

What is an Outlier?

Outlier is a data point that stands out significantly from the rest of the data.

It can be an extremely high or low value compared to the other observations in a dataset.

Outliers can be caused by measurement errors, natural variations in the data, or even unexpected discoveries.

Types of Outliers

There are 3 main types of outliers:

Global outliers: Stand out from the entire dataset, like a lone wolf.

Contextual outliers: Depend on their surroundings, like a high sale at a clothing store.

Collective outliers: Groups that deviate together, like a cluster of oddly high values.

What is the Outlier Detection Method?

Outlier detection is a method used to find unusual or abnormal data points in a set of information.

Imagine you have a group of friends, and you're all about the same age, but one person is much older or younger than the rest.

That person would be considered an outlier because they stand out from the usual pattern.

In data, outliers are points that deviate significantly from the majority, and detecting them helps identify unusual patterns or errors in the information.

This method is like finding the odd one out in a group, helping us spot data points that might need special attention or investigation.

How to Treat Outliers?

There are several ways to treat outliers in a dataset, depending on the nature of the outliers and the problem being solved.

Here are some of the most common ways of treating outlier values.

Trimming

It excludes the outlier values from our analysis.

By applying this technique, our data becomes thin when more outliers are present in the dataset. Its main advantage is its fastest nature.

Capping

In this technique called “outlier detection,” we cap our data to set limits.

For instance, if we decide on a specific value, any data point above or below that value is considered an outlier.

The number of outliers in the dataset then gives us insight into that capping number.

like setting a boundary and saying, “Anything beyond this point is unusual,” and by doing so, we identify and count the outliers in our data.

For example, if you’re working on the income feature, you might find that people above a certain income level behave similarly to those with a lower income. In this case, you can cap the income value at a level that keeps that intact and accordingly treat the outliers.

Treating outliers as a missing value: By assuming outliers as the missing observations, treat them accordingly, i.e., same as missing values imputation.

Discretization

In the method of outlier detection, we create groups and categorize the outliers into a specific group, making them follow the same behavior as the other points in that group.

This approach is often referred to as Binning.

Binning is a way of organizing data, especially in outlier detection, where we group similar items together, helping us identify and understand patterns more effectively.

How to Detect Outliers?

For Normal Distributions

Use empirical relations of Normal distribution.

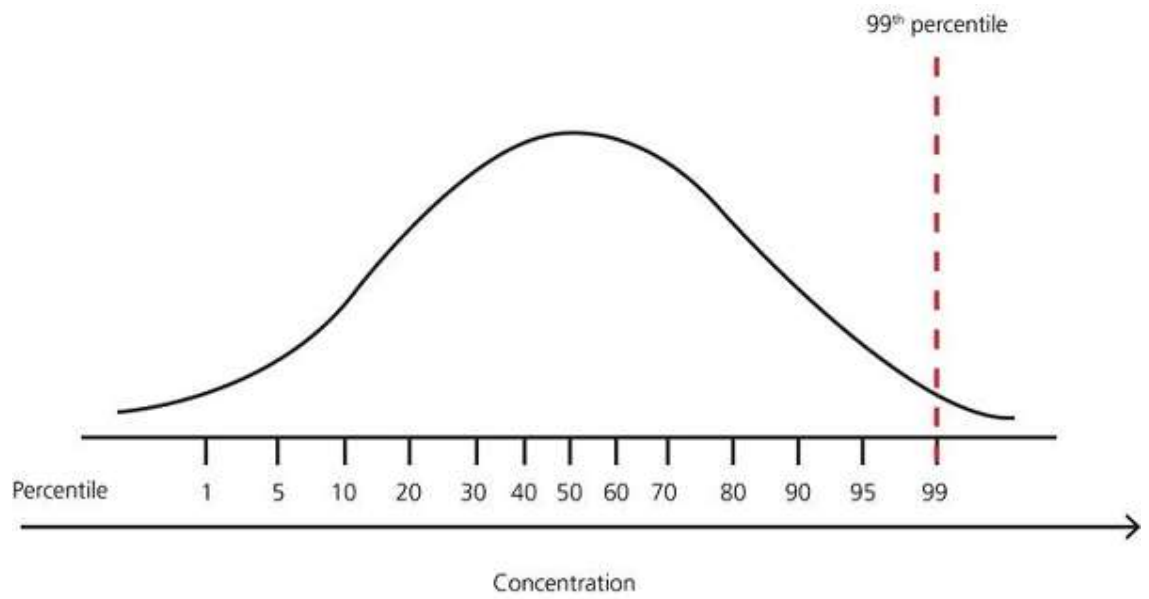
The data points that fall below $\text{mean} - 3 \cdot (\text{sigma})$ or above $\text{mean} + 3 \cdot (\text{sigma})$ are outliers, where mean and sigma are the average value and standard deviation of a particular column.

