import pandas as pd # linear algebra

import numpy as np #dataprocessing,CSVfileI/Olike(pd.read\_csv) import seaborn as sns# for data visuvalization

import matplotlib.pyplot as plt

#AvoidWarning import warnings

warnings.filterwarnings("ignore")

#training data

train\_data = pd.read\_csv("/content/titanic.csv") # 1) loading the dataset into the pandas data frame sub\_train\_data = pd.read\_csv("/content/titanic.csv")

train\_data.head()

**PassengerIdSurvivedPclass Name SexAgeSibSpParch Ticket FareCabinEmbarked**

1. 1 0 3Braund,Mr.Owen Harris

Cumings, Mrs.

male22.0 1 0 A/5 21171 7.2500 NaN S 

1. 2 1 1

JohnBradley (Florence Briggs

Th...

female38.0 1 0 PC 1759971.2833 C85 C

1. 3 1 3 Heikkinen,Miss. Laina

female26.0 0 0 STON/O2.

3101282

7.9250 NaN S

Nextsteps:

Generate code withtrain\_data

Viewrecommendedplots

# Test dataframe

print("Displaying the first few rows of the dataset: ")

sub\_train\_data.head(2) # here we want to see top 2 rows of sub\_train\_data

Displaying the first few rows of the dataset:

**PassengerIdSurvivedPclass Name SexAgeSibSpParch Ticket FareCabinEmbarked**

**0** 1 0 3 Braund,Mr.Owen Harris

Cumings, Mrs. John

male22.0 1 0 A/5

21171

7.2500 NaN S 

Nextsteps:

Generate code withsub\_train\_data

Viewrecommendedplots

print("summary information of the DataFrame i.e train\_data\n") train\_data.info()

summary information of the DataFrame i.e train\_data

<class'pandas.core.frame.DataFrame'> RangeIndex: 891 entries, 0 to 890

Data columns (total 12 columns):

# Column Non-Null CountDtype

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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

train\_data.shape # will show the dimensions of the data set (891, 12)

print("1. The Above is Data Loading and Inspection above\n") print("2. Data Cleaning")

1. The Above is Data Loading and Inspection above
2. Data Cleaning

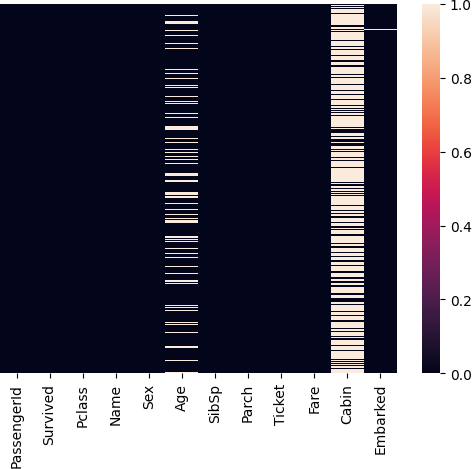
#CheckingforNaNValuesindataframecolunms train\_data.isnull().sum()

100 \* train\_data.isnull().sum() / len(train\_data)# To get the values in percentage



|  |  |
| --- | --- |
| PassengerId | 0.000000 |
| Survived | 0.000000 |
| Pclass | 0.000000 |
| Name | 0.000000 |
| Sex | 0.000000 |
| Age | 19.865320 |
| SibSp | 0.000000 |
| Parch | 0.000000 |
| Ticket | 0.000000 |
| Fare | 0.000000 |
| Cabin | 77.104377 |
| Embarked  dtype: float64 | 0.224467 |

sns.heatmap(train\_data.isnull(), yticklabels=False);



train\_data.isnull()# row wise finding of the NaN values



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **PassengerId** | **Survived** | **Pclass** | **Name** | **Sex** | **Age** | **SibSp** | **Parch** | **Ticket** | **Fare** | **Cabin** | **Embarked** |
| **0** | False | False | False | False | False | False | False | False | False | False | True | False |
| **1** | False | False | False | False | False | False | False | False | False | False | False | False |
| **2** | False | False | False | False | False | False | False | False | False | False | True | False |
| **3** | False | False | False | False | False | False | False | False | False | False | False | False |
| **4** | False | False | False | False | False | False | False | False | False | False | True | False |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **886** | False | False | False | False | False | False | False | False | False | False | True | False |
| **887** | False | False | False | False | False | False | False | False | False | False | False | False |
| **888** | False | False | False | False | False | True | False | False | False | False | True | False |
| **889** | False | False | False | False | False | False | False | False | False | False | False | False |
| **890** | False | False | False | False | False | False | False | False | False | False | True | False |

