**Missing Values and Outlier Detection**

Data is the backbone of machine learning and artificial intelligence, but not all datasets are perfect. Often, datasets contain missing values and outliers that need to be handled before building any model.

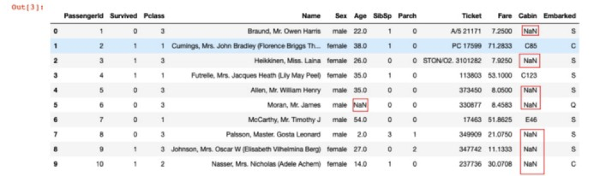
Missing values are the ones that are not present in the dataset, either because the data was not collected or because it was lost during processing. Outliers, on the other hand, are values that are significantly different from other values in the dataset. These can be due to errors in data collection or data processing.

**Handling Missing Values**

## **How Is a Missing Value Represented in a Dataset?**

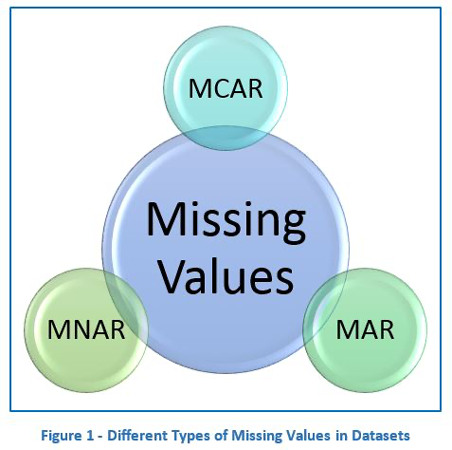
In the dataset, the blank shows the missing values.

In Pandas, usually, missing values are represented by **NaN**. It stands for **Not a Number**.



## **Types of Missing Values**

Formally the missing values are categorized as follows:



#### **Missing Completely At Random (MCAR)**

In MCAR, the probability of data being missing is the same for all the observations. In this case, there is no relationship between the missing data and any other values observed or unobserved (the data which is not recorded) within the given dataset. That is, missing values are completely independent of other data. There is no pattern.

In the case of MCAR data, the value could be missing due to human error, some system/equipment failure, loss of sample, or some unsatisfactory technicalities while recording the values. For Example, suppose in a library there are some overdue books. Some values of overdue books in the computer system are missing. The reason might be a human error, like the librarian forgetting to type in the values. So, the missing values of overdue books are not related to any other variable/data in the system. It should not be assumed as it’s a rare case. The advantage of such data is that the statistical analysis remains unbiased.

#### **Missing At Random (MAR)**

MAR data means that the reason for missing values can be explained by variables on which you have complete information, as there is some relationship between the missing data and other values/data. In this case, the data is not missing for all the observations. It is missing only within sub-samples of the data, and there is some pattern in the missing values.

For example, if you check the survey data, you may find that all the people have answered their ‘Gender,’ but ‘Age’ values are **mostly** missing for people who have answered their ‘Gender’ as ‘female.’ (The reason being most of the females don’t want to reveal their age.)

So, the probability of data being missing depends only on the observed value or data. In this case, the variables ‘Gender’ and ‘Age’ are related. The reason for missing values of the ‘Age’ variable can be explained by the ‘Gender’ variable, but you can not predict the missing value itself.

Suppose a poll is taken for overdue books in a library. Gender and the number of overdue books are asked in the poll. Assume that most of the females answer the poll and men are less likely to answer. So why the data is missing can be explained by another factor, that is gender. In this case, the statistical analysis might result in bias. Getting an unbiased estimate of the parameters can be done only by modeling the missing data.

#### **Missing Not At Random (MNAR)**

Missing values depend on the unobserved data. If there is some structure/pattern in missing data and other observed data **can not explain** it, then it is considered to be Missing Not At Random (MNAR).

If the missing data does not fall under the MCAR or MAR, it can be categorized as MNAR. It can happen due to the reluctance of people to provide the required information. A specific group of respondents may not answer some questions in a survey.

For example, suppose the name and the number of overdue books are asked in the poll for a library. So most of the people having no overdue books are likely to answer the poll. People having more overdue books are less likely to answer the poll. So, in this case, the missing value of the number of overdue books depends on the people who have more books overdue.

Another example is that people having less income may refuse to share some information in a survey or questionnaire.

In the case of MNAR as well, the statistical analysis might result in bias.

## **Why Do We Need to Care About Handling Missing Data?**

It is important to handle the missing values appropriately.

* Many machine learning algorithms fail if the dataset contains missing values. However, algorithms like K-nearest and Naive Bayes support data with missing values.
* You may end up building a biased machine learning model, leading to incorrect results if the missing values are not handled properly.
* Missing data can lead to a lack of precision in the statistical analysis.

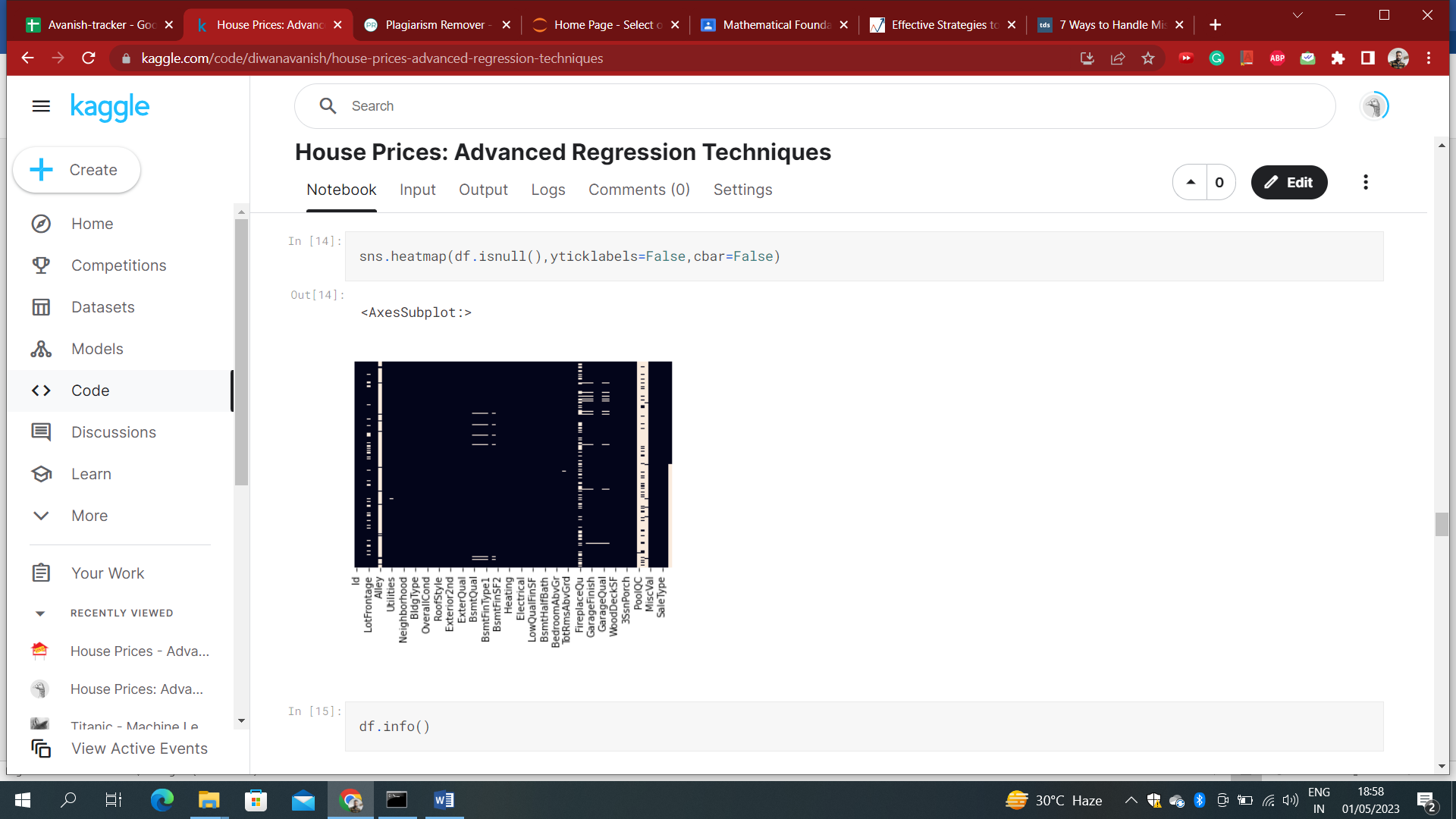
## **Handling Missing Values**

The most common way of dealing with missing values is to remove them from the dataset. However, this approach is not always recommended, especially if the missing values are too many. Instead, one can choose imputation method.

1. Find the total number of missing values from the entire dataset

*df.isnull().sum()*

**2**. Visualize the null data



White line or point showing the missing values.

**3**. There are 2 ways one can delete the missing data values:

Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values. If the missing value is of the type Missing Not At Random (MNAR), then it should not be deleted.

If the missing value is of type Missing At Random (MAR) or Missing Completely At Random (MCAR) then it can be deleted (In the analysis, all cases with available data are utilized, while missing observations are assumed to be completely random (MCAR) and addressed through pairwise deletion).

Deleting the entire row (listwise deletion):

If a row has many missing values, you can drop the entire row. If every row has some (column) value missing, you might end up deleting the whole data. The code to drop the entire row is as follows:

*df = df.dropna(axis=0)*

*df.isnull().sum()*

Deleting the entire column:

If a certain column has many missing values, then you can choose to drop the entire column. The code to drop the entire column is as follows:

*df.drop(columns = ['Id','Alley','PoolQC','Fence','MiscFeature'],inplace = True)*

*df.isnull().sum()*

### **Imputing the Missing Value**

There are many imputation methods for replacing the missing values. You can use different python libraries such as Pandas, and Sci-kit Learn to do this. Let’s go through some of the ways of replacing the missing values.

**Replacing with an arbitrary value**

If you can make an educated guess about the missing value, then you can replace it with some arbitrary value using the following code. E.g., in the following code, we are replacing the missing values of the ‘Dependents’ column with ‘0’.

IN:

#Replace the missing value with '0' using 'fiilna' method

train\_df['Dependents'] = train\_df['Dependents'].fillna(0)

train\_df[‘Dependents'].isnull().sum()

OUT:

0

**Replacing with the mean, median and mode**

This is the most common method of imputing missing values of numeric columns. If there are outliers, then the mean will not be appropriate. In such cases, outliers need to be treated first.

The median is the middlemost value. It’s better to use the median value for imputation in the case of outliers.

