DATA CLEANING

11 October 2023 20:21

Data Cleaning or cleansing also known as Data Preprocessing is the process of detecting and correcting(or removing) corrupt or inaccurate records from record set, table or database and referers to identifying incomplete incorrect inaccurate or irrelevant parts of the data and then replacing , modifying or deleting the dirty or coarse data

Data cleaning is essential because raw data is often noisy, incomplete, and inconsistent, **which can negatively impact the accuracy and reliability of the insights derived from it.**

1. Data inspection and exploration:

This step involves understanding the data by inspecting its structure and identifying missing values, outliers, and inconsistencies.

- Check the duplicate rows.
- Python3

```
df.duplicated()
```

Output:

```
0      False
1      False
       ...
889    False
890    False
Length: 891, dtype: bool
```

- ## How to Handle Duplicate Values.?

Handling duplicate values in a dataset typically involves various techniques, depending on the specific context and requirements of the data analysis. Here are some common approaches to handle duplicate values:

1. **Identify Duplicates:** Begin by identifying the duplicate values in the dataset to understand the extent of the duplication and the specific fields or columns affected.
2. **Remove Duplicates:** If the duplicates are unnecessary and do not provide valuable information, consider removing them from the dataset. This can be done using the appropriate function or method in your chosen programming language or software.
3. **Aggregate Duplicates:** In some cases, duplicates may represent valid data that needs to be aggregated. This could involve calculating summary statistics, such as means, medians, or modes, for the duplicated values.
4. **Flag Duplicates:** You can create an additional variable or column to indicate the presence of duplicates, allowing you to maintain the original data while being aware of the duplicated entries.

- Check the data information using df.info()

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

From the above data info, we can see that Age and Cabin have an unequal number of counts. And some of the columns are categorical and have data type objects and some are integer and float values.

Check the categorical and numerical columns

- Python3

```
# Categorical columns
cat_col = [col for col in df.columns if df[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in df.columns if df[col].dtype != 'object']
print('Numerical columns :',num_col)
```

Check the total number of unique values in the Categorical column

- `df[cat_col].nunique()`

Output:

```
Name        891
Sex           2
Ticket      681
Cabin       147
Embarked      3
dtype: int64
```

2. Removal of unwanted observations

This includes deleting duplicate/ redundant or irrelevant values from your dataset. Duplicate observations most frequently arise during data collection and Irrelevant observations are those that don't actually fit the specific problem that you're trying to solve.

- Redundant observations alter the efficiency to a great extent as the data repeats and may add towards the correct side or towards the incorrect side, thereby producing unfaithful results.
- Irrelevant observations are any type of data that is of no use to us and can be removed directly.

3. Handling missing data:

Missing data is a common issue in real-world datasets, and it can occur due to various reasons such as human errors, system failures, or data collection issues. Various techniques can be used to handle missing data, such as imputation, deletion, or substitution.

Let’s check the % missing values columns-wise for each row using df.isnull() it checks whether the values are null or not and gives returns boolean values. and .sum() will sum the total number of null values rows and we divide it by the total number of rows present in the dataset then we multiply to get values in % i.e per 100 values how much values are null.

HOW TO HANDLE MISSING VALUES.?

Handling missing values in a dataset is crucial to ensure accurate and reliable data analysis. Here are some common strategies for handling missing values:

1. Identify Missing Values: Begin by identifying the missing values in the dataset and understanding the extent of the missing data across different variables.

2. Remove Missing Values: If the amount of missing data is relatively small and randomly distributed, removing the rows or columns with missing values can be an effective approach. However, this should be done cautiously, as it may lead to a loss of valuable information.

3. Imputation: Imputation involves filling in the missing values with estimated or predicted values. Common imputation techniques include replacing missing values with the mean, median, mode, or a constant value, or using more model imputation methods such as regression imputation or k-nearest neighbors (KNN) imputation.

