steps to clean and apply a ML model on the Titanic dataset

1. Load the Titanic dataset.
2. Raw data inspection and understanding.
3. Data Quality Checks.
4. Statistical Summary and Visualization.
5. Data Cleaning (Handling Missing Values, Outliers, Duplicates).
6. Feature Engineering (Creating new features, Encoding categorical variables).
7. Data Normalization/Standardization.
8. ML readiness (Splitting data into training and testing sets).
9. Model Selection and Training.

1. Problem Statement & Goal
2. Load Titanic Dataset
3. Initial Data Inspection
4. Data Quality Report
5. Univariate EDA
6. Bivariate EDA
7. Statistical Measures (IQR, Skewness, Kurtosis)
8. Outlier Detection & Treatment
9. Missing Value Strategy
10. Feature Engineering
11. Encoding & Scaling
12. Train-Test Split
13. Baseline ML Model
14. Improved ML Model
15. Final Evaluation & Learnings

Raw Data Inspection and Understanding

- Load the dataset using pandas.
- shape,columns,datatypes,info,describe,head,tail
- first loook at rows

In [443…

```
import pandas as pd
import numpy as np
```

```
import seaborn as sns
```

In [444…

```
df = sns.load_dataset('titanic')
```

In [445…

```
df.head()
```

Out[445…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adul
0	0	3	male	22.0	1	0	7.2500	S	Third	man	
1	1	1	female	38.0	1	0	71.2833	C	First	woman	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	
4	0	3	male	35.0	0	0	8.0500	S	Third	man	

In [446…

```
df.shape
```

Out[446…

```
(891, 15)
```

In [447…

```
df.columns
```

Out[447…

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [448…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    category
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

In [449…

```
df.describe()
```

Out[449…

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Data Quality Checks

- Check for missing values.
- Check for duplicates.
- Check for inconsistent data types.
- invvalid values

In [450…

```
df.isnull().sum()
```

Out[450…

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [451…

```
missing_pct = df.isnull().mean() * 100
missing_pct.sort_values(ascending=False)
```

Out[451…

```
deck           77.216611
age            19.865320
embarked        0.224467
embark_town     0.224467
sex             0.000000
pclass          0.000000
survived        0.000000
fare            0.000000
parch           0.000000
sibsp           0.000000
class           0.000000
adult_male      0.000000
who             0.000000
alive           0.000000
alone           0.000000
dtype: float64
```

In [452…

```
df.duplicated().sum()
```

Out[452…

```
np.int64(107)
```

In [453…

```
for col in df.select_dtypes(include="object"):
    print(f"{col}: {df[col].nunique()} unique values")
```

```
sex: 2 unique values
embarked: 3 unique values
who: 3 unique values
embark_town: 3 unique values
alive: 2 unique values
```

In [454…

```
df.describe(include="all")
```

Out[454…

	survived	pclass	sex	age	sibsp	parch	fare
count	891.000000	891.000000	891	714.000000	891.000000	891.000000	891.000000
unique	NaN	NaN	2	NaN	NaN	NaN	NaN
top	NaN	NaN	male	NaN	NaN	NaN	NaN
freq	NaN	NaN	577	NaN	NaN	NaN	NaN
mean	0.383838	2.308642	NaN	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	NaN	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	NaN	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	NaN	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	NaN	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	NaN	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	NaN	80.000000	8.000000	6.000000	512.329200

🧠 Data Quality Summary (write this in markdown)

In your notebook, add a markdown cell like:

Data Quality Findings

Missing values in age, deck, embark_town

Deck has excessive missing data

No duplicate rows

Fare appears heavily skewed

Target leakage column present (alive)

Exploratory Data Analysis (EDA)

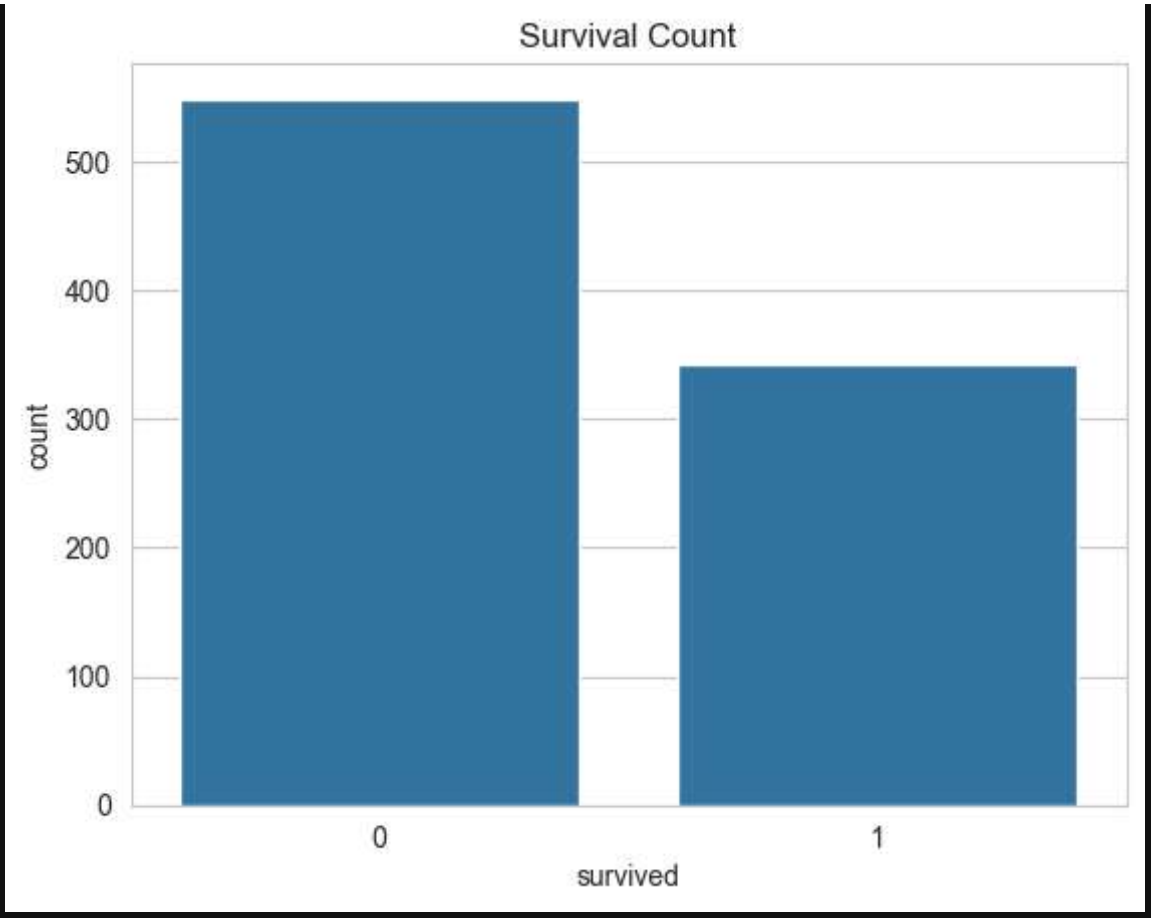
- univariate analysis
- Bivariate analysis
- Multivariate analysis
- correlation analysis
- Distribution of numerical features

In [455…

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")
```

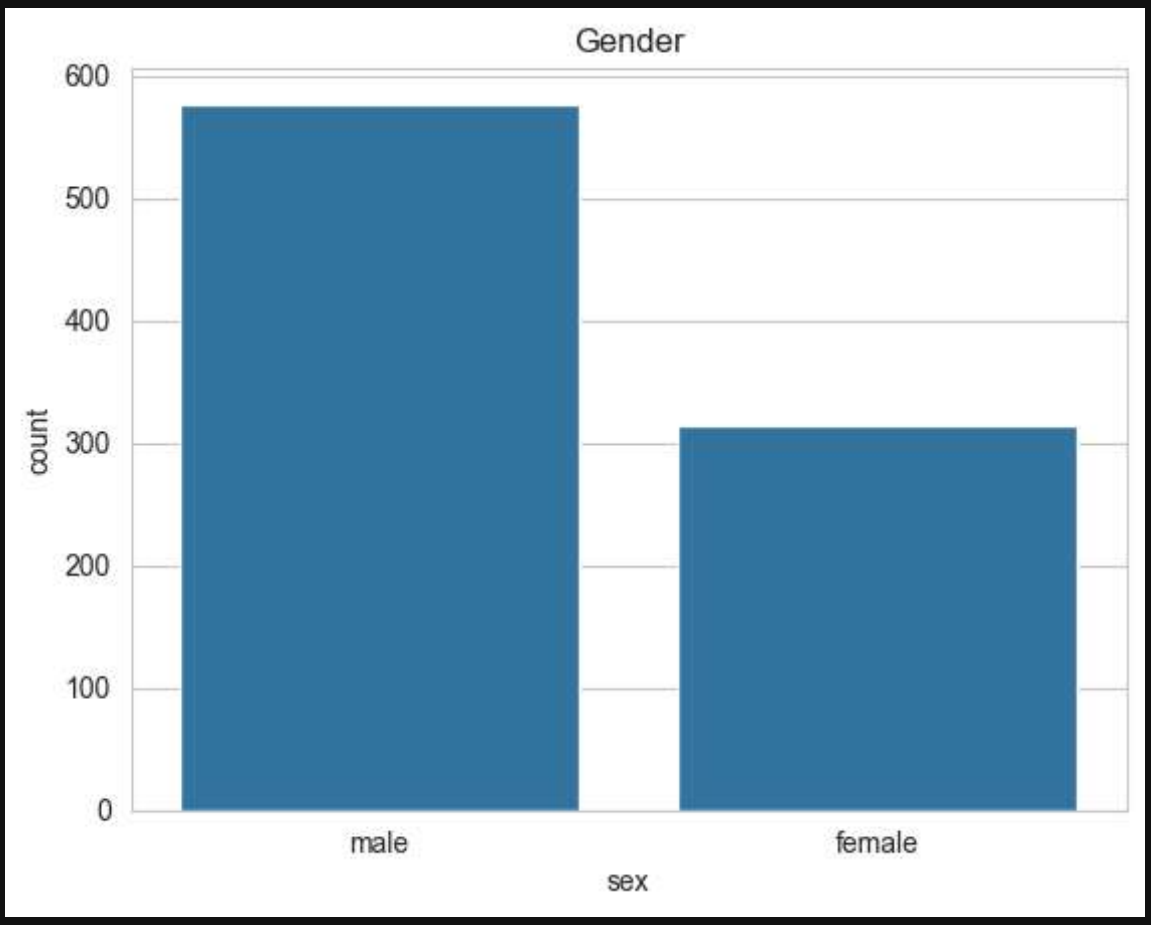
In [456…

```
# analysis target
sns.countplot(x='survived', data=df)
plt.title('Survival Count')
plt.show()
```



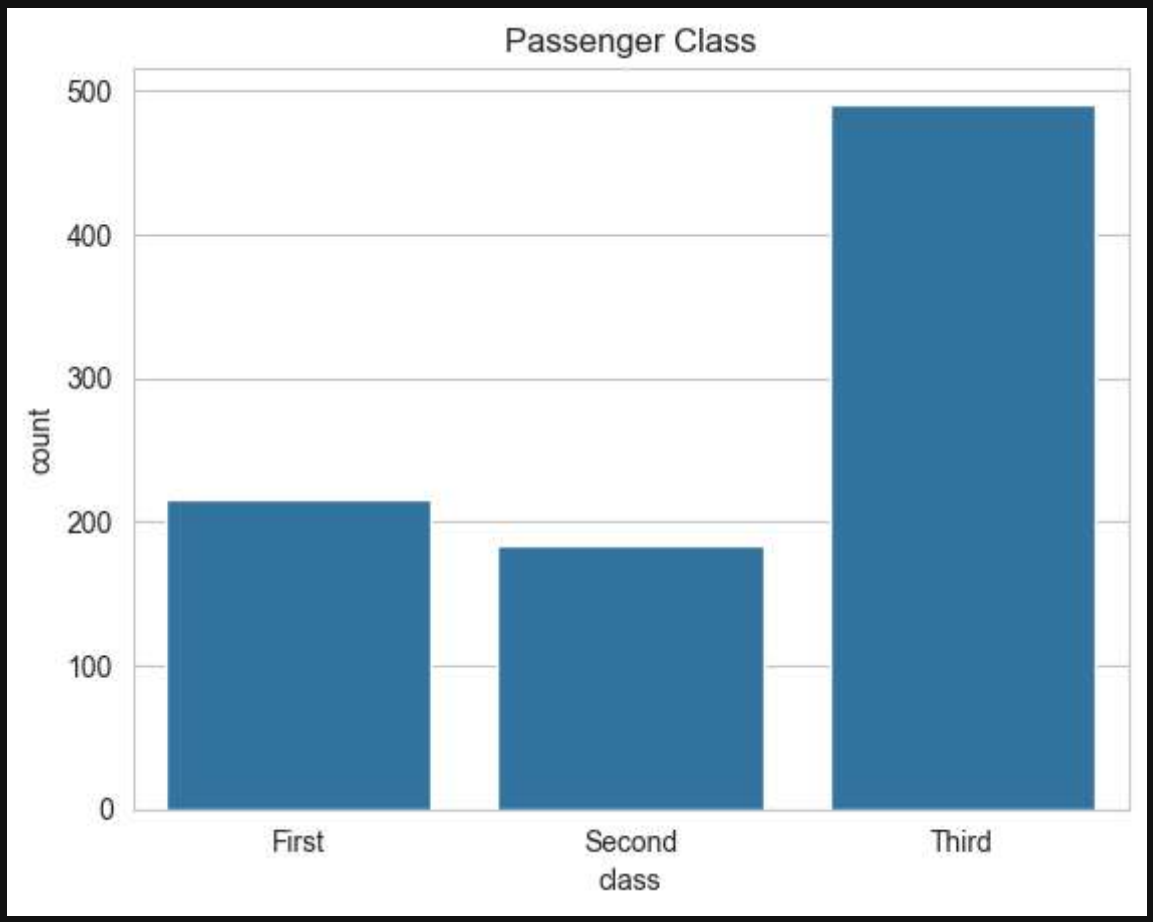
In [457…