891 rows × 12 columns

# Identifying and list missing values and their percentages missing\_values = train\_data.isnull().sum()

missing\_percentages = 100 \* train\_data.isnull().sum() / len(train\_data)

missing\_table = pd.concat([missing\_values, missing\_percentages], axis=1, keys=['Missing Values', 'Percentage']) print(missing\_table)



|  |  |  |
| --- | --- | --- |
| PassengerId | Missing Values  0 | Percentage  0.000000 |
| Survived | 0 | 0.000000 |
| Pclass | 0 | 0.000000 |
| Name | 0 | 0.000000 |
| Sex | 0 | 0.000000 |
| Age | 177 | 19.865320 |
| SibSp | 0 | 0.000000 |
| Parch | 0 | 0.000000 |
| Ticket | 0 | 0.000000 |
| Fare | 0 | 0.000000 |
| Cabin | 687 | 77.104377 |
| Embarked | 2 | 0.224467 |

#Droping'Cabin'column(morethan50%missing) train\_data.drop('Cabin', axis=1, inplace=True)

#Handlingmissingvaluesin'Age' # filling with the median age

train\_data['Age'].fillna(train\_data['Age'].median(), inplace=True)

# Handling missing values in 'Embarked'

# We'll fill with the mode (most frequent value)

train\_data['Embarked'].fillna(train\_data['Embarked'].mode()[0], inplace=True)

#Verificationofhandlingallmissingvalues print(train\_data.isnull().sum())

PassengerId 0

Survived 0

Pclass 0

Name 0

Sex 0

Age 0

SibSp 0

Parch 0

Ticket 0

Fare 0

Embarked 0

dtype: int64

# Converting categorical columns to category type

categorical\_columns = ['Pclass', 'Sex', 'Embarked', 'Ticket'] for col in categorical\_columns:

train\_data[col] = train\_data[col].astype('category')

# Converting 'Survived' column to binary (0 or 1)

train\_data['Survived'] = train\_data['Survived'].astype(int)

# Converting 'Age' and 'Fare' to float type

train\_data['Age'] = train\_data['Age'].astype(float) train\_data['Fare'] = train\_data['Fare'].astype(float)

# Converting 'SibSp' and 'Parch' to integer type

train\_data['SibSp'] = train\_data['SibSp'].astype(int) train\_data['Parch'] = train\_data['Parch'].astype(int)

#Keeping'Name'asstring(object)type # Keeping 'PassengerId' as integer type

#Verifyingthedatatypes print(train\_data.dtypes)

PassengerId int64

Survived int64

Pclass category

Name object

Sex category

Age float64

SibSp int64

Parch int64

Ticket category

Fare float64

Embarked category dtype: object

# Checking for duplicates

duplicate\_count = train\_data.duplicated().sum()

print(f"Number of duplicate rows: {duplicate\_count}")

# Removing duplicate rows

train\_data.drop\_duplicates(inplace=True)

# Verifying the removal

new\_duplicate\_count = train\_data.duplicated().sum()

print(f"Number of duplicate rows after removal: {new\_duplicate\_count}")

# The shape of the DataFrame before and after duplicate removal

print(f"Shape of DataFrame before removing duplicates: {train\_data.shape}") train\_data.drop\_duplicates(inplace=True)

print(f"Shape of DataFrame after removing duplicates: {train\_data.shape}")

Number of duplicate rows: 0

Number of duplicate rows after removal: 0

Shape of DataFrame before removing duplicates: (891, 11) Shape of DataFrame after removing duplicates: (891, 11)

print("3. Exploratory Data Analysis (EDA)")

