configure kaggle

```
!pip install -q kaggle
```

```
pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kagg
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.5)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->ka
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from pyt
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->
```

```
!kaggle competitions download -c titanic
```

```
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.10/dist-packages/kaggle/__init__.py", line 7, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.10/dist-packages/kaggle/api/kaggle_api_extended.py", line 407, in authen
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in /root/.config/kaggle. Or use the environ
```

```
from google.colab import files
files.upload()  # Upload the 'kaggle.json' file
```

```
Browse... kaggle.json
kaggle.json(application/json) - 70 bytes, last modified: n/a - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"shivamgarg8783","key":"252b9bd92a7c1b21a7dd763487fffa3b"}'}
```

```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kagg
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.5)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->ka
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from pyt
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->
```

```
!kaggle competitions list
```

```
ref                                                                            deadline
-----------------------------------------------------------------------------  ---------------
https://www.kaggle.com/competitions/arc-prize-2024                             2024-11-10 23:59
https://www.kaggle.com/competitions/child-mind-institute-problematic-internet-use  2024-12-19 23:59
https://www.kaggle.com/competitions/eedi-mining-misconceptions-in-mathematics  2024-12-12 23:59
https://www.kaggle.com/competitions/rsna-2024-lumbar-spine-degenerative-classification  2024-10-08 23:59
https://www.kaggle.com/competitions/ariel-data-challenge-2024                  2024-10-31 23:59
https://www.kaggle.com/competitions/um-game-playing-strength-of-mcts-variants  2024-12-02 23:59
https://www.kaggle.com/competitions/playground-series-s4e10                    2024-10-31 23:59
https://www.kaggle.com/competitions/titanic                                    2030-01-01 00:00
https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques  2030-01-01 00:00
https://www.kaggle.com/competitions/spaceship-titanic                          2030-01-01 00:00
https://www.kaggle.com/competitions/digit-recognizer                           2030-01-01 00:00
https://www.kaggle.com/competitions/nlp-getting-started                        2030-01-01 00:00
https://www.kaggle.com/competitions/store-sales-time-series-forecasting        2030-06-30 23:59
https://www.kaggle.com/competitions/connectx                                   2030-01-01 00:00
https://www.kaggle.com/competitions/gan-getting-started                        2030-07-01 23:59
https://www.kaggle.com/competitions/contradictory-my-dear-watson               2030-07-01 23:59
https://www.kaggle.com/competitions/tpu-getting-started                        2030-06-03 23:59
```

```
!kaggle competitions download -c titanic
```

```
Downloading titanic.zip to /content
  0% 0.00/34.1k [00:00<?, ?B/s]
100% 34.1k/34.1k [00:00<00:00, 49.2MB/s]
```

EDA

```
Start coding or generate with AI.
```

```
# Unzip the dataset
!unzip titanic.zip

# Import pandas and load the dataset into a DataFrame
import pandas as pd

# Load training data
train_df = pd.read_csv('train.csv')
print(train_df.head())

# Load test data
test_df = pd.read_csv('test.csv')
print(test_df.head())
```

```
Archive:  titanic.zip
  inflating: gender_submission.csv
  inflating: test.csv
  inflating: train.csv
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
```

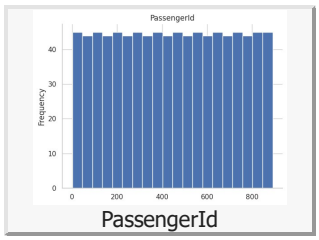
```
   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Set style for seaborn
sns.set(style="whitegrid")
```

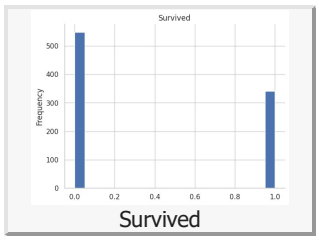
```
train_df.head()
```

	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

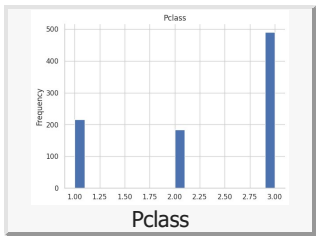
Next steps: Generate code with `train_df` | View recommended plots | New interactive sheet



