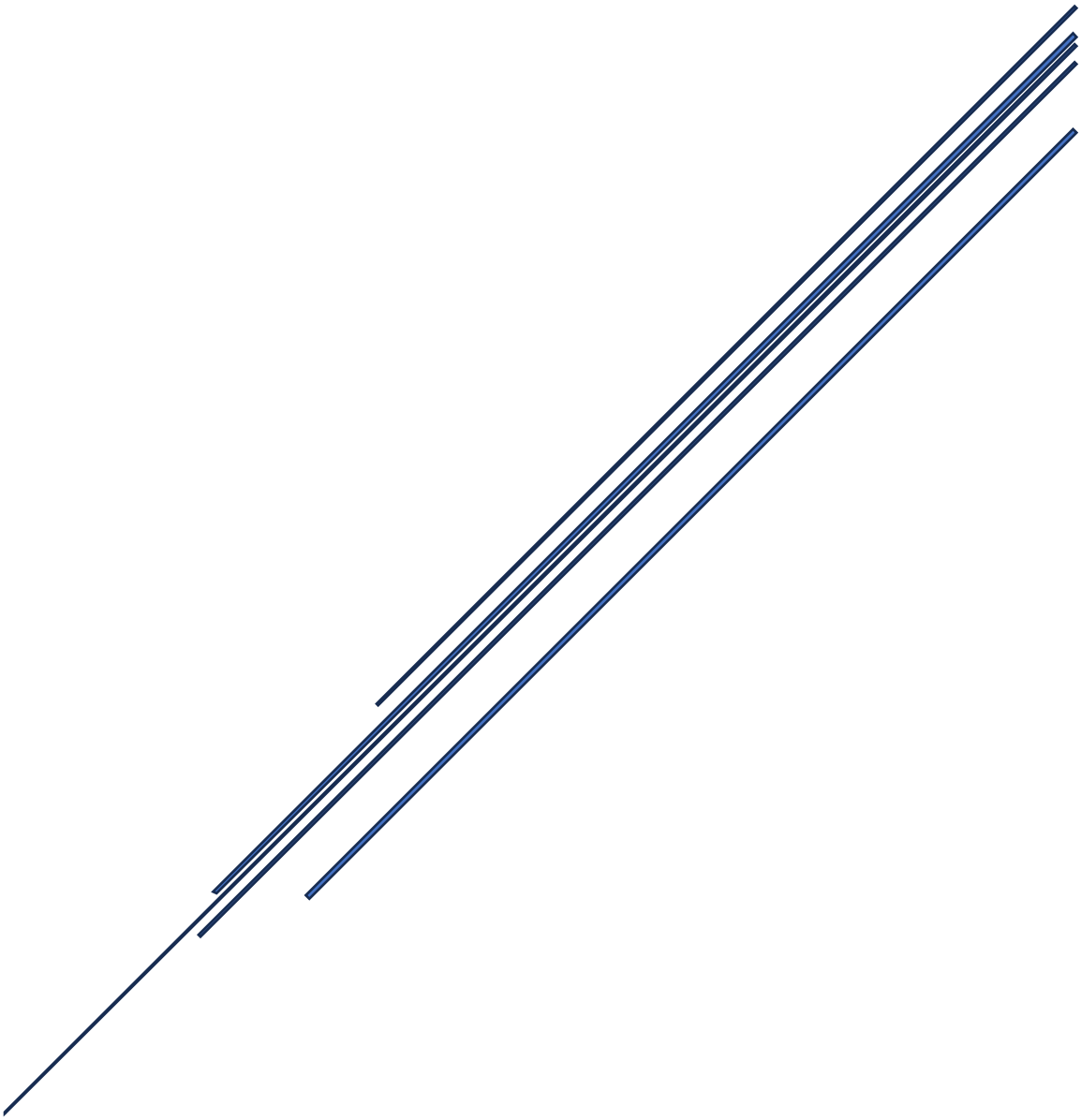
FEDERICO ARITON

Data cleaning and Preprocessing Report



Index

Data Cleaning and Preprocessing Report: Titanic Dataset

Introduction

This report outlines the steps we took to clean and preprocess the Titanic dataset. The goal was to prepare the data for further analysis by addressing missing values, correcting data types, and handling outliers.

Step 1: Understanding the Dataset

Before cleaning the data, we first needed to understand its structure and content.

We loaded the dataset and checked the basic information, including the number of rows, columns, and data types.

We identified that some columns had missing values, and some columns had encoded categorical data.

We noticed some columns, like Embarked, had numerical values (0.0, 1.0, 2.0) instead of the original categories (C, Q, S).

Step 2: Renaming and Correcting Columns

The objective is to ensure all column names are correct and understandable.

Actions:

We noticed that the Survived column was incorrectly labeled as 2urvived. We corrected it to Survived.

We also identified and removed irrelevant columns like those named zero, which seemed to contain no useful data.

Renaming the columns and removing

In [5]:
```
# Rename columns to more appropriate names
df.rename(columns={
    '2urvived': 'Survived',
    'Passengerid': 'PassengerId',
    # Drop irrelevant columns that seem to have no value
}, inplace=True)

```

In [6]:
```
# Dropping all columns that start with 'zero' as they seem to be placeholder
df.drop(columns=[col for col in df.columns if 'zero' in col], inplace=True)

# Verify the changes
df.columns
```

Out[6]:
```
Index(['PassengerId', 'Age', 'Fare', 'Sex', 'sibsp', 'Parch', 'Pclass',
       'Embarked', 'Survived'],
      dtype='object')
```

Figure 1 Code of renaming the columns and removing

Step 3: Handling Missing Values

The objective is to fill in missing values to avoid errors in analysis, we focused on the Embarked column, which had a couple of missing values.