```
sns.countplot(x='sex',data=df)
plt.title('Gender')
plt.show()
```



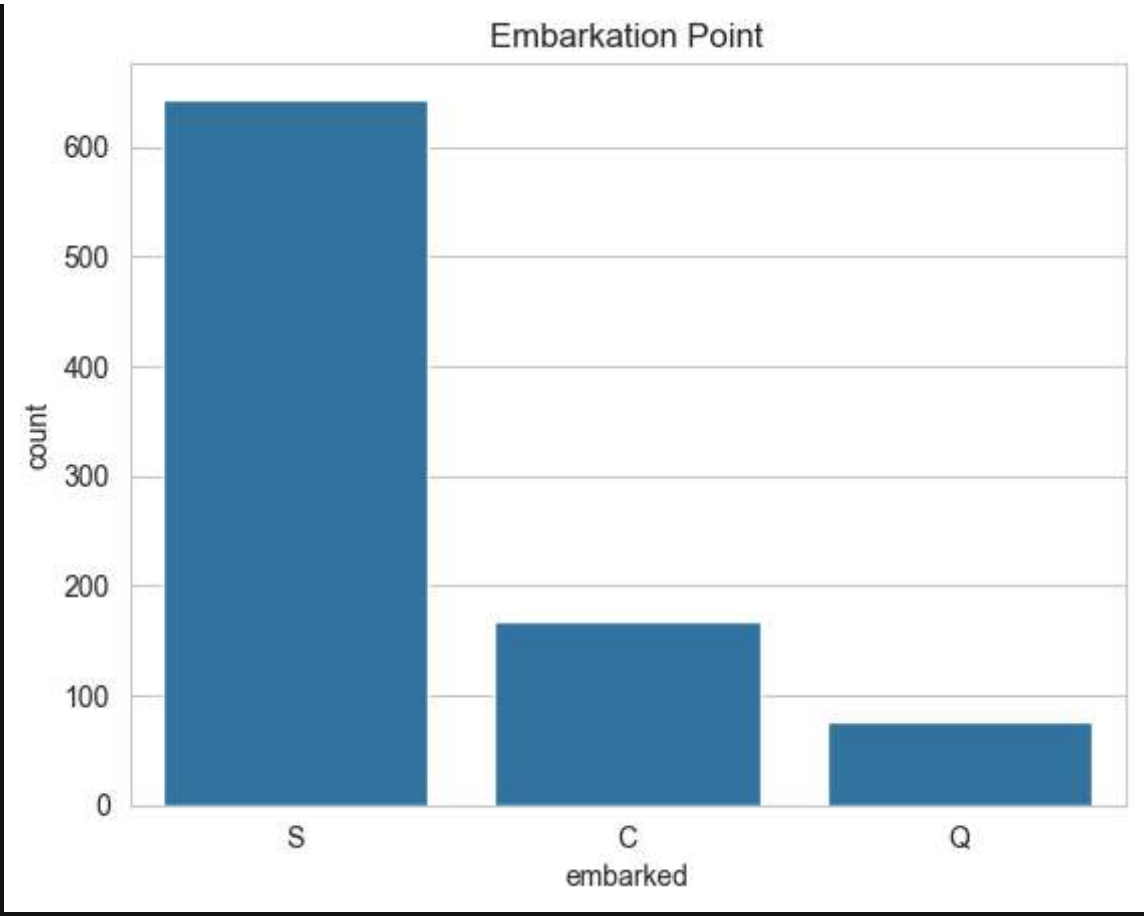
In [458…

```
sns.countplot(x='class',data=df)
plt.title('Passenger Class')
plt.show()
```



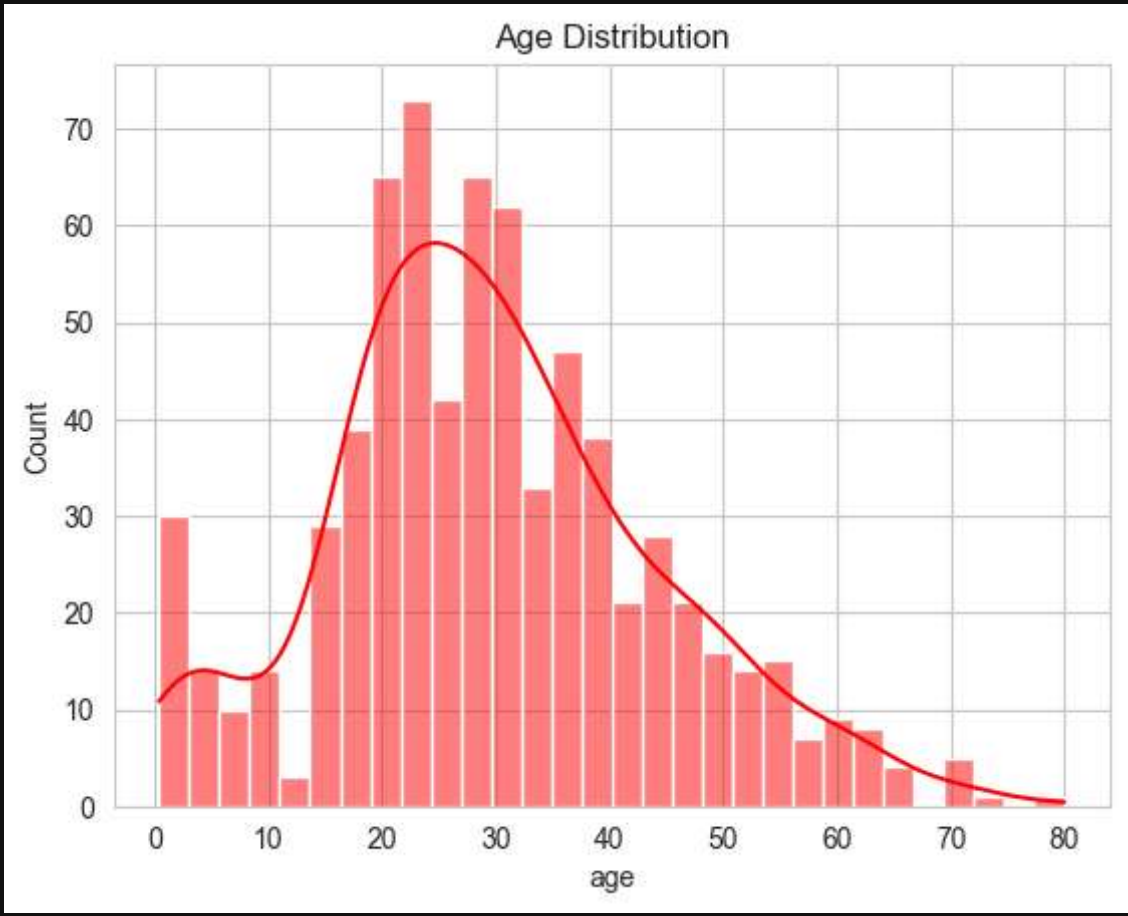
In [459…

```
sns.countplot(x='embarked',data=df)
plt.title('Embarkation Point')
plt.show()
```



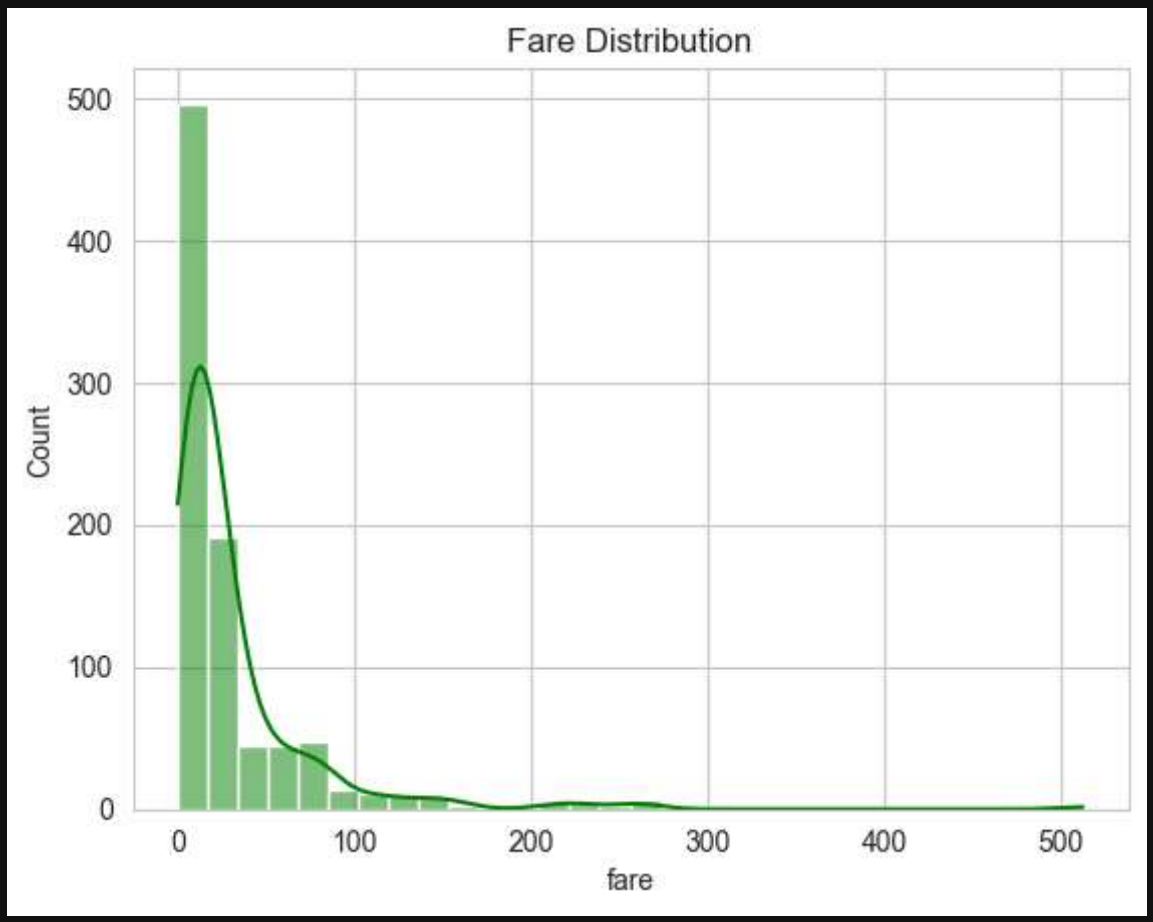
In [460…

```
sns.histplot(df['age'].dropna(), bins=30, kde=True, color='red')
plt.title('Age Distribution')
plt.show()
```



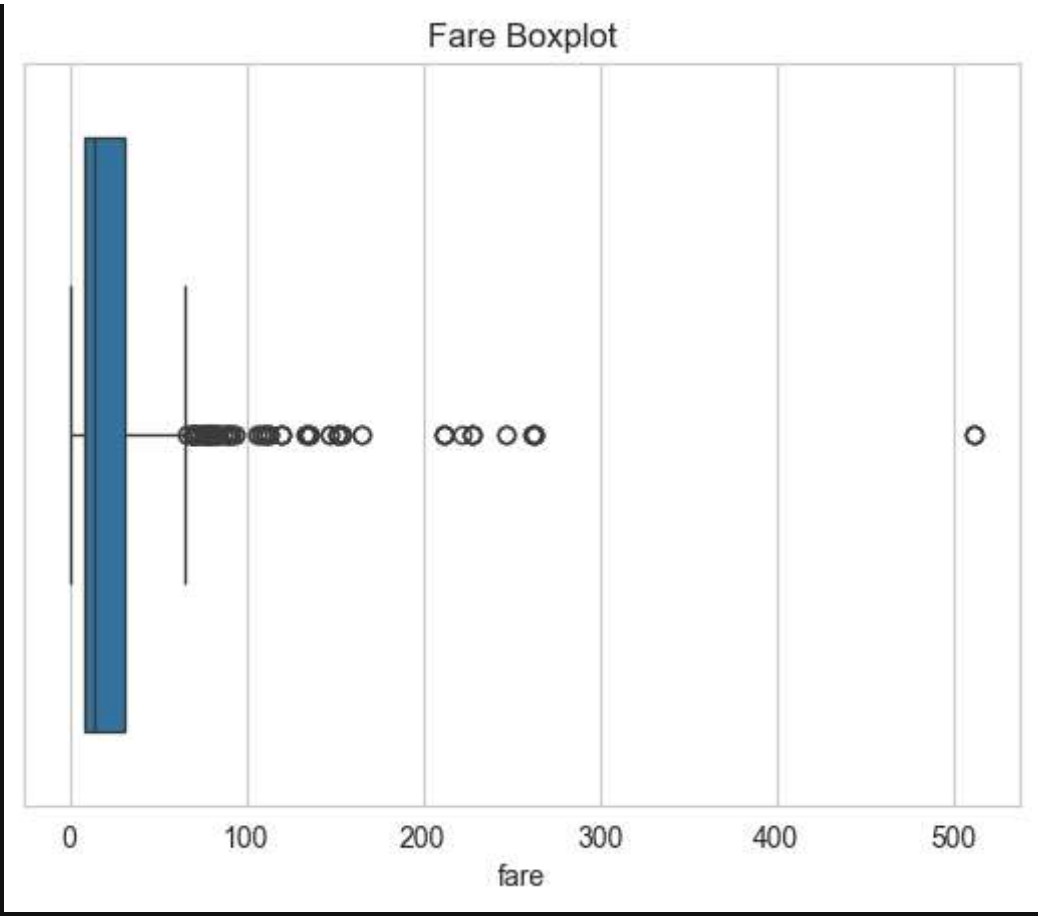
In [461…

```
sns.histplot(df['fare'], bins=30, kde=True, color='green')
plt.title('Fare Distribution')
plt.show()
```



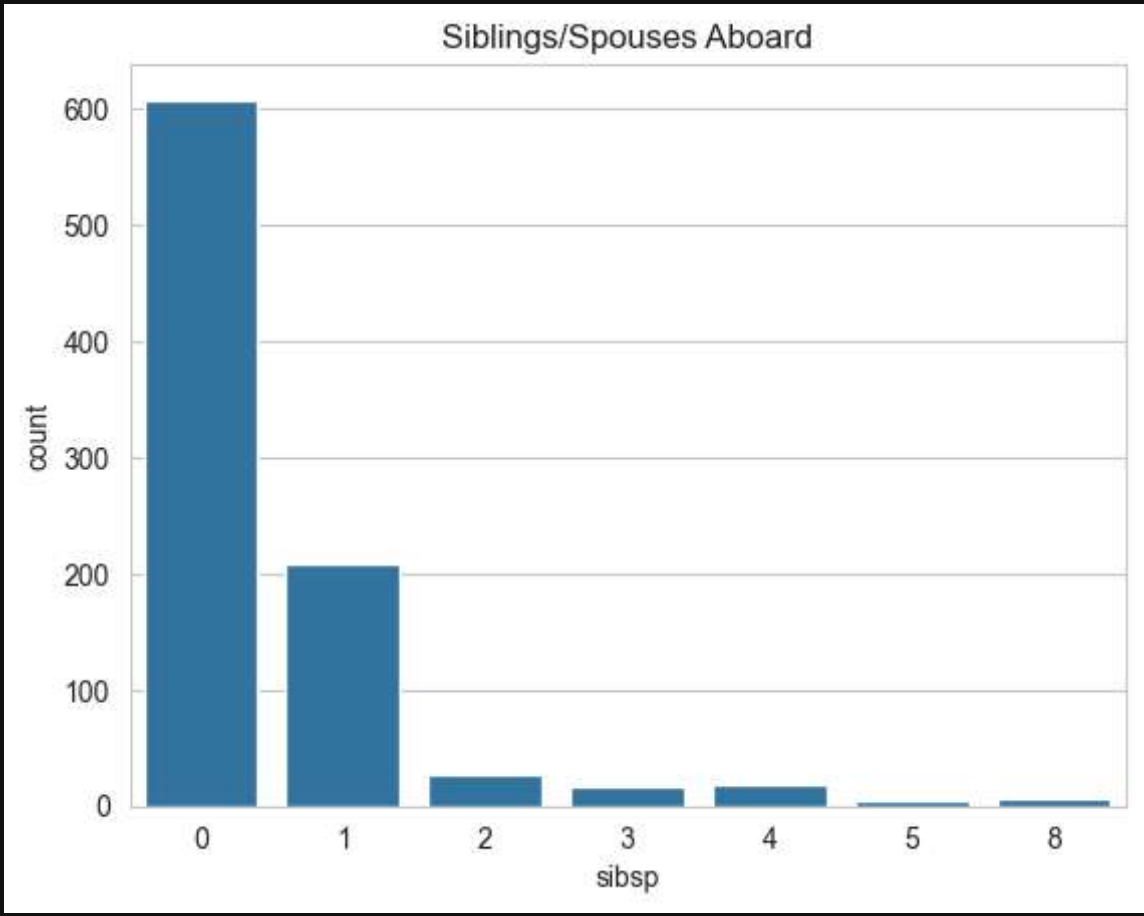
In [462…