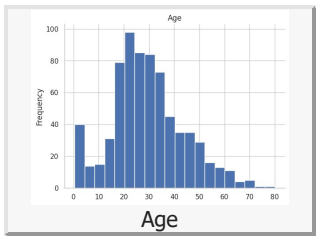
PassengerId



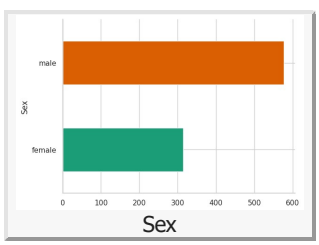
Survived



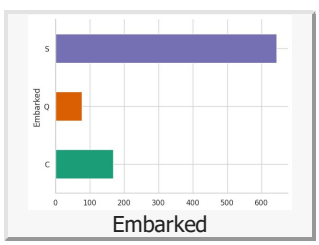
Pclass



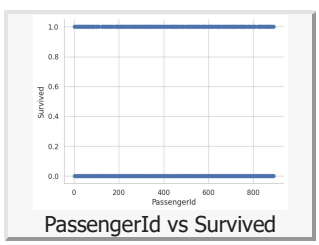
Age



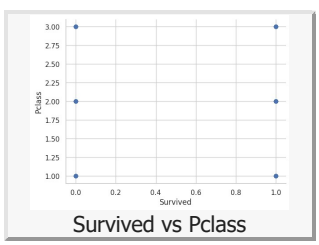
Sex



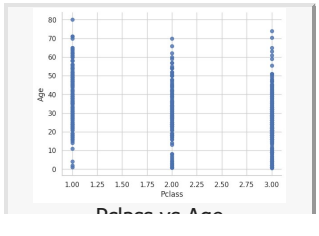
Embarked

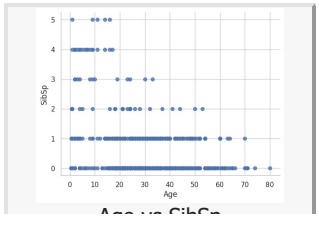


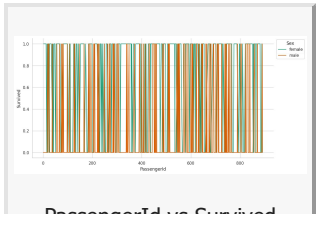
PassengerId vs Survived

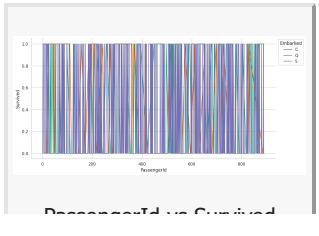


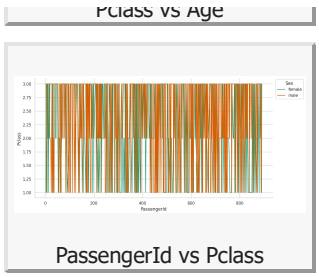
Survived vs Pclass

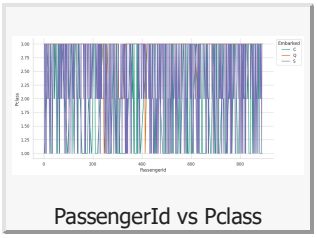


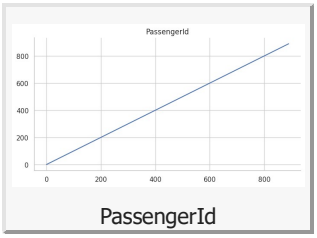


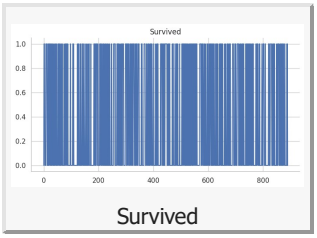


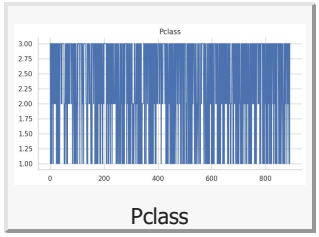


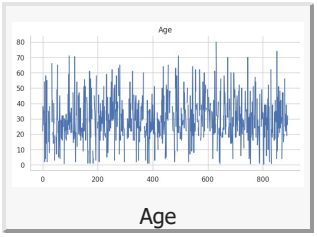


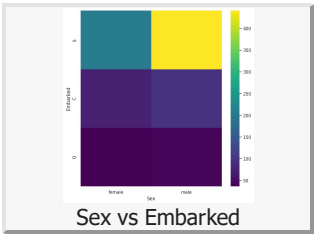


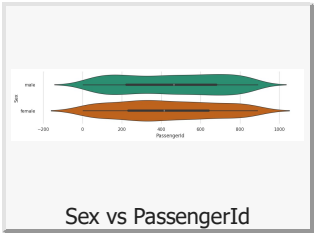


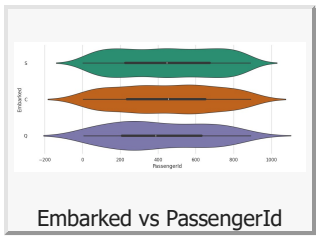


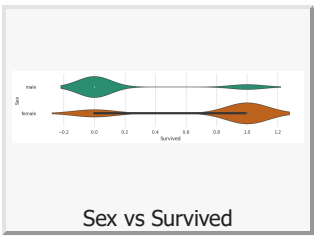


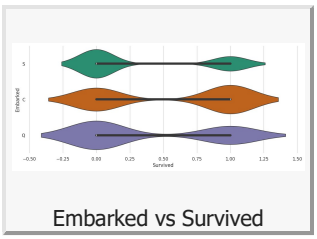






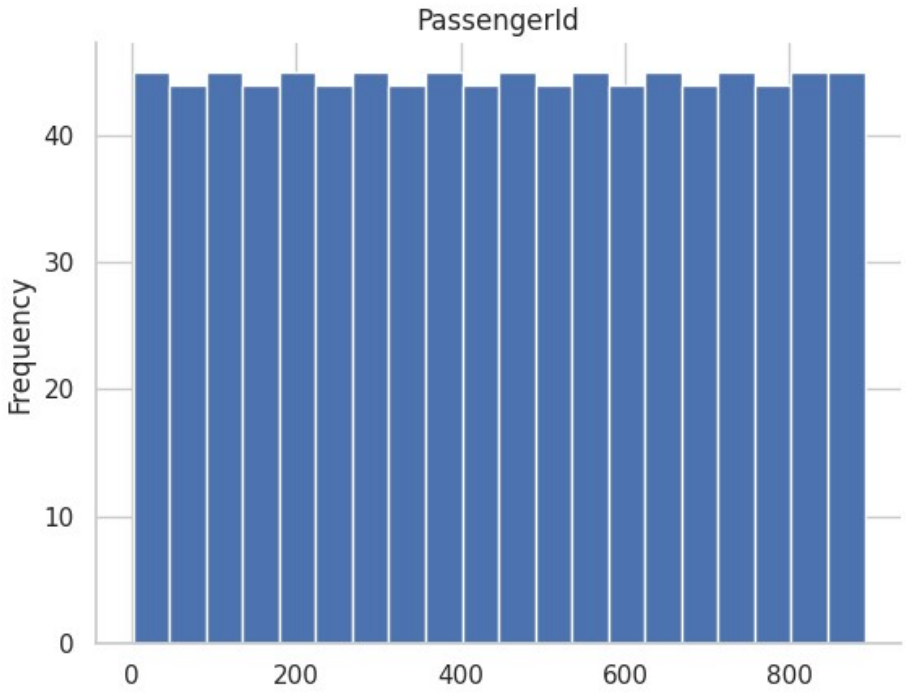






PassengerId

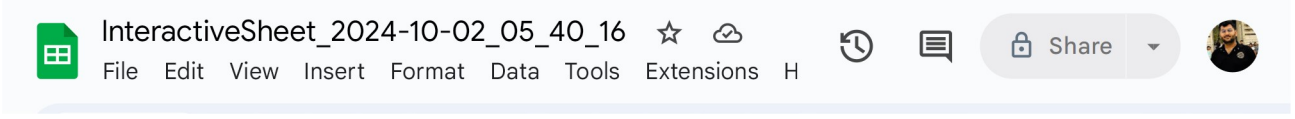
Show code

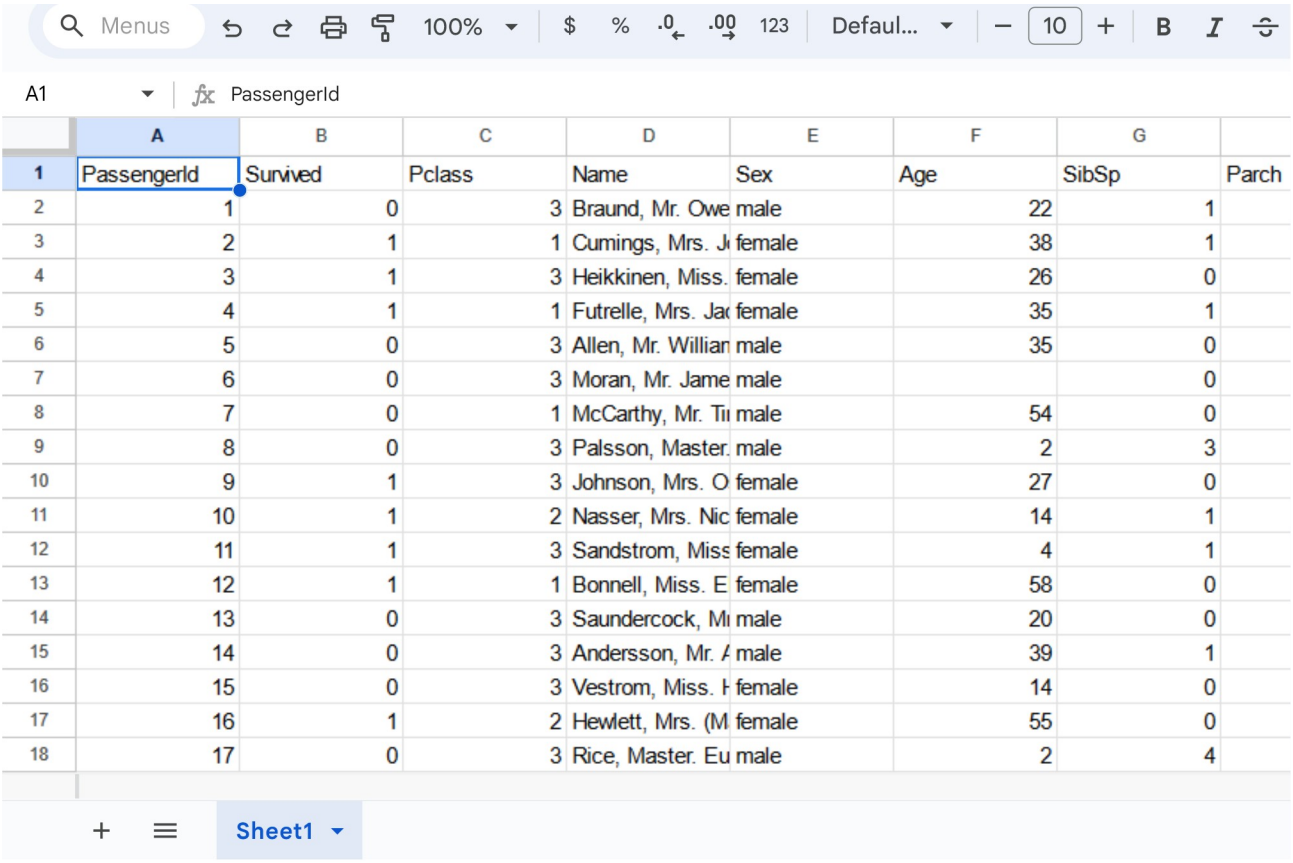



```
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=train_df)
```

https://docs.google.com/spreadsheets/d/1nyiPlQwT9TguAWEdIrQdA3DSXyMoxDoeOR49ZyPWWos#gid=0

```
/usr/local/lib/python3.10/dist-packages/google/colab/sheets.py:31: FutureWarning: DataFrame.applymap ha
  return frame.applymap(_clean_val).replace({np.nan: None})
```





	A	B	C	D	E	F	G	
1	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
2	1	0	3	Braund, Mr. Owe	male	22	1	
3	2	1	1	Cumings, Mrs. J	female	38	1	
4	3	1	3	Heikkinen, Miss.	female	26	0	
5	4	1	1	Futrelle, Mrs. Ja	female	35	1	
6	5	0	3	Allen, Mr. Willian	male	35	0	
7	6	0	3	Moran, Mr. Jame	male		0	
8	7	0	1	McCarthy, Mr. Ti	male	54	0	
9	8	0	3	Palsson, Master.	male	2	3	
10	9	1	3	Johnson, Mrs. O	female	27	0	
11	10	1	2	Nasser, Mrs. Nic	female	14	1	
12	11	1	3	Sandstrom, Miss	female	4	1	
13	12	1	1	Bonnell, Miss. E	female	58	0	
14	13	0	3	Saundercock, Mr	male	20	0	
15	14	0	3	Andersson, Mr. A	male	39	1	
16	15	0	3	Vestrom, Miss. H	female	14	0	
17	16	1	2	Hewlett, Mrs. (M	female	55	0	
18	17	0	3	Rice, Master. Eu	male	2	4	

- PassengerId: A unique identifier for each passenger.
- Survived: Indicates whether the passenger survived (1) or died (0).
- Pclass: Passenger class (1st, 2nd, or 3rd).
- Name: Passenger's name.
- Sex: Passenger's gender (male or female).
- Age: Passenger's age.
- SibSp: Number of siblings or spouses aboard.
- Parch: Number of parents or children aboard.
- Ticket: Ticket number.
- Fare: Passenger's fare.
- Cabin: Cabin number (if available).
- Embarked: Port of embarkation (S = Southampton, C = Cherbourg, Q = Queenstown).

```
# Check the structure and info of the dataset
train_df.info()

# Check for missing values
train_df.isnull().sum()

# Summary statistics
train_df.describe()
```

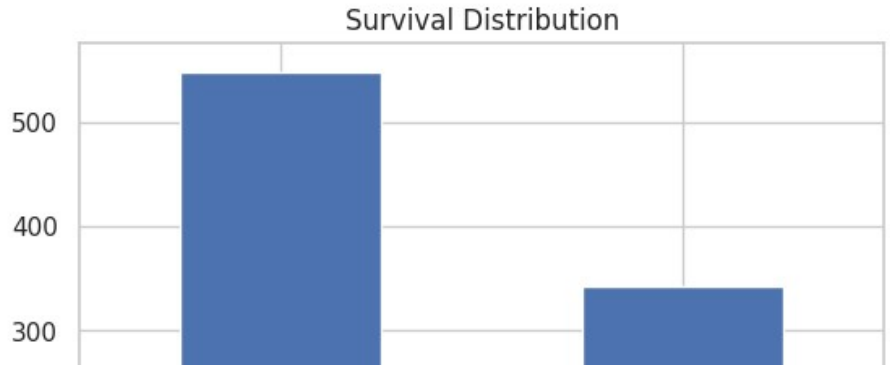
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
```

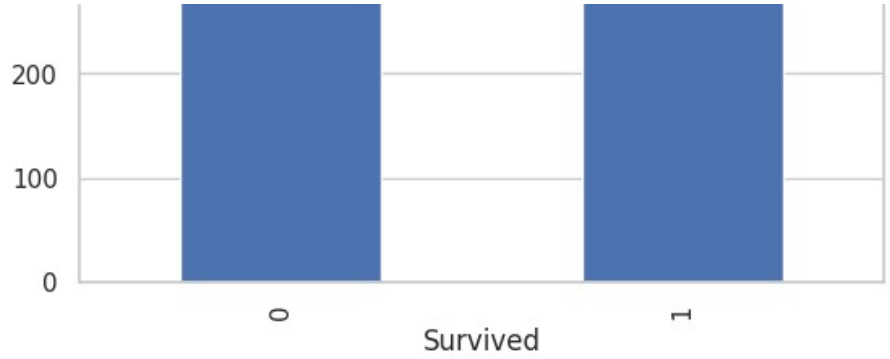
```
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

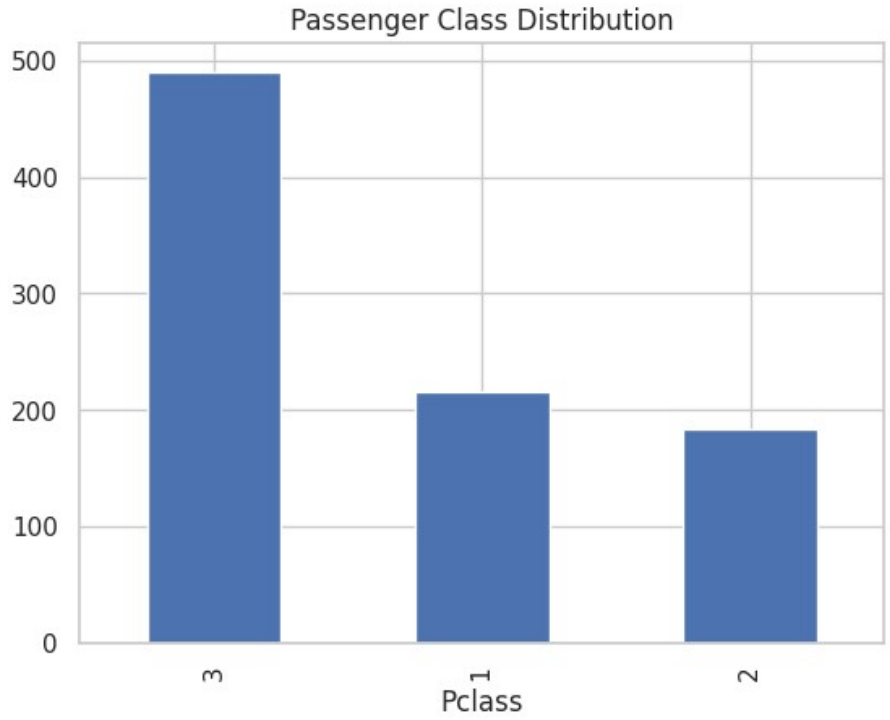
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

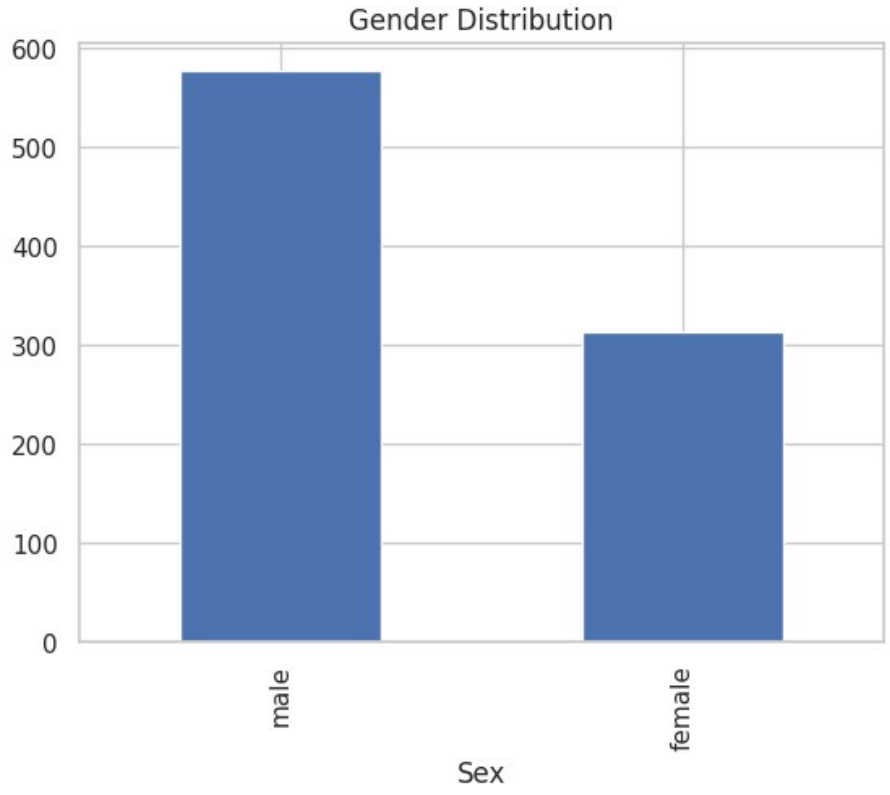
- PassengerId: The minimum and maximum values are 1 and 891, respectively, indicating that there are 891 passengers in the dataset.
- Survived: The mean is 0.383838, suggesting that approximately 38% of passengers survived.
- Pclass: The mean is 2.308642, indicating that the majority of passengers were in the 3rd class.
- Age: The mean age is 29.699118, with a standard deviation of 14.526497. This suggests that the ages are relatively evenly distributed around the mean, with some variation.
- SibSp: The mean is 0.523008, indicating that most passengers were not traveling with siblings or spouses.
- Parch: The mean is 0.381594, suggesting that most passengers were not traveling with parents or children.
- Fare: The mean fare is 32.204208, with a standard deviation of 49.693429. This indicates that there is a wide range of fares, with some passengers paying significantly more than others

```
# Distribution of target variable 'Survived'
train_df['Survived'].value_counts().plot(kind='bar', title='Survival Distribution')
plt.show()

# Distribution of 'Pclass' (Passenger Class)
train_df['Pclass'].value_counts().plot(kind='bar', title='Passenger Class Distribution')
plt.show()

# Distribution of 'Sex'
train_df['Sex'].value_counts().plot(kind='bar', title='Gender Distribution')
plt.show()
```