MODEL IMPUTATION:
Ex: KNN

K-nearest neighbors (KNN) imputation is a method used to handle missing values in a dataset by estimating the missing data points based on the values of their nearest neighbours. **This technique is particularly useful for imputing missing numerical data in a multivariate dataset.** Here's how the KNN imputation method works:

1. **Identify Nearest Neighbors**: For each observation with missing values, the KNN algorithm identifies its k nearest neighbors based on the **available features or variables**. The distance between observations is typically calculated using metrics such as **Euclidean distance or Manhattan distance.**

2. **Calculate Imputed Value:** Once the nearest neighbors are identified, the missing values are imputed using the average (or weighted average) of the values of the corresponding feature from the neighboring observations.

3. **Weighting:** In some variations of the KNN imputation, **the contribution of each neighbor to the imputed value is weighted based on the proximity of the neighbor. This means that closer neighbors have a greater influence on the imputed value than those farther away.**

4. **Parameter Selection:** The choice of the value for k, the number of nearest neighbors, is crucial and can impact the imputation results. A small value of k may lead to high variance, while a large value of k may lead to high bias.

KNN imputation is a useful technique when the missing values occur randomly and the dataset has a sufficient number of observations. However, it is important to be cautious when applying KNN imputation, as it can be computationally intensive, especially for large datasets. Additionally, the choice of the distance metric and the value of k should be carefully considered to ensure the accuracy and reliability of the imputed values.

SEGMENTATION

Segmentation in the context of handling missing values involves creating distinct segments or categories to distinguish data points with missing values from those without missing values. This approach allows for the separate treatment of missing values, enabling more targeted and effective strategies for handling them. Here's an example of segmentation in the context of missing values:
Suppose you have a dataset that includes information about customer transactions, including customer age, income, and transaction amount. Some entries have missing values for the income variable. To handle these missing values using segmentation, you could create a new category specifically for the missing income

values. This might involve:

1. Creating a New Category: You could create a new category or segment labeled "Missing" within the income variable to represent entries with missing income values.

2. Separate Treatment: With the "Missing" segment, you can apply specific imputation techniques, such as mean imputation or KNN imputation, to only the entries in this segment, rather than the entire dataset.

3. Distinct Analysis: You can perform separate analyses for the "Missing" segment to understand the impact of missing values on other variables or to explore potential correlations or patterns associated with the missing income data.
By segmenting the data in this manner, you can handle missing values more effectively and gain insights into the potential reasons behind the missing data. This approach allows for a more nuanced and tailored analysis that takes into account the distinct characteristics of the missing values within the dataset.

```
import pandas as pd
import numpy as np

# Creating a sample dataset with missing values
data = {'Age': [30, 40, 25, np.nan, 35, np.nan, 45, 50],
        'Income': [50000, np.nan, 40000, np.nan, 60000, np.nan, 80000, 75000],
        'Transaction': [100, 200, 150, 120, 180, 220, 250, 300]}

df = pd.DataFrame(data)

# Segmentation for missing values
df['Income_Category'] = np.where(df['Income'].isnull(), 'Missing', 'Available')

# Imputation for missing values
mean_income = df[df['Income_Category'] == 'Available']['Income'].mean()
df['Income'] = df['Income'].fillna(mean_income)

# Displaying the DataFrame
print(df)
```

- Python3

```
round((df1.isnull().sum()/df1.shape[0])*100,2)
```

Output:

```
PassengerId     0.00
Survived        0.00
Pclass          0.00
Sex             0.00
Age            19.87
SibSp           0.00
Parch           0.00
Fare            0.00
Cabin          77.10
Embarked        0.22
dtype: float64
```

We cannot just ignore or remove the missing observation. They must be handled carefully as they can be an indication of something important.