```
sns.boxplot(x=df['fare'])
plt.title('Fare Boxplot')
plt.show()
```



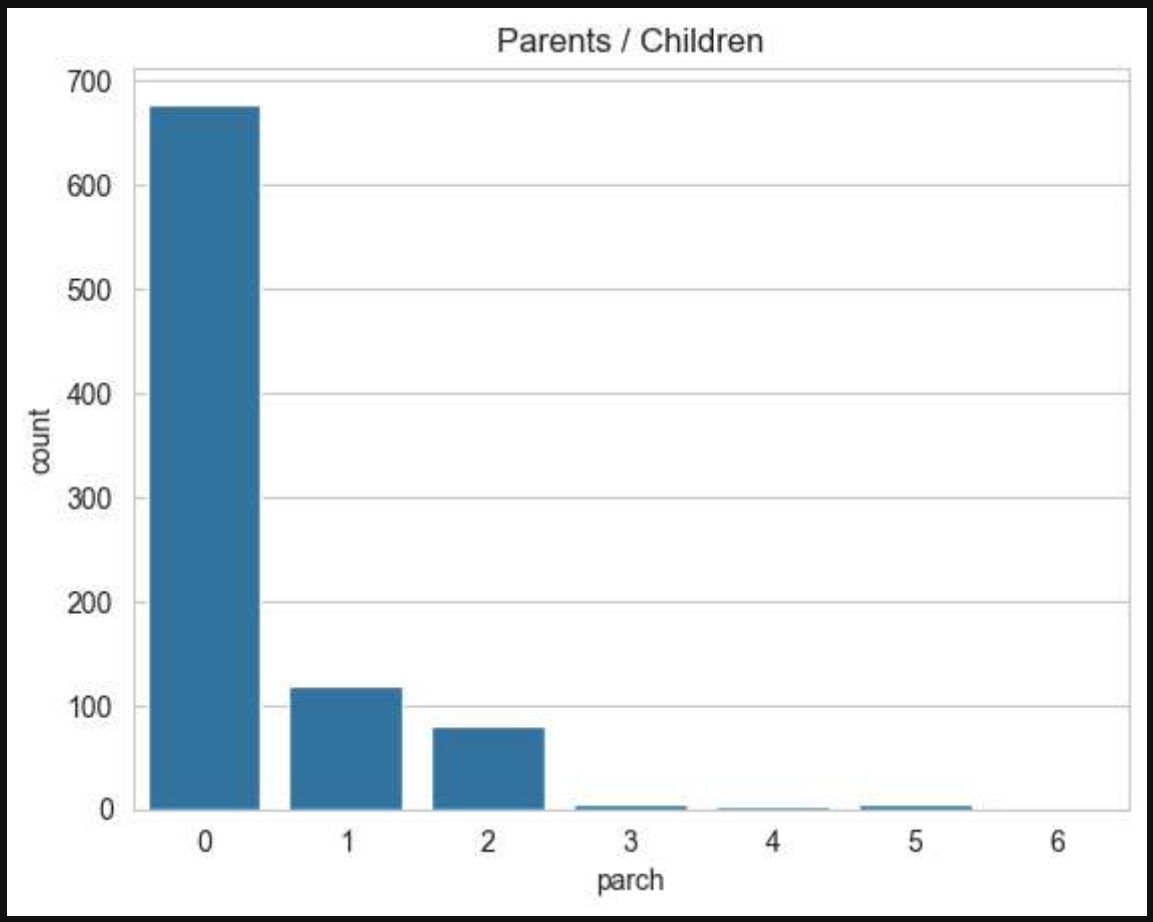
In [463…

```
sns.countplot(x='sibsp',data=df)
plt.title('Siblings/Spouses Aboard')
plt.show()
```



In [464…

```
sns.countplot(x="parch", data=df)
plt.title("Parents / Children")
plt.show()
```



🗨️ What you learned in Step 4

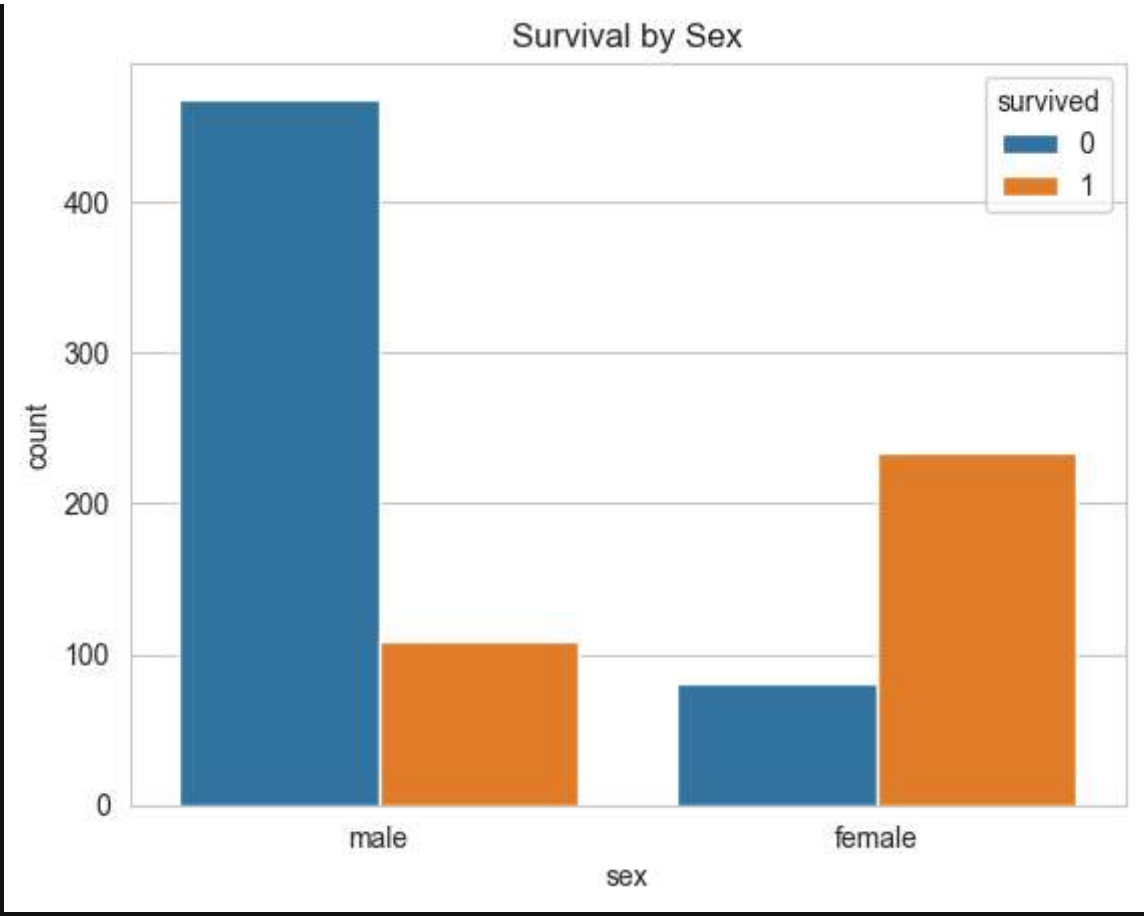
✔ Dataset imbalance ✔ Which features are skewed ✔ Which features are categorical vs numeric ✔ Where outliers probably exist ✔ Which features need transformation later

Bivariate Analysis

- Relationships between features and target variable (survived)

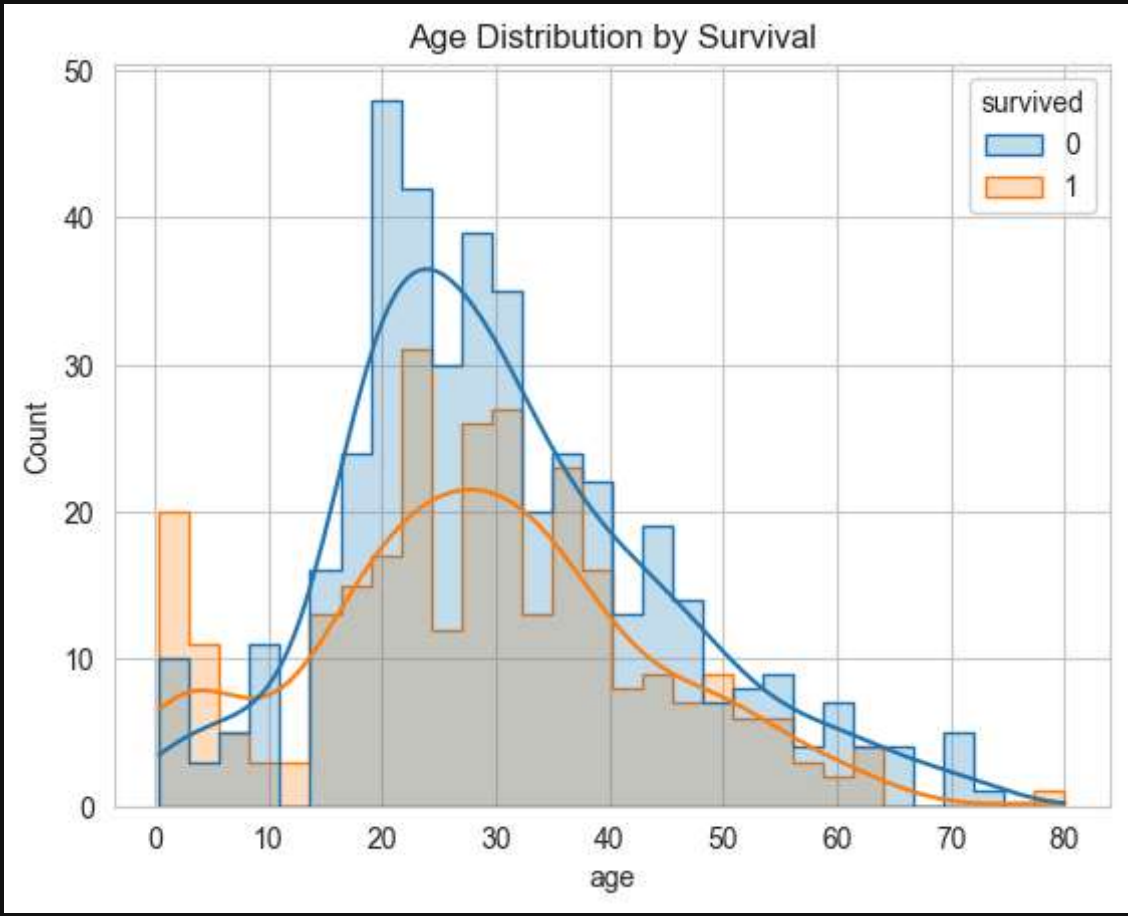
In [465…

```
sns.countplot(x="sex", hue="survived", data=df)
plt.title("Survival by Sex")
plt.show()
```



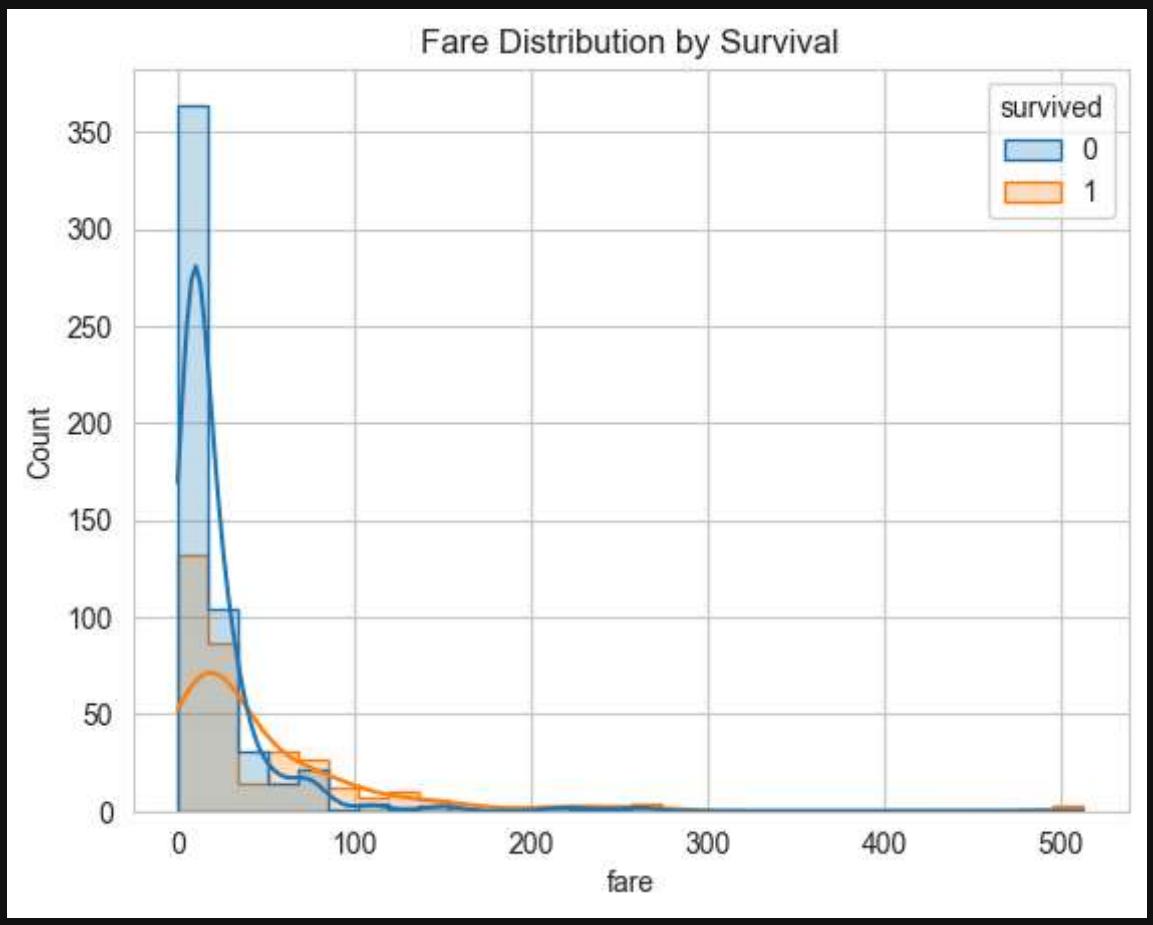
In [466…