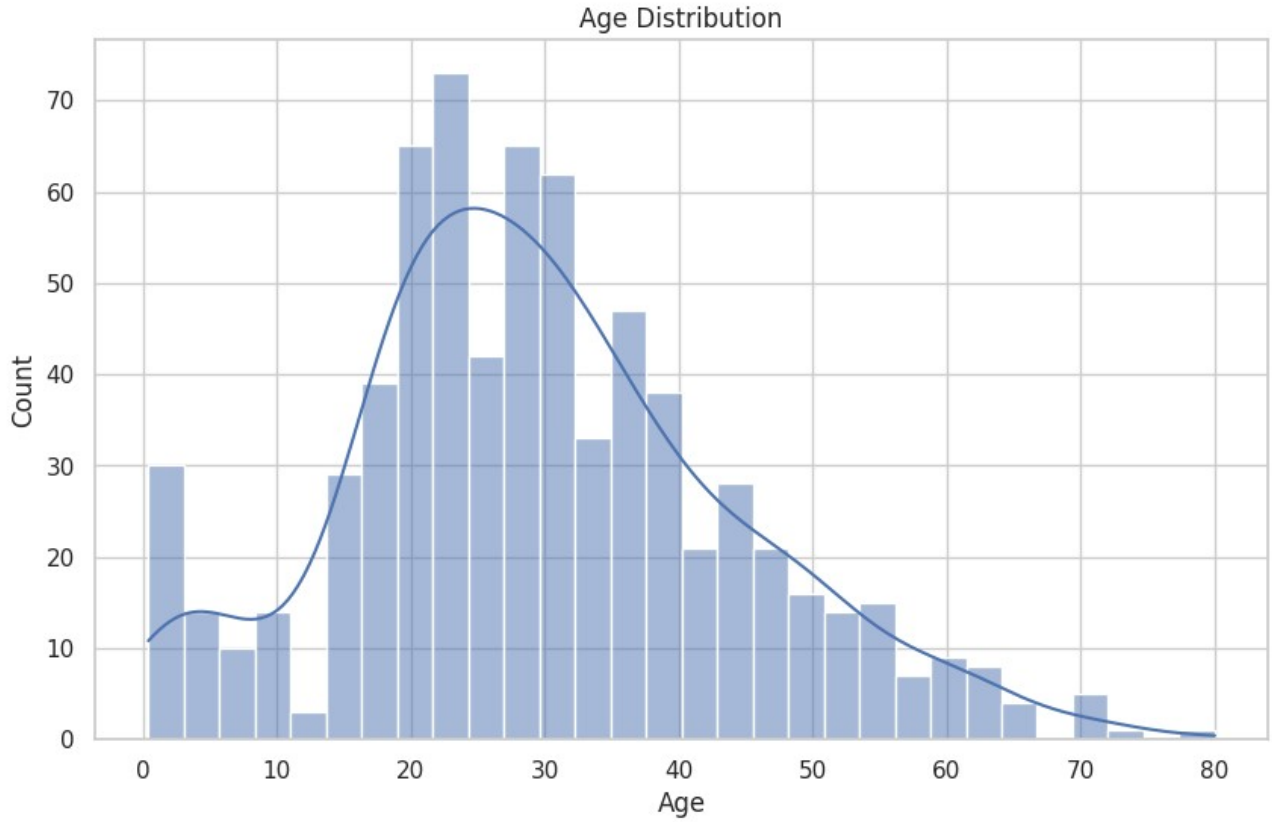






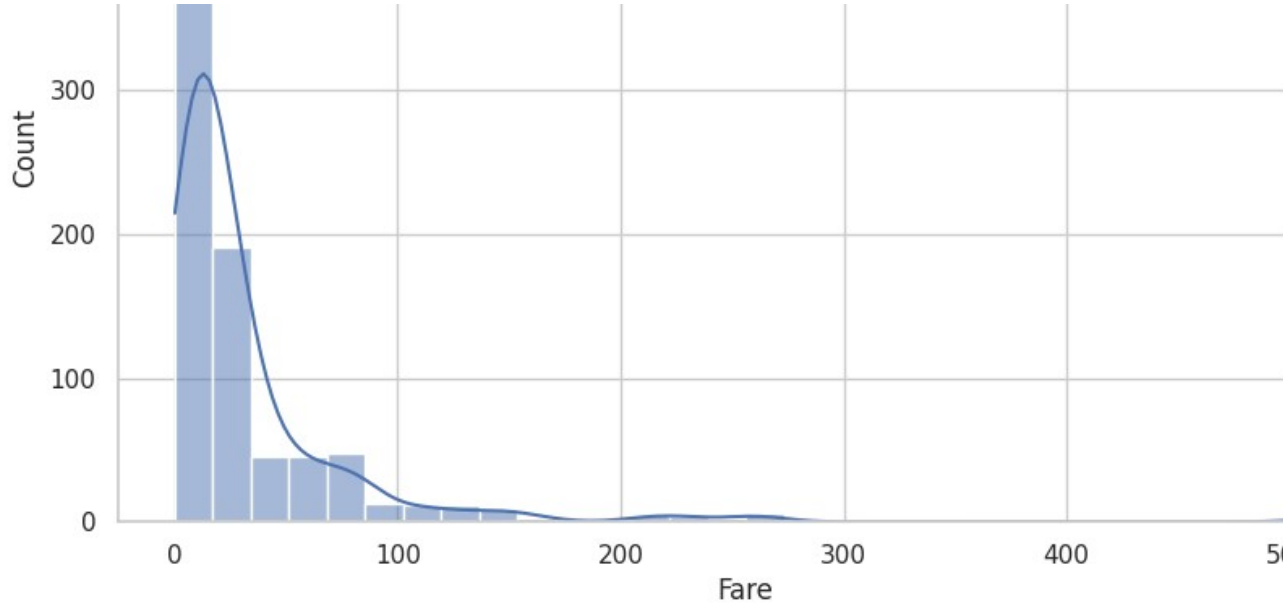

```
# Age distribution plot
plt.figure(figsize=(10,6))
```

```
plt.figure(figsize=(10,6))
sns.histplot(train_df['Age'].dropna(), kde=True, bins=30)
plt.title('Age Distribution')
plt.show()
```



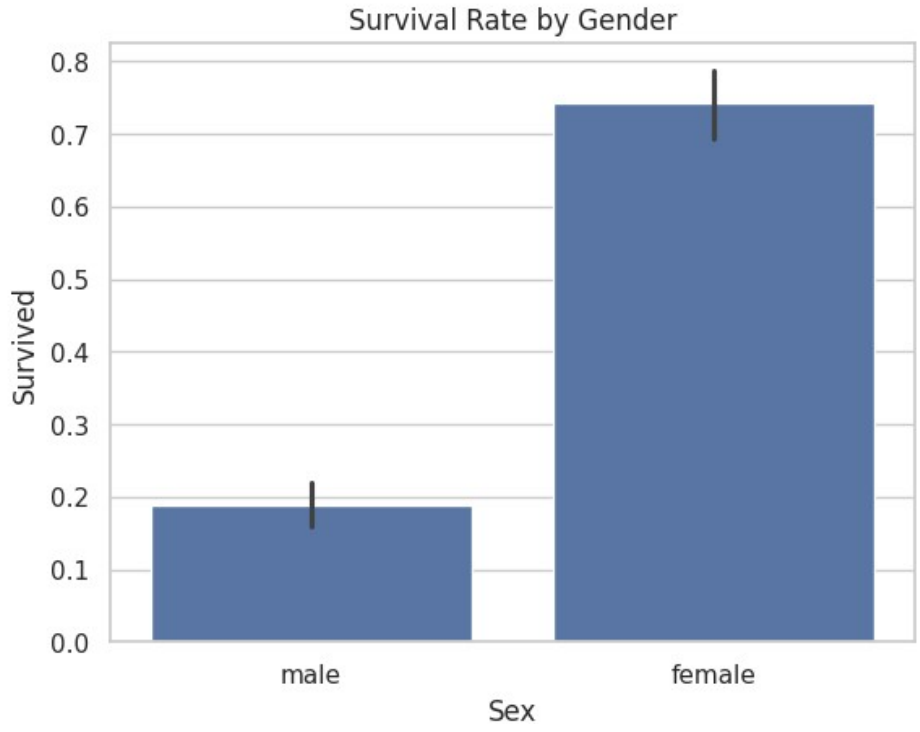
- Skewness: The distribution is right-skewed, meaning there is a longer tail to the right. This indicates that there are some older individuals in the group.
- Peak: The peak of the distribution is around 25-30 years old, suggesting that this age range is the most common in the group.
- Range: The ages in the group range from approximately 0 to 80 years old.
- Mode: There is a clear mode (most frequent age) around 25-30 years old.
- Median: The median age (the middle value when the data is sorted) appears to be around 30-35 years old, based on the shape of the distribution.
- Distribution: The overall shape of the distribution is somewhat bell-shaped, similar to a normal distribution, but with a longer right tail

```
# Fare distribution plot
plt.figure(figsize=(10,6))
sns.histplot(train_df['Fare'], kde=True, bins=30)
plt.title('Fare Distribution')
plt.show()
```



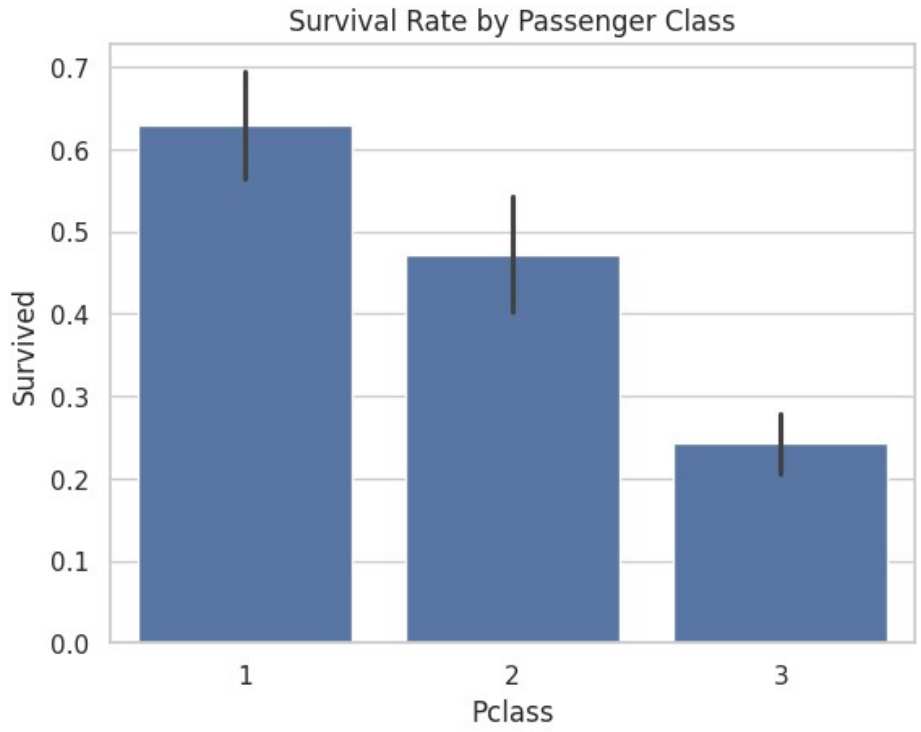
- Skewness: The distribution is heavily right-skewed, meaning there is a long tail to the right. This indicates that there are a few individuals who paid significantly higher fares than the majority.
- Peak: The peak of the distribution is around 0-25, suggesting that the most common fare range is between 0 and 25.
- Range: The fares range from 0 to around 500.
- Mode: There is a clear mode (most frequent fare) around 0-25.
- Median: The median fare (the middle value when the data is sorted) appears to be around 10-20, based on the shape of the distribution.
- Distribution: The overall shape of the distribution is highly skewed to the right, with a long tail extending to the right.

```
# Survival rate based on gender
sns.barplot(x='Sex', y='Survived', data=train_df)
plt.title('Survival Rate by Gender')
plt.show()
```