For Skewed Distributions

- Use Inter-Quartile Range (IQR) proximity rule.
- The data points that fall below $Q1 - 1.5 \text{ IQR}$ or above the third quartile $Q3 + 1.5 \text{ IQR}$ are outliers, where $Q1$ and $Q3$ are the 25th and 75th percentile of the dataset, respectively. IQR represents the inter-quartile range and is given by $Q3 - Q1$.

For Other Distributions

- Use a percentile-based approach.
- For Example, data points that are far from the 99% percentile and less than 1 percentile are considered an outlier.



How to Detect and Remove Outliers in Python

Z-score Treatment

Assumption: The features are normally or approximately normally distributed.

Step 1: Importing necessary dependencies

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

Step 2: Read and load the dataset

```
df = pd.read_csv('placement.csv')  
df.sample(5)
```

Step 3: Plot the distribution plots for the features

```
import warnings  
warnings.filterwarnings('ignore')  
plt.figure(figsize=(16,5))  
plt.subplot(1,2,1)  
sns.distplot(df['cgpa'])  
plt.subplot(1,2,2)  
sns.distplot(df['placement_exam_marks'])  
plt.show()
```

Step 4: Finding the boundary values

```
print("Highest allowed",df['cgpa'].mean() + 3*df['cgpa'].std())  
print("Lowest allowed",df['cgpa'].mean() - 3*df['cgpa'].std())
```

Output:

Highest allowed 8.808933625397177

Lowest allowed 5.113546374602842

Step 5: Finding the outliers

```
df[(df['cgpa'] > 8.80) | (df['cgpa'] < 5.11)]
```

Step 6: Trimming of outliers

```
new_df = df[(df['cgpa'] < 8.80) & (df['cgpa'] > 5.11)]  
new_df
```

Step 7: Capping on outliers

```
upper_limit = df['cgpa'].mean() + 3*df['cgpa'].std()  
lower_limit = df['cgpa'].mean() - 3*df['cgpa'].std()
```

Step 8: Now, apply the capping

```
df['cgpa'] = np.where(
    df['cgpa'] > upper_limit,
    upper_limit,
    np.where(
        df['cgpa'] < lower_limit,
        lower_limit,
        df['cgpa']
    )
)
```

Step 9: Now, see the statistics using the “Describe” function

```
df['cgpa'].describe()
```

Output:

```
count    1000.000000
mean       6.961499
std        0.612688
min        5.113546
25%        6.550000
50%        6.960000
75%        7.370000
max        8.808934
Name: cgpa, dtype: float64
```

This completes our Z-score-based technique!

IQR Based Filtering

Used when our data distribution is skewed.