```
sns.histplot(data=df,x='age',hue='survived',bins=30,kde=True,element='step')

plt.title("Age Distribution by Survival")
plt.show()
```



In [467…

```
sns.histplot(
    data=df,
    x= "fare",
    hue="survived",
    bins=30,
    kde=True,
    element="step"
)

plt.title("Fare Distribution by Survival")
plt.show()
```

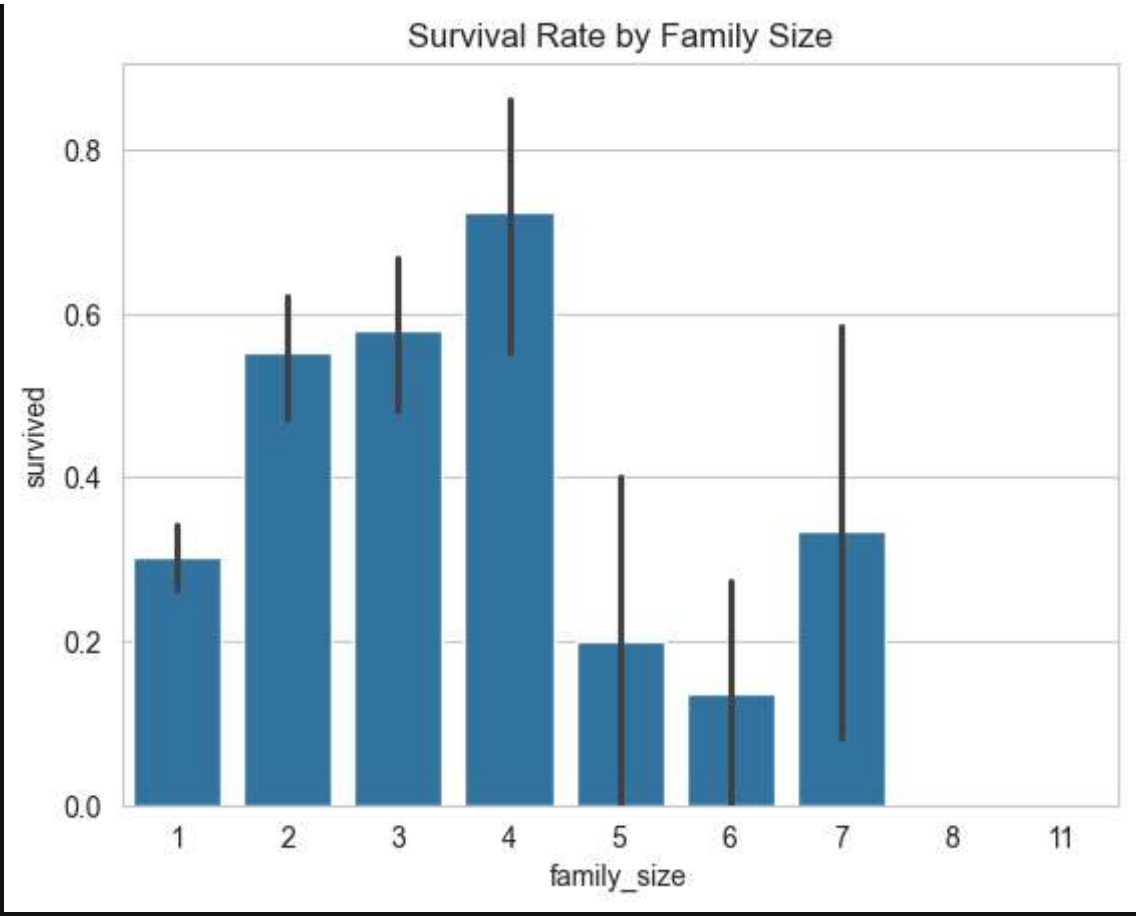


In [468…

```
df["family_size"] = df["sibsp"] + df["parch"] + 1
```

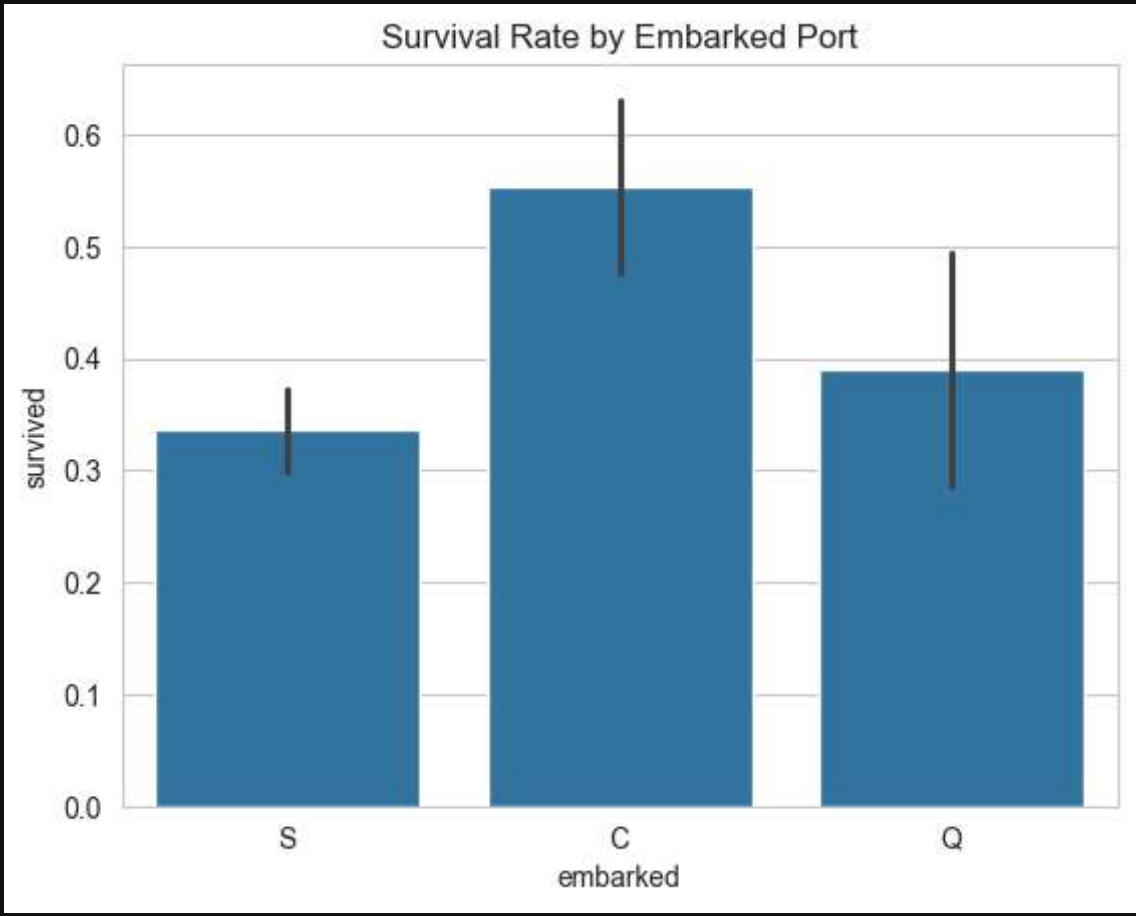
In [469…

```
sns.barplot(x='family_size', y='survived', data=df)
plt.title('Survival Rate by Family Size')
plt.show()
```



In [470…

```
sns.barplot(x="embarked", y="survived", data=df)
plt.title("Survival Rate by Embarked Port")
plt.show()
```



Multivariate Analysis

- Correlation heatmap

In [471…

```
num_df = df.select_dtypes(include=["int64", "float64"])
```

In [472…

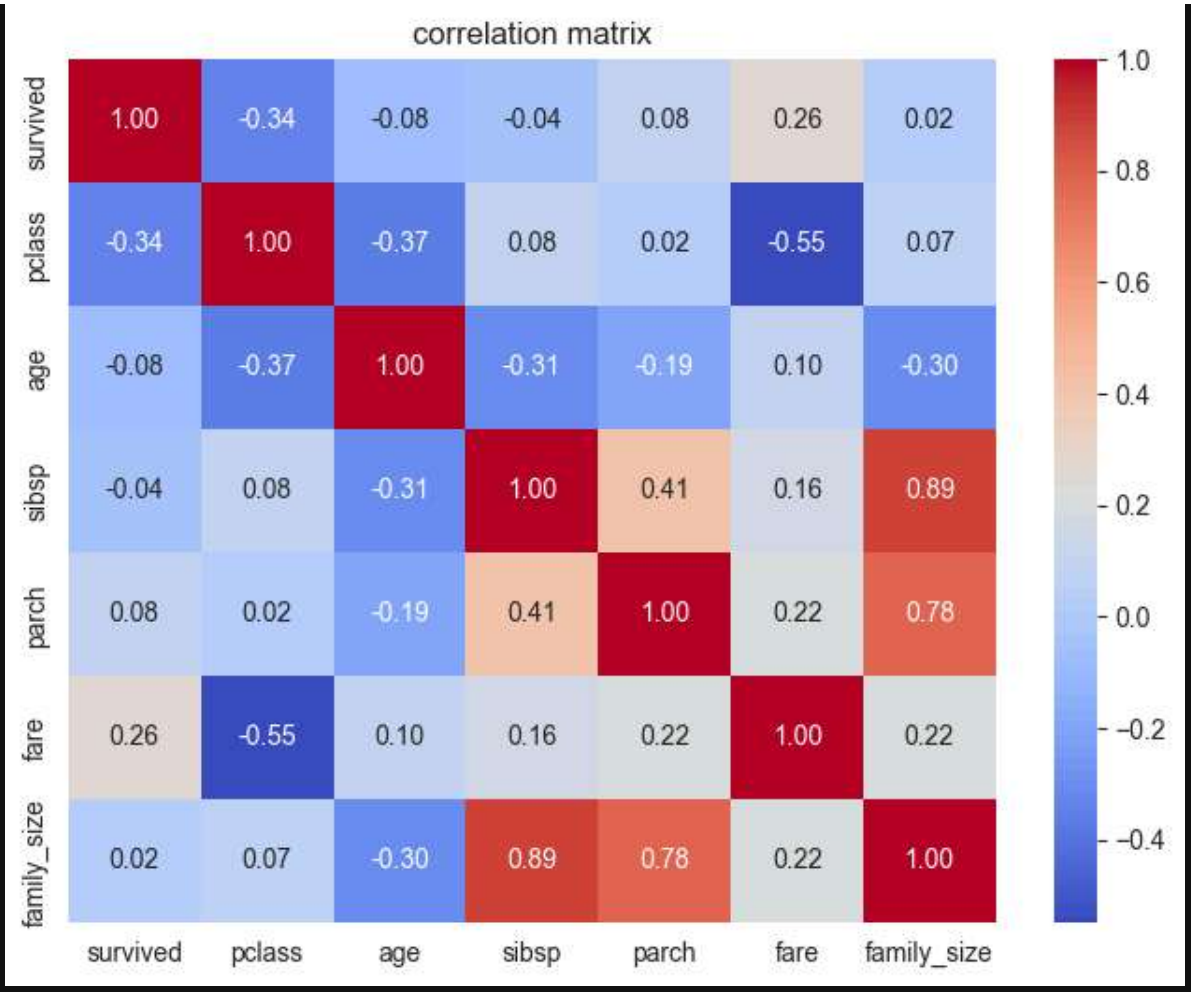
```
corr = num_df.corr()
corr
```

Out[472…

	survived	pclass	age	sibsp	parch	fare	family_size
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	0.016639
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.065997
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	-0.301914
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	0.890712
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	0.783111
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	0.217138
family_size	0.016639	0.065997	-0.301914	0.890712	0.783111	0.217138	1.000000

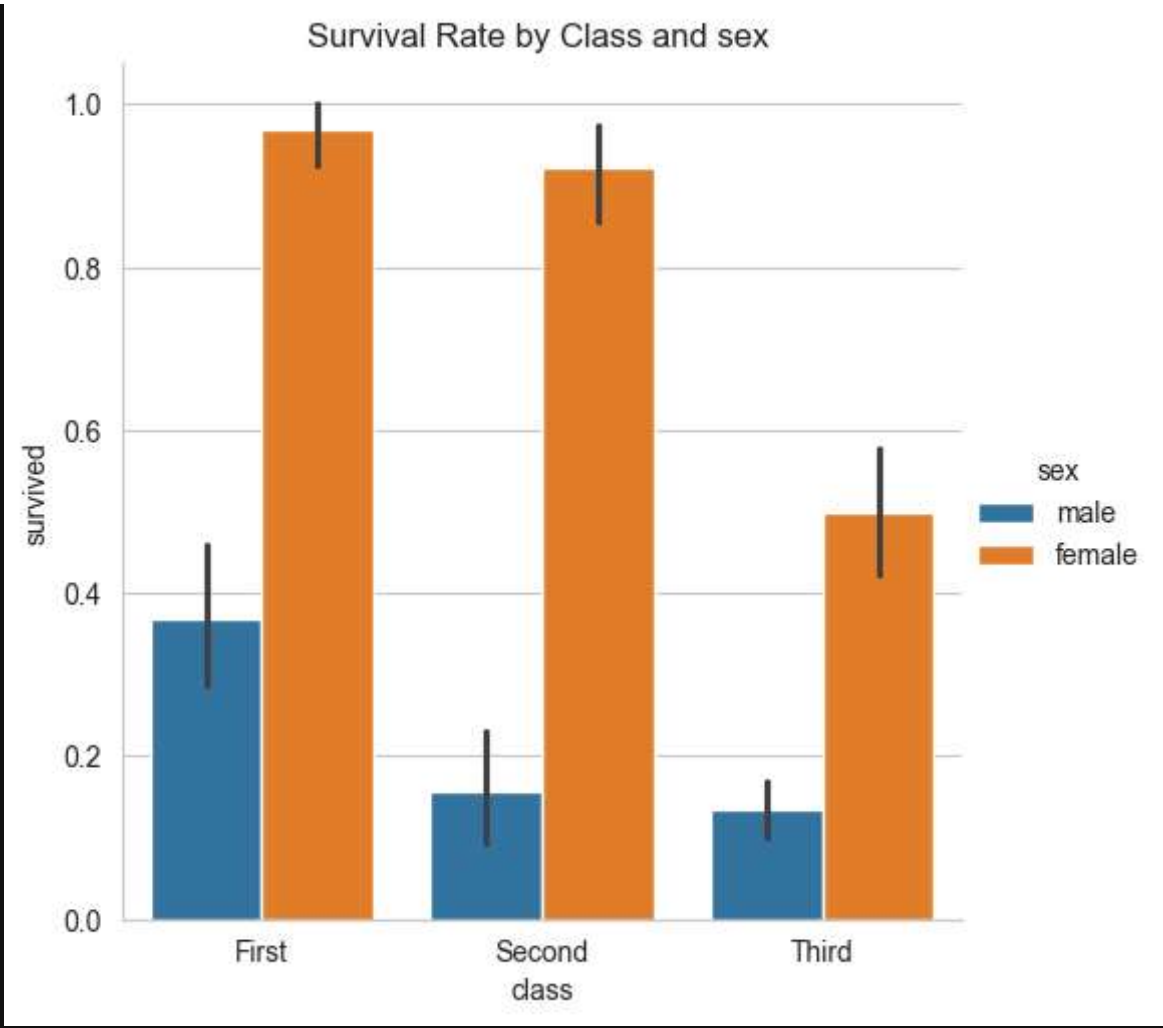
In [473…