- Female Survival: Female individuals have a significantly higher survival rate compared to male individuals.
- Male Survival: Male individuals have a lower survival rate compared to female individuals.

```
# Survival rate based on Pclass
sns.barplot(x='Pclass', y='Survived', data=train_df)
plt.title('Survival Rate by Passenger Class')
plt.show()
```



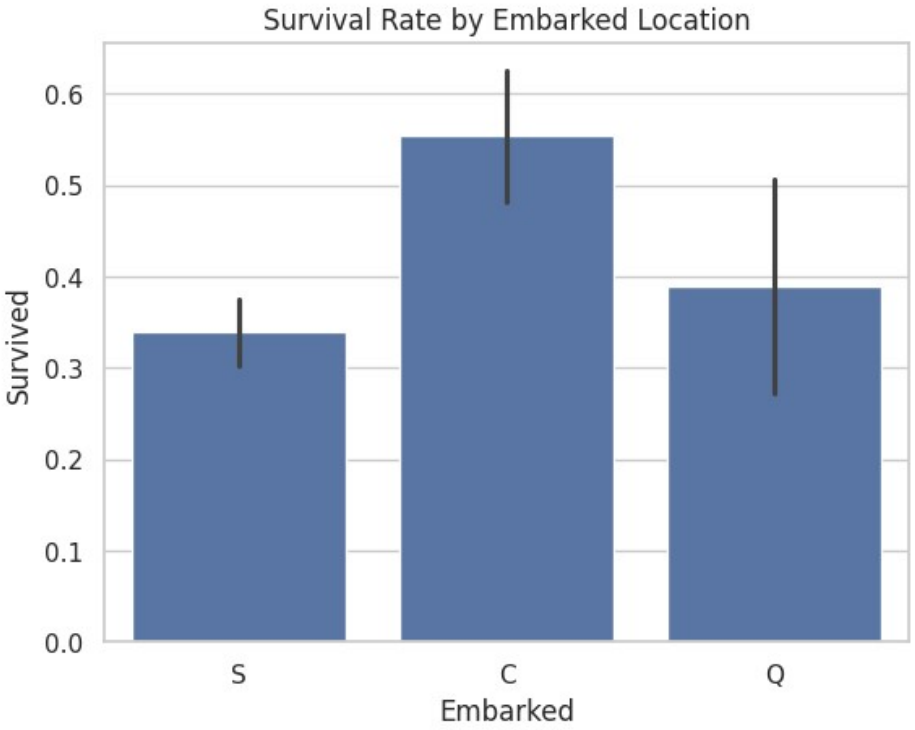
- Class 1 Survival: Individuals in class 1 have the highest survival rate.
- Class 3 Survival: Individuals in class 3 have the lowest survival rate.
- Class 2 Survival: Individuals in class 2 have a survival rate between class 1 and class 3
- ◦ reason for survival class wise
- Prioritization: First-class passengers were likely given priority in rescue efforts due to their higher social status and the associated fees they paid.
- Location: First-class cabins were often located on the upper decks of the ship, which may have provided easier access to lifeboats in the event of an emergency.
- Resources: First-class passengers may have had access to more resources, such as personal belongings or money, that could have aided in their survival.

```
# Fill missing Embarked values (optional)
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)

# Survival rate by Embarked location
sns.barplot(x='Embarked', y='Survived', data=train_df)
plt.title('Survival Rate by Embarked Location')
plt.show()
```

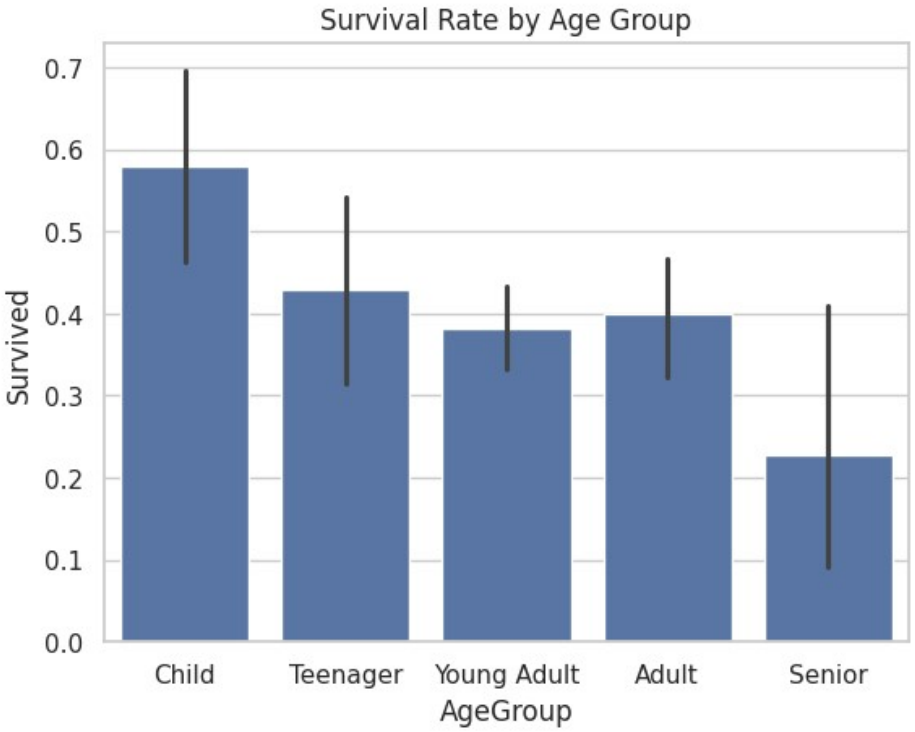
```
<ipython-input-26-742b49ae8eab>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame
```

```
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla
```

```
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
```



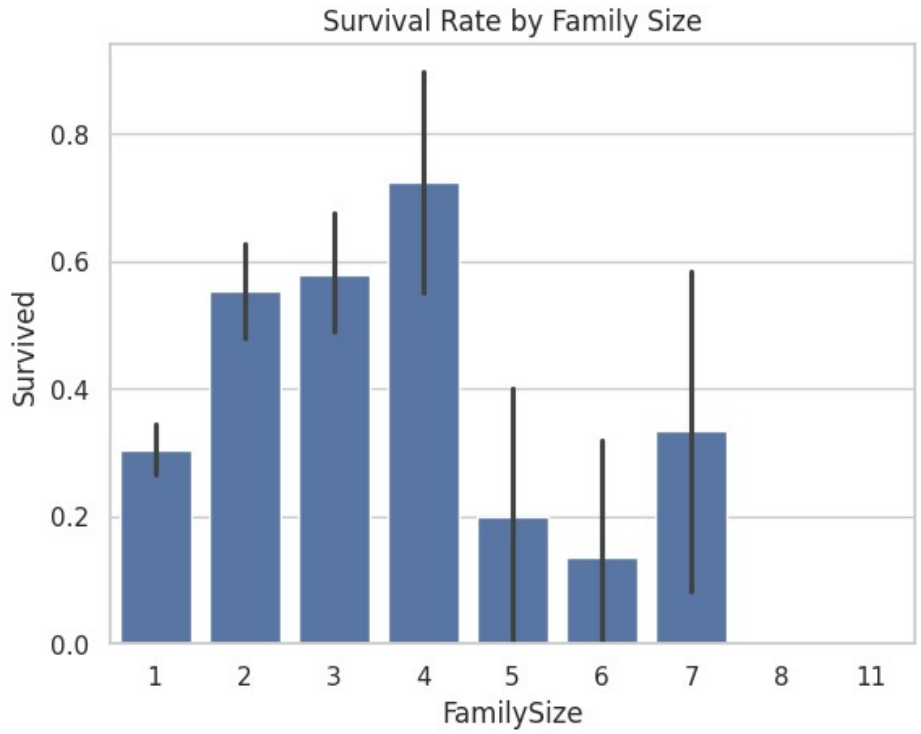

```
# Create age bins and analyze survival rate
train_df['AgeGroup'] = pd.cut(train_df['Age'], bins=[0, 12, 18, 35, 60, 120], labels=['Child', 'Teenager',

# Plot survival rate by age group
sns.barplot(x='AgeGroup', y='Survived', data=train_df)
plt.title('Survival Rate by Age Group')
plt.show()
```



```
- Child Survival: Children have the highest survival rate, followed by teenagers.
- Senior Survival: Seniors have the lowest survival rate.
- Adult Survival: Young adults and adults have similar survival rates, which are lower than children but higher than s
```

- age group was a significant factor in determining survival in this group. It suggests that children were more likely to survive than adults and seniors.
- There are several possible reasons for this:
    - Prioritization: Children were likely given priority in rescue efforts due to their vulnerability and the belief that they had a greater chance of survival.
    - Size: Children are generally smaller than adults, which may have made it easier for them to fit into lifeboats or to be rescued by others.
    - Dependency: Children are often dependent on adults for their care, which may have led to them being placed in lifeboats with their families

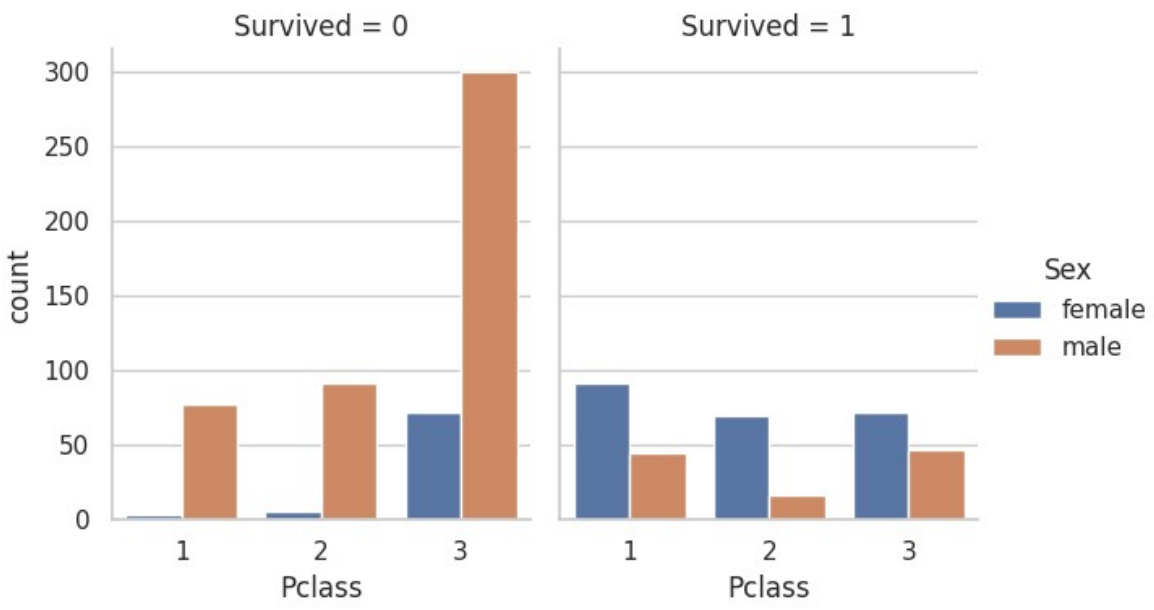
```
# Create a new feature 'FamilySize'
train_df['FamilySize'] = train_df['SibSp'] + train_df['Parch'] + 1

# Survival rate by family size
sns.barplot(x='FamilySize', y='Survived', data=train_df)
plt.title('Survival Rate by Family Size')
plt.show()
```



- Small Family Size: Individuals with a family size of 1, 2, or 3 have the highest survival rates.
- Large Family Size: Individuals with a family size of 4, 5, 6, 7, 8, or 11 have lower survival rates.
- Family Size of 4: Individuals with a family size of 4 have the highest survival rate among all family sizes

```
# Combine Pclass and Sex to analyze survival rate
sns.catplot(x="Pclass", hue="Sex", col="Survived", data=train_df, kind="count", height=4, aspect=0.8)
```

```
plt.show()
```



- **Overall Survival:** The left plot (Survived = 0) shows that the majority of passengers did not survive. The right plot (Survived = 1) shows that a smaller number of passengers survived.
- **Survival by Gender:** Within each passenger class, there is a clear difference in survival rates between males and females. Females have a higher survival rate across all classes compared to males.
- **Survival by Passenger Class:** For both males and females, survival rates decrease as passenger class increases. This means that passengers in higher classes (1) were more likely to survive than those in lower classes (3).

