01_cleaning_walkthrough

June 24, 2025

[1]:
```python
# Step 1: Import the libraries we'll use
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler

# This makes sure we can see all the columns
pd.set_option('display.max_columns', None)

# Step 2: Load the Titanic CSV from the data folder
df = pd.read_csv('../data/Titanic-Dataset.csv')  # if your file was named train.
 ↪csv, use that instead
df.head()
```

```
Matplotlib is building the font cache; this may take a moment.
```

[1]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[2]:

```
# Step 3: Let's check for missing values and data types
print("  Data Types:\n", df.dtypes)
print("\n  Missing values in each column:")
print(df.isnull().sum())
```

```
  Data Types:
 PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object

  Missing values in each column:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

[3]:

```
# Step 4: Fixing missing values

# 1. Fill missing Age with the median
df['Age'].fillna(df['Age'].median(), inplace=True)

# 2. Fill missing Embarked with the mode (most frequent)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# 3. Drop the Cabin column entirely
df.drop(columns='Cabin', inplace=True)

# Check again to confirm all missing values are gone
df.isnull().sum()
```

```
C:\Users\dutta\AppData\Local\Temp\ipykernel_8556\3291964984.py:4: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Age'].fillna(df['Age'].median(), inplace=True)
C:\Users\dutta\AppData\Local\Temp\ipykernel_8556\3291964984.py:7: FutureWarning:
A value is trying to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
```

```
[3]: PassengerId    0
     Survived       0
     Pclass         0
     Name           0
     Sex            0
     Age            0
     SibSp          0
     Parch          0
     Ticket         0
     Fare           0
     Embarked       0
     dtype: int64
```

```
[4]: # Step 5: Convert text columns into numbers

     from sklearn.preprocessing import LabelEncoder

     # Create encoder
     le = LabelEncoder()
```

```
# Encode 'Sex' and 'Embarked'
df['Sex'] = le.fit_transform(df['Sex'])
df['Embarked'] = le.fit_transform(df['Embarked'])

# Drop 'Name' and 'Ticket' columns
df.drop(columns=['Name', 'Ticket'], inplace=True)

# Display first few rows to see changes
df.head()
```

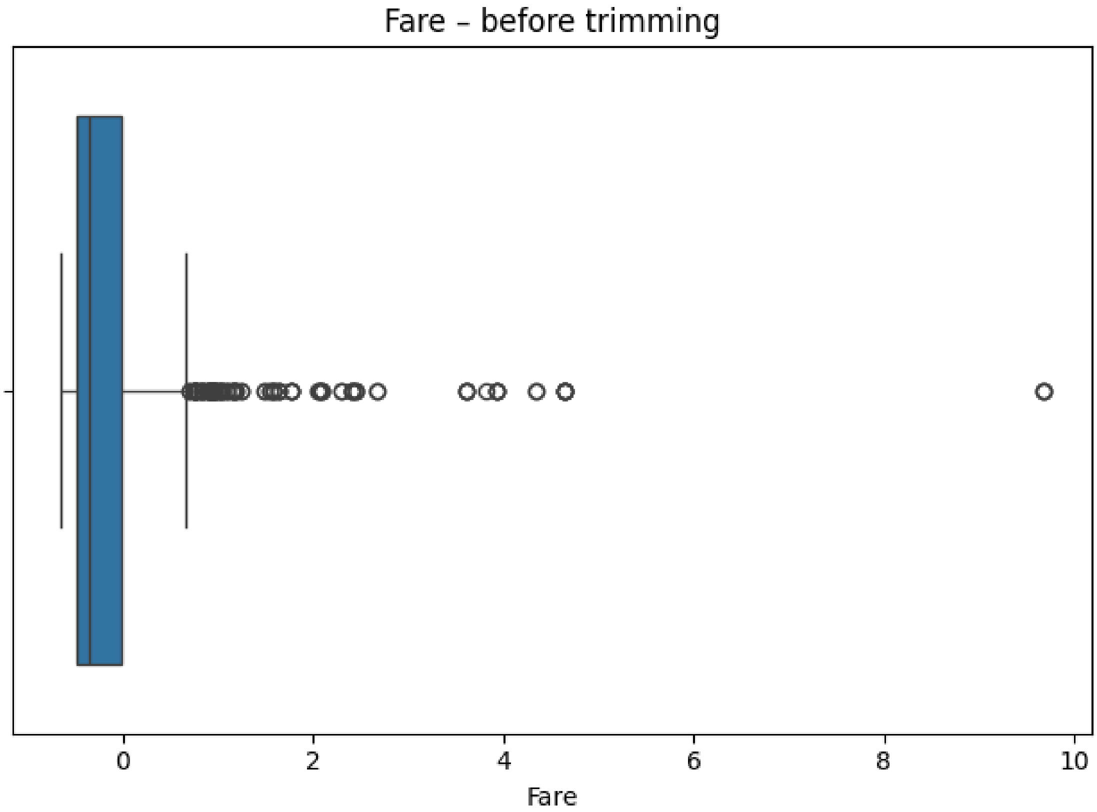
```
[4]:    PassengerId  Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
     0            1         0       3    1  22.0      1      0   7.2500         2
     1            2         1       1    0  38.0      1      0  71.2833         0
     2            3         1       3    0  26.0      0      0   7.9250         2
     3            4         1       1    0  35.0      1      0  53.1000         2
     4            5         0       3    1  35.0      0      0   8.0500         2
```