```
## 0.7 > strng corelation

plt.figure(figsize=(8,6))
sns.heatmap(corr,annot=True,cmap="coolwarm", fmt='.2f')
plt.title("correlation matrix")
plt.show()
```



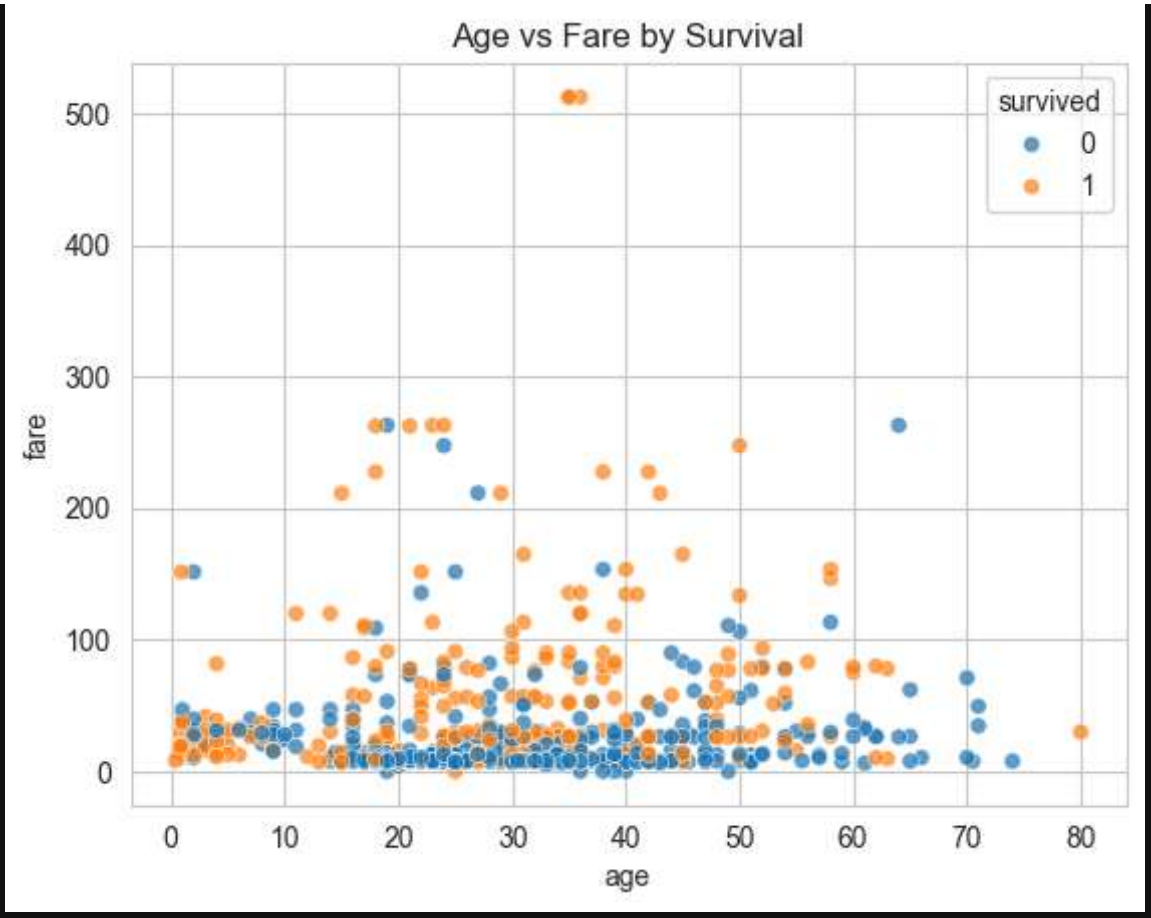
In [474…

```
sns.catplot(
    x='class',
    y='survived',
    data=df,
    hue='sex',
    kind='bar',
    height=5
)
plt.title("Survival Rate by Class and sex ")
plt.show()
```



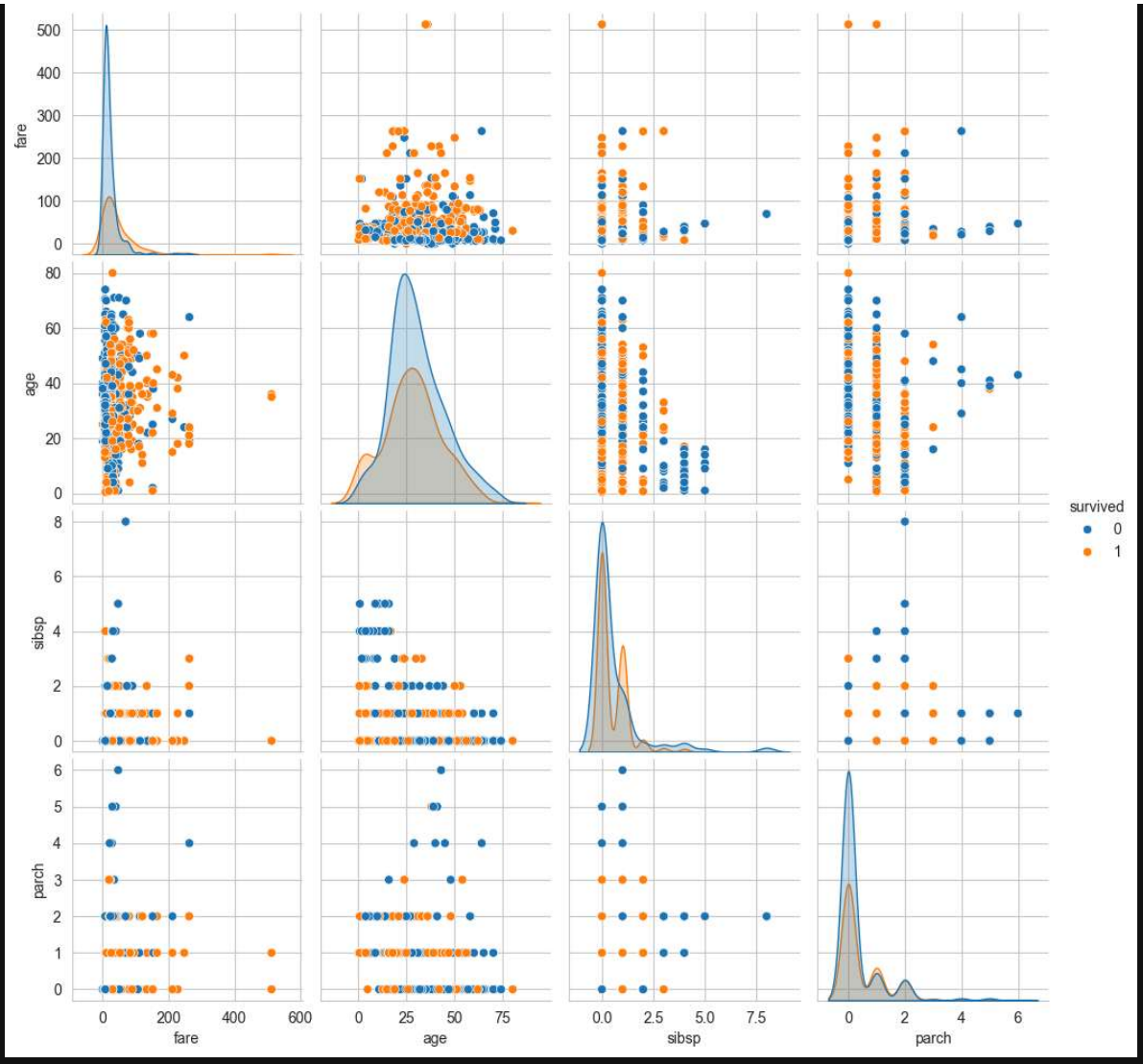
In [475…

```
sns.scatterplot(x="age",y="fare",hue="survived",data=df,alpha=0.7)
plt.title("Age vs Fare by Survival")
plt.show()
```



In [476…

```
sns.pairplot(df[["survived","fare","age","sibsp","parch"]],hue="survived")
plt.show()
```



🧠 What Step 6 gives you

✔ Which features interact ✔ Which ones carry similar information ✔ Which features deserve engineering ✔ Which models may work better

Statistical understanding of data

- Central Tendency (Mean, Median, Mode)
- Dispersion (Range, Variance, Standard Deviation)
- IQR (Interquartile Range)
- Skewness and Kurtosis
- Visualizations (Histograms, Box Plots, Scatter Plots)
- interpretation of statistical measures

In [477…

```
num_df = df.select_dtypes(include=["int64", "float64"])
num_df.columns
```

Out[477…

```
Index(['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare', 'family_size'], d
type='object')
```

In [478…

```
num_df.mean()
```

Out[478…

```
survived        0.383838
pclass          2.308642
age            29.699118
sibsp           0.523008
parch           0.381594
fare           32.204208
family_size     1.904602
dtype: float64
```

In [479…

```
num_df.median()
```

Out[479…

```
survived        0.0000
pclass          3.0000
age            28.0000
sibsp           0.0000
parch           0.0000
fare           14.4542
family_size     1.0000
dtype: float64
```

In [480…

```
num_df.mode()
```

Out[480…

	survived	pclass	age	sibsp	parch	fare	family_size
0	0	3	24.0	0	0	8.05	1

In [481…

```
num_df[["age","fare"]].median()
```

Out[481…

```
age     28.0000
fare    14.4542
dtype: float64
```

In [482…

```
num_df.std()
```

Out[482…

```
survived        0.486592
pclass          0.836071
age            14.526497
sibsp           1.102743
parch           0.806057
fare           49.693429
family_size     1.613459
dtype: float64
```

In [483…

```
# interpretation rule for skewness
# - A skewness value > 1 or < -1 indicates a highly skewed distribution.
# - A skewness value between 0.5 and 1 or between -0.5 and -1 indicates a modera
# - A skewness value between -0.5 and 0.5 indicates a approximately symmetric di

num_df.skew()
```

Out[483…

```
survived       0.478523
pclass        -0.630548
age            0.389108
sibsp          3.695352
parch          2.749117
fare           4.787317
family_size    2.727441
dtype: float64
```

In [484…

```python
# higher the kurtosis value, the heavier the tails of the distribution.
# high kurtosis indicates more outliers.
# lower/negative kurtosis indicates lighter tails.

num_df.kurtosis()
```

Out[484…

```
survived       -1.775005
pclass         -1.280015
age             0.178274
sibsp          17.880420
parch           9.778125
fare           33.398141
family_size     9.159666
dtype: float64
```

In [485…

```python
Q1 = num_df.quantile(0.25)
Q3 = num_df.quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

lower_bound ,    upper_bound
```

Out[485…

```
(survived       -1.5000
 pclass          0.5000
 age            -6.6875
 sibsp          -1.5000
 parch           0.0000
 fare          -26.7240
 family_size    -0.5000
 dtype: float64,
 survived        2.5000
 pclass          4.5000
 age            64.8125
 sibsp           2.5000
 parch           0.0000
 fare           65.6344
 family_size     3.5000
 dtype: float64)
```

In [486…

```python
outliers = num_df[(num_df < lower_bound) | (num_df > upper_bound)]

outliers.shape
```

Out[486…

```
(891, 7)
```

Statistical conclusions (WRITE THIS)

Add a markdown cell:

Statistical Findings

Fare is highly right-skewed with high kurtosis

Fare contains many extreme outliers

Age is mildly skewed with missing values

Mean is unreliable for fare → use median / robust methods

Our options (decision table) Method When to use Remove Measurement error Cap (Winsorize) Extreme but valid Transform Skewed distributions Keep Tree models

For Titanic fare → Transform + Robust Scaling

Data Cleaning

- Handling Missing Values
- Outliers treatment
- Fixing Skewness
- Duplicates Removal
- removing leakage & useless features

In [487…

```
df_clean = df.copy()
```

In [488…

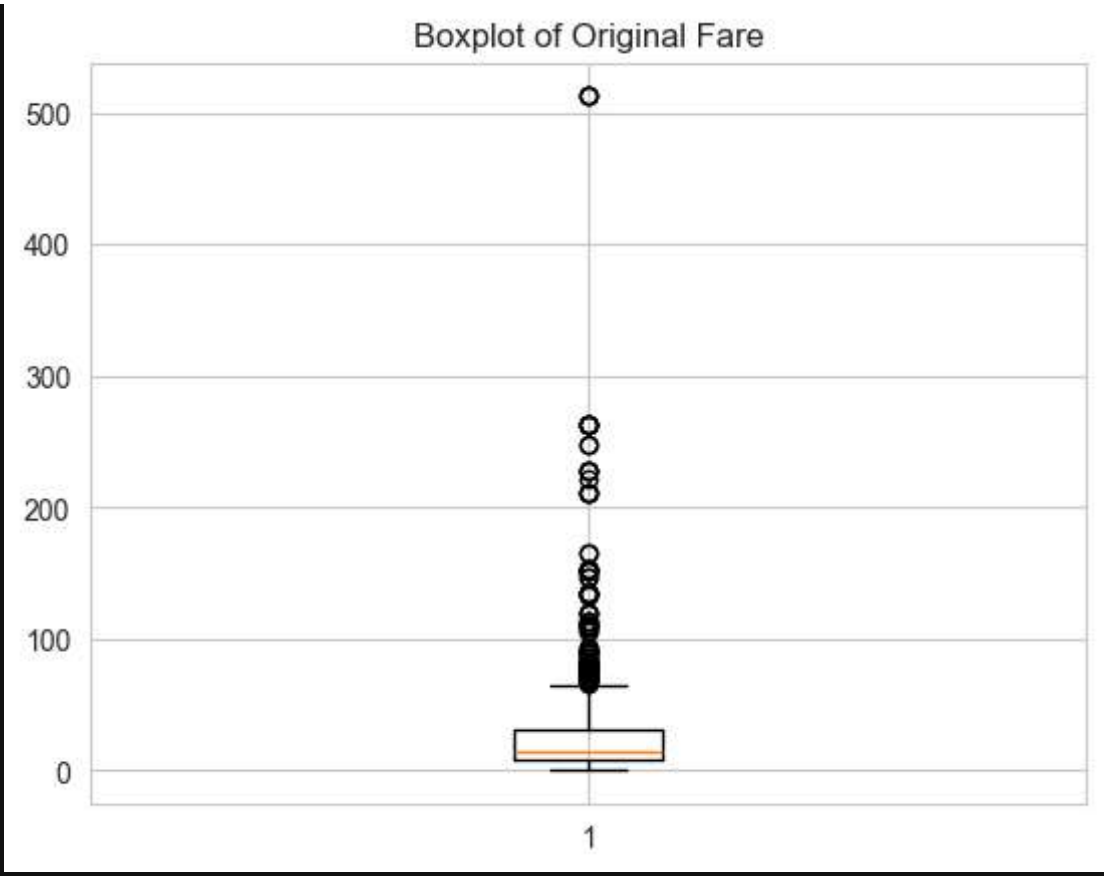
```
Q1 = df_clean["fare"].quantile(0.25)
Q3 = df_clean["fare"].quantile(0.75)
IQR = Q3 - Q1


lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
```

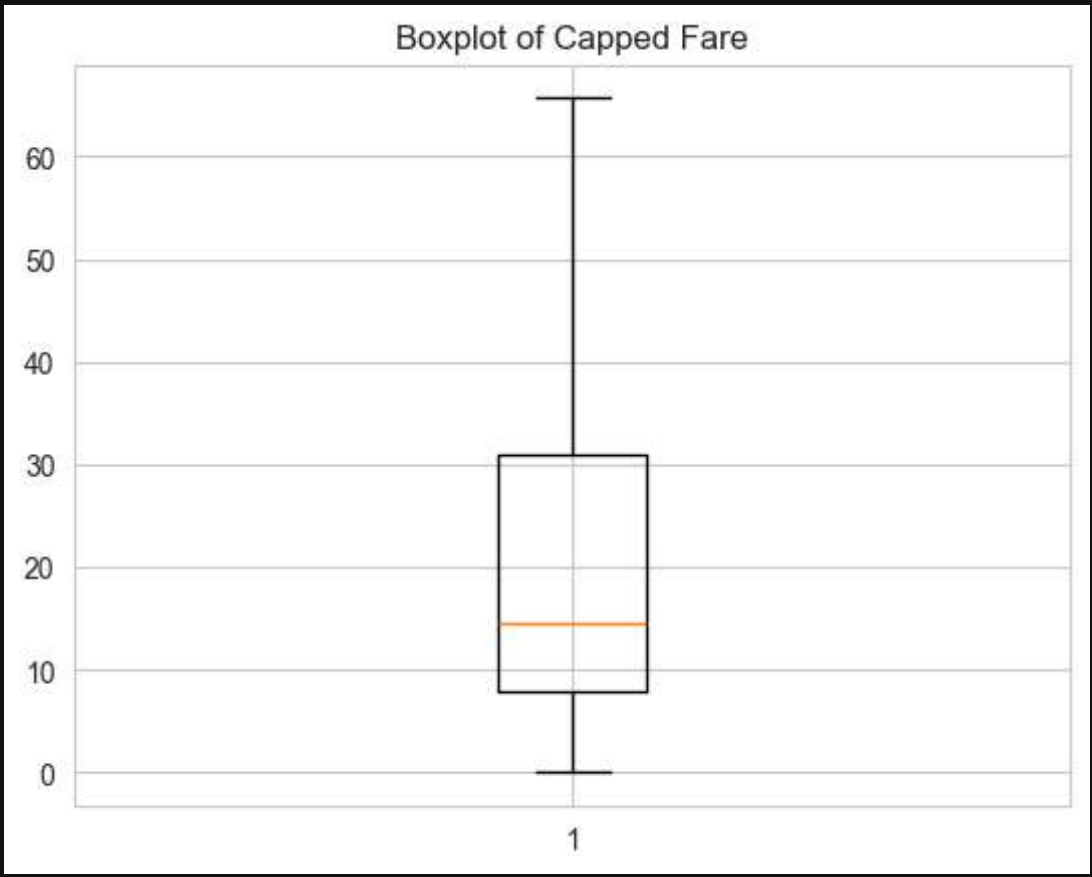
In [489…

```
df_clean["fare_capped"] = df_clean["fare"].clip(lower, upper)
```

In [490…

```
plt.boxplot(df["fare"].dropna())
plt.title("Boxplot of Original Fare")
plt.show()

plt.boxplot(df_clean["fare_capped"].dropna())
plt.title("Boxplot of Capped Fare")
plt.show()
```



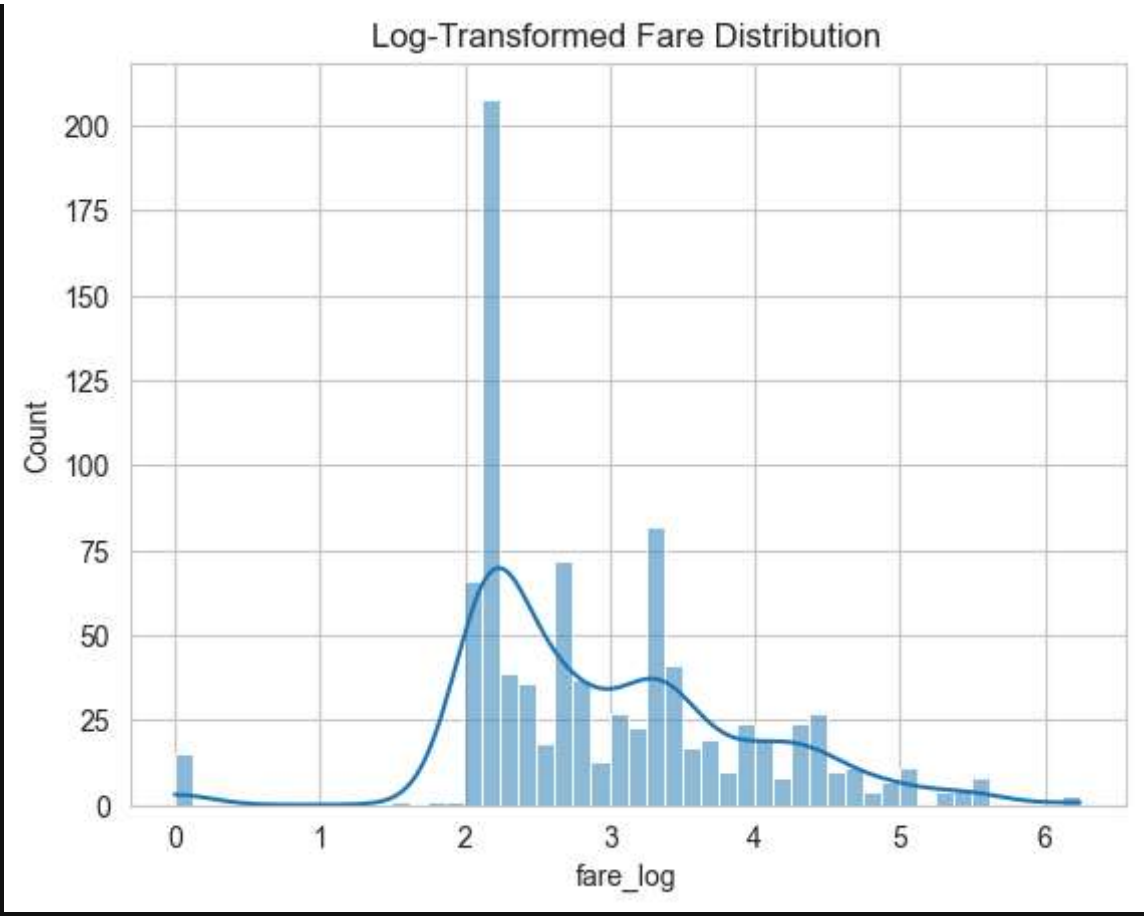


In [491…

```
# Log Transformation
df_clean["fare_log"] = np.log1p(df_clean["fare"])
```

In [492…

```
sns.histplot(data=df_clean["fare_log"], bins= 50 , kde=True)
plt.title("Log-Transformed Fare Distribution")
plt.show()
```



🧠 What you learned in Step 8

✔ Outliers ≠ bad data ✔ Capping > removing for real-world data ✔ Transformation beats deletion ✔ Cleaning depends on the model you plan to use

In [493…

```
## Missing value tratment

df_clean.isnull().sum()
```

Out[493…

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
family_size      0
fare_capped      0
fare_log         0
dtype: int64
```

In [494…

```
# decide drop vs impite(key logic )
# deck highly missing adds noise
```

```
# drop

df_clean.drop(columns="deck", inplace=True)
```

In [495…

```
# embark_town
# similar to embarked same info different format

df_clean.drop(columns="embark_town", inplace=True)
```

In [496…

```
# check most frequent values

df_clean['embarked'].value_counts()
```

Out[496…

```
embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

In [497…

```
# fill na value with mode values that is "S"

df_clean["embarked"].fillna("S", inplace=True)
```

In [498…

```
df_clean["age"].median()
```

Out[498…

```
np.float64(28.0)
```

In [499…

```
# fill the nan valus of age with the median  of the age column

df_clean["age"].fillna(df_clean["age"].median(), inplace=True)
```

In [500…

```
df_clean.isnull().sum()
```

Out[500…

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
alive          0
alone          0
family_size    0
fare_capped    0
fare_log       0
dtype: int64
```

Feature Engineering

- Creating new features
- Encoding categorical variables
- scaling features if necessary

- normalization/standardization
- Dimensionality Reduction if needed
- feature selection techniques

In [501…

```
df_clean["family_size"] = df_clean["sibsp"] + df_clean["parch"]+1
```

In [502…

```
# is_alone is basically family_size 1

df_clean["is_alone"] = (df_clean["family_size"] == 1).astype(int)
```

In [503…

```
# title extraction from name (safe: handle datasets without 'name' column)
if 'name' in df_clean.columns:
        df_clean["title"] = df_clean["name"].str.extract(r" ([A-Za-z]+)\.", expa
else:
        def _derive_title(row):
                if row.get('sex') == 'male':
                        return 'Mr'
                if row.get('sex') == 'female':
                        age = row.get('age')
                        try:
                                if pd.notnull(age) and age < 18:
                                        return 'Miss'
                        except Exception:
                                pass
                        return 'Mrs'
                return 'Unknown'
        df_clean["title"] = df_clean.apply(_derive_title, axis=1)
```

In [504…

```
df_clean
```

Out[504…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	3	male	22.0	1	0	7.2500	S	Third	man
1	1	1	female	38.0	1	0	71.2833	C	First	woman
2	1	3	female	26.0	0	0	7.9250	S	Third	woman
3	1	1	female	35.0	1	0	53.1000	S	First	woman
4	0	3	male	35.0	0	0	8.0500	S	Third	man
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man
887	1	1	female	19.0	0	0	30.0000	S	First	woman
888	0	3	female	28.0	1	2	23.4500	S	Third	woman
889	1	1	male	26.0	0	0	30.0000	C	First	man
890	0	3	male	32.0	0	0	7.7500	Q	Third	man

891 rows × 18 columns

In [505…

```
# we dived the passangers on the basess of their rang and gender
```

```
df_clean["titles"] = df_clean["title"].replace(["Lady", "Countess", "Capt", "Col
     "Major", "Rev", "Sir", "Jonkheer", "Dona"],
    "Rare"
    )

df_clean["title"] = df_clean["title"].replace(["Mlle","Ms"],"Miss")

df_clean["title"]= df_clean["title"].replace("Mme","Mrs")
```

In [506…

```
df_clean["title"].value_counts()
```

Out[506…

```
title
Mr      577
Mrs     259
Miss     55
Name: count, dtype: int64
```

In [507…

```
## group the people will help us most of the  times

df_clean["age_group"] = pd.cut(
    df_clean["age"],
    bins=[0, 12, 20, 35, 60, 80],
    labels=["Child", "Teenager", "Young Adult", "Adult", "Senior"]
)
```

In [508…

```
df_clean
```

Out[508…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	3	male	22.0	1	0	7.2500	S	Third	man
1	1	1	female	38.0	1	0	71.2833	C	First	woman
2	1	3	female	26.0	0	0	7.9250	S	Third	woman
3	1	1	female	35.0	1	0	53.1000	S	First	woman
4	0	3	male	35.0	0	0	8.0500	S	Third	man
...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man
887	1	1	female	19.0	0	0	30.0000	S	First	woman
888	0	3	female	28.0	1	2	23.4500	S	Third	woman
889	1	1	male	26.0	0	0	30.0000	C	First	man
890	0	3	male	32.0	0	0	7.7500	Q	Third	man

891 rows × 20 columns

In [509…

```
df_clean["fare_group"] = pd.qcut(df_clean["fare"],q=4 ,labels=["Low","Mid","High

df_clean
```

Out[509…

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who
0	0	3	male	22.0	1	0	7.2500	S	Third	man
1	1	1	female	38.0	1	0	71.2833	C	First	woman
2	1	3	female	26.0	0	0	7.9250	S	Third	woman
3	1	1	female	35.0	1	0	53.1000	S	First	woman
4	0	3	male	35.0	0	0	8.0500	S	Third	man
…	…	…	…	…	…	…	…	…	…	…
886	0	2	male	27.0	0	0	13.0000	S	Second	man
887	1	1	female	19.0	0	0	30.0000	S	First	woman
888	0	3	female	28.0	1	2	23.4500	S	Third	woman
889	1	1	male	26.0	0	0	30.0000	C	First	man
890	0	3	male	32.0	0	0	7.7500	Q	Third	man

891 rows × 21 columns

In [510…

```
# we will now drop some columns that we no longer uase
# we have got the insights from them and created new columns
if "name" in df_clean.columns:
    df_clean.drop(columns=["name"], inplace=True)
```

🧠 What you achieved in Step 10

✅ Converted raw info into meaningful signals ✅ Reduced noise ✅ Improved model learning ability ✅ Built features humans understand

This is advanced feature engineering, not toy examples.

Markdown summary

Feature Engineering

Created family_size and is_alone

Extracted and grouped passenger titles

Binned age and fare to capture non-linearity

Dropped high-cardinality identifiers

Encoding and scaling

- Label Encoding
- One-Hot Encoding
- Scaling (Standardization/Normalization)
- Dimensionality Reduction (if needed)

In [511…

```
df_clean.dtypes
```

Out[511…

```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
alive            object
alone              bool
family_size       int64
fare_capped     float64
fare_log        float64
is_alone          int64
title            object
titles           object
age_group      category
fare_group     category
dtype: object
```

In [512…

```
df_clean[["survived","alive"]]
```

Out[512…

	survived	alive
0	0	no
1	1	yes
2	1	yes
3	1	yes
4	0	no
…	…	…
886	0	no
887	1	yes
888	0	no
889	1	yes
890	0	no

891 rows × 2 columns

In [513…

```
## we will now remove the duplicate values or the vlaues that are similar in nat
df_clean.drop(columns=["alive"], inplace=True)
```

In [514…

```
## we will use one-hot encoding

cat_cols = df_clean.select_dtypes(include=["object","category"]).columns
```

In [515…

```
## we will now apply the encoding

df_encoded = pd.get_dummies(
    df_clean,
    columns = cat_cols,
    drop_first=True
)
```

In [516…

```
df_encoded.head()
```

Out[516…

	survived	pclass	age	sibsp	parch	fare	adult_male	alone	family_size	fare_ca
0	0	3	22.0	1	0	7.2500	True	False	2	7
1	1	1	38.0	1	0	71.2833	False	False	2	65
2	1	3	26.0	0	0	7.9250	False	True	1	7
3	1	1	35.0	1	0	53.1000	False	False	2	53
4	0	3	35.0	0	0	8.0500	True	True	1	8

5 rows × 30 columns

In [517…

```python
df_encoded.dtypes
```

Out[517…

```
survived                   int64
pclass                     int64
age                      float64
sibsp                      int64
parch                      int64
fare                     float64
adult_male                  bool
alone                       bool
family_size                int64
fare_capped              float64
fare_log                 float64
is_alone                   int64
sex_male                    bool
embarked_Q                  bool
embarked_S                  bool
class_Second                bool
class_Third                 bool
who_man                     bool
who_woman                   bool
title_Mr                    bool
title_Mrs                   bool
titles_Mr                   bool
titles_Mrs                  bool
age_group_Teenager          bool
age_group_Young Adult       bool
age_group_Adult             bool
age_group_Senior            bool
fare_group_Mid              bool
fare_group_High             bool
fare_group_VeryHigh         bool
dtype: object
```

In [518…

```python
# now we will scale the numerical features
## note all models dont need scaling

num_cols = ["age","fare","fare_log","family_size"]
```

In [519…

```python
# we will use robust scaling

from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])
```

In [520…

```python
df_encoded
```

Out[520…

	survived	pclass	age	sibsp	parch	fare	adult_male	alone	family_siz
0	0	3	-0.461538	1	0	-0.312011	True	False	1.(
1	1	1	0.769231	1	0	2.461242	False	False	1.(
2	1	3	-0.153846	0	0	-0.282777	False	True	0.(
3	1	1	0.538462	1	0	1.673732	False	False	1.(
4	0	3	0.538462	0	0	-0.277363	True	True	0.(
…	…	…	…	…	…	…	…	…	.
886	0	2	-0.076923	0	0	-0.062981	True	True	0.(
887	1	1	-0.692308	0	0	0.673281	False	True	0.(
888	0	3	0.000000	1	2	0.389604	False	False	3.(
889	1	1	-0.153846	0	0	0.673281	True	True	0.(
890	0	3	0.307692	0	0	-0.290356	True	True	0.(

891 rows × 30 columns

In [521…

```
df_encoded.shape
```

Out[521…

```
(891, 30)
```

In [522…

```
df_encoded.isnull().sum().sum()
```

Out[522…

```
np.int64(0)
```

🧠 What you achieved in Step 11

✅ Converted real-world data into ML-ready matrix ✅ Prevented leakage ✅ Scaled correctly (not blindly) ✅ Built something reusable

📝 Markdown summary

Encoding & Scaling

One-hot encoded categorical variables

Dropped leakage column

Scaled numeric features using RobustScaler

Final dataset contains only numeric values

ML readiness

- Splitting data into training and testing sets
- Ensuring no data leakage

- choosing the right features for modeling

In [523…
```
# now we start ML from train test split

x = df_encoded.drop(columns=["survived"])
y = df_encoded["survived"]
```

In [524…
```
x.shape, y.shape
```

Out[524…
```
((891, 29), (891,))
```

In [525…
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_st
```

In [526…
```
## verify the split
y_train.value_counts(normalize=True)
```

Out[526…
```
survived
0    0.616573
1    0.383427
Name: proportion, dtype: float64
```

In [527…
```
y_test.value_counts(normalize=True)
```

Out[527…
```
survived
0    0.614525
1    0.385475
Name: proportion, dtype: float64
```

In [528…
```
# we will use logistic regression as our first model

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)

model.fit(x_train,y_train)
```

Out[528…
```
▾ LogisticRegression
▸ Parameters
```

In [529…
```
y_pred = model.predict(x_test)
```

In [530…
```
# we will now check the accuracy of our model
from sklearn.metrics import accuracy_score , classification_report

accuracy = accuracy_score(y_test,y_pred)
print(f"the accuray pf the mosdel is { accuracy}")

report = classification_report(y_test,y_pred)

print(f" {report }")
```

```
the accuray pf the mosdel is 0.8324022346368715
              precision    recall  f1-score   support

           0       0.84      0.89      0.87       110
           1       0.81      0.74      0.77        69

    accuracy                           0.83       179
   macro avg       0.83      0.82      0.82       179
weighted avg       0.83      0.83      0.83       179
```

🧠 What you learned in Step 13

✅ Built first real ML model ✅ Evaluated properly ✅ Understood metrics beyond accuracy ✅ Created a benchmark to beat

confusion mnetrics and error analysis

- to understand why and how the model makes mistakes

In [531…

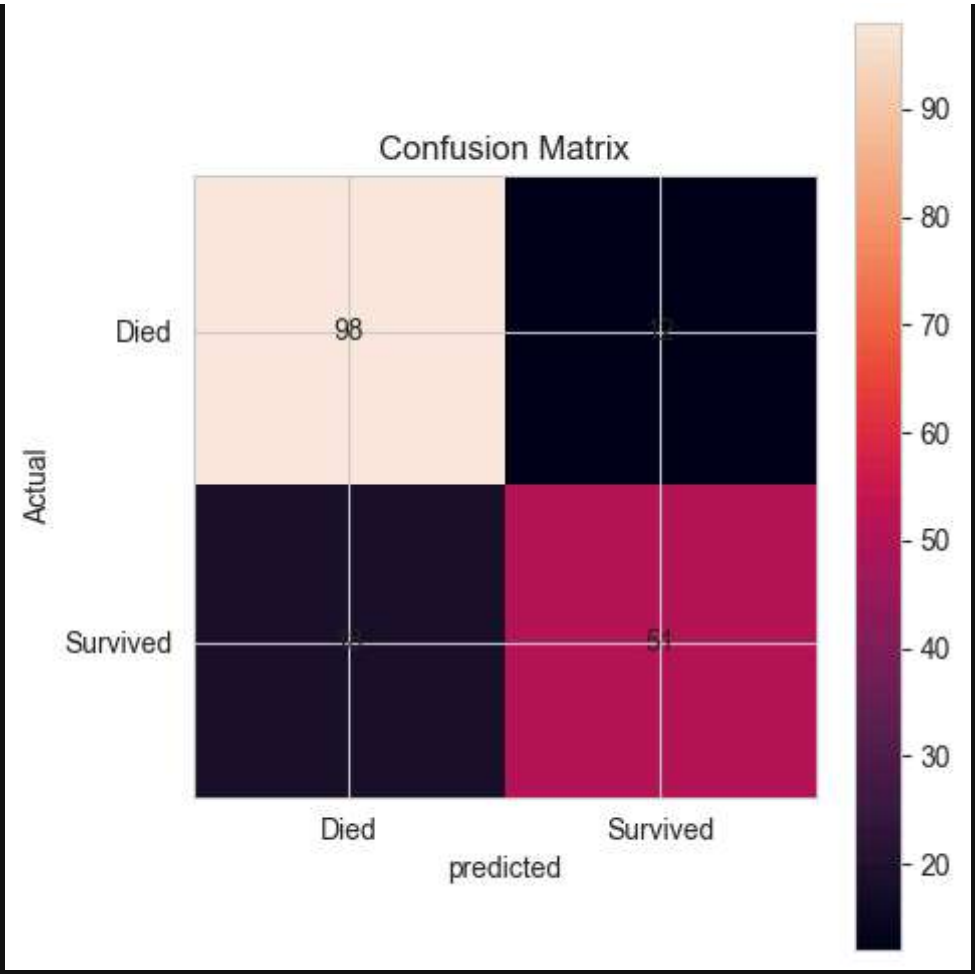
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

cm
```

Out[531…

```
array([[98, 12],
       [18, 51]])
```

In [532…

```
import matplotlib.pyplot as plt

plt.figure(figsize=(5,6))
plt.imshow(cm)
plt.colorbar()

plt.xticks([0,1],["Died","Survived"])
plt.yticks([0,1],["Died","Survived"])

for i in range(2):
    for j in range(2):
        plt.text(j,i,cm[i,j],ha="center", va="center")

plt.xlabel("predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [533…
```
cm[1,0]
```

Out[533…
```
np.int64(18)
```

In [534…
```
misclassified = x_test[y_test != y_pred]
misclassified.head()
```

Out[534…

	pclass	age	sibsp	parch	fare	adult_male	alone	family_size	fare_cap
553	3	-0.461538	0	0	-0.313093	True	True	0.0	7.
712	1	1.538462	1	0	1.626091	True	False	1.0	52.
455	3	0.076923	0	0	-0.284041	True	True	0.0	7.
65	3	0.000000	1	1	0.034284	True	False	2.0	15.
297	1	-2.000000	1	2	5.937556	False	False	3.0	65.

5 rows × 29 columns

In [535…
```
misclassified["actual"] = y_test[y_test != y_pred]
```

```
C:\Users\mohda\AppData\Local\Temp\ipykernel_23044\2361411544.py:1: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  misclassified["actual"] = y_test[y_test != y_pred]
```

🧠 What you learned in Step 14

✔ Read confusion matrices correctly ✔ Identified critical errors ✔ Connected model mistakes to data ✔ Prepared for smarter feature engineering

FEATURE IMPORTANCE (MODEL INTERPRETATION)

Goal: Understand which features drive survival predictions.

In [536…

```
import numpy as np

feature_importance = np.abs(model.coef_[0])

feature_names = x_train.columns

importance_df = (
    pd.DataFrame(
        {
            "feature":feature_names,
            "importance": feature_importance
        }
    )
    .sort_values(by="importance",ascending=False)
)
```

In [537…

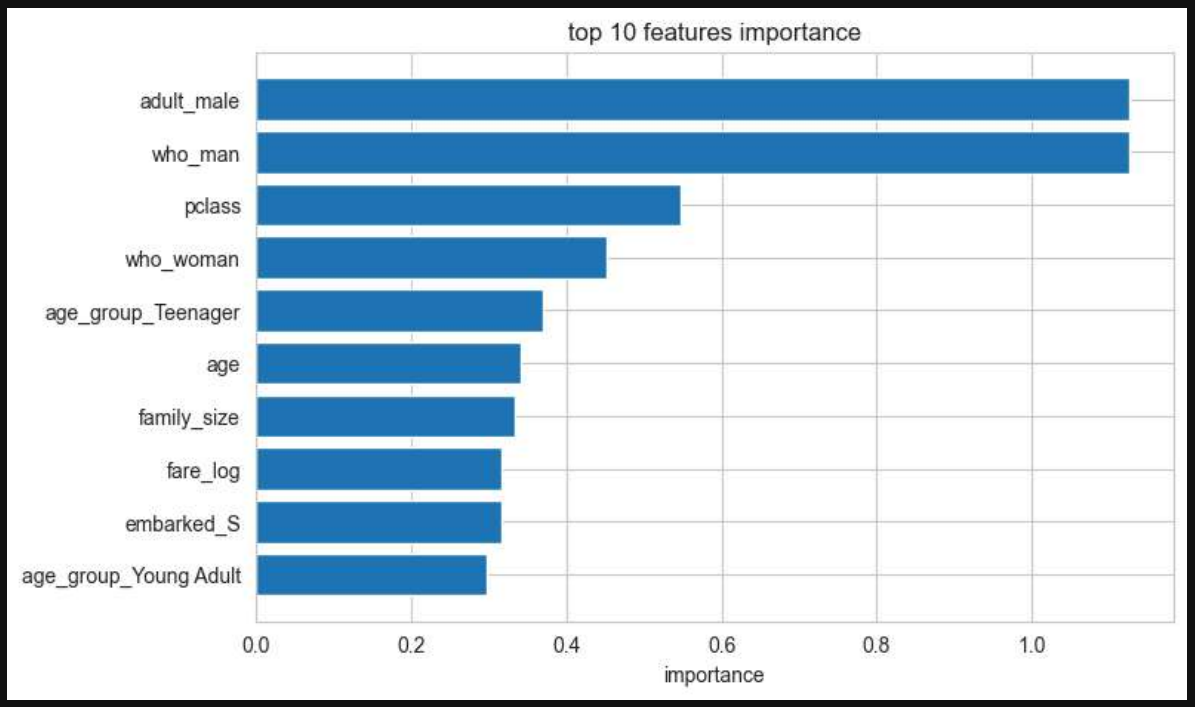
```
importance_df.head(10)
```

Out[537…

	feature	importance
5	adult_male	1.125765
16	who_man	1.125765
0	pclass	0.548094
17	who_woman	0.452127
22	age_group_Teenager	0.370314
1	age	0.340822
7	family_size	0.334167
9	fare_log	0.317628
13	embarked_S	0.316850
23	age_group_Young Adult	0.296890

In [538…

```
top = importance_df.head(10)

plt.figure(figsize=(8,5))
plt.barh(top["feature"],top["importance"])
plt.gca().invert_yaxis()
plt.xlabel("importance")
plt.title("top 10 features importance")
plt.show()
```



In [539…

```
coef_df = pd.DataFrame({
    "feature":feature_names,
    "coefficient":model.coef_[0]
}).sort_values(by="coefficient")

coef_df.head(10)
coef_df.tail(10)
```

Out[539…

	feature	coefficient
21	titles_Mrs	-0.104886
14	class_Second	-0.062290
4	fare	-0.058545
3	parch	-0.049657
8	fare_capped	0.021473
26	fare_group_Mid	0.155492
27	fare_group_High	0.200113
12	embarked_Q	0.247466
9	fare_log	0.317628
17	who_woman	0.452127

🧠 What you learned in Step 15

✔ Interpreted a real ML model ✔ Verified logic vs reality ✔ Detected potential leakage ✔ Prepared for feature engineering

FEATURE ENGINEERING (LEVEL UP 🚀)

Goal: Create smarter features from raw data.

We turn data → information → signal.

In [540…

```
df["family_size"] = df["sibsp"] + df["parch"] + 1
df["family_size"].value_counts()
```

Out[540…

```
family_size
1     537
2     161
3     102
4      29
6      22
5      15
7      12
11      7
8       6
Name: count, dtype: int64
```

In [541…

```
df["is_alone"] = (df["family_size"] == 1).astype(int)
df["is_alone"].value_counts()
```

Out[541…

```
is_alone
1    537
0    354
Name: count, dtype: int64
```

In [542…

```
df["fare_per_person"] = df["fare"]/df["family_size"]
```

In [543…

```
df["age_group"] = pd.cut(
    df["age"],
    bins=[1,12,18,35,60,100],
    labels=["child","teen","adult","middle_aged","senior"]
)
```

In [544…

```
df["age_group"].value_counts()
```

Out[544…

```
age_group
adult          358
middle_aged    195
teen            70
child           55
senior          22
Name: count, dtype: int64
```

In [545…

```
df = pd.get_dummies(
    df,
    columns=["age_group"],
    drop_first=True
)
```

In [546…

```
df.drop(columns=["sibsp", "parch"], inplace=True)
```

In [547…

```
df.drop(columns=["deck","embark_town","alive"],inplace=True)
```

In [548…

```
df.isnull().sum()
```

Out[548…

```
survived                   0
pclass                     0
sex                        0
age                      177
fare                       0
embarked                   2
class                      0
who                        0
adult_male                 0
alone                      0
family_size                0
is_alone                   0
fare_per_person            0
age_group_teen             0
age_group_adult            0
age_group_middle_aged      0
age_group_senior           0
dtype: int64
```

In [549…

```
df["age"].fillna(df["age"].median(), inplace=True)
```

In [550…

```
df.isnull().sum()
```

Out[550…

```
survived                 0
pclass                   0
sex                      0
age                      0
fare                     0
embarked                 2
class                    0
who                      0
adult_male               0
alone                    0
family_size              0
is_alone                 0
fare_per_person          0
age_group_teen           0
age_group_adult          0
age_group_middle_aged    0
age_group_senior         0
dtype: int64
```

In [551…

```
cat_cols = ["sex","embarked","class","who"]
```

In [552…

```
# install scikit-learn if missing

from sklearn.preprocessing import LabelEncoder

# only encode columns that actually exist in df to avoid KeyError
cols_to_encode = [col for col in cat_cols if col in df.columns]

for col in cols_to_encode:
    le = LabelEncoder()
    # fill missing values with a placeholder and ensure string type for consiste
    # df[col] = df[col].fillna("missing").astype(str)
    df[col] = le.fit_transform(df[col])
```

In [553…

```
x = df.drop(columns=["survived"])
y = df["survived"]
```

In [554…

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train , y_test = train_test_split(
    x,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
    )
```

🧠 What you learned in Step 16

✔ Created meaningful features ✔ Reduced noise ✔ Injected domain knowledge ✔ Set stage for performance jump

RETRAIN MODEL & COMPARE RESULTS

Goal: Prove that feature engineering actually improved the model.

