of survival.
- ✓ Cabin information was available for only 204 out of 891 passengers.

CONSIDERATIONS

- Survival patterns (e.g., gender and class advantages) reflect **social norms of 1912**, which may not apply universally.
- Interpreting findings within that context is key for accurate conclusions.
- EDA is **descriptive**, not predictive. While patterns are insightful, they should be verified using **modeling and statistical testing**.
- Some relationships may appear strong visually but may not hold statistically.

LINK TO THE NOTEBOOK: https://www.kaggle.com/code/jedidahwavinya/titanic-case-studyeda

CONCLUSION

- In conclusion, this Exploratory Data Analysis journey has underscored the importance of truly understanding data before jumping into complex modeling or predictions.
- Through systematic cleaning, visualization, and interpretation, I uncovered key patterns, trends, and irregularities that offer valuable insights into the dataset's structure and behavior.
- One of the most powerful aspects of EDA is its ability to turn a seemingly ordinary collection of numbers into a narrative, one that highlights not just what the data shows, but what it means.
- Whether it was identifying correlations, spotting outliers, or understanding distributions, each discovery deepened our appreciation of how raw data reflects real-world phenomena.
- Ultimately, this process has laid a strong foundation for any subsequent steps in the data science lifecycle, whether predictive modeling, clustering, or decision-making.
- It has also served as a reminder that good analysis starts not with algorithms, but with thoughtful questions and a willingness to explore.