The two most common ways to deal with missing data are:

- Dropping observations with missing values.
- The fact that the value was missing may be informative in itself.
- Plus, in the real world, you often need to make predictions on new data even if some of the features are missing!
  As we can see from the above result that Cabin has 77% null values and Age has 19.87% and Embarked has 0.22% of null values. So, it's not a good idea to fill 77% of null values. So, we will drop the Cabin column. Embarked column has only 0.22% of null values so, we drop the null values rows of Embarked column.

- Python3

```
df2 = df1.drop(columns='Cabin')
df2.dropna(subset=['Embarked'], axis=0, inplace=True)
df2.shape
```

Output:

```
(889, 9)
```

- Imputing the missing values from past observations.
- Again, "missingness" is almost always informative in itself, and you should tell your algorithm if a value was missing.
- Even if you build a model to impute your values, you're not adding any real information. You're just reinforcing the patterns already provided by other features.
  From the above describe table, we can see that there are very less differences between the mean and median i..e 29.6 and 28. So, here we can do any one from mean imputation or Median imputations.

  Note:

- Mean imputation is suitable when the data is normally distributed and has no extreme outliers.
- Median imputation is preferable when the data contains outliers or is skewed.
- Python3

```
# Mean imputation
df3 = df2.fillna(df2.Age.mean())
# Let's check the null values again
df3.isnull().sum()
```

Output:

```
PassengerId    0
Survived       0
Pclass         0
Sex            0
Age            0
SibSp          0
Parch          0
Fare           0
Embarked       0
dtype: int64
```

INTERPOLATION

Interpolation is a mathematical technique used to estimate unknown or intermediate values between two known values. It involves constructing a function or curve that closely approximates the relationship between data points, allowing for the prediction of values within the range of known data points.

Linear and polynomial interpolation are two common techniques used to estimate unknown values between known data points in a dataset. Here's how each method works:

1. **Linear Interpolation:**
   - Linear interpolation assumes a linear relationship between two known data points and uses this line to estimate unknown values between them.
   - Given two data points (x1, y1) and (x2, y2), where x1 and x2 are known data points and y1 and y2 are the corresponding values, linear interpolation estimates the value y for a new point x, where x1 < x < x2.

- The formula for linear interpolation is:
  $y=y1+(x2-x1)(y2-y1)*(x-x1)$.
- Linear interpolation assumes a straight-line relationship between data points and is simple to implement but may not accurately capture more complex relationships in the data.

2. **Polynomial Interpolation:**
   - Polynomial interpolation **fits a polynomial function to the known data points and uses this function** to estimate the values between data points.
   - The degree of the polynomial used for interpolation depends on the number of known data points and the complexity of the data.
   - The formula for a polynomial interpolation function of degree n is:
     $P(x)=a0+a1*x+a2*x2+...+an*xn$.
   - Polynomial interpolation can provide a more accurate estimation compared to linear interpolation, especially for datasets with more complex relationships.
   - However, high-degree polynomials can lead to overfitting and may not generalize well to new data.

   Both linear and polynomial interpolation methods are valuable tools for estimating unknown values within a dataset. The choice between the two methods depends on the nature of the data and the desired level of accuracy and complexity in the estimation.

3. **Spline Interpolation**: Spline interpolation divides the dataset into smaller segments and fits separate polynomial functions to each segment. This method can provide a smoother and more continuous estimation of the unknown values.

WHERE TO APPLY LINEAR , POLYNOMIAL AND SPLINE IMPUTATION.?

1. Linear Interpolation:
   - Linear interpolation is typically applied in situations where the relationship between data points is assumed to be linear. It is commonly used in engineering, physics, and finance for tasks such as trajectory estimation, simple curve fitting, and time series analysis.
   - Linear interpolation is also useful in creating simple visualizations and in filling in missing data points in a linear context.