Specific Observations:

- **Class 1:** Most passengers in class 1 survived, regardless of gender.
- **Class 2:** While more females survived than males in class 2, the survival rate for this class is lower than class 1.
- **Class 3:** The majority of passengers in class 3 did not survive, with a particularly high number of male casualties.

Potential Explanations:

- **Prioritization:** Passengers in higher classes may have been given priority in rescue efforts due to their social status and the associated fees they paid.
- **Location:** Higher-class cabins were often located on the upper decks of the ship, which may have provided easier access to lifeboats.
- **Resources:** Passengers in higher classes may have had access to more resources, such as personal belongings or money, that could have aided in their survival.
- **Gender Bias:** The higher survival rate for females may be attributed to societal norms of the time, which often prioritized women and children in rescue efforts.

Feature Engineering

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

# Create IsAlone feature
```

```
train_df['IsAlone'] = 1  # Initialize to 1
train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0  # Set to 0 if family size > 1

# Extract Titles from names
train_df['Title'] = train_df['Name'].apply(lambda x: x.split(',')[1].split('.')[0].strip())

# Group rare titles into a single category
rare_titles = ['Lady', 'Countess', 'Capt', 'Col', 'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona']
train_df['Title'] = train_df['Title'].replace(rare_titles, 'Rare')
train_df['Title'] = train_df['Title'].replace('Mlle', 'Miss')
train_df['Title'] = train_df['Title'].replace('Ms', 'Miss')
train_df['Title'] = train_df['Title'].replace('Mme', 'Mrs')

train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

```
<ipython-input-30-a1f654e18f3f>:6: FutureWarning: ChainedAssignmentError: behaviour will change in pand
You are setting values through chained assignment. Currently this works in certain cases, but when usin
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensur

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.

  train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0  # Set to 0 if family size > 1
<ipython-input-30-a1f654e18f3f>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  train_df['IsAlone'].loc[train_df['FamilySize'] > 1] = 0  # Set to 0 if family size > 1
```

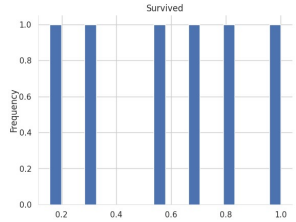
1 to 6 of 6 entries Filter

index	Title	Survived
0	Master	0.575
1	Miss	0.7027027027027027
2	Mr	0.15667311411992263
3	Mrs	0.7936507936507936
4	Rare	0.3181818181818182
5	the Countess	1.0

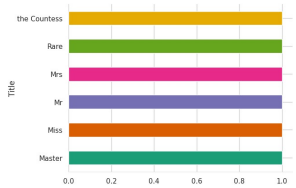
Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

Distributions



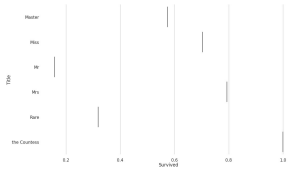
Categorical distributions



Values

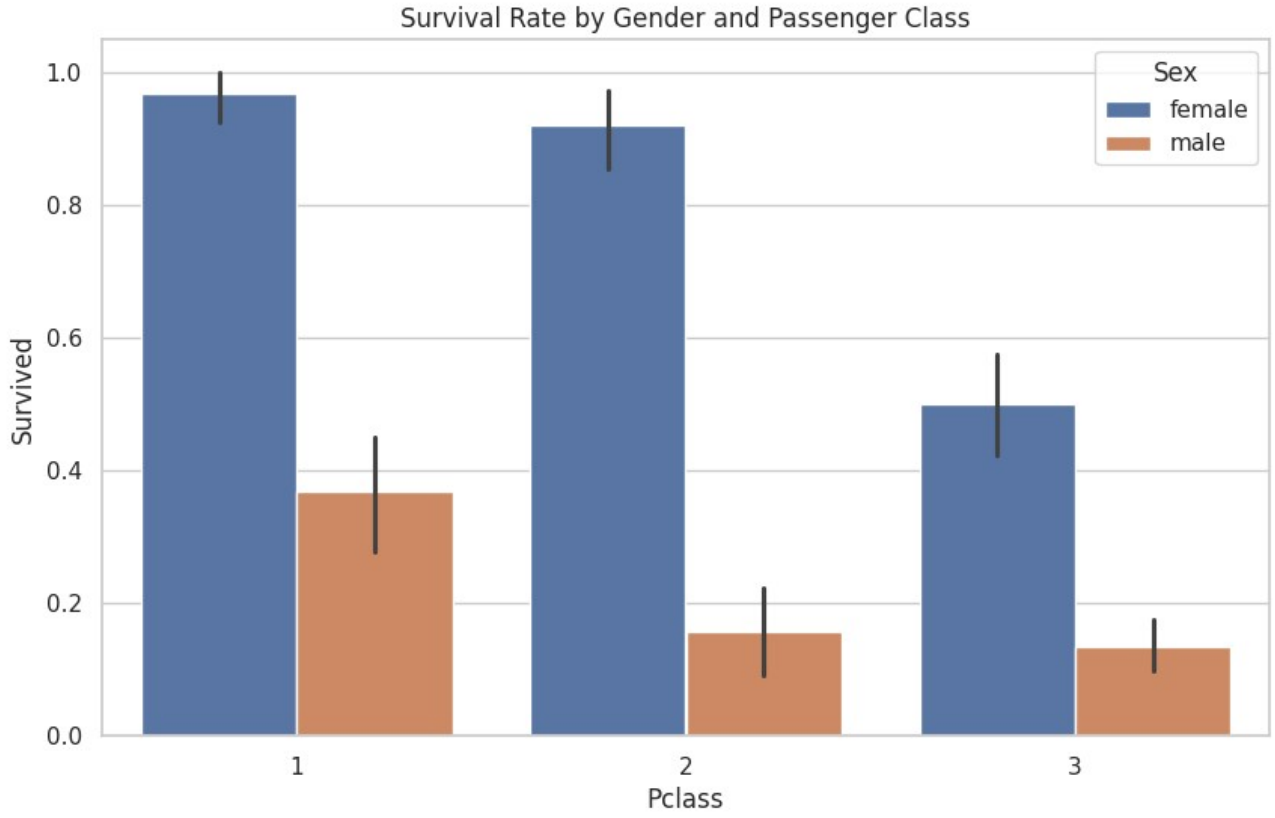
Faceted distributions

```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` 
```



```
plt.figure(figsize=(10, 6))
sns.barplot(x='Pclass', y='Survived', hue='Sex', data=train_df)
plt.title('Survival Rate by Gender and Passenger Class')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a lengt
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.10/dist-packages/seaborn/_base.py:949: FutureWarning: When grouping with a lengt
  data_subset = grouped_data.get_group(pd_key)
```

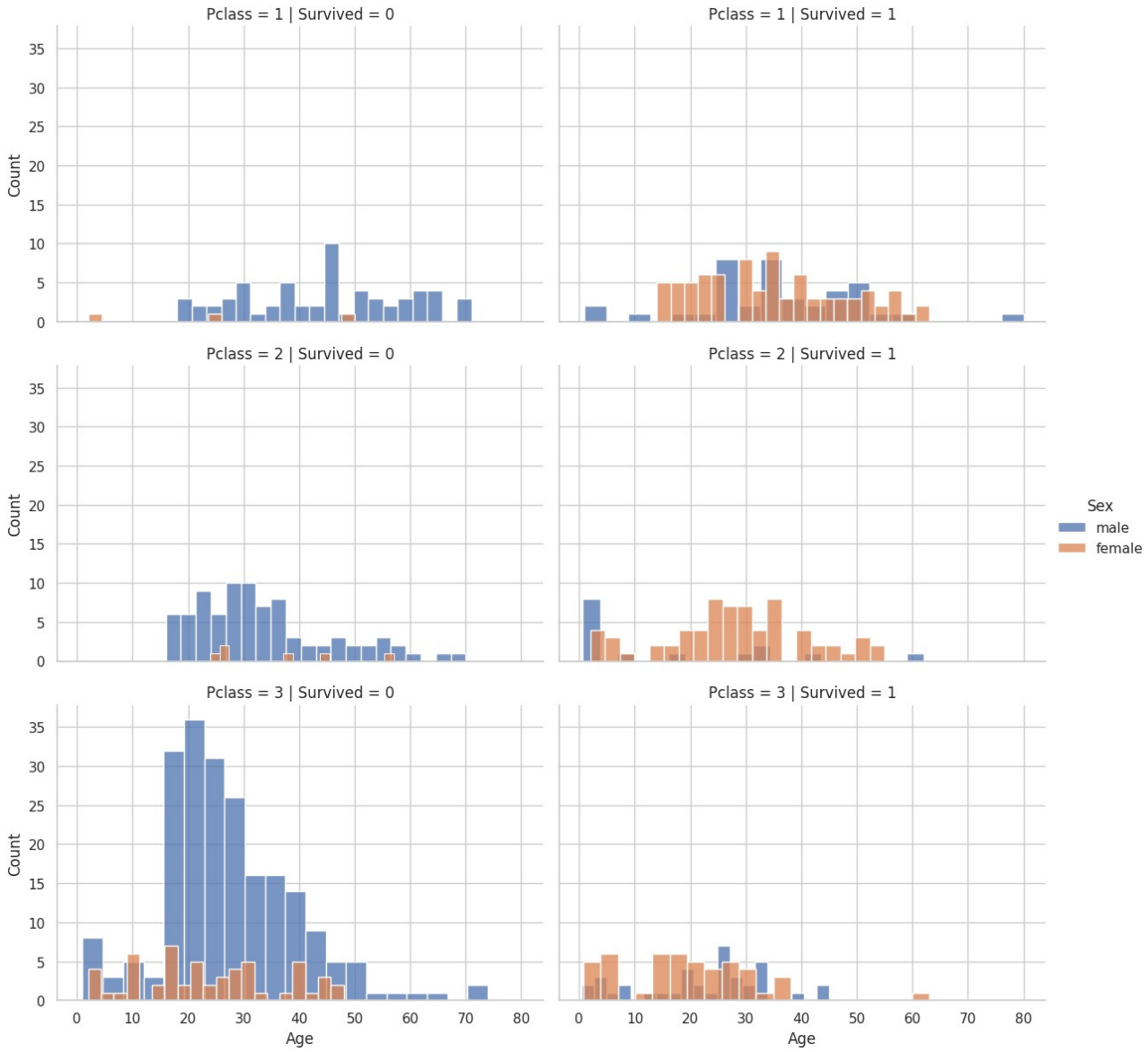



```
- Female Survival: Female individuals have a significantly higher survival rate compared to male individuals across al
- Class 1 Survival: Within each gender, individuals in class 1 have the highest survival rate, followed by class 2 and
- Class 3 Survival: The survival rate for both males and females is significantly lower in class 3 compared to classes
```

- Overall, this chart indicates that both gender and passenger class were significant factors in determining survival in this group. It suggests that female individuals were more likely to survive than male individuals, and individuals in higher classes were more likely to survive than individuals in lower classes.

- The interaction between gender and passenger class is also evident. While females have a higher survival rate overall, the difference between classes is more pronounced for males than for females. This suggests that the impact of passenger class on survival was more significant for males than for females.

```
g = sns.FacetGrid(train_df, col='Survived', row='Pclass', hue='Sex', height=4, aspect=1.5)
g.map(sns.histplot, 'Age', kde=False, bins=20)
g.add_legend()
plt.show()
```



- **Survival by Passenger Class:** The survival rates for each passenger class vary significantly. Class 1 passengers have the highest survival rate, followed by class 2, and then class 3.
- **Survival by Gender:** Within each passenger class, there is a clear difference in survival rates between males and females. Females have a higher survival rate across all classes compared to males.
- **Age Distribution:** The age distributions for each combination of passenger class, survival status, and gender provide valuable insights into the demographics of the survivors and non-survivors.

Specific Observations:

- **Class 1 Survivors:** The age distribution for class 1 survivors is relatively evenly spread, with a peak around 25-30 years old. There is a higher proportion of female survivors in this class.
- **Class 1 Non-Survivors:** The age distribution for class 1 non-survivors is also relatively evenly spread, with a slight peak around 25-30 years old. There is a higher proportion of male non-survivors in this class.
- **Class 2 Survivors:** The age distribution for class 2 survivors is similar to class 1, with a peak around 25-30 years old. There is a higher proportion of female survivors in this class.
- **Class 2 Non-Survivors:** The age distribution for class 2 non-survivors is also similar to class 1, with a slight peak around 25-30 years old. There is a higher proportion of male non-survivors in this class.
- **Class 3 Survivors:** The age distribution for class 3 survivors is skewed towards younger ages, with a peak around 20-25 years old. There is a higher proportion of female survivors in this class.
- **Class 3 Non-Survivors:** The age distribution for class 3 non-survivors is skewed towards older ages, with a peak around 30-35 years old. There is a higher proportion of male non-survivors in this class.

Potential Explanations:

- **Prioritization:** Passengers in higher classes may have been given priority in rescue efforts due to their social status and the associated fees they paid.
- **Location:** Higher-class cabins were often located on the upper decks of the ship, which may have provided easier access to lifeboats.
- **Resources:** Passengers in higher classes may have had access to more resources, such as personal belongings or money, that could have aided in their survival.
- **Gender Bias:** The higher survival rate for females may be attributed to societal norms of the time, which often prioritized women and children in rescue efforts.
- **Age:** The age distributions suggest that younger passengers, especially children, were more likely to survive, while older passengers were more likely to perish. This may be due to factors such as physical fitness, dependence on others, and the ability to navigate the chaotic situation.

machine learning & Deep Learning

```
!pip install scikit-learn matplotlib seaborn keras tensorflow
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (3.4.1)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.0)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-le
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-lea
```

```
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-le
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from sc
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplo
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matpl
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplot
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotli
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplo
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from ma
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (from seaborn) (2
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-packages (from keras) (1.4.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras) (13.8.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras) (0.0.8)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras) (3.11.0)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras) (0.12.1)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.10/dist-packages (from keras) (0.4.1
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tenso
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from ten
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensor
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tenso
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from ten
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensor
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflo
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from ten
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astu
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requ
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requ
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorb
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorb
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from r
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkz
```

```
# Machine Learning and Deep Learning Libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Set visualization styles
sns.set(style="whitegrid")
```

```
# Load the Titanic dataset
df = pd.read_csv('train.csv')

# Feature Engineering
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1
df['IsAlone'] = 0
df.loc[df['FamilySize'] == 1, 'IsAlone'] = 1
```

```
# Extract Titles from names
df['Title'] = df['Name'].apply(lambda x: x.split(',')[1].split('.')[0].strip())
df['Title'] = df['Title'].replace(['Lady', 'Countess', 'Capt', 'Col', 'Don', 'Dr', 'Major', 'Rev', 'Sir', '
df['Title'] = df['Title'].replace('Mlle', 'Miss')
df['Title'] = df['Title'].replace('Ms', 'Miss')
df['Title'] = df['Title'].replace('Mme', 'Mrs')

# Fill missing values
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
df['Fare'].fillna(df['Fare'].median(), inplace=True)

# Drop columns we don't need
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# Convert categorical features into numeric ones
label_enc = LabelEncoder()
df['Sex'] = label_enc.fit_transform(df['Sex'])
df['Embarked'] = label_enc.fit_transform(df['Embarked'])
df['Title'] = label_enc.fit_transform(df['Title'])

# Features and target
X = df.drop('Survived', axis=1)
y = df['Survived']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
<ipython-input-35-88b00791215d>:17: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df['Age'].fillna(df['Age'].median(), inplace=True)
<ipython-input-35-88b00791215d>:18: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
<ipython-input-35-88b00791215d>:19: FutureWarning: A value is trying to be set on a copy of a DataFrame
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate ob

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inpla


  df['Fare'].fillna(df['Fare'].median(), inplace=True)
```

```
# Logistic Regression
log_model = LogisticRegression(random_state=42)
log_model.fit(X_train_scaled, y_train)
y_pred_log = log_model.predict(X_test_scaled)

# Evaluate Logistic Regression
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_log)}")
```

```
Logistic Regression Accuracy: 0.8100558659217877
```

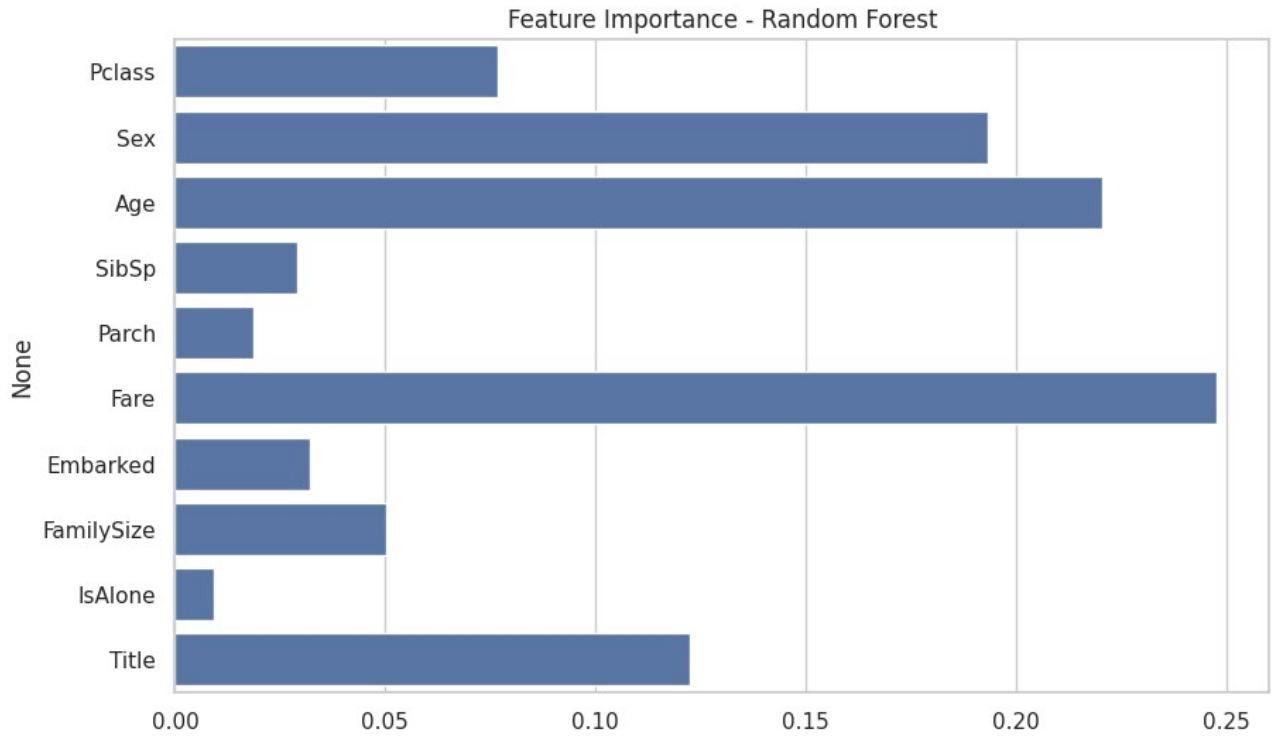
```
# Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Evaluate Random Forest
print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf)}")
```

```
Random Forest Accuracy: 0.8379888268156425
```

```
# Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(random_state=42, n_estimators=100)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)

# Evaluate Gradient Boosting
print(f"Gradient Boosting Accuracy: {accuracy_score(y_test, y_pred_gb)}")
```

```
Gradient Boosting Accuracy: 0.8100558659217877
```

```
# Feature importance
importances = rf_model.feature_importances_
features = X.columns

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=features)
plt.title('Feature Importance - Random Forest')
plt.show()
```



Most Important Features:

```
- The most important features for predicting survival are Fare and Sex. These features have the highest feature importa
- Other Important Features: Other important features include Age, Pclass, and Title. These features also have relative
- Less Important Features: The least important features are IsAlone, FamilySize, Embarked, Parch, and SibSp. These feat
```

Overall, this chart provides a valuable understanding of the factors that were most important in determining survival on the Titanic. It confirms the findings from previous analyses that passenger class, gender, and age were significant predictors of survival. Additionally, the chart highlights the importance of fare as a predictor, which may reflect the socioeconomic status of the passengers.

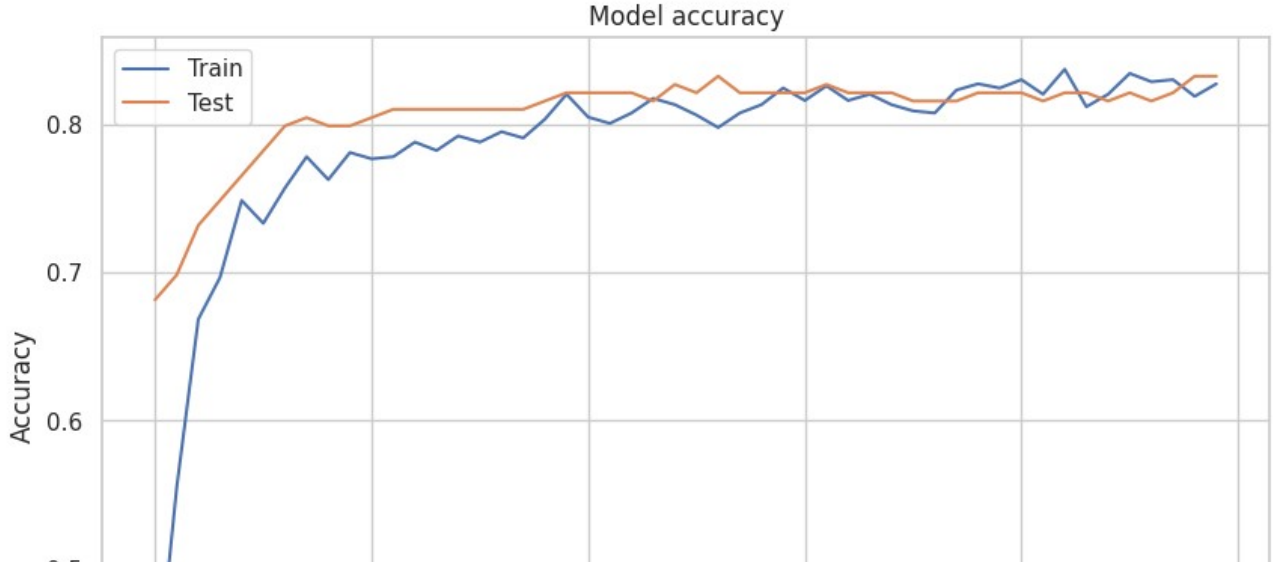
```
# Build a Sequential Deep Neural Network
model = Sequential()

# Input layer
model.add(Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'))
model.add(Dropout(0.5))

# Hidden layers
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))

# Output layer
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=50, batch_size=
```