```
[5]: # Step 6: Standardize Age and Fare so they're on the same scale
     from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()

     # Fit the scaler on Age and Fare, then transform them
     df[['Age', 'Fare']] = scaler.fit_transform(df[['Age', 'Fare']])

     # Quick sanity-check: the new mean should be ~0 and std ~1
     df[['Age', 'Fare']].describe().loc[['mean', 'std']]
```

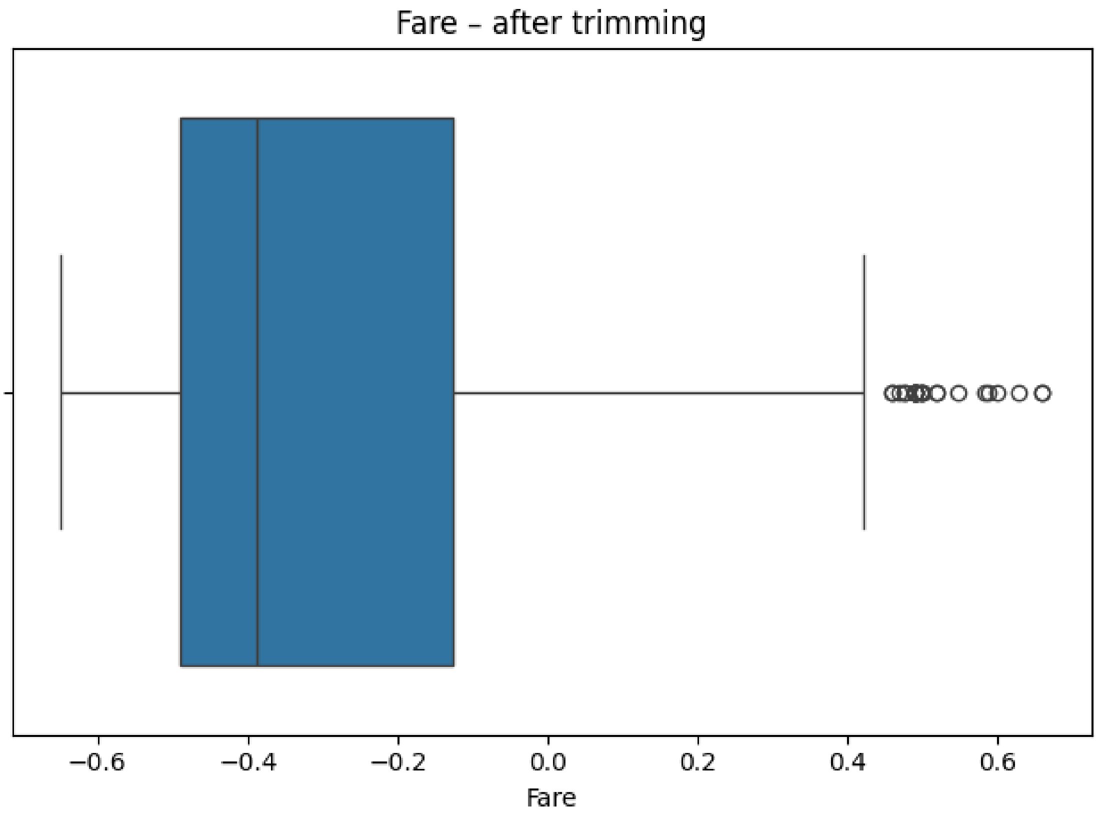
```
[5]:                Age          Fare
     mean  2.272780e-16  3.987333e-18
     std   1.000562e+00  1.000562e+00
```

```
[6]: # Step 7: Visualise and trim Fare outliers

     import seaborn as sns
     import matplotlib.pyplot as plt
     from pathlib import Path

     # 1  Boxplot BEFORE trimming
     sns.boxplot(x=df['Fare']).set_title('Fare – before trimming')
     plt.tight_layout()

     # Save the picture to images/ so you can show it in GitHub
     Path('../images').mkdir(exist_ok=True)
     plt.savefig('../images/fare_boxplot_before.png')
     plt.show()
```

```
# 2  IQR method to define "too high" or "too low"
q1, q3 = df['Fare'].quantile([0.25, 0.75])
iqr = q3 - q1
lower, upper = q1 - 1.5 * iqr, q3 + 1.5 * iqr

# 3  Keep only rows within the bounds
before = len(df)
df = df[(df['Fare'] >= lower) & (df['Fare'] <= upper)]
after = len(df)

print(f"Removed {before - after} outlier rows. Rows left: {after}")

# 4  Boxplot AFTER trimming (quick check)
sns.boxplot(x=df['Fare']).set_title('Fare - after trimming')
plt.tight_layout()
plt.savefig('../images/fare_boxplot_after.png')
plt.show()
```



```
Removed 116 outlier rows. Rows left: 775
```



[7]:

```
# Step 8: Save the final cleaned dataset
df.to_csv('../data/titanic_cleaned.csv', index=False)
print(f"Cleaned data written to data/titanic_cleaned.csv    (rows: {len(df)})")
```

```
Cleaned data written to data/titanic_cleaned.csv    (rows: 775)
```

[ ]: