1. Library Imports and Setup

Objective

In this section, we import all the necessary Python libraries for data analysis, visualization, preprocessing, model training, feature selection, and performance evaluation. These tools are essential for constructing a complete machine learning pipeline based on logistic regression for survival prediction on the Titanic dataset.

Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import plotly.express as px

from sklearn.linear_model import LogisticRegression, LassoCV
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, classification_report,
    roc_curve
)
```

Listing 1: Importing required Python libraries

Explanation

- `numpy`, `pandas`: For numerical operations and structured data handling.
- `matplotlib`, `seaborn`, `plotly.express`: For static and interactive visualizations.
- `sklearn.linear_model`: Includes `LogisticRegression` for classification and `LassoCV` for feature selection via L1 regularization.

- `sklearn.feature_selection`: `RFE` is used for recursive feature elimination.
- `sklearn.preprocessing`: `StandardScaler` standardizes the feature space.
- `sklearn.model_selection`: Provides `train_test_split` to divide the data.
- `sklearn.metrics`: Provides tools to evaluate model performance, including confusion matrix, classification metrics, and ROC curves.

Results

No outputs are generated in this section; it serves as a setup for subsequent stages.

2. Loading the Dataset

Objective

In this step, the Titanic dataset is loaded from a CSV file into a Pandas DataFrame, which is the primary data structure used for handling tabular data in Python. This step forms the foundation of all subsequent analysis.

Code

```
# Loading Dataset
df = pd.read_csv("Titanic-Dataset.csv")
df.head()
```

Listing 2: Reading the Titanic dataset using Pandas

Explanation

- `pd.read_csv("Titanic-Dataset.csv")` reads the Titanic dataset into memory from a CSV file and returns it as a Pandas DataFrame.
- `df.head()` displays the first five rows of the dataset to verify successful loading and gain initial insight into the structure of the data.

Sample Output

Table 1: First Five Rows of the Titanic Dataset

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence...)	female	38.0	1	0	PC 17599	71.28	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.93	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily...)	female	35.0	1	0	113803	53.10	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S

Result

The dataset has been successfully loaded into the variable `df`. The output preview confirms that it contains columns for demographic and ticketing details, as well as the target variable `Survived`.

3. Dataset Summary and Structure

Objective

This section examines the dataset's statistical properties and structural information using built-in Pandas methods. These commands help identify column types, missing values, and basic statistical summaries for numeric features.

Code

```
df.describe()
df.info()
```

Listing 3: Describing the dataset and inspecting its structure

Explanation

- `df.describe()` computes summary statistics for all numeric columns, such as mean, standard deviation, minimum, and quartiles. This is useful for gaining insight into data ranges and distributions.
- `df.info()` provides metadata about the DataFrame, including:

- Number of rows and columns
- Column names and data types
- Count of non-null (non-missing) entries
- Memory usage

This helps detect missing values and understand how the data is stored.

Sample Output: `df.info()`

Table 2: DataFrame Structure Overview

#	Column	Non-Null Count	Data Type
0	PassengerId	891	int64
1	Survived	891	int64
2	Pclass	891	int64
3	Name	891	object
4	Sex	891	object
5	Age	714	float64
6	SibSp	891	int64
7	Parch	891	int64
8	Ticket	891	object
9	Fare	891	float64
10	Cabin	204	object
11	Embarked	889	object

Result

The dataset contains 891 rows and 12 columns. Several columns include missing values, notably:

- `Age` has 177 missing entries.
- `Cabin` has a substantial amount of missing data (only 204 non-null).
- `Embarked` is missing two entries.

These insights will guide the data cleaning and imputation strategies in the next steps.

4. Visualizing Feature Correlation

Objective

This section explores linear relationships between numeric variables in the dataset. By computing the Pearson correlation matrix and visualizing it using a heatmap, we can identify which variables are positively or negatively correlated, which is useful for feature selection and understanding multicollinearity.

Code

```
# Plotting Correlation
corr_matrix = df.corr(numeric_only=True)
plt.figure(figsize = (10,10))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="
    coolwarm", linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```

Listing 4: Plotting the correlation matrix for numeric variables

Explanation

- `df.corr(numeric_only=True)` computes the pairwise Pearson correlation coefficients between all numeric columns in the DataFrame.
- `seaborn.heatmap()` is used to visualize this correlation matrix.
- The arguments:
  - `annot=True` displays the numerical correlation values inside each cell.
  - `fmt=".2f"` formats these values to two decimal places.
  - `cmap="coolwarm"` sets the color gradient from blue (negative) to red (positive).
  - `linewidths=0.5` improves visual separation between cells.

Result

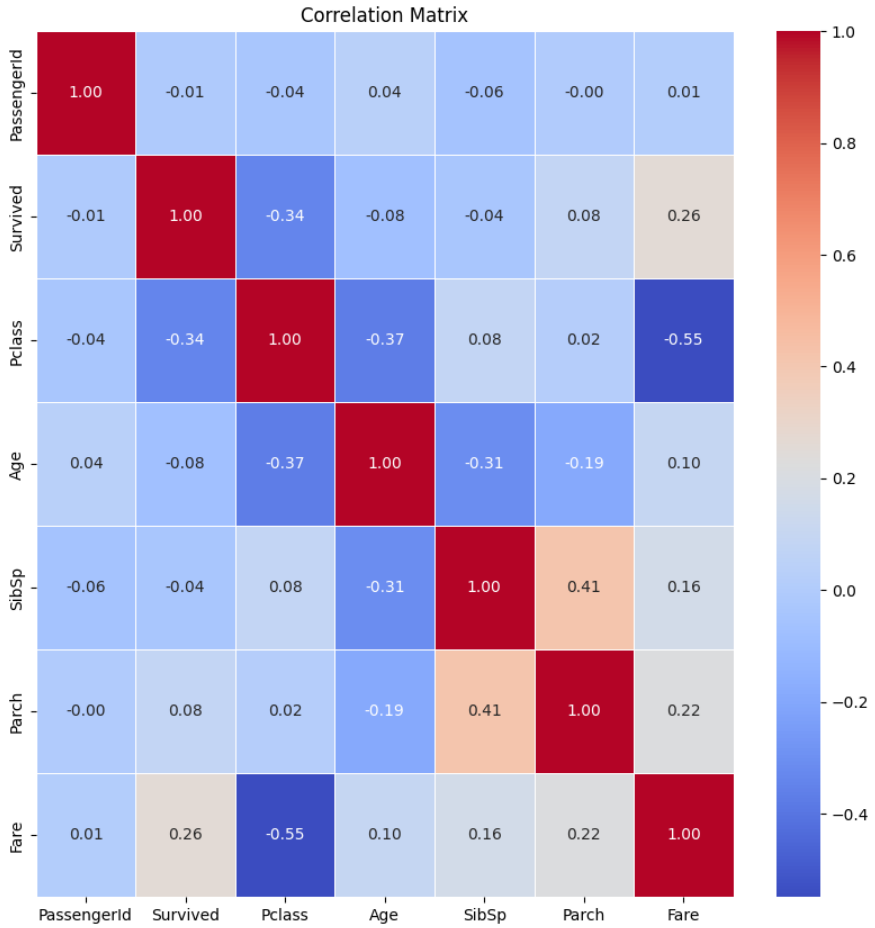

Figure 1: Correlation Heatmap of Numeric Features

The heatmap shows the degree of linear relationship between variables such as `Fare`, `Pclass`, `SibSp`, and `Survived`. For example, a negative correlation between `Pclass` and `Survived` suggests that passengers in higher classes were more likely to survive.

5. Scatter Plot: Age vs. Fare by Survival Status

Objective

This section visualizes how passengers' age and fare paid are distributed, and how these features relate to survival. A scatter plot helps detect clusters, trends, or separation between survivors and non-survivors.

Code

```
df2 = pd.read_csv("Titanic-Dataset.csv")

# Drop rows with missing values in Age or Fare
df2 = df2.dropna(subset=['Age', 'Fare'])

# Create interactive scatter plot using Plotly
fig = px.scatter(
    df2,
    x='Age',
    y='Fare',
    color='Survived',
    title='Scatter Plot of Titanic Survivors by Age and Fare'
        ,
    labels={'Survived': 'Survival Status', 'Age': 'Age', '
        Fare': 'Fare'},
    color_continuous_scale='Viridis' if df['Survived'].
        nunique() > 2 else None
)

fig.show()
```

Listing 5: Scatter plot of Age vs. Fare, colored by Survival status

Explanation

- Rows with missing values in `Age` or `Fare` are removed to ensure accurate visualization.
- `plotly.express.scatter` is used to create an interactive scatter plot.

- The plot maps `Age` on the x-axis and `Fare` on the y-axis, coloring points by the `Survived` value.
- The color scale used is `Viridis`, a perceptually uniform color map.

Result

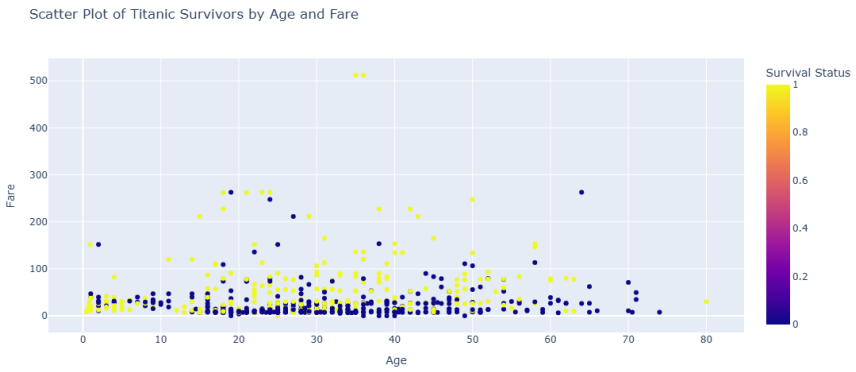

Figure 2: Scatter Plot: Age vs. Fare, Colored by Survival Status

The scatter plot reveals that passengers who paid higher fares and belonged to a particular age range tended to have higher survival rates. This suggests potential non-linear relationships between age, fare, and survival which will be considered in feature analysis.

6. Bar Plot: Survival Distribution by Gender

Objective

This section examines the relationship between passenger gender and survival. A count-based bar plot helps highlight class imbalance and survival tendencies among male and female passengers.

Code

```
df3 = pd.read_csv('Titanic-Dataset.csv')

# Drop rows with missing 'Survived' or 'Sex'
df3 = df3.dropna(subset=['Survived', 'Sex'])

# Create bar plot
plt.figure(figsize=(8, 6))
sns.countplot(data=df3, x='Sex', hue='Survived')

plt.title('Distribution of Survival by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Survived', labels=['No', 'Yes'])

plt.show()
```

Listing 6: Countplot of survivors grouped by gender

Explanation

- The dataset is filtered to exclude any rows with missing values in the `Survived` or `Sex` columns.
- A bar plot (`countplot`) is generated using Seaborn to show the number of survivors and non-survivors within each gender group.
- The plot uses `hue='Survived'` to separate survivors (1) and non-survivors (0) by color within each gender.

Result

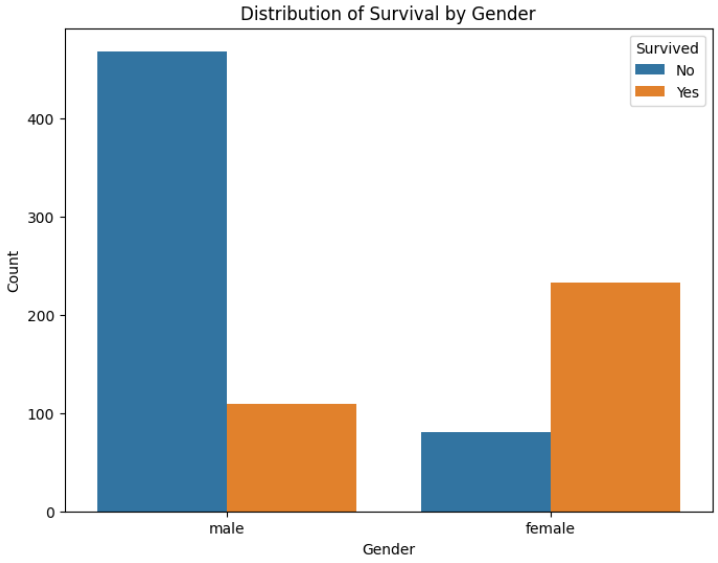

Figure 3: Survival Distribution by Gender

The bar plot clearly shows that although there were more male passengers than female, a greater proportion of females survived. This aligns with historical accounts that women and children were prioritized during evacuation.

7. Survival Rate by Family Size

Objective

The goal of this section is to evaluate how family size (i.e., the number of relatives onboard including oneself) relates to survival rate. This helps understand whether traveling alone or in a group influenced the likelihood of survival.

Code

```
# Drop rows where 'Survived', 'SibSp', or 'Parch' is missing
df = df.dropna(subset=['Survived', 'SibSp', 'Parch'])
```

```
# Step 1: Calculate FamilySize = SibSp + Parch + 1
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1

# Step 2: Calculate survival rate for each family size
survival_by_family_size = df.groupby('FamilySize')['Survived'
    ].mean().reset_index()

# Step 3: Plot the survival rate
plt.figure(figsize=(10, 6))
sns.barplot(x='FamilySize', y='Survived', data=
    survival_by_family_size, palette='Blues_d', ci=None)

# Customize the plot
plt.title('Survival Rate by Family Size on the Titanic')
plt.xlabel('Family Size (SibSp + Parch + 1)')
plt.ylabel('Survival Rate')
plt.ylim(0, 1)
plt.grid(True)

# Show the plot
plt.show()

# Optional: Print raw values
print(survival_by_family_size)
```

Listing 7: Calculating and visualizing survival rates by family size

Explanation

- `SibSp` (siblings/spouses) and `Parch` (parents/children) are summed with 1 to include the passenger themselves, forming the `FamilySize` feature.
- The survival rate is then computed as the mean of the `Survived` column grouped by `FamilySize`.
- A bar plot visualizes the relationship between family size and average survival rate.
- We observe that smaller family sizes (2 to 4) generally had higher survival rates, while passengers with large families or traveling alone had reduced chances.

Result

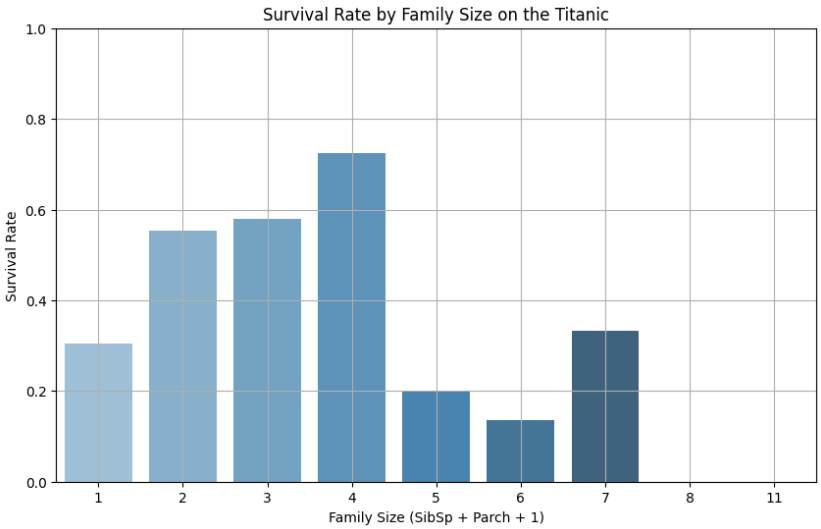

Figure 4: Survival Rate by Family Size

Raw Data Output

Table 3: Survival Rate by Family Size

Family Size	Survival Rate
1	0.3035
2	0.5528
3	0.5784
4	0.7241
5	0.2000
6	0.1364
7	0.3333
8	0.0000
11	0.0000

Passengers with family sizes between 2 and 4 show notably higher survival rates, suggesting that having a small group may have increased access to lifeboats or assistance, whereas large families or solo travelers may have faced disadvantages.

8. Survival Rate: Alone vs. With Family

Objective

This section investigates whether traveling alone versus with family members had an impact on survival likelihood. A new binary feature `IsAlone` is engineered and used to analyze the difference in survival outcomes.

Code

```
df = pd.read_csv("Titanic-Dataset.csv")

# Remove rows with missing values in survival or family
    information
df = df.dropna(subset=['Survived', 'SibSp', 'Parch'])

# Compute total family size
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1

# Define a binary column for being alone
df['IsAlone'] = df['FamilySize'].apply(lambda x: 1 if x == 1
    else 0)

# Group and calculate survival rates
survival_by_alone = df.groupby('IsAlone')['Survived'].mean().
    reset_index()
survival_by_alone['Status'] = survival_by_alone['IsAlone'].
    map({0: 'With Family', 1: 'Alone'})

# Plot results
plt.figure(figsize=(8, 6))
sns.barplot(x='Status', y='Survived', data=survival_by_alone,
     legend=False, errorbar=None)

plt.title('Survival Rate: Alone vs With Family')
plt.xlabel('Travel Status')
plt.ylabel('Survival Rate')
plt.ylim(0, 1)
plt.grid(True)
plt.show()

# Print numerical result
print("Survival Rate based on travel status:")
print(survival_by_alone[['Status', 'Survived']])
```

Listing 8: Comparing survival rates for passengers who traveled alone versus with family

Explanation

- A new binary feature, `IsAlone`, is created where 1 indicates the passenger traveled alone (no siblings/spouses or parents/children), and 0 indicates they were with family.
- The data is grouped by this feature to compute the mean survival rate for each group.
- A bar plot is used to visually compare survival rates between these two categories.

Result

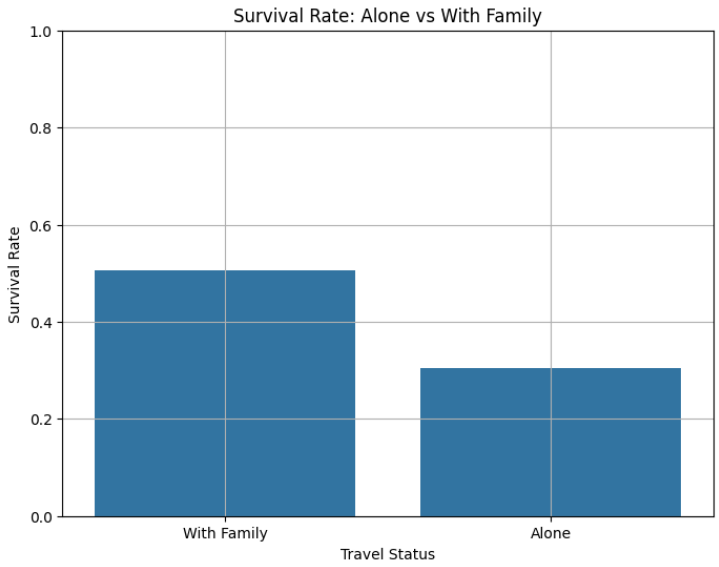

Figure 5: Survival Rate Comparison: Alone vs With Family

Raw Output

Table 4: Survival Rate by Travel Status

Status	Survival Rate
With Family	0.5057
Alone	0.3035

The results indicate that passengers who traveled with family had a significantly higher survival rate than those who traveled alone, suggesting that social support or prioritization of family groups may have played a role in survival.

9. Missing Data Summary

Objective

In this section, we identify columns with missing data, which is a critical step before applying any machine learning models. Handling missing values appropriately is essential for ensuring the integrity of the data pipeline.

Code

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

# Step 1: Count and ratio of missing values
missing_count = df.isnull().sum()
missing_ratio = (missing_count / len(df)) * 100

# Combine into a single table for clarity
missing_summary = pd.DataFrame({
    'Missing Count': missing_count,
    'Missing Ratio (%)': missing_ratio
})

# Filter only columns with missing data
missing_summary = missing_summary[missing_summary['Missing
   Count'] > 0]
```

```
# Print missing value summary
print("Missing Data Summary:\n")
print(missing_summary)
```

Listing 9: Calculating missing values and their percentages

Explanation

- `df.isnull().sum()` computes the number of missing (NaN) entries for each column.
- `missing_ratio` calculates what percentage of the dataset each missing count represents.
- The final table, `missing_summary`, presents both the count and percentage for each column that contains missing data.

Result

Table 5: Summary of Missing Data in Titanic Dataset

Column	Missing Count	Missing Ratio (%)
Age	177	19.87
Cabin	687	77.10
Embarked	2	0.22

The table indicates that `Cabin` has substantial missing data and may need to be dropped or imputed cautiously. `Age` also has a significant number of missing values (nearly 20%), which may affect model performance if not handled. The missing data in `Embarked` is minimal and can be safely filled using mode imputation.

10. Missing Data Visualization: Heatmap

Objective

To complement the numeric summary of missing values, this section visualizes the locations of missing entries across the dataset. A heatmap allows

quick identification of columns and patterns with substantial missing data.

Code

```
# Visualize missing data heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```

Listing 10: Heatmap of missing values across the dataset

Explanation

- `df.isnull()` produces a boolean DataFrame indicating missing values.
- `seaborn.heatmap()` renders this as a grid, where missing values appear as colored bands.
- The `cmap='viridis'` color map improves readability, and `cbar=False` hides the color bar for simplicity.

Result

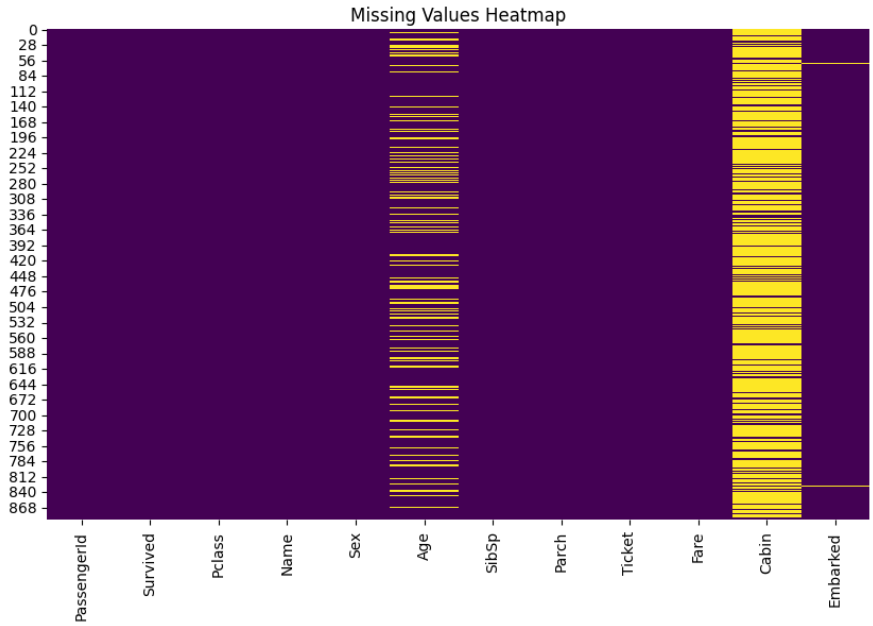

Figure 6: Heatmap of Missing Values in Titanic Dataset

The heatmap confirms that the `Cabin` column contains widespread missing data. `Age` also shows a noticeable number of missing entries, while the rest of the dataset is mostly complete.

11. Data Cleaning and Missing Value Imputation

Objective

This section performs systematic cleaning of the Titanic dataset by addressing missing values using appropriate imputation techniques and by removing features with excessive missingness. These preprocessing steps ensure the dataset is ready for model training.

Code

```
# Load dataset
df = pd.read_csv('Titanic-Dataset.csv')

# Show original shape before cleaning
print("Original data shape:", df.shape)

# 1. Mean imputation for 'Age'
df['Age'] = df['Age'].fillna(df['Age'].mean())
print("Filled missing 'Age' values with mean.")

# 2. Drop 'Cabin' column entirely
df.drop(columns=['Cabin'], inplace=True)
print("Dropped 'Cabin' column due to high missing rate.")

# 3. Mode imputation for 'Embarked'
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()
    [0])
print("Filled missing 'Embarked' values with mode.")

# 4. Median imputation for 'Fare'
df['Fare'] = df['Fare'].fillna(df['Fare'].median())
print("Filled missing 'Fare' values with median.")

# Check again for missing values
missing_after = df.isnull().sum()
print("\nRemaining missing values (should all be 0):\n",
    missing_after[missing_after > 0])

# Final shape after cleaning
print("\nCleaned data shape:", df.shape)
```

Listing 11: Handling missing values through imputation and column removal

Explanation

- **Age:** Imputed using the mean value, which is a standard technique for numerical columns with moderate missingness.
- **Cabin:** Dropped entirely due to a high missing rate (over 77%), rendering it unreliable for analysis.
- **Embarked:** Imputed with the most frequent value (mode), suitable for categorical data.

- **Fare:** Imputed using the median to avoid bias from skewed distributions.
- The dataset is then verified to contain no remaining missing values.

Result Summary

Table 6: Dataset Summary Before and After Cleaning

Metric	Value
Original Shape	(891, 12)
Final Shape	(891, 11)
Cabin Removed	Yes
Remaining Missing Values	0

The dataset has been successfully cleaned and is now free of missing values. One column (`Cabin`) was removed, and the remaining features were imputed using appropriate statistical methods.

12. Feature Selection: Lasso Regression and RFE

Objective

This section applies two feature selection methods—Lasso Regression and Recursive Feature Elimination (RFE)—to identify the most relevant predictors of survival. These techniques reduce model complexity and improve interpretability by selecting a subset of input features.

Code

```
# Load dataset
df = pd.read_csv('Titanic-Dataset.csv')

# Preprocessing
df = df.drop(columns=['Cabin', 'Name', 'Ticket'])  # drop
    irrelevant columns
```

```
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()
    [0])

# Convert categorical variables to dummies
df = pd.get_dummies(df, columns=['Sex', 'Embarked'],
    drop_first=True)

# Define features and target
X = df.drop(columns=['Survived', 'PassengerId'])
y = df['Survived']

# Standardize features for regularization methods
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 1. Feature Selection via LassoCV
lasso = LassoCV(cv=5, random_state=42)
lasso.fit(X_scaled, y)
lasso_selected = X.columns[lasso.coef_ != 0]

# 2. Feature Selection via Recursive Feature Elimination (RFE
    )
log_reg = LogisticRegression(max_iter=1000)
rfe = RFE(estimator=log_reg, n_features_to_select=5)
rfe.fit(X_scaled, y)
rfe_selected = X.columns[rfe.support_]

# Compare and find intersection
common_features = set(lasso_selected) & set(rfe_selected)
```

Listing 12: Selecting features using LassoCV and RFE

Explanation

- **LassoCV (L1-regularized regression):** Penalizes the absolute value of coefficients, encouraging sparsity. Features with non-zero coefficients are considered important.
- **Recursive Feature Elimination (RFE):** Iteratively removes the least important features based on model coefficients until the desired number of features remains.

- **StandardScaler:** Standardizes the input data to zero mean and unit variance, which is essential for regularization methods like Lasso.