Step-1: Import necessary dependencies

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

Step-2: Read and load the dataset

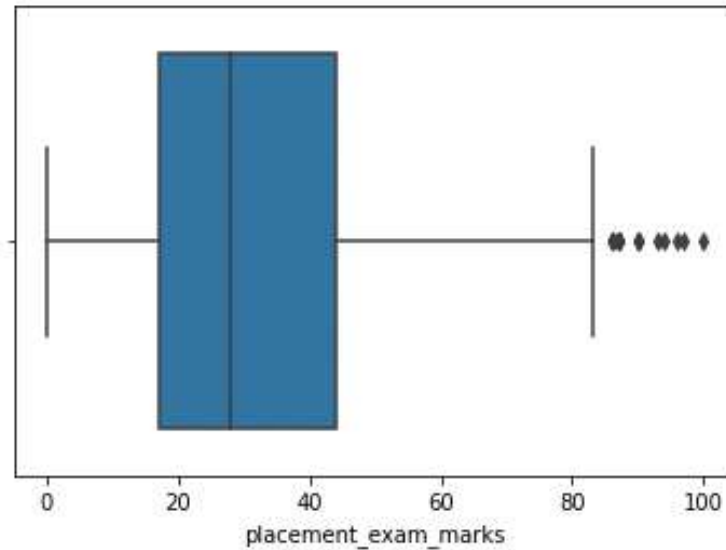
```
df = pd.read_csv('placement.csv')
df.head()
```

Step-3: Plot the distribution plot for the features

```
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['cgpa'])
plt.subplot(1,2,2)
sns.distplot(df['placement_exam_marks'])
plt.show()
```

Step-4: Form a box-plot for the skewed feature

```
sns.boxplot(df['placement_exam_marks'])
```



Step-5: Finding the IQR

```
percentile25 = df['placement_exam_marks'].quantile(0.25)
```

```
percentile75 = df['placement_exam_marks'].quantile(0.75)
```

Step-6: Finding the upper and lower limits

```
upper_limit = percentile75 + 1.5 * iqr
```

```
lower_limit = percentile25 - 1.5 * iqr
```

Step-7: Finding outliers

```
df[df['placement_exam_marks'] > upper_limit]
```

```
df[df['placement_exam_marks'] < lower_limit]
```

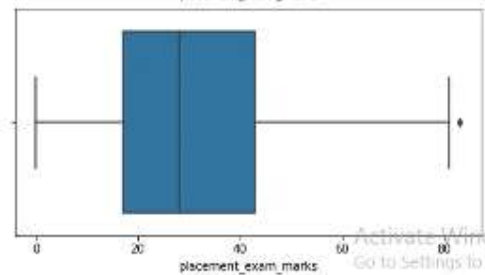
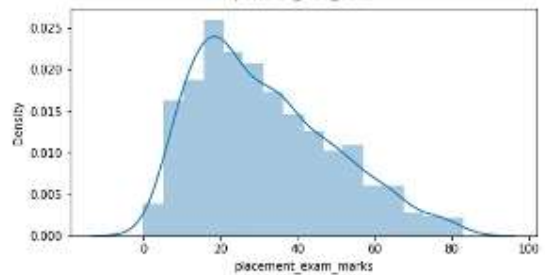
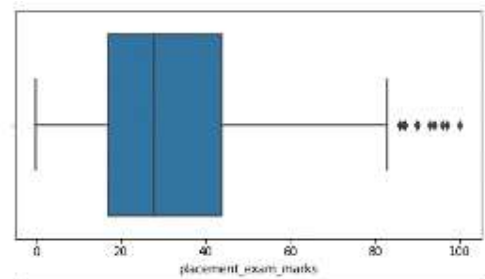
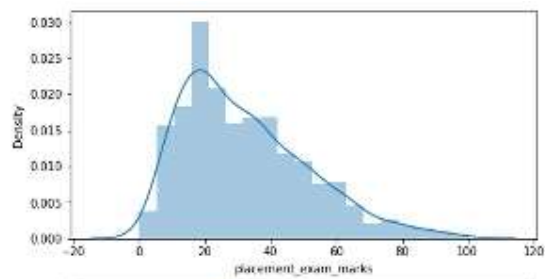
Step-8: Trimming outliers

```
new_df = df[df['placement_exam_marks'] < upper_limit]
```

```
new_df.shape
```