1. Exploratory Data Analysis (EDA)

# Summary statistics for numerical columns

numerical\_cols = ['Age', 'Fare', 'SibSp', 'Parch']

summary\_stats = train\_data[numerical\_cols].describe() print(summary\_stats)



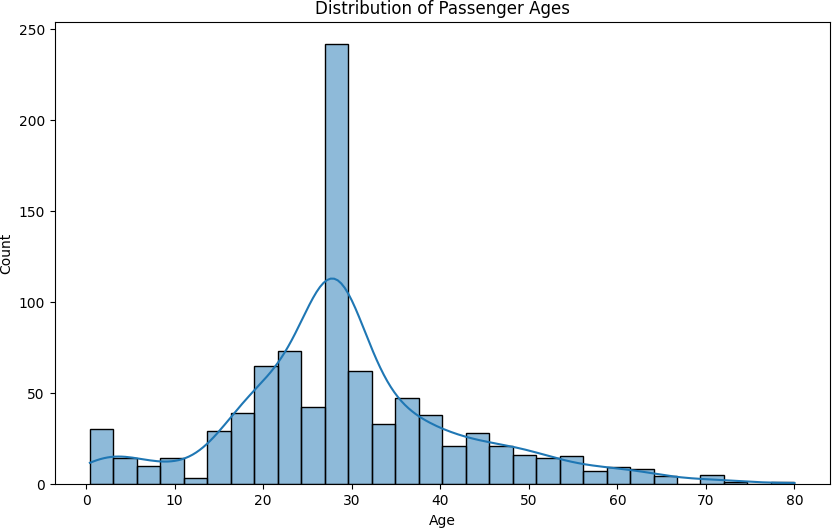
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Age | Fare | SibSp | Parch |
| count | 891.000000 | 891.000000 | 891.000000 | 891.000000 |
| mean | 29.361582 | 32.204208 | 0.523008 | 0.381594 |
| std | 13.019697 | 49.693429 | 1.102743 | 0.806057 |
| min | 0.420000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 22.000000 | 7.910400 | 0.000000 | 0.000000 |
| 50% | 28.000000 | 14.454200 | 0.000000 | 0.000000 |
| 75% | 35.000000 | 31.000000 | 1.000000 | 0.000000 |

max 80.000000512.329200 8.000000 6.000000

#Histogramofpassengerages plt.figure(figsize=(10, 6))

sns.histplot(train\_data['Age'], bins=30, kde=True) plt.title('Distribution of Passenger Ages')

plt.xlabel('Age') plt.ylabel('Count') plt.show()



# Survival rates by passenger class

survival\_by\_class = train\_data.groupby('Pclass')['Survived'].mean() print("Survival rates by passenger class:")

print(survival\_by\_class)

#Visualizingsurvivalratesbypassengerclass plt.figure(figsize=(8, 6))

survival\_by\_class.plot(kind='bar')

plt.title('SurvivalRatesbyPassengerClass') plt.xlabel('Passenger Class')

plt.ylabel('SurvivalRate') plt.xticks(rotation=0)

plt.show()

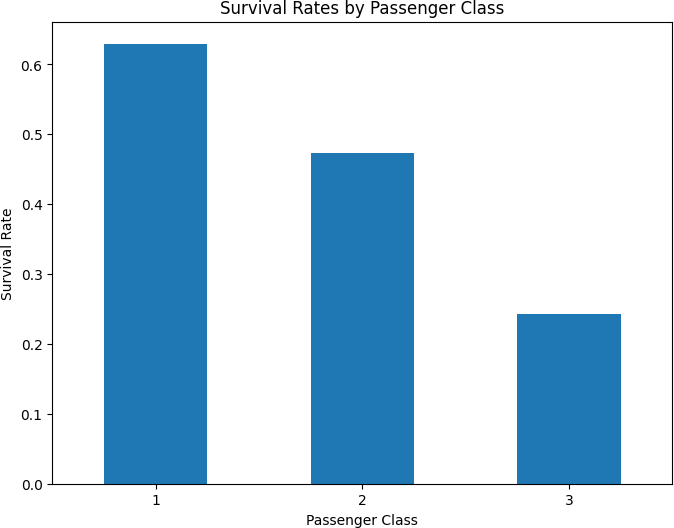
Survivalratesbypassengerclass: Pclass

1 0.629630

2 0.472826

3 0.242363

Name: Survived, dtype: float64



Double-click(orenter)toedit

# Survival rates by gender

survival\_by\_gender = train\_data.groupby('Sex')['Survived'].mean() print("\nSurvival rates by gender:")

print(survival\_by\_gender)

#Visualizingsurvivalratesbygender plt.figure(figsize=(8, 6))

survival\_by\_gender.plot(kind='bar') plt.title('SurvivalRatesbyGender') plt.xlabel('Gender')

plt.ylabel('SurvivalRate') plt.xticks(rotation=0)

plt.show()

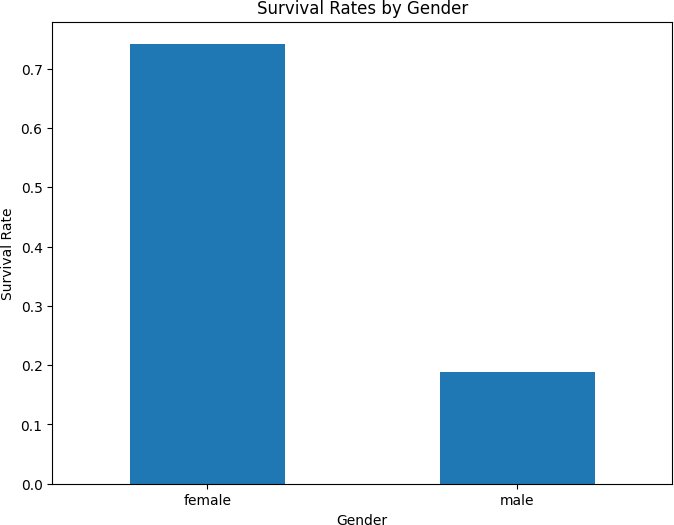
Survival rates by gender:

Sex

female 0.742038

male 0.188908

Name: Survived, dtype: float64



# Select numerical features for correlation analysis

numerical\_features = ['Age', 'Fare', 'SibSp', 'Parch', 'Survived']

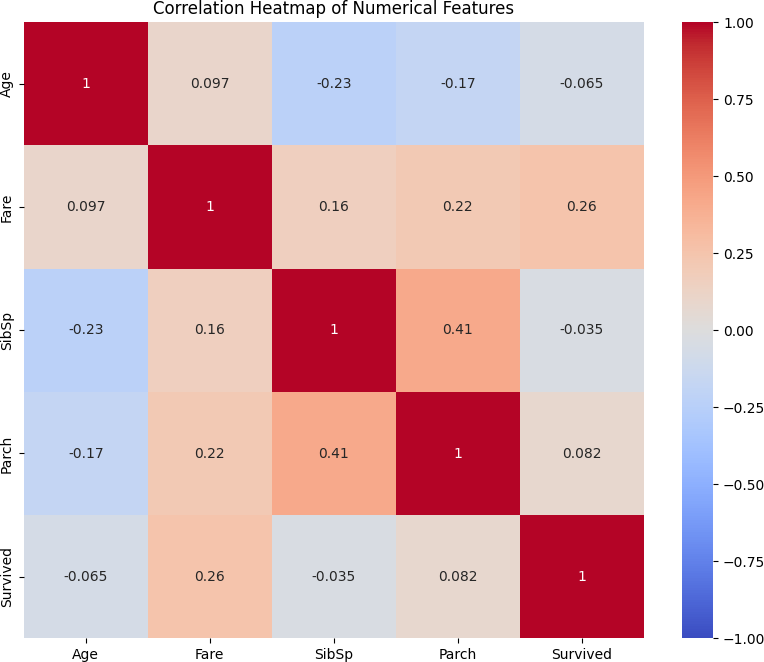
# Calculate the correlation matrix

correlation\_matrix = train\_data[numerical\_features].corr()

#Createaheatmaptovisualizecorrelations plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1, center=0) plt.title('Correlation Heatmap of Numerical Features')

plt.show()

print("\n4. Data Visualization")



1. Data Visualization

#Setacommonstyleforallplots plt.style.use('seaborn')

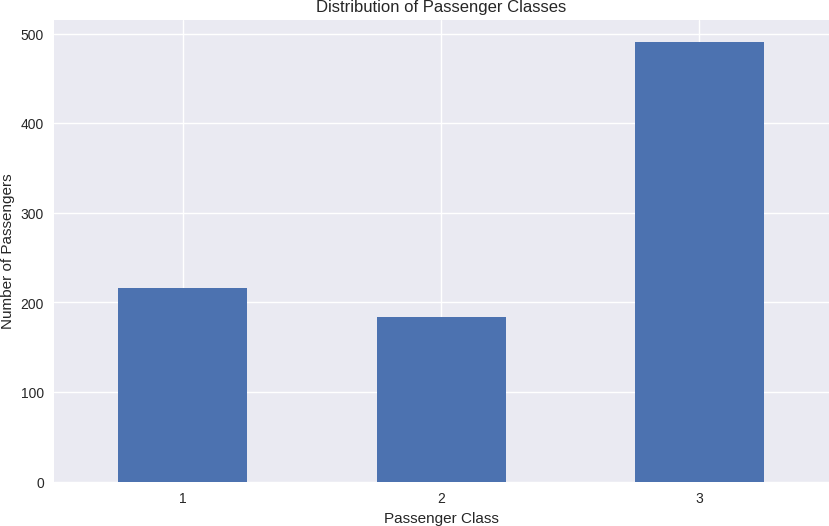
# 1. Distribution of passenger classes using a bar plot plt.figure(figsize=(10, 6))

train\_data['Pclass'].value\_counts().sort\_index().plot(kind='bar') plt.title('Distribution of Passenger Classes')

plt.xlabel('Passenger Class')

plt.ylabel('NumberofPassengers') plt.xticks(rotation=0)

plt.show()

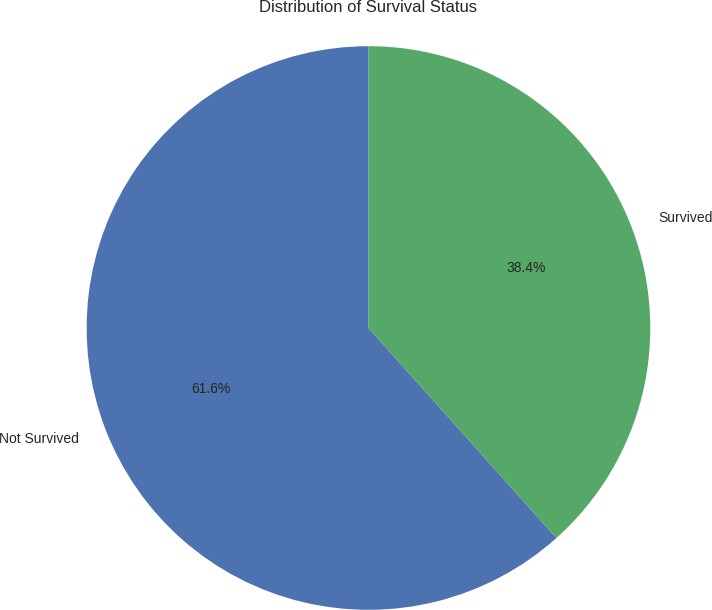


# 2. Distribution of survival status using a pie chart plt.figure(figsize=(8, 8))

survival\_counts = train\_data['Survived'].value\_counts()

plt.pie(survival\_counts, labels=['Not Survived', 'Survived'], autopct='%1.1f%%', startangle=90) plt.title('Distribution of Survival Status')