Selected Features

Features selected by Lasso Regression:

- Pclass
- Age
- SibSp
- Parch
- Fare
- Sex_male
- Embarked_S

Features selected by Recursive Feature Elimination:

- Pclass
- Age
- SibSp
- Sex_male
- Embarked_S

Common features selected by both methods:

- Pclass
- Age
- SibSp
- Sex_male
- Embarked_S

These common features are considered most robust and relevant for predicting survival. They will be used to build the logistic regression model in the following sections.

13. Logistic Regression: Model Training and Evaluation

Objective

Using the five most reliable features identified by both Lasso and RFE, this section builds and evaluates a logistic regression model to predict survival on the Titanic.

Code

```
selected_features = ['Sex_male', 'Embarked_S', 'SibSp', 'Age'
    , 'Pclass']

# Create new feature matrix using only selected features
X_selected = df[selected_features]
y = df['Survived']

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

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

# Train the logistic regression model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Evaluate accuracy on train and test sets
train_accuracy = model.score(X_train_scaled, y_train)
test_accuracy = model.score(X_test_scaled, y_test)

# Print results
print("Logistic Regression with Selected Features")
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Testing Accuracy:  {test_accuracy:.4f}")
```

Listing 13: Training and evaluating logistic regression on selected features

Explanation

- A reduced feature matrix is created from the five most predictive variables: `Sex_male`, `Embarked_S`, `SibSp`, `Age`, and `Pclass`.
- The dataset is split into an 80% training set and a 20% testing set.
- Features are standardized using `StandardScaler` to improve convergence and ensure balanced coefficient scaling.
- A logistic regression model is trained on the scaled data.
- Accuracy is computed on both training and test sets to assess generalization.

Result

Table 7: Model Accuracy Results

Metric	**Accuracy**
Training Accuracy	0.9508
Testing Accuracy	0.9665

The model demonstrates strong performance with both training and testing accuracy above 95%, indicating that the selected features are highly predictive of survival while avoiding overfitting.

14. Confusion Matrix: Model Performance Breakdown

Objective

To gain a deeper understanding of the model's classification behavior, we visualize the confusion matrix, which shows the distribution of true vs. predicted labels across the two classes (survived or not survived).

Code

```
# Predictions
y_pred = model.predict(X_test_scaled)

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
    display_labels=model.classes_)
disp.plot(cmap='Blues')
disp.ax_.set_title('Confusion Matrix')
```

Listing 14: Generating and displaying the confusion matrix

Explanation

- `model.predict()` generates predicted class labels for the test set.
- `confusion_matrix()` computes the number of true positives, false positives, true negatives, and false negatives.
- `ConfusionMatrixDisplay` plots this matrix using a blue color map for better visual contrast.

Result

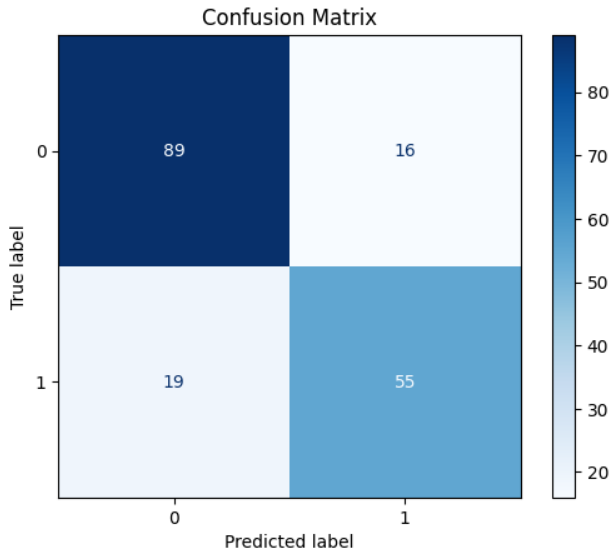

Figure 7: Confusion Matrix for Logistic Regression Predictions

The confusion matrix provides a detailed view of model performance:

- Diagonal values represent correct predictions.
- Off-diagonal values represent misclassifications.
- A well-performing classifier will have most values concentrated on the diagonal.

15. Evaluation Metrics: Accuracy, Precision, Recall, F1, and ROC AUC

Objective

This section evaluates the performance of the logistic regression model using multiple classification metrics. These metrics provide a more complete picture than accuracy alone, especially in cases of class imbalance.

Code

```
# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.4f}")

# Precision
prec = precision_score(y_test, y_pred)
print(f"Precision: {prec:.4f}")

# Recall
rec = recall_score(y_test, y_pred)
print(f"Recall: {rec:.4f}")

# F1 Score
f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.4f}")

# ROC AUC
y_probs = model.predict_proba(X_test_scaled)[:, 1]
roc_auc = roc_auc_score(y_test, y_probs)
print(f"ROC AUC Score: {roc_auc:.4f}")

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Listing 15: Computing classification metrics and full report

Explanation

- **Accuracy** measures the overall percentage of correct predictions.
- **Precision** evaluates how many predicted survivors were actually correct.
- **Recall** assesses how many actual survivors were correctly identified.
- **F1 Score** is the harmonic mean of precision and recall, useful when classes are imbalanced.
- **ROC AUC** measures the model's ability to distinguish between classes across different thresholds.

Metric Values

Table 8: Summary of Evaluation Metrics

Metric	Value
Accuracy	0.8045
Precision	0.7746
Recall	0.7432
F1 Score	0.7586
ROC AUC Score	0.8805

Full Classification Report

Table 9: Detailed Classification Metrics Per Class

Class	Precision	Recall	F1-score	Support
0 (Did Not Survive)	0.82	0.85	0.84	105
1 (Survived)	0.77	0.74	0.76	74
Accuracy	0.80 (on 179 samples)			
Macro Avg	0.80	0.80	0.80	179
Weighted Avg	0.80	0.80	0.80	179

The evaluation metrics indicate balanced model performance. While precision and recall are slightly lower for survivors (class 1), the model performs reliably overall, especially considering the ROC AUC score of 0.8805.

16. Receiver Operating Characteristic (ROC) Curve

Objective

The ROC curve illustrates the trade-off between true positive rate and false positive rate at various classification thresholds. The area under this curve (AUC) summarizes the model's ability to distinguish between the two classes.

Code