Actions:

We examined the distribution of the Embarked values to determine how many passengers boarded at each port.

We determined that 0.0, 1.0, and 2.0 likely represented S (Southampton), C (Cherbourg), and Q (Queenstown) respectively.

Since S (Southampton) was the most common embarkation point, we filled the missing Embarked values with 0.0, which corresponds to Southampton.

```
2.0    914
0.0    270
1.0    123
Name: Embarked, dtype: int64
```

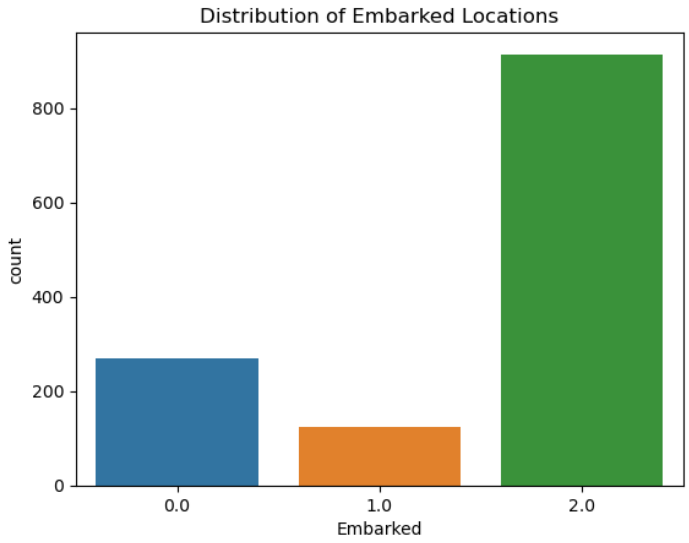


Figure 2 Histogram of the Distribution of Embarked Location

Step 4: Data Type Conversion

The objective is to ensure that each column has the correct data type to optimize storage and processing.

Actions:

We identified that the Pclass, Embarked, and Survived columns should be treated as categorical data because they represent discrete categories (e.g., ticket class, port of embarkation, and survival status).

We converted Pclass and Embarked from their original types to the category type.

We also converted Survived to category since it represents a binary outcome (0 for not survived, 1 for survived).

We verified the data types after conversion to ensure that Pclass, Embarked, and Survived columns were correctly set as categorical.

Data Type Conversion

```
In [15]:
df.columns
```

```
Out[15]: Index(['PassengerId', 'Age', 'Fare', 'Sex', 'sibsp', 'Parch', 'Pclass',
       'Embarked', 'Survived'],
      dtype='object')
```

```
In [16]:
# Convert 'Pclass' and 'Embarked' to categorical data types
df['Pclass'] = df['Pclass'].astype('category')
df['Embarked'] = df['Embarked'].astype('category')

# Convert 'Survived' to categorical
df['Survived'] = df['Survived'].astype('category')

# Verify the changes
df.dtypes

```

```
Out[16]: PassengerId       int64
Age             float64
Fare            float64
Sex               int64
sibsp             int64
Parch             int64
Pclass         category
Embarked       category
Survived       category
dtype: object
```

```
In [17]:
# Check for duplicate rows
duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# Drop duplicates if any
df.drop_duplicates(inplace=True)

```

```
Number of duplicate rows: 0
```

Figure 3 Code of Data Type Conversion

Step 5: Handling Outliers

The objective is to Identify and appropriately handle outliers to improve the accuracy of the analysis, we focused on the Fare and Age columns, as these are common areas where outliers occur.

Actions:

We used visualizations (boxplots and histograms) to see the distribution of values and identify potential outliers.

We applied the Interquartile Range (IQR) method to calculate the boundaries for what counts as an outlier.

IQR: Measures the middle 50% of the data to define typical values.

Outliers: Values that are much higher or lower than the rest of the data.

For Fare: We found that there were a few passengers who paid significantly more than others, which could skew our analysis if not handled.

For Age: We identified ages that were unusually high or low.

Decision: We discussed options for handling outliers, including removing them or capping them at the upper bound. We decided to cap the outliers, which means replacing them with the nearest acceptable value.

In [18]:
```python
# Example: Handling outliers in the 'Fare' column using the IQR method
Q1 = df['Fare'].quantile(0.25)
Q3 = df['Fare'].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
df = df[(df['Fare'] >= lower_bound) & (df['Fare'] <= upper_bound)]

# Display the shape of the dataset after removing outliers
print(df.shape)
```

```
(1138, 9)
```

Figure 4 Coding of handling outliers

Step 6: Finalizing the Cleaned Dataset

Save the cleaned dataset for future analysis.

Actions:

After making all the corrections and adjustments, we saved the cleaned dataset to a new file called cleaned_titanic_data.csv.

Conclusion

The data cleaning process involved carefully examining the data, making informed decisions on how to handle missing values, outliers, and other anomalies. By following these steps, we ensured that the dataset is now in a better condition for any further analysis or modeling.