Comprehensive Data Cleaning and Summary Statistics Analysis

Overview

Data cleaning is a crucial step in any data analysis project. It involves identifying and rectifying inaccuracies or inconsistencies in the data, which can significantly affect the outcomes of analysis and modeling. This project focuses on cleaning a dataset and performing summary statistics analysis using Python's Pandas and NumPy libraries. The dataset in question is assumed to be a CSV file, commonly used for storing tabular data. In this case, we will work with a dataset named `train.csv`.

Objectives of This Analysis

1. **Data Cleaning:** Prepare the dataset by addressing missing values, correcting data types, and handling outliers.
2. **Summary Statistics:** Compute key summary statistics for numerical columns, which provide insights into the data's characteristics.

Step 1: Load and Read the Data

```
python
Copy code
import pandas as pd
import numpy as np

data_path = r"D:\# DATA SCIENCE\# PROJECTS\- PROJECTS
INTERNSHIPS\TECHNOHACKS SOLUTIONS - DATA ANALYSIS\COMPREHENSIVE DATA
CLEANING AND SUMMARY STATISTICS ANALYSIS\Data\train.csv"

train_df = pd.read_csv(data_path)
```

In this first step, we import the necessary libraries: **Pandas** for data manipulation and **NumPy** for numerical operations. We define the `data_path`, which points to the location of the dataset file. The dataset is loaded into a Pandas DataFrame named `train_df` using the `pd.read_csv()` function. This function reads a comma-separated values (CSV) file into a DataFrame, which is the primary data structure in Pandas for data analysis.

Step 2: Initial Data Exploration

Step 2.1: Display Initial Data Information

```
python
Copy code
```

```
print("\nInitial Data Info:")
print(train_df.info())
```

Here, we use the `info()` method to display a concise summary of the DataFrame, including the number of entries, the data types of each column, and the memory usage. This information is essential for understanding the structure of the dataset and identifying potential issues.

Step 2.2: Check for Duplicates

```
python
Copy code
duplicates = train_df.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicates}")
if duplicates > 0:
    train_df.drop_duplicates(inplace=True)
    print("Duplicates removed.")
```

In this section, we check for duplicate rows using the `duplicated()` method, which returns a boolean Series indicating whether each row is a duplicate. The `sum()` function counts the number of duplicate rows. If any duplicates are found, the `drop_duplicates()` method is called to remove them, and we confirm this action by printing a message.

Step 2.3: Check and Correct Data Types

```
python
Copy code
print("\nData Types Before Correction:")
print(train_df.dtypes)

# Correcting data types
train_df['Survived'] = train_df['Survived'].astype('category')
train_df['Embarked'] = train_df['Embarked'].astype('category')
train_df['Age'] = train_df['Age'].astype(float)
train_df['Fare'] = train_df['Fare'].astype(float)

print("\nData Types After Correction:")
print(train_df.dtypes)
```

We examine the current data types of each column using the `dtypes` attribute. Correcting data types is crucial for ensuring accurate analysis and modeling. We convert the 'Survived' and 'Embarked' columns to categorical data types, which is efficient for storage and speeds up processing. We also ensure that 'Age' and 'Fare' are of float type, which is appropriate for numerical calculations. After correcting the data types, we print the updated types to confirm the changes.

Step 3: Display Initial Statistical Summary

```
python
Copy code
print("\nInitial Statistical Summary:")
print(train_df.describe(include='all'))
```

The `describe()` method provides a statistical summary of the DataFrame. By setting `include='all'`, we obtain descriptive statistics for both numerical and categorical columns. This summary includes metrics such as count, mean, median, standard deviation, and quantiles for numerical columns, and frequency counts for categorical columns.

Step 4: Check for Missing Values

```
python
Copy code
missing_values = train_df.isnull().sum()
print("\nMissing Values Before Cleaning:")
print(missing_values[missing_values > 0])
```

In this step, we identify missing values in the dataset using the `isnull()` method, which returns a DataFrame of the same shape with Boolean values indicating the presence of nulls. We then use `sum()` to count the missing values for each column and print the results, focusing only on those with missing data.

Step 5: Data Cleaning - Remove Missing 'Age' Values

```
python
Copy code
train_df = train_df.dropna(subset=['Age'])
```

We address missing values in the 'Age' column by removing rows where 'Age' is null. The `dropna()` method with the `subset` parameter allows us to specify the column to check for missing values. This decision may depend on the analysis requirements; for instance, 'Age' is often a critical variable in survival analysis.

Step 6: Fill Missing 'Fare' Values with the Median

```
python
Copy code
train_df['Fare'].fillna(train_df['Fare'].median(), inplace=True)
```

For the 'Fare' column, we choose to fill missing values with the median fare. Using the median is appropriate here, as it is less affected by outliers compared to the mean. The `fillna()` method updates the DataFrame in place by replacing nulls with the calculated median.