Step-9: Compare the plots after trimming

```
plt.figure(figsize=(16,8))  
plt.subplot(2,2,1)  
sns.distplot(df['placement_exam_marks'])  
plt.subplot(2,2,2)  
sns.boxplot(df['placement_exam_marks'])  
plt.subplot(2,2,3)  
sns.distplot(new_df['placement_exam_marks'])  
plt.subplot(2,2,4)  
sns.boxplot(new_df['placement_exam_marks'])  
plt.show()
```

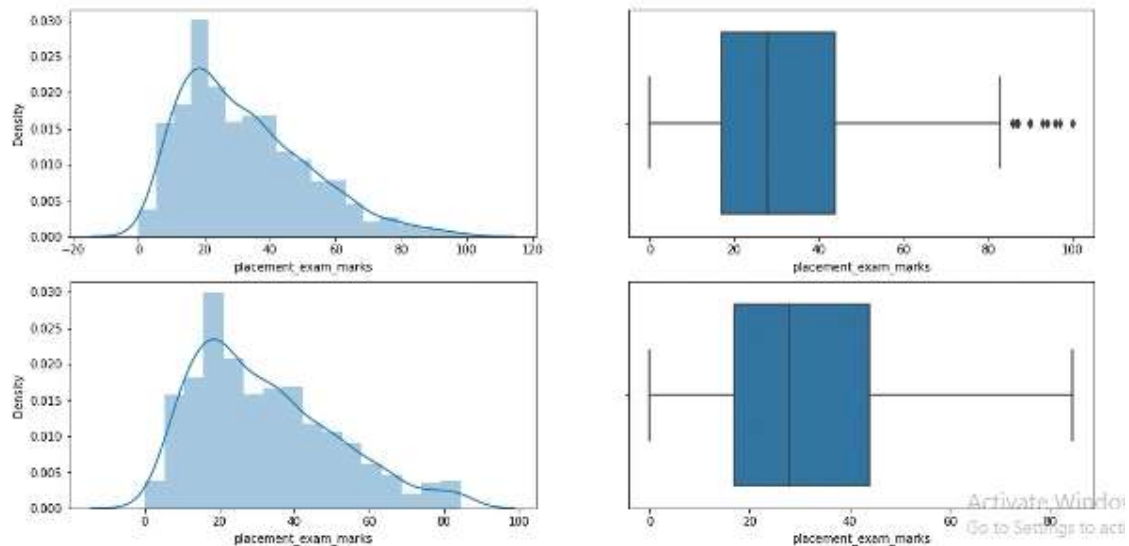


Step-10: Capping

```
new_df_cap = df.copy()
new_df_cap['placement_exam_marks'] = np.where(
    new_df_cap['placement_exam_marks'] > upper_limit,
    upper_limit,
    np.where(
        new_df_cap['placement_exam_marks'] < lower_limit,
        lower_limit,
        new_df_cap['placement_exam_marks']
    )
)
```

Step-11: Compare the plots after capping

```
plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df['placement_exam_marks'])
plt.subplot(2,2,2)
sns.boxplot(df['placement_exam_marks'])
plt.subplot(2,2,3)
sns.distplot(new_df_cap['placement_exam_marks'])
plt.subplot(2,2,4)
sns.boxplot(new_df_cap['placement_exam_marks'])
plt.show()
```



This completes our IQR-based technique!

Percentile Method

This technique works by setting a particular **threshold** value, which is decided based on our problem statement.

While we remove the outliers using capping, then that particular method is known as Winsorization.

Here, we always maintain symmetry on both sides, meaning if we remove 1% from the right, the left will also drop by 1%.

Steps to follow for the percentile method:

Step-1: Import necessary dependencies

```
import numpy as np
import pandas as pd
```