     You can use the interp1d function from the scipy.interpolate module for linear interpolation.
     From scipy.interpolate importinterp1d

2. Polynomial Interpolation:
   - Polynomial interpolation is used when the relationship between data points is more complex and can be better represented by a polynomial function. It is commonly used in fields such as numerical analysis, signal processing, and data modelling for tasks such as curve fitting, data approximation, and function approximation.
   - Polynomial interpolation allows for a more flexible representation of the data, capturing non-linear relationships and complex patterns more accurately than linear interpolation.

     You can use the polyfit and poly1d functions from the numpy library for polynomial interpolation.
     np.polyfit(x,y,degree=2)

   Both linear and polynomial interpolation techniques are valuable tools for estimating values between data points, and the choice between the two depends on the underlying data relationships and the desired level of complexity in the estimation. It is essential to carefully consider the characteristics of the data and the specific requirements of the analysis when choosing the appropriate interpolation method.

3. **Spline interpolation,**
   Spline interpolation is particularly useful in situations where the data exhibits complex, non-linear relationships and requires a more flexible and smooth estimation method. Here are some specific scenarios where spline interpolation is commonly applied:

1. **Computer Graphics and Animation**: Spline interpolation is widely used in computer graphics and animation for tasks such as creating smooth curves and generating realistic motion paths. It allows for the creation of natural and visually appealing animations by ensuring smooth transitions between keyframes.

2. **Geographic Information Systems (GIS):** Spline interpolation is applied in GIS for tasks such as terrain modeling, surface reconstruction, and contour mapping. It helps in creating continuous and smooth surfaces

from irregularly spaced data points, enabling the visualization and analysis of geographic features.

3. **Biomedical Imaging**: Spline interpolation finds application in biomedical imaging for tasks such as image reconstruction and interpolation of medical images. It aids in creating high-resolution and continuous images from sparse data points, facilitating accurate visualization and analysis of biomedical data.

4. You can use the UnivariateSpline function from the scipy.interpolate module for spline interpolation.
```
fromscipy.interpolate importUnivariateSpline
UnivariateSpline(x_known, y_known, k=3)
```

4. Handling outliers:

Outliers are extreme values that deviate significantly from the majority of the data. They can negatively impact the analysis and model performance. Techniques such as clustering, interpolation, or transformation can be used to handle outliers.

BOXPLOT(FOR CAPPING):

Capping of outliers, also known as winsorizing, is a method used to handle extreme values or outliers in a dataset by setting a threshold or limit beyond which values are capped or truncated

To check the outliers, We generally use a box plot. A box plot, also referred to as a box-and-whisker plot, is a graphical representation of a dataset's distribution. It shows a variable's median, quartiles, and potential outliers. The line inside the box denotes the median, while the box itself denotes the interquartile range (IQR). The whiskers extend to the most extreme non-outlier values within 1.5 times the IQR. Individual points beyond the whiskers are considered potential outliers. A box plot offers an easy-to-understand overview of the range of the data and makes it possible to identify outliers or skewness in the distribution.

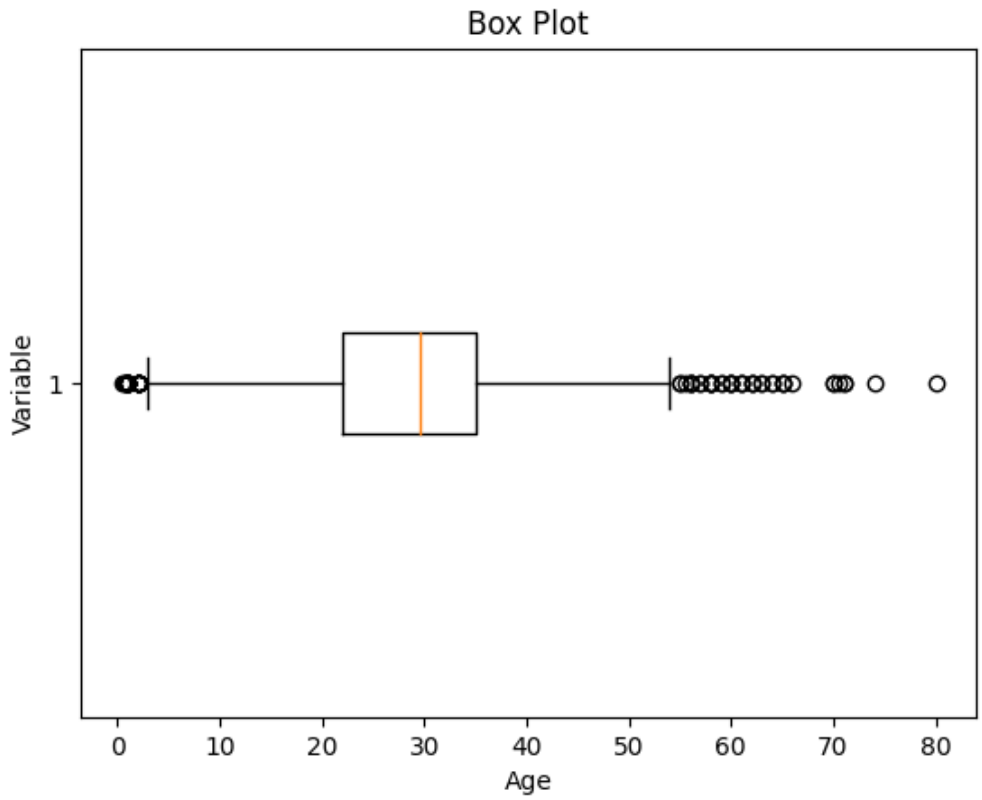
Let's plot the box plot for Age column data.

- Python3

```
import matplotlib.pyplot as plt

plt.boxplot(df3['Age'], vert=False)
plt.ylabel('Variable')
plt.xlabel('Age')
plt.title('Box Plot')
plt.show()
```

Output:



Box Plot

As we can see from the above Box and whisker plot, Our age dataset has outliers values. The values less than 5 and more 55 are outliers.

- Python3

```
# calculate summary statistics
mean = df3['Age'].mean()
std  = df3['Age'].std()

# Calculate the lower and upper bounds
lower_bound = mean - std*2
upper_bound = mean + std*2

print('Lower Bound :',lower_bound)
print('Upper Bound :',upper_bound)

# Drop the outliers
df4 = df3[(df3['Age'] >= lower_bound)
                & (df3['Age'] <= upper_bound)]
```

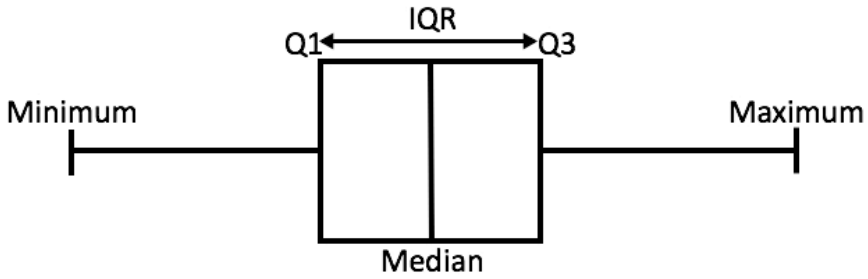
Output:

```
Lower Bound : 3.705400107925648
Upper Bound : 55.578785285332785
```

Similarly, we can remove the outliers of the remaining columns.

IQR(IQR for transformation):

IQR stands for Interquartile Range, which is a measure of statistical spread. It is used to describe the variability in a dataset by focusing on the spread of the middle 50% of the data. The IQR is particularly useful in identifying the spread of data points while being robust to outliers, unlike the range or standard deviation.



- *minimum* is the minimum value in the dataset,
- and *maximum* is the maximum value in the dataset.

  So the difference between the two tells us about the range of dataset.
- The *median* is the median (or centre point), also called second quartile, of the data (resulting from the fact that the data is ordered).
- *Q1* is the first quartile of the data, i.e., to say 25% of the data lies between *minimum* and *Q1*.
- *Q3* is the third quartile of the data, i.e., to say 75% of the data lies between *minimum* and *Q3*.

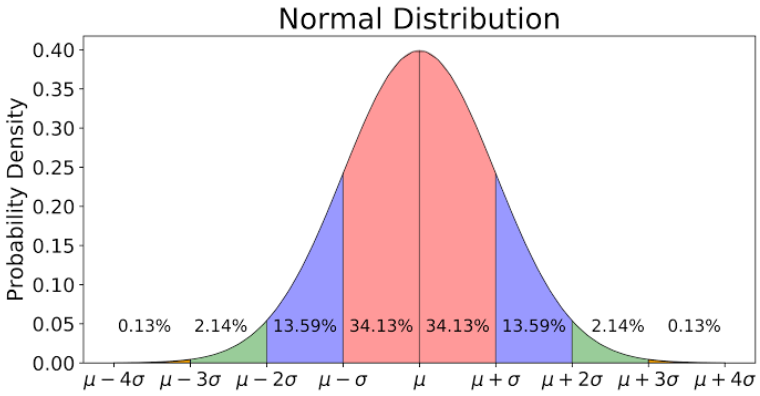
  The difference between *Q3* and *Q1* is called the **Inter-Quartile Range** or **IQR**.
  IQR = Q3 - Q1

  To detect the outliers using this method, we define a new range, let's call it decision range, and any data point lying outside this range is considered as outlier and is accordingly dealt with. The range is as given below:
  Lower Bound: **(Q1 - 1.5 * IQR)**Upper Bound: **(Q3 + 1.5 * IQR)**

  Any data point less than the *Lower Bound or* more than the *Upper Bound* is considered as an outlier.

WHY DO WE TAKE ONLY 1.5 TIMES THE IQR?

For example, let's say our data follows, our beloved, Gaussian Distribution



- About 68.26% of the whole data lies within one standard deviation (**<σ**) of the mean (μ), taking both sides into account, the pink region in the figure
- About 95.44% of the whole data lies within two standard deviations (**2σ**) of the mean (μ), taking both sides into account, the pink+blue region in the figure.
- About 99.72% of the whole data lies within three standard deviations (**<3σ**) of the mean (μ), taking both sides into account, the pink+blue+green region in the figure.

- And the rest 0.28% of the whole data lies outside three standard deviations (**>3σ**) of the mean (μ), taking both sides into account, the little red region in the figure. **And this part of the data is considered as outliers**.
- The first and the third quartiles, *Q1* and *Q3*, lies at **-0.675σ** and **+0.675σ** from the mean, respectively.

Let's calculate the IQR decision range in terms of σ
Taking scale = 1:

Lower Bound:
= Q1 - 1 * IQR
= Q1 - 1 * (Q3 - Q1)
= -0.675σ - 1 * (0.675 - [-0.675])σ
= -0.675σ - 1 * 1.35σ
= -2.025σ
Upper Bound:
= Q3 + 1 * IQR
= Q3 + 1 * (Q3 - Q1)
= 0.675σ + 1 * (0.675 - [-0.675])σ
= 0.675σ + 1 * 1.35σ
= 2.025σ

So, when scale is taken as 1, then according to IQR Method any data which lies beyond 2.025σ from the mean (μ), on either side, shall be considered as outlier. But as we know, upto 3σ, on either side of the μ ,the data is useful. So we cannot take scale = 1, because this makes the decision range too exclusive, means this results in too much outliers. In other words, the decision range gets so small (compared to 3σ) that it considers some data points as outliers, which is not desirable.

Taking scale = 2:
Lower Bound:

= Q1 - 2 * IQR
= Q1 - 2 * (Q3 - Q1)
= -0.675σ - 2 * (0.675 - [-0.675])σ
= -0.675σ - 2 * 1.35σ
= -3.375σ

Upper Bound:
= Q3 + 2 * IQR
= Q3 + 2 * (Q3 - Q1)
= 0.675σ + 2 * (0.675 - [-0.675])σ
= 0.675σ + 2 * 1.35σ
= 3.375σ

So, when scale is taken as 2, then according to IQR Method any data which lies beyond 3.375σ from the mean (μ), on either side, shall be considered as outlier. But as we know, upto 3σ, on either side of the μ ,the data is useful. So we cannot take scale = 2, because this makes the decision range too inclusive, means this results in too few outliers. In other words, the decision range gets so big (compared to 3σ) that it considers some outliers as data points, which is not desirable either.

Taking scale = 1.5:
Lower Bound:

= Q1 - 1.5 * IQR
= Q1 - 1.5 * (Q3 - Q1)
= -0.675σ - 1.5 * (0.675 - [-0.675])σ
= -0.675σ - 1.5 * 1.35σ
= -2.7σ

Upper Bound:
= Q3 + 1.5 * IQR
= Q3 + 1.5 * (Q3 - Q1)
= 0.675σ + 1.5 * (0.675 - [-0.675])σ
= 0.675σ + 1.5 * 1.35σ
= 2.7σ

When scale is taken as 1.5, then according to IQR Method any data which lies beyond 2.7σ from the mean (μ), on either side, shall be considered as outlier. And this decision range is the closest to what Gaussian Distribution tells us, i.e., 3σ. In other words, this makes the decision rule closest to what Gaussian Distribution considers for outlier detection, and this is exactly what we wanted.

*** Side Note ***

To get exactly 3σ, we need to take the scale = 1.7, but then 1.5 is more "symmetrical" than 1.7 and we've always been a little more inclined towards symmetry, aren't we!?

Also, IQR Method of Outlier Detection is not the only and definitely not the best method for outlier detection, so a bit trade-off is legible and accepted.

Zscore(Removal of Outlier):

Z-score method is another method for detecting outliers. This method is generally used when a variable' distribution looks close to Gaussian. Z-score is the number of standard deviations a value of a variable is away from the variable' mean.

Z-Score = (X-mean) / Standard deviation

when the values of a variable are converted to Z-scores, then the distribution of the variable is called standard normal distribution with mean=0 and standard deviation=1. The Z-score method requires a cut-off specified by the user, to identify outliers. The widely used lower end cut-off is -3 and the upper end cut-off is +3. The reason behind using these cut-offs is, 99.7% of the values lie between -3 and +3 in a standard normal distribution.

When do we use IQR boxplot and Zscore for outlier detection?

The choice between using the IQR (Interquartile Range) method with box plots and the Z-score method for outlier detection depends on the characteristics of the data and the specific requirements of the analysis. Here's when each method is typically used:

1. IQR and Box Plots:
   - Use the IQR method with box plots when the data is not assumed to follow a normal distribution and may contain significant outliers. Box plots provide a visual representation of the data distribution, making it easy to identify the presence of outliers based on the whiskers of the box plot.
   - Box plots are particularly useful for comparing the spread and skewness of different datasets and for identifying potential anomalies or extreme values that lie outside the upper and lower whiskers.

2. Z-Score Method:
   - Use the Z-score method when the data is assumed to follow a normal distribution or when the objective is to quantitatively assess the deviation of each data point from the mean in terms of standard deviations.
   - The Z-score method is effective for identifying outliers based on predefined threshold values, such as a certain number of standard deviations from the mean. It provides a standardized approach to outlier detection and can be applied to both small and large datasets.

In practice, it's common to use both methods in tandem to ensure a comprehensive assessment of the data. This dual approach helps in validating the presence of outliers and in making informed decisions about whether to retain, adjust, or remove the identified outliers based on the context and objectives of the analysis.