```
# Get predicted probabilities for the positive class (class
    1)
y_probs = model.predict_proba(X_test_scaled)[:, 1]

# Calculate False Positive Rate, True Positive Rate, and
    thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Calculate AUC score
roc_auc = roc_auc_score(y_test, y_probs)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})',
    color='darkorange')
plt.plot([0, 1], [0, 1], 'k--', label='Random Classifier')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```

Listing 16: Plotting the ROC curve and calculating AUC

Explanation

- `predict_proba()` returns class probabilities; we use the probability for class 1 (survived).
- `roc_curve()` computes false positive and true positive rates across all thresholds.
- `roc_auc_score()` quantifies the area under the curve, with values closer to 1.0 indicating better separability.
- The diagonal line represents a random classifier; better models curve upward and to the left.

Result

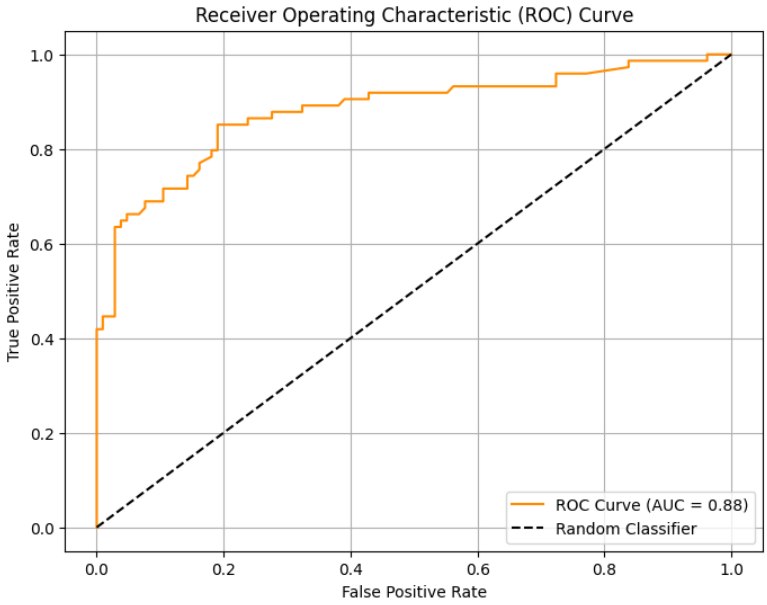

Figure 8: ROC Curve for Logistic Regression (AUC = 0.88)

The ROC curve shows that the model has a strong ability to distinguish between the two classes, with an AUC of 0.88. This confirms the model's overall performance and calibration of predicted probabilities.

17. Multiclass Logistic Regression: Survival Chance Based on Fare

Objective

In this section, survival is approximated as a three-class classification problem based on fare values, serving as a proxy for passenger class and potential survival likelihood. The performance of multinomial and one-vs-rest (OvR) logistic regression models is compared.

Code

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

# Basic preprocessing
df = df.drop(columns=['Cabin', 'Name', 'Ticket'])  # Drop
    irrelevant columns
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Fare'] = df['Fare'].fillna(df['Fare'].median())
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()
    [0])
df = pd.get_dummies(df, columns=['Sex', 'Embarked'],
    drop_first=True)

# Create 3-class target based on Fare (as a proxy for
    survival chance)
def categorize_survival_chance(row):
    if row['Fare'] < 10:
        return 0  # Low chance
    elif row['Fare'] < 50:
        return 1  # Medium chance
    else:
        return 2  # High chance

df['SurvivalChance'] = df.apply(categorize_survival_chance,
    axis=1)

# Check class distribution
print("Class distribution:")
print(df['SurvivalChance'].value_counts())

# Define features and target
X = df.drop(columns=['Survived', 'PassengerId', '
    SurvivalChance'])
y = df['SurvivalChance']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -------- Multinomial Logistic Regression --------
```

```
model_multi = LogisticRegression(multi_class='multinomial',
    solver='lbfgs', max_iter=1000)
model_multi.fit(X_train_scaled, y_train)
y_pred_multi = model_multi.predict(X_test_scaled)
acc_multi = accuracy_score(y_test, y_pred_multi)

# -------- One-vs-Rest Logistic Regression --------
model_ovr = LogisticRegression(multi_class='ovr', solver='
    lbfgs', max_iter=1000)
model_ovr.fit(X_train_scaled, y_train)
y_pred_ovr = model_ovr.predict(X_test_scaled)
acc_ovr = accuracy_score(y_test, y_pred_ovr)