Step-2: Read and Load the dataset

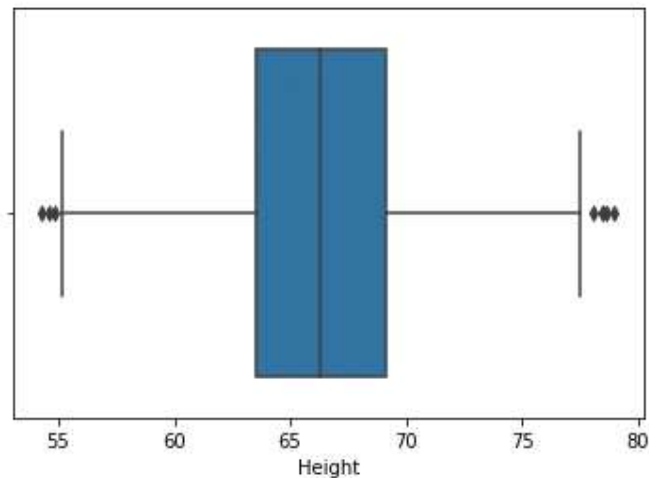
```
df = pd.read_csv('weight-height.csv')
df.sample(5)
```

Step-3: Plot the distribution plot of the “height” feature

```
sns.distplot(df['Height'])
```

Step-4: Plot the box-plot of the “height” feature

```
sns.boxplot(df['Height'])
```



Step-5: Finding the upper and lower limits

```
upper_limit = df['Height'].quantile(0.99)
```

```
lower_limit = df['Height'].quantile(0.01)
```

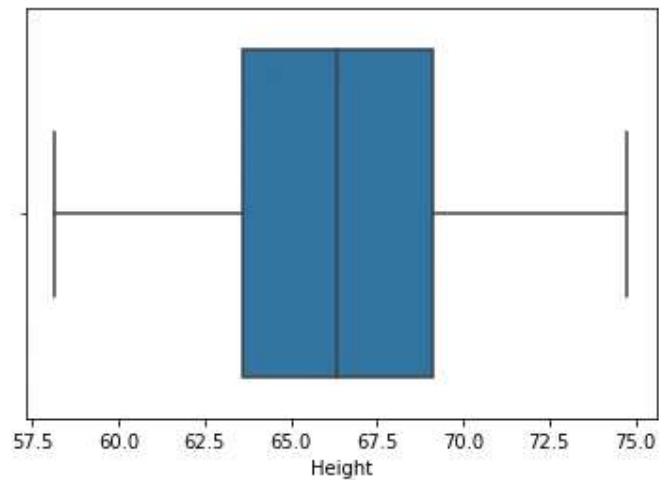
Step-6: Apply trimming

```
new_df = df[(df['Height'] <= 74.78) & (df['Height'] >= 58.13)]
```

Step-7: Compare the distribution and box-plot after trimming

```
sns.distplot(new_df['Height'])
```

```
sns.boxplot(new_df['Height'])
```



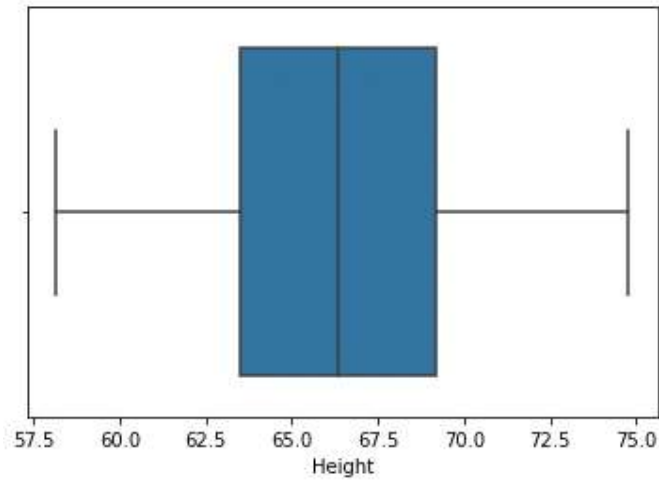
Step-8: Apply Capping (Winsorization)

```
df['Height'] = np.where(df['Height'] >= upper_limit,  
                        upper_limit,  
                        np.where(df['Height'] <= lower_limit,  
                                lower_limit,  
                                df['Height']))
```

Step-9: Compare the distribution and box-plot after capping

```
sns.distplot(df['Height'])
```

```
sns.boxplot(df['Height'])
```



This completes our percentile-based technique!