In [555…

```python
from sklearn.linear_model import LogisticRegression

model_fe = LogisticRegression(max_iter=1000)
model_fe.fit(x_train,y_train)
```

Out[555…

```
▾ LogisticRegression
▸ Parameters
```

In [556…

```python
y_pred_fe = model_fe.predict(x_test)
```

In [557…

```python
from sklearn.metrics import accuracy_score, classification_report

print("Accuracy:", accuracy_score(y_test, y_pred_fe))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_fe))
```

```
Accuracy: 0.8268156424581006

Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.89      0.86       110
           1       0.81      0.72      0.76        69

    accuracy                           0.83       179
   macro avg       0.82      0.81      0.81       179
weighted avg       0.83      0.83      0.82       179
```

In [558…

```python
from sklearn.metrics import confusion_matrix

cm_fe = confusion_matrix(y_test, y_pred_fe)
cm_fe
```

Out[558…

```
array([[98, 12],
       [19, 50]])
```

🧠 What you learned in Step 17

✔ How to validate improvements correctly ✔ Why features > fancy models ✔ How to compare models scientifically ✔ How ML progress is measured

Model Building

- Selecting appropriate ML algorithms
- Training the model
- Hyperparameter tuning
- Evaluating model performance using relevant metrics

In [559…

```python
# we will now head towards using stronger models
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier(
    max_depth=5,
    random_state=42
)

dt.fit(x_train, y_train)
```

Out[559…

```
▾ DecisionTreeClassifier
▸ Parameters
```

In [560…

```
y_pred_dt = dt.predict(x_test)
```

In [561…

```
from sklearn.metrics import classification_report, accuracy_score

print(f"accuracy is {accuracy_score(y_test,y_pred_dt)}")

print(classification_report(y_test,y_pred_dt))
```

```
accuracy is 0.7988826815642458
              precision    recall  f1-score   support

           0       0.84      0.83      0.83       110
           1       0.73      0.75      0.74        69

    accuracy                           0.80       179
   macro avg       0.79      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179
```

In [562…

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(
    n_estimators = 200,
    max_depth=8,
    random_state=42
)


rf.fit(x_train,y_train)
```

Out[562…

```
▾ RandomForestClassifier
▸ Parameters
```

In [563…

```
y_pred_rf = rf.predict(x_test)
```

In [564…

```
print(f"the accuracy is {accuracy_score(y_test,y_pred_rf)}")

print(classification_report(y_test,y_pred_rf))
```

```
the accuracy is 0.8100558659217877
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       110
           1       0.78      0.71      0.74        69

    accuracy                           0.81       179
   macro avg       0.80      0.79      0.80       179
weighted avg       0.81      0.81      0.81       179
```

now we will compare the accuracy of all modles developed and check which onwe iss the best for us

In [565…

```
results = pd.DataFrame({
    "Model":["Logistic Regression","Descision Tree","Random Forest"],
    "Accuracy":[
        accuracy_score(y_test,y_pred_fe),
        accuracy_score(y_test,y_pred_dt),
        accuracy_score(y_test,y_pred_rf),
    ]
})
```

In [566…

```
results
```

Out[566…

	Model	Accuracy
0	Logistic Regression	0.826816
1	Descision Tree	0.798883
2	Random Forest	0.810056

4️⃣ What you should observe

Logistic → stable, interpretable

Tree → captures interactions

Forest → best recall + F1 (usually)

If RF beats LR by a lot → non-linear patterns exist.

🧠 What you learned in Step 18

✔ Why model choice matters ✔ How ensembles reduce variance ✔ When linear models fail ✔ How to compare models honestly

FEATURE IMPORTANCE (TREE-BASED MODELS)

Goal: See what Random Forest believes actually matters.

In [567…

```
import numpy as np
rf_importance = rf.feature_importances_

rf_df = (
    pd.DataFrame({
        "feature": x_train.columns,
        "importance":rf_importance
    }).sort_values(by="importance",ascending=False)
)
```

In [568…

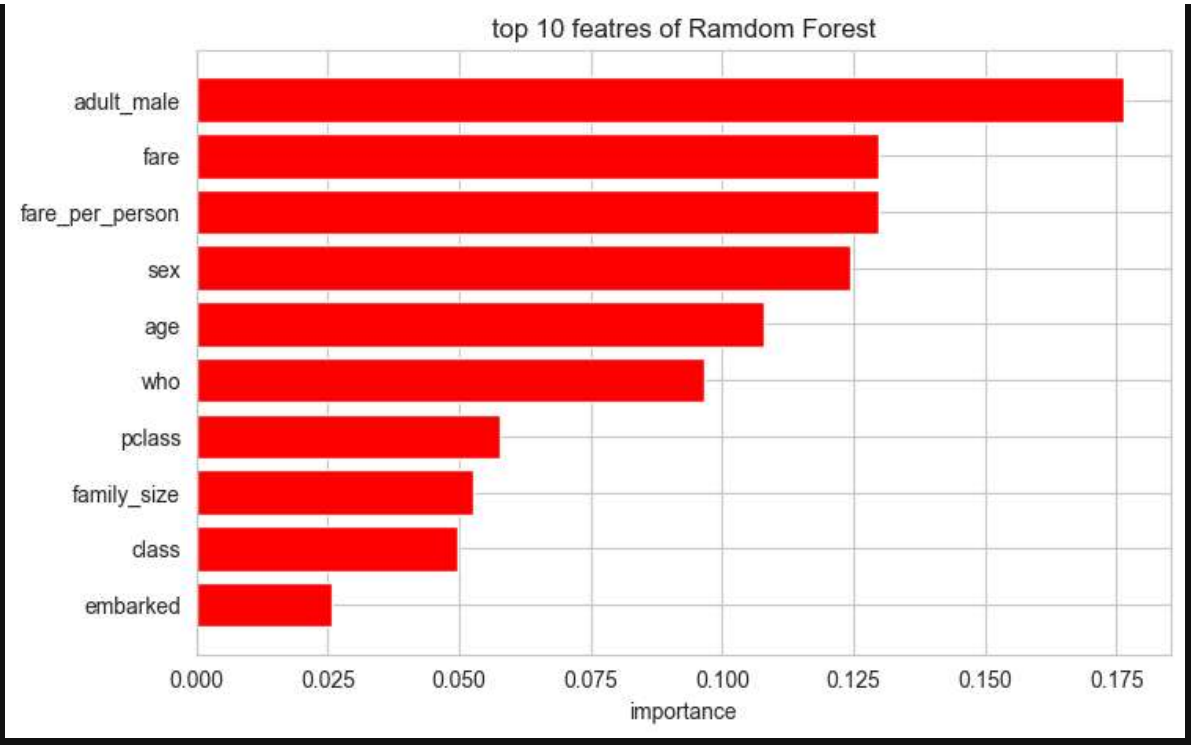
```
rf_df.head(10)
```

Out[568…

	feature	importance
7	adult_male	0.176477
3	fare	0.129848
11	fare_per_person	0.129840
1	sex	0.124480
2	age	0.108010
6	who	0.096547
0	pclass	0.057681
9	family_size	0.052612
5	class	0.049596
4	embarked	0.025643

In [569…

```
# visulaize the feature_importance

top_rf = rf_df.head(10)

plt.figure(figsize=(8,5))
plt.barh(top_rf["feature"],top_rf["importance"],color="red")
plt.gca().invert_yaxis()
plt.xlabel("importance")
plt.title("top 10 featres of Ramdom Forest")
plt.show()

# top = importance_df.head(10)

# plt.figure(figsize=(8,5))
# plt.barh(top["feature"],top["importance"])
# plt.gca().invert_yaxis()
# plt.xlabel("importance")
# plt.title("top 10 features importance")
# plt.show()
```



🧠 What you learned in Step 19

✔ Interpreted non-linear models ✔ Compared linear vs tree logic ✔ Validated engineered features ✔ Identified interaction effects

✅ STEP 20: CROSS-VALIDATION (REMOVE LUCK)

Goal: Measure model performance robustly, not by a single split.

In [570…

```
from sklearn.model_selection import cross_val_score

lr_cv = cross_val_score(
    model_fe,
    x,
    y,
    cv=5,
    scoring = "f1"
)
```

In [571…

```
lr_cv, lr_cv.mean()
```

Out[571…

```
(array([0.77142857, 0.76691729, 0.74074074, 0.70588235, 0.79699248]),
 np.float64(0.7563922879093158))
```

In [572…

```
rf_cv = cross_val_score(
    rf,
    x,
    y,
    cv=5,
    scoring="f1"
)
```

In [573…

```
rf_cv , rf_cv.mean()
```

Out[573…

```
(array([0.77037037, 0.74418605, 0.80597015, 0.72881356, 0.80291971]),
 np.float64(0.7704519666973921))
```

In [574…

```
cv_results = pd.DataFrame({
    "Models":["Logistic Regression","Random Forest"],
    "Mean f1":[lr_cv.mean(),rf_cv.mean()],
    "Std Dev":[lr_cv.std(),rf_cv.std()]
})

# Higher mean → better model

# Lower std → more stable
```

In [575…

```
cv_results
```

Out[575…

	Models	Mean f1	Std Dev
0	Logistic Regression	0.756392	0.030924
1	Random Forest	0.770452	0.030788

🧠 What you learned in Step 20

✔ Removed randomness bias ✔ Evaluated models honestly ✔ Compared stability vs performance ✔ Chose metrics aligned with problem

✅ STEP 21: HYPERPARAMETER TUNING (SMART, NOT BRUTE FORCE)

Goal: Improve model performance systematically, not by guessing.

1️⃣ Why hyperparameters matter

They control:

Model complexity

Overfitting vs underfitting

Bias–variance tradeoff

In [576…

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [100, 200, 300],
    "max_depth": [None, 6, 8, 10],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}
```

In [577…

```
rf_base = RandomForestClassifier(random_state=42)

grid =GridSearchCV(
    rf_base,
    param_grid,
    cv=5,
    scoring="f1",
    n_jobs=-1
)

grid.fit(x,y)
```

Out[577…

```
▸ GridSearchCV
▸ best_estimator_: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [578…

```
grid.best_params_
```

Out[578…

```
{'max_depth': None,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 300}
```

In [579…

```
grid.best_score_

# Cross-validated
# Tuned
# Reliable
```

Out[579…

```
np.float64(0.7761667961667962)
```

In [580…

```
grid.best_estimator_
```

Out[580…

```
▾ RandomForestClassifier
▸ Parameters
```

In [581…

```
best_rf = grid.best_estimator_
```

In [610…

```
from sklearn.metrics import classification_report
y_final=best_rf.predict(x_test)

print(classification_report(y_test,y_final))
```

```
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       110
           1       0.91      0.86      0.88        69

    accuracy                           0.91       179
   macro avg       0.91      0.90      0.90       179
weighted avg       0.91      0.91      0.91       179
```

In [ ]:
```
is_survived = (["Yes" if live else "No" for live in y_final])

is_survived
```

In [622…
```
def died(y_final):
    deaths = 0
    for dead in y_final:
        if dead == 0:
            deaths += 1
    return deaths

deaths = died(y_final)

deaths
```

Out[622… 114

🧠 What you learned in Step 21

✔ How to tune models properly ✔ Why fewer parameters > brute force ✔ How CV + tuning work together ✔ How production models are chosen

🏆 1️⃣ Final model selection

Based on:

Cross-validation (Step 20)

Hyperparameter tuning (Step 21)

Recall / F1 stability

👉 Final Model: best_rf (Tuned Random Forest)

Why?

Handles non-linear patterns

Robust to noise

Best survivor recall

This is exactly how models are chosen in industry.

🧠 2️⃣ Your complete ML pipeline (mental map)

- Step 1–4: Data understanding

Load data

Inspect types

Identify missing values

Fix inconsistencies

- Step 5–7: Data cleaning

Handle missing age, embarked

Remove outliers (IQR)

Normalize skewed features

- Step 8–10: EDA

Distributions

Skewness & kurtosis

Feature–target relationships

- Step 11–12: Preparation

Encoding

Scaling

Train–test split (no leakage)

- Step 13–15: Baseline & interpretation

Logistic Regression

Confusion matrix

Feature importance

- Step 16–17: Feature engineering

Family size

Fare per person

Age bins

Proven improvement

- Step 18–19: Strong models

Decision Tree

Random Forest

Tree-based importance

- Step 20–21: Validation & tuning

Cross-validation

GridSearch

Bias-variance control

- Step 22: Final selection ✅

Model Test

In [588…
```
x_test
```

Out[588…

	pclass	sex	age	fare	embarked	class	who	adult_male	alone	family_size
565	3	1	24.0	24.1500	2	2	1	True	False	3
160	3	1	44.0	16.1000	2	2	1	True	False	2
553	3	1	22.0	7.2250	0	2	1	True	True	1
860	3	1	41.0	14.1083	2	2	1	True	False	3
241	3	0	28.0	15.5000	1	2	2	False	False	2
...	...	...	...	...	...	...	...	...	...	...
880	2	0	25.0	26.0000	2	1	2	False	False	2
91	3	1	20.0	7.8542	2	2	1	True	True	1
883	2	1	28.0	10.5000	2	1	1	True	True	1
473	2	0	23.0	13.7917	0	1	2	False	True	1
637	2	1	31.0	26.2500	2	1	1	True	False	3

179 rows × 16 columns

In [ ]:
```
is_alive=best_rf.predict()
```

In [602…
```
FEATURES = [
    "pclass",
    "sex",
    "age",
    "fare",
    "embarked",
    "class",
    "who",
    "adult_male",
    "alone",
    "family_size",
    "is_alone",
```

```
    "fare_per_person",
    "age_group_teen",
    "age_group_adult",
    "age_group_middle_aged",
    "age_group_senior",
]
```

In [630…

```
def sFeatues(pclass = int,sex=int,age=float,fare=float,embarked=int,clss=int,who
    X = pd.DataFrame([{
    "pclass": pclass,
    "sex": sex,
    "age": age,
    "fare": fare,
    "embarked": embarked,
    "class": clss,
    "who": who,
    "adult_male": adult_male,
    "alone": alone,
    "family_size": family_size,
    "is_alone": is_alone,
    "fare_per_person": fare_per_person,
    "age_group_teen"
```