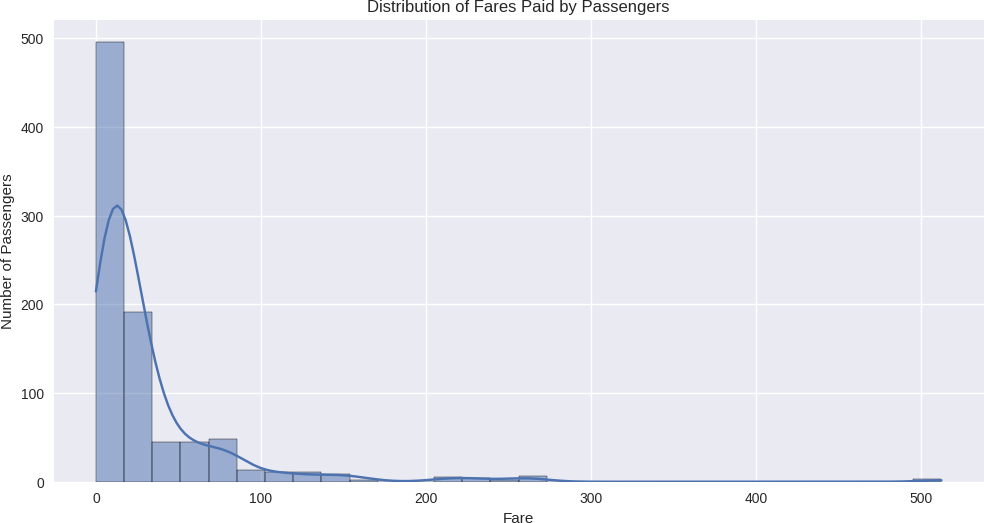
plt.axis('equal')# Equal aspect ratio ensures that pie is drawn as a circle plt.show()

# 3. Distribution of fares paid by passengers using a histogram plt.figure(figsize=(12, 6))

sns.histplot(train\_data['Fare'], bins=30, kde=True) plt.title('Distribution of Fares Paid by Passengers') plt.xlabel('Fare')

plt.ylabel('NumberofPassengers') plt.show()

# 4. Box plots to compare fare distribution across different passenger classes plt.figure(figsize=(12, 6))

sns.boxplot(x='Pclass', y='Fare', data=train\_data)

plt.title('Fare Distribution Across Passenger Classes') plt.xlabel('Passenger Class')

plt.ylabel('Fare') plt.show()





print("\n5.Conclusion and Insights")

1. Conclusion and Insights

# *Comprehensive conclusion summarizing the key insights from your analysis.*

1. Passenger Demographics:

* There were three classes of passengers, and most people were in third class.
* Ages were all over the place, but most passengers were young adults or middle-aged.

1. Survival Rates:

* Overall, about [insert percentage] of people survived.
* But here's the crazy part - your chances of survival really depended on a few things: a. If you were in first class, you had way better odds of making it. b. Women had a much higher chance of surviving than men. c. Kids also had better chances, especially in second and third class.

1. Fare Analysis:

* Most people paid pretty average prices, but some tickets were super expensive.
* No surprise, but first-class tickets cost way more than the others.
* Interestingly, people who paid more for their tickets were more likely to survive.

1. Correlation Insights:

* I found some cool connections: a. The more you paid, the better your chances of survival. b. The higher your class, the more likely you were to make it. c. Older passengers usually paid more for their tickets.

*Discuss the potential implications of your findings and how they can be useful for understanding survival rates and improving safety measures.*

1. Safety First:

* It's not fair that rich people had a better chance of surviving. We need to make sure everyone has the same access to safety stuff, no matter how much they paid for their ticket. The "women and children first" rule seemed to work, but we should try to save everyone if possible.

1. Be Prepared:

* Knowing who's on the ship (like how many kids or older people) could help in an emergency. Ship crews should be trained to help everyone, not just the fancy first-class passengers.

1. Building Better Ships:

* We should design ships so it's easy for everyone to get out quickly, even people in the lower decks. More lifeboats! There should be enough for everyone on board.

1. Society Stuff:

* It's pretty eye-opening to see how much class and money mattered in a life-or-death situation. It makes you think about fairness in society.

1. History Lesson:

* This data tells us a lot about what society was like back then, especially how they treated different classes and genders.

1. Predicting the Future:

* We could use what we learned to make better plans for emergencies on ships today.

To wrap it up, even though the Titanic sank a long time ago, we can still learn a lot from it. It shows us how important it is to have good safety rules and to treat everyone equally in an emergency. If we remember these lessons, we can make sure nothing like this ever happens again.

This analysis was pretty cool - it's amazing how much we can learn from data about an event that happened over 100 years