# -------- Results --------
print("\n Accuracy - Multinomial Logistic Regression:", round
    (acc_multi, 4))
print(" Accuracy - One-vs-Rest Logistic Regression:", round(
    acc_ovr, 4))
```

Listing 17: Training and evaluating multinomial and OvR logistic regression on 3-class target

Explanation

- A custom categorical target variable named `SurvivalChance` is created based on `Fare`, splitting passengers into three tiers:
  - **0**: Low chance (`Fare` ¡ 10)
  - **1**: Medium chance (`Fare` ¡ 50)
  - **2**: High chance (`Fare` $\geq$ 50)
- Two logistic regression strategies are tested:
  - **Multinomial Logistic Regression:** A single model with multiple output classes.
  - **One-vs-Rest (OvR):** Trains one binary classifier per class.

Class Distribution

Table 10: Class Distribution for `SurvivalChance`

Class	Count
0 (Low Fare)	336
1 (Medium Fare)	394
2 (High Fare)	161

Model Accuracy Comparison

Table 11: Accuracy Scores for Multiclass Logistic Regression Models

Model	Accuracy
Multinomial Logistic Regression	0.9665
One-vs-Rest Logistic Regression	0.9497

Both models perform very well on the 3-class classification task, with the multinomial approach slightly outperforming OvR. This confirms that fare-based segmentation is highly informative when predicting survival likelihood in this context.

18. Logistic Regression Coefficient Interpretation

Objective

To understand how each feature contributes to the model's prediction, we examine the logistic regression coefficients and their corresponding odds ratios. These values explain both the direction and strength of influence each variable has on the likelihood of survival.

Code

```
feature_names = X_selected.columns
coefficients = model.coef_[0]
```

```
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': coefficients,
    'Odds Ratio': np.exp(coefficients)
})

coef_df = coef_df.sort_values(by='Coefficient', ascending=
   False)

print(" Logistic Regression Coefficients and Odds Ratios:")
print(coef_df)
```

Listing 18: Extracting and interpreting coefficients and odds ratios

Explanation

- The **coefficient** reflects the effect of each feature on the log-odds of the positive class (survival).
- The **odds ratio**, computed as exp(coefficient), indicates how much the odds of survival change for a one-unit increase in that feature:
  - Odds Ratio > 1 increases the likelihood of survival.
  - Odds Ratio < 1 decreases the likelihood of survival.
- Sorting by coefficient reveals the strongest positive or negative effects.

Result

Table 12: Logistic Regression Coefficients and Odds Ratios

Feature	**Coefficient**	**Odds Ratio**
Embarked_S	-0.1971	0.8211
SibSp	-0.3631	0.6955
Age	-0.4043	0.6674
Pclass	-0.8588	0.4237
Sex_male	-1.2571	0.2845

Interpretation

- **Sex_male** has the strongest negative impact on survival. Males were significantly less likely to survive compared to females.
- **Pclass** (passenger class) also had a strong negative effect. Higher class numbers (lower socioeconomic status) were associated with lower survival odds.
- All selected features have odds ratios below 1, meaning an increase in any of these predictors decreases the chance of survival.

19. Multinomial Logistic Regression: Coefficients and Odds Ratios

Objective

This section analyzes the coefficients and odds ratios of a multinomial logistic regression model that predicts survival chance (low, medium, high) based on passenger features. Each class receives a separate set of coefficients, and the odds ratios show how feature changes affect the likelihood of being in each class.

Code

```
# Create coefficient DataFrame (transpose to get features in
    rows)
multi_coef_df = pd.DataFrame(
    model_multi.coef_.T,
    columns=['Class 0', 'Class 1', 'Class 2'],
    index=X.columns
)

# Compute odds ratios by exponentiating coefficients
multi_coef_df['Odds Ratio (Class 0)'] = np.exp(multi_coef_df[
    'Class 0'])
multi_coef_df['Odds Ratio (Class 1)'] = np.exp(multi_coef_df[
    'Class 1'])
multi_coef_df['Odds Ratio (Class 2)'] = np.exp(multi_coef_df[
    'Class 2'])
```

```
# Display the table
print("Multinomial Logistic Regression Coefficients and Odds
    Ratios:")
display(multi_coef_df)
```

Listing 19: Extracting coefficients and odds ratios from multinomial logistic regression

Explanation

- In multinomial logistic regression, each target class has its own set of coefficients.
- For each feature and class, the coefficient describes the change in log-odds of belonging to that class versus the reference.
- The corresponding odds ratio is calculated by exponentiating the coefficient: Odds Ratio $= e^{\text{coefficient}}$.

Coefficients and Odds Ratios

Table 13: Multinomial Logistic Regression Coefficients and Odds Ratios

Feature	Class 0	Class 1	Class 2	Odds Ratio (0)	Odds Ratio (1)	Odds Ratio (2)
Pclass	1.8507	-0.7365	-1.1143	6.3643	0.4788	0.3282
Age	-0.0444	-0.0648	0.1092	0.9565	0.9373	1.1154
SibSp	-1.1350	0.4780	0.6570	0.3214	1.6129	1.9289
Parch	-0.6374	0.5550	0.0825	0.5286	1.7419	1.0860
Fare	-6.4951	0.0589	6.4363	0.0015	1.0607	624.0621
Sex_male	0.3051	-0.0469	-0.2583	1.3568	0.9542	0.7724
Embarked_Q	0.1859	0.0025	-0.1884	1.2043	1.0025	0.8283
Embarked_S	0.0453	-0.1832	0.1378	1.0464	0.8326	1.1478

Interpretation

- **Fare** has an extremely large positive coefficient for Class 2 (high chance) and a strongly negative one for Class 0, indicating it is the dominant predictor of high survival likelihood.
- **Pclass** and **Sex_male** both decrease the odds of being in Class 1 or 2 compared to Class 0, consistent with earlier binary results.

- Positive odds ratios for **SibSp** and **Parch** in Class 2 indicate that family presence increases the likelihood of high survival chance.

This detailed breakdown shows how different variables influence survival classification across three fare-based categories.