Titanic Survival Prediction Project: Full Code Explanation

# Step 1: Import Required Libraries

In this step, we import all the necessary libraries used in the project.  
  
```python  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
```  
  
1. \*\*pandas (pd)\*\*: Used for data manipulation and analysis.  
 - It provides powerful data structures like `DataFrame`, which allows us to load, clean, and manipulate tabular data easily.  
 - Example: `df = pd.read\_csv('file.csv')` reads a CSV file into a DataFrame.  
 - If we skip this, we cannot load or manipulate data efficiently.  
  
2. \*\*numpy (np)\*\*: Provides support for numerical operations.  
 - Often used with pandas to perform mathematical operations on arrays.  
 - If omitted, operations like handling missing data or mathematical transformations might become cumbersome.  
  
3. \*\*matplotlib.pyplot (plt)\*\*: A plotting library used for visualizing data.  
 - Example: `plt.plot(x, y)` creates a simple line graph.  
 - Without it, we lose the ability to visualize patterns, distributions, and relationships.  
  
4. \*\*seaborn (sns)\*\*: Built on top of matplotlib; provides attractive statistical graphics.  
 - Example: `sns.barplot(x='Sex', y='Survived', data=df)` shows survival rates by gender.  
 - If skipped, our visualizations would be less informative and aesthetic.  
  
5. \*\*sklearn.model\_selection.train\_test\_split\*\*: Splits the dataset into training and testing sets.  
 - Ensures we train the model on one part and test it on another to evaluate performance.  
 - Without it, model evaluation would be unreliable.  
  
6. \*\*sklearn.linear\_model.LogisticRegression\*\*: The main algorithm used for this project.  
 - Logistic Regression is a classification algorithm used when the target is binary (e.g., survived or not).  
 - Chosen for its simplicity and interpretability.  
 - If not used, we wouldn’t be able to train our classifier.  
  
7. \*\*sklearn.metrics\*\*: Provides tools to evaluate the model (confusion matrix, classification report, accuracy).  
 - Helps us understand how well our model performs.

# Step 2: Load the Dataset

```python  
df = pd.read\_csv('titanic.csv')  
df.head()  
```  
  
- This line loads the Titanic dataset from a CSV file into a pandas DataFrame called `df`.  
- `df.head()` displays the first five rows for a quick overview of the dataset.  
- If not executed, we won't have any data to work with.

# Step 3: Explore the Dataset (EDA)

EDA helps us understand the structure and key patterns in the data.  
  
```python  
df.isnull().sum()  
df.info()  
```  
  
- These lines check for missing values and provide data types and counts.  
- Crucial for identifying data quality issues.  
  
```python  
sns.barplot(data=df, x='Sex', y='Survived')  
plt.show()  
```  
  
- Visualizes survival rate by gender using Seaborn.  
- Helps discover that more females survived.  
  
Similar plots are used for 'Pclass' and age group using `pd.cut()` to create bins.

# Step 4: Data Cleaning

```python  
df['Age'].fillna(df['Age'].median(), inplace=True)  
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)  
df.drop('Cabin', axis=1, inplace=True)  
df.dropna(inplace=True)  
```  
  
- Missing values are handled here.  
- `fillna()` replaces missing values to avoid model training errors.  
- `drop()` removes unnecessary or highly incomplete columns.  
  
Without cleaning, the model could crash or perform poorly.

# Step 5: Feature Selection and Encoding

```python  
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']  
X = df[features]  
y = df['Survived']  
X = pd.get\_dummies(X, drop\_first=True)  
```  
  
- We select the input features and target variable.  
- `get\_dummies()` performs one-hot encoding to convert categorical variables into numeric format.  
- `drop\_first=True` avoids multicollinearity.  
  
If skipped, the model won’t accept categorical inputs.

# Step 6: Split Data

```python  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
```  
  
- Splits the data into training and test sets (80% train, 20% test).  
- Ensures fair model evaluation.

# Step 7: Train Model

```python  
model = LogisticRegression(max\_iter=1000)  
model.fit(X\_train, y\_train)  
```  
  
- Initializes and trains the Logistic Regression model.  
- `max\_iter=1000` ensures convergence for complex datasets.  
  
Why Logistic Regression?  
- Suitable for binary classification (0 = not survived, 1 = survived).  
- Fast and interpretable.

# Step 8: Evaluate Model

```python  
y\_pred = model.predict(X\_test)  
print(confusion\_matrix(y\_test, y\_pred))  
print(classification\_report(y\_test, y\_pred))  
print(accuracy\_score(y\_test, y\_pred))  
```  
  
- Compares predictions to true labels.  
- `confusion\_matrix`: shows TP, FP, TN, FN.  
- `classification\_report`: includes precision, recall, F1-score.  
- `accuracy\_score`: overall correct predictions ratio.

# Step 9: Bonus Visualization

```python  
sex\_survival = df.groupby('Sex')['Survived'].mean()  
sns.barplot(x=sex\_survival.index, y=sex\_survival.values)  
plt.title('Survival Rate by Sex')  
```  
  
- Final visualization showing average survival rate by gender.  
- Reinforces insight: females had a higher survival rate.