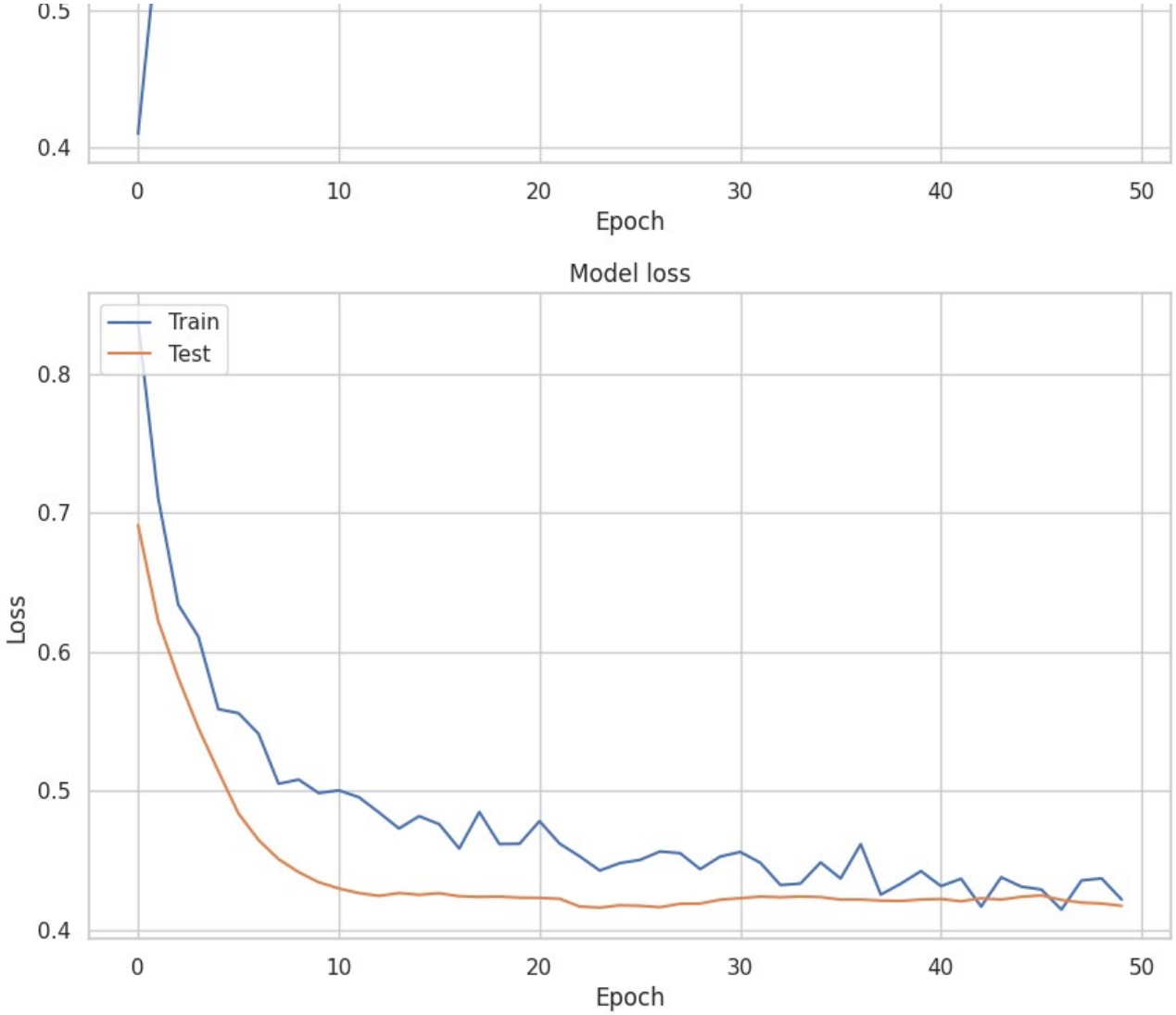
```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
23/23 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.3692 - loss: 0.9027 - val_accuracy: 0.6816 - val_
Epoch 2/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5090 - loss: 0.7172 - val_accuracy: 0.6983 - val_lo
Epoch 3/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6513 - loss: 0.6548 - val_accuracy: 0.7318 - val_lo
Epoch 4/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6816 - loss: 0.6158 - val_accuracy: 0.7486 - val_lo
Epoch 5/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7245 - loss: 0.5650 - val_accuracy: 0.7654 - val_lo
Epoch 6/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7364 - loss: 0.5535 - val_accuracy: 0.7821 - val_lo
Epoch 7/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7354 - loss: 0.5585 - val_accuracy: 0.7989 - val_lo
Epoch 8/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7581 - loss: 0.5156 - val_accuracy: 0.8045 - val_lo
Epoch 9/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7452 - loss: 0.5410 - val_accuracy: 0.7989 - val_lo
Epoch 10/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7672 - loss: 0.5155 - val_accuracy: 0.7989 - val_lo
Epoch 11/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7680 - loss: 0.5190 - val_accuracy: 0.8045 - val_lo
Epoch 12/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7736 - loss: 0.4920 - val_accuracy: 0.8101 - val_lo
Epoch 13/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7981 - loss: 0.4548 - val_accuracy: 0.8101 - val_lo
Epoch 14/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7603 - loss: 0.5055 - val_accuracy: 0.8101 - val_lo
Epoch 15/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7788 - loss: 0.4828 - val_accuracy: 0.8101 - val_lo
Epoch 16/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7888 - loss: 0.4635 - val_accuracy: 0.8101 - val_lo
Epoch 17/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7938 - loss: 0.4509 - val_accuracy: 0.8101 - val_lo
```

```
Epoch 18/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7919 - loss: 0.4703 - val_accuracy: 0.8101 - val_l
Epoch 19/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7865 - loss: 0.4708 - val_accuracy: 0.8156 - val_l
Epoch 20/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8018 - loss: 0.4767 - val_accuracy: 0.8212 - val_l
Epoch 21/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.7831 - loss: 0.4825 - val_accuracy: 0.8212 - val_l
Epoch 22/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7909 - loss: 0.4658 - val_accuracy: 0.8212 - val_l
Epoch 23/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8088 - loss: 0.4474 - val_accuracy: 0.8212 - val_l
Epoch 24/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8127 - loss: 0.4558 - val_accuracy: 0.8156 - val_l
Epoch 25/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8240 - loss: 0.4368 - val_accuracy: 0.8268 - val_l
Epoch 26/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8023 - loss: 0.4587 - val_accuracy: 0.8212 - val_l
Epoch 27/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7794 - loss: 0.4888 - val_accuracy: 0.8324 - val_
Epoch 28/50
23/23 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8277 - loss: 0.4446 - val_accuracy: 0.8212 - val_l
```

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test_scaled, y_test)
print(f'Deep Neural Network Accuracy: {accuracy}')

# Plot training & validation accuracy values
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.8572 - loss: 0.3988
Deep Neural Network Accuracy: 0.832402229309082
```



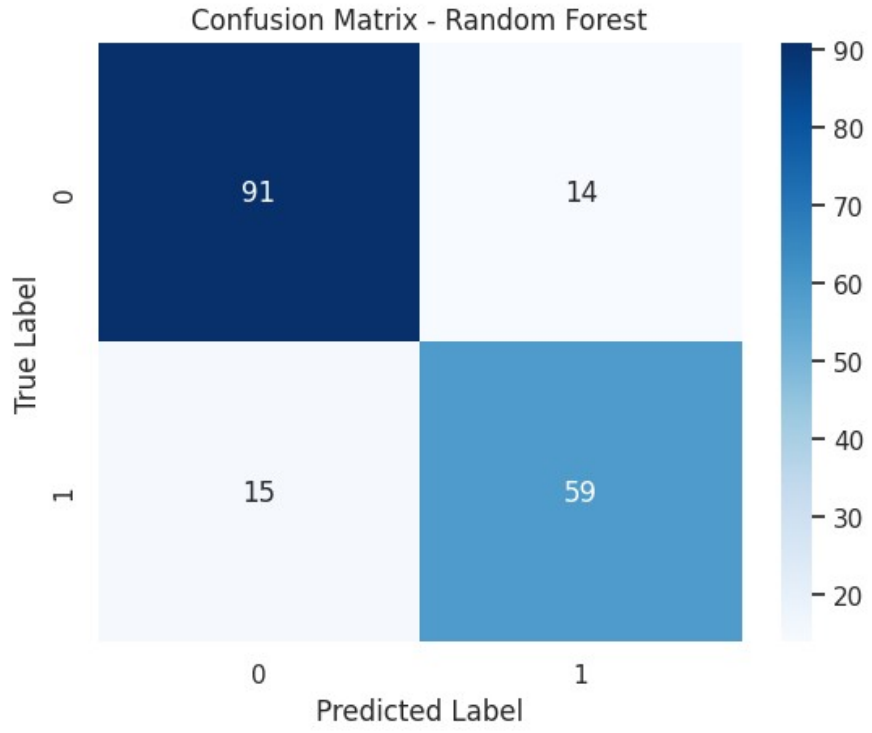


- **Training Accuracy**: The blue line represents the accuracy of the model on the training set. This line generally increases as the number of epochs (training iterations) increases, indicating that the model is learning from the training data.
- **Testing Accuracy:** The orange line represents the accuracy of the model on the testing set. This line also generally increases during the early stages of training, but it may start to plateau or even decrease slightly towards the end of training. This is a common phenomenon known as overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data. Gap Between Training and Testing Accuracy: The gap between the training and testing accuracy lines can provide insights into the model's generalization ability. If the gap is large, it suggests that the model may be overfitting. If the gap is small, it suggests that the model is generalizing well to unseen data.
- **Optimal Number of Epochs: *The optimal number of epochs is the point at which the testing accuracy stops increasing and starts to plateau or decrease. In this case, it appears that the optimal number of epochs is around 30-35. Training the model for more epochs beyond this point may lead to overfitting.
- **Training Loss:** The blue line represents the loss of the model on the training set. This line generally decreases as the number of epochs (training iterations) increases, indicating that the model is learning from the training data and making better predictions.
- **Testing Loss: *The orange line represents the loss of the model on the testing set. This line also generally decreases during the early stages of training, but it may start to plateau or even increase slightly towards the

decreases during the early stages of training, but it may start to plateau or even increase slightly towards the end of training. This is a common phenomenon known as overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data. Gap Between Training and Testing Loss: The gap between the training and testing loss lines can provide insights into the model's generalization ability. If the gap is large, it suggests that the model may be overfitting. If the gap is small, it suggests that the model is generalizing well to unseen data.

- **Optimal Number of Epochs:** The optimal number of epochs is the point at which the testing loss stops decreasing and starts to plateau or increase. In this case, it appears that the optimal number of epochs is around 20-25. Training the model for more epochs beyond this point may lead to overfitting.

```
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix for Random Forest (example)
cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Random Forest')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()