Step 7: Handle Missing Values for 'Embarked' and 'Cabin'

```
python
Copy code
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
train_df['Cabin'].fillna('Unknown', inplace=True)
```

For the 'Embarked' column, we fill missing values with the mode (the most frequently occurring value) using the `mode()` method. For the 'Cabin' column, we choose to fill missing values with 'Unknown'. Depending on the analysis goals, we could also opt to drop the 'Cabin' column altogether if it has too many missing values.

Step 8: Check for Missing Values After Cleaning

```
python
Copy code
missing_values_after = train_df.isnull().sum()
print("\nMissing Values After Cleaning:")
print(missing_values_after[missing_values_after > 0])
```

After completing the cleaning process, we check for any remaining missing values to ensure that our efforts have successfully addressed the issues. This verification step is crucial for confirming the integrity of the cleaned dataset.

Step 9: Check for Outliers Before Handling (Age)

```
python
Copy code
z_scores_before = np.abs((train_df['Age'] - train_df['Age'].mean()) /
train_df['Age'].std())
outliers_before_age = train_df[z_scores_before >= 3]
print("\nOutliers for 'Age' Before Handling:")
print(outliers_before_age)
```

We use the Z-score method to identify outliers in the 'Age' column. The Z-score measures the number of standard deviations a data point is from the mean. We consider a Z-score of 3 or more (or -3 or less) as an outlier. This step helps us understand the distribution of age values and identify extreme cases that may need special attention.

Step 10: Check for Outliers Before Handling (Fare)

```
python
Copy code
Q1_before = train_df['Fare'].quantile(0.25)
```

```
Q3_before = train_df['Fare'].quantile(0.75)
IQR_before = Q3_before - Q1_before
outliers_before_fare = train_df[(train_df['Fare'] < (Q1_before - 1.5 *
IQR_before)) | (train_df['Fare'] > (Q3_before + 1.5 * IQR_before))]
print("\nOutliers for 'Fare' Before Handling:")
print(outliers_before_fare)
```

We employ the Interquartile Range (IQR) method to detect outliers in the 'Fare' column. The IQR is the difference between the first (Q1) and third (Q3) quartiles. We identify outliers as values that fall below Q1−1.5×IQRQ1 - 1.5 \times IQRQ1−1.5×IQR or above Q3+1.5×IQRQ3 + 1.5 \times IQRQ3+1.5×IQR. This approach effectively captures extreme values that could skew the analysis.

Step 11: Remove Outliers Using Z-Score Method (Age)

```
python
Copy code
z_scores = np.abs((train_df['Age'] - train_df['Age'].mean()) /
train_df['Age'].std())
train_df = train_df[z_scores < 3]

# Check for Outliers After Handling (Age)
z_scores_after = np.abs((train_df['Age'] - train_df['Age'].mean()) /
train_df['Age'].std())
outliers_after_age = train_df[z_scores_after >= 3]

print("\nOutliers for 'Age' After Handling:")
print(outliers_after_age)
```

After identifying outliers in the 'Age' column, we remove them from the DataFrame by applying the Z-score threshold. We then recheck for outliers to confirm that our outlier removal process was effective. This step is vital for ensuring the robustness of subsequent analyses.

Step 12: Remove Outliers Using IQR Method (Fare)

```
python
Copy code
Q1 = train_df['Fare'].quantile(0.25)
Q3 = train_df['Fare'].quantile(0.75)
IQR = Q3 - Q1
train_df = train_df[(train_df['Fare'] >= (Q1 - 1.5 * IQR)) &
(train_df['Fare'] <= (Q3 + 1.5 * IQR))]

# Check for Outliers After Handling (Fare)
Q1_after = train_df['Fare'].quantile(0.25)
Q3_after = train_df['Fare'].quantile(0.75)
IQR_after = Q3_after - Q1_after
outliers_after_fare = train_df[(train_df['Fare'] < (Q1_after - 1.5 *
IQR_after)) | (train_df['Fare'] > (Q3_after + 1.5 * IQR_after))]
```

```
print("\nOutliers for 'Fare' After Handling:")
print(outliers_after_fare)
```

Similarly, we remove outliers from the 'Fare' column using the IQR method. After applying the filtering, we check again for outliers to ensure our approach has worked correctly. This comprehensive outlier handling ensures that our dataset is ready for accurate analysis.

Step 13: Final Statistical Summary

```
python
Copy code
print("\nFinal Statistical Summary:")
print(train_df.describe(include='all'))
```

Finally, we provide an updated statistical summary of the cleaned DataFrame. This summary reflects the adjustments made during the data cleaning process and allows us to analyze the impact of those changes on the dataset's statistics.

Conclusion

This project demonstrated a comprehensive approach to data cleaning and summary statistics analysis using Python. By following these steps, you can ensure that your dataset is well-prepared for further analysis, modeling, and decision-making. Data cleaning is an iterative process, and the methods applied here can be adapted to various datasets and analysis contexts.