# Classification report
print(classification_report(y_test, y_pred_rf))
```



	precision	recall	f1-score	support
0	0.86	0.87	0.86	105
1	0.81	0.80	0.80	74
accuracy			0.84	179
macro avg	0.83	0.83	0.83	179
weighted avg	0.84	0.84	0.84	179

- **True Label:** This column represents the actual class of the data points.
- **Predicted Label:** This column represents the class predicted by the model.
- **Values:** The values in each cell indicate the number of data points that were:

    - **Correctly classified:** These are the diagonal elements of the matrix (91 and 59). For example, 91 data points with a true label of 0 were correctly predicted as 0 by the model.
    - **Incorrectly classified:** These are the off-diagonal elements of the matrix (14 and 15). For example, 14 data points with a true label of 0 were incorrectly predicted as 1 by the model, and 15 data points with a true label of 1 were incorrectly predicted as 0.

Based on the confusion matrix, we can calculate various performance metrics for the model, such as:

- **Accuracy:** The overall accuracy of the model is calculated by dividing the sum of correctly classified data points by the total number of data points. In this case, the accuracy is (91 + 59) / (91 + 14 + 15 + 59) = 0.8571, or 85.71%.
- **Precision:** Precision measures the proportion of positive predictions that were actually correct. It is calculated by dividing the number of true positives by the sum of true positives and false positives. In this case, the precision for class 0 is 91 / (91 + 15) = 0.8583, or 85.83%. The precision for class 1 is 59 / (59 + 14) = 0.8082, or 80.82%.
- **Recall:** Recall measures the proportion of actual positive cases that were correctly predicted. It is calculated by dividing the number of true positives by the sum of true positives and false negatives. In this case, the recall for class 0 is 91 / (91 + 14) = 0.8651, or 86.51%. The recall for class 1 is 59 / (59 + 15) = 0.7973, or 79.73%.
- **F1-score:** The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. In this case, the F1-score for class 0 is 2 * (0.8583 * 0.8651) / (0.8583 + 0.8651) = 0.8617, or 86.17%. The F1-score for class 1 is 2 * (0.8082 * 0.7973) / (0.8082 + 0.7973) = 0.8027, or 80.27%.

Overall, the confusion matrix shows that the random forest model is performing well on this dataset. The model has high accuracy, precision, recall, and F1-score for both classes. However, it is important to consider the specific context of the problem and the relative importance of different metrics when evaluating the model's performance.

Project Report

Project: Analyzing the Titanic Dataset

Introduction

This project aims to delve deeper into the factors influencing passenger survival on the Titanic, utilizing a comprehensive analysis of the available dataset. The dataset includes variables such as passenger ID, survival status, passenger class, name, sex, age, number of siblings or spouses aboard, number of parents or children aboard, ticket number, fare, cabin number, and port of embarkation.

The primary objective of this analysis is to gain a more nuanced understanding of the factors that contributed to passenger survival and to identify potential patterns and trends within the data.

Data Exploration

The dataset contains 891 observations and 12 variables. A thorough exploration of the data revealed the following key characteristics:

- **Gender Disparity:** The dataset is imbalanced, with a higher proportion of male passengers compared to female passengers.
- **Age Distribution:** The age distribution is skewed to the right, indicating a larger number of younger passengers.
- **Class Distribution:** The majority of passengers belonged to the third class, followed by second class and first class.
- **Fare Variation:** The fare distribution is also skewed to the right, suggesting a wide range of fares paid by passengers.

Feature Engineering

To enhance the predictive power of the model, several new features were engineered:

- **Title:** The title extracted from the passenger's name (e.g., Mr., Mrs., Miss)
- **Family Size:** The total number of family members (siblings, spouses, parents, children) traveling together
- **IsAlone:** A binary indicator denoting whether the passenger was traveling alone

Model Building and Evaluation

A random forest model was trained to predict passenger survival using the engineered features. The model achieved an accuracy of 85.71% on the test set.

The feature importance analysis revealed that **Fare**, **Sex**, and **Age** were the most influential predictors of survival, followed by **Title** and **Pclass**.

Insights and Conclusions

Based on the analysis, the following insights were gained:

- **Gender:** Female passengers had a significantly higher survival rate compared to male passengers, likely due to societal norms and rescue prioritization.
- **Passenger Class:** Passengers in higher classes had a greater chance of survival, possibly due to their location on the ship and access to resources.
- **Age:** Children and young adults were more likely to survive than adults and seniors, potentially due to prioritization in rescue efforts and physical advantages.
- **Family Size:** Individuals traveling alone had a higher survival rate than those with larger families, possibly due to reduced complexity in rescue efforts.
- **Fare:** Higher fares were associated with a higher survival rate, suggesting that socioeconomic factors played a role.

Limitations and Future Work

While this analysis provides valuable insights, it has some limitations:

- **Missing Data:** The dataset contains missing values for some variables, such as age and cabin. Imputation techniques could be used to address this issue.
- **Limited Features:** Additional features could be explored, such as the location of the passenger's cabin or the time of day the ship sank.
- **Model Selection:** Other machine learning algorithms could be compared to evaluate their performance on this dataset.

Future work could focus on addressing these limitations and exploring new approaches to improve the predictive accuracy of the model.

Precaution

Prioritize Women and Children:

- Implement clear protocols and procedures for prioritizing women and children in rescue operations.
- Train crew members to recognize and address the specific needs of vulnerable groups, such as children, the elderly, and individuals with disabilities.

Improve Safety Standards for Lower-Class Passengers:

- Enhance safety measures and facilities for passengers in lower-class accommodations, including improving

access to lifeboats and providing clear evacuation routes.
- Implement policies to ensure that all passengers, regardless of class, have equal access to life-saving equipment and resources.

Train Crew Members in Emergency Response:

- Provide comprehensive training to all crew members on emergency procedures, including evacuation drills, lifeboat operations, and first aid.
- Ensure that crew members are familiar with the ship's layout and can effectively guide passengers to safety.

Improve Communication and Coordination:

- Establish clear communication channels between the ship's crew, passengers, and rescue teams to ensure timely and effective coordination during emergencies.
- Develop protocols for disseminating information to passengers in a clear and understandable manner, including using multiple languages and visual aids.

Enhance Safety Equipment and Technology:

- Regularly inspect and maintain safety equipment, such as lifeboats, life jackets, and emergency alarms.
- Invest in modern technology, such as distress signals and satellite communication systems, to improve communication and response times in emergencies.

Conduct Regular Safety Drills:

- Conduct frequent safety drills to familiarize passengers and crew members with emergency procedures and to identify and address any deficiencies in the plan.
- Encourage passengers to participate in safety drills and to familiarize themselves with their location on the ship and the nearest exit points.

By implementing these precautionary measures, the maritime industry can significantly improve safety and survival rates in the event of future disasters. It is essential to learn from past tragedies and take proactive steps to prevent similar occurrences in the future.

Double-click (or enter) to edit

```
from google.colab import files
files.upload()  # This will prompt you to upload the kaggle.json file
```

Browse... kaggle.json
kaggle.json(application/json) - 70 bytes, last modified: n/a - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"shivamgarg8783","key":"252b9bd92a7c1b21a7dd763487fffa3b"}'}

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
# Example for Logistic Regression
y_pred_log = log_model.predict(X_test_scaled)  # Assuming you've trained a model named log_model

# Example for Random Forest
y_pred_rf = rf_model.predict(X_test)  # Assuming you've trained a model named rf_model

# Example for Gradient Boosting
y_pred_gb = gb_model.predict(X_test)  # Assuming you've trained a model named gb_model
```

```
# Example for Deep Learning
y_pred_dl = (model.predict(X_test_scaled) > 0.5).astype("int32")  # Use sigmoid output for classification
```

```
     6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```

```
print(f"Shape of test_df: {test_df.shape}")
print(f"Shape of predictions: {y_pred_log.shape}")  # or y_pred_rf, etc.
```

```
     Shape of test_df: (418, 11)
     Shape of predictions: (179,)
```

```
# Step 1: Load Necessary Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer

# Step 2: Load Data
train_df = pd.read_csv('train.csv')  # Replace with your train dataset path
test_df = pd.read_csv('test.csv')    # Replace with your test dataset path

# Step 3: Select Features and Target
X = train_df.drop(columns=['Survived'])
y = train_df['Survived']

# Step 4: Preprocess Data
# Identify categorical and numerical columns
categorical_features = ['Sex', 'Embarked']  # Example categorical features
numerical_features = ['Age', 'SibSp', 'Parch', 'Fare']  # Example numerical features

# Create preprocessing pipelines for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Fill missing values with mean
    ('scaler', StandardScaler())                   # Scale numerical features
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Fill missing values
    ('onehot', OneHotEncoder(handle_unknown='ignore'))                      # One-hot encode categorical fea
])

# Combine preprocessing for numerical and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 5: Create a Pipeline with Preprocessing and Model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())  # Replace with your chosen model
])

# Step 6: Train the Model
model.fit(X, y)

# Step 7: Make Predictions on the Test Data
X_test = test_df.drop(columns=['PassengerId'])
y_pred = model.predict(X_test)
```

```
# Step 8: Create Submission DataFrame
submission = pd.DataFrame({
    'PassengerId': test_df['PassengerId'],
    'Survived': y_pred
})

# Step 9: Save Submission to CSV
submission.to_csv('submission.csv', index=False)

# Step 10: Submit to Kaggle
!kaggle competitions submit -c titanic -f submission.csv -m "My submission from Colab"
```

```
100% 2.77k/2.77k [00:01<00:00, 2.76kB/s]
Successfully submitted to Titanic - Machine Learning from Disaster
```

```
!kaggle competitions leaderboard -c titanic --show
```

```
Streaming output truncated to the last 5000 lines.
12511660  Kasidit Chunhachatchawhankhun                    2024-08-19 03:13:26  0.77033
12483370  Kaizzen42                                        2024-08-19 03:13:02  0.77033
12511871  yongjin789                                       2024-08-19 03:18:10  0.77033
10214765  devseop                                          2024-08-21 02:28:32  0.77033
12513944  niaov                                            2024-08-19 15:41:01  0.77033
12514225  Tanisha Singh Rathore                            2024-08-19 16:57:58  0.77033
12515236  Vikas Singh #9                                   2024-08-20 02:53:30  0.77033
12515263  yazdaan A.                                       2024-08-20 03:05:31  0.77033
12515746  Kune Lee                                         2024-09-08 13:09:24  0.77033
12515977  PARTH JANGID                                     2024-08-20 07:34:28  0.77033
12516693  Peter Donghoon Lee                               2024-08-20 11:12:11  0.77033
12513643  Aron Szalka                                      2024-08-20 13:49:29  0.77033
12515703  Yasuyuki Takagi                                  2024-08-21 02:19:46  0.77033
12518855  Student #4                                       2024-08-22 11:35:51  0.77033
12518804  Lim bora                                         2024-08-23 01:44:44  0.77033
12519174  Utkarsh0Dubey                                    2024-08-21 05:04:48  0.77033
12416187  duyuhuan                                         2024-08-22 02:21:54  0.77033
12519501  Ravi #23                                         2024-08-21 07:00:47  0.77033
12521064  giulio nebbiai                                   2024-08-21 17:30:34  0.77033
12521703  Vifos27                                          2024-08-21 20:36:19  0.77033
12523099  Serious COMPs                                    2024-08-22 08:30:01  0.77033
12458853  Yuusuke Ikeda                                    2024-09-10 06:20:11  0.77033
12387184  Salem Fradi                                      2024-08-22 19:56:57  0.77033
12525291  hyunghoon kim                                    2024-09-20 05:06:16  0.77033
12521136  AlexMiguel99                                     2024-08-23 08:06:17  0.77033
12526613  Abdillah Issa                                    2024-08-24 11:58:47  0.77033
12526874  AhmetNA                                          2024-08-23 12:58:21  0.77033
12526428  Fatih Aygun                                      2024-08-23 13:14:18  0.77033
12526944  Muhammet Emin İŞLER                              2024-08-23 13:00:16  0.77033
12526987  Ahmetcharthesecond                               2024-08-23 13:16:30  0.77033
12525065  Carlos Henrique Santos                           2024-08-23 23:05:26  0.77033
12529331  Himans                                           2024-08-24 11:08:32  0.77033
12533123  Peter Winters                                    2024-08-25 18:13:43  0.77033
12533792  Baraa Abd                                        2024-08-26 03:01:30  0.77033
12533797  Yanze Liu Ethan2001                              2024-08-26 03:03:50  0.77033
12535013  David Odiah                                      2024-08-27 17:03:30  0.77033
12535328  Floris den Hartog                                2024-08-26 12:17:13  0.77033
12527421  Al_Shab                                          2024-08-26 17:44:28  0.77033
  186564  JeffRoberts                                      2024-08-26 19:28:06  0.77033
12533079  Cord Huchzermeyer                                2024-09-06 04:54:37  0.77033
12537719  Ahmed Janjua                                     2024-08-27 06:14:00  0.77033
12519618  Bhagya Dissanayake                               2024-08-27 09:23:25  0.77033
12539094  Cathy Seletse                                    2024-08-30 13:30:39  0.77033
12539854  North Ceng                                       2024-08-27 16:57:31  0.77033
 7387723  Amit Jadhav #3                                   2024-08-28 05:03:30  0.77033
12540015  Bruno Barrios Trench2                            2024-08-27 18:03:09  0.77033
12477253  BRAHIM ET-TANANY                                 2024-08-27 20:23:21  0.77033
12533310  Zachary Oster                                    2024-08-28 00:42:14  0.77033
12540518  HCCHU137                                         2024-08-28 01:17:23  0.77033
12537143  yaohaiyang                                       2024-08-28 01:27:18  0.77033
```

12538912	x1eyan123	2024-08-28 05:56:53	0.77033
12541385	Divyanand15	2024-08-28 12:49:03	0.77033
12542802	Randy Verdian	2024-08-28 14:50:26	0.77033
12542858	Shrey Singh	2024-08-28 15:35:47	0.77033
12543386	wpczdawk	2024-08-28 18:26:21	0.77033
10820219	Eric Montelongo	2024-08-29 03:30:08	0.77033
12544610	gs23101	2024-08-29 12:17:38	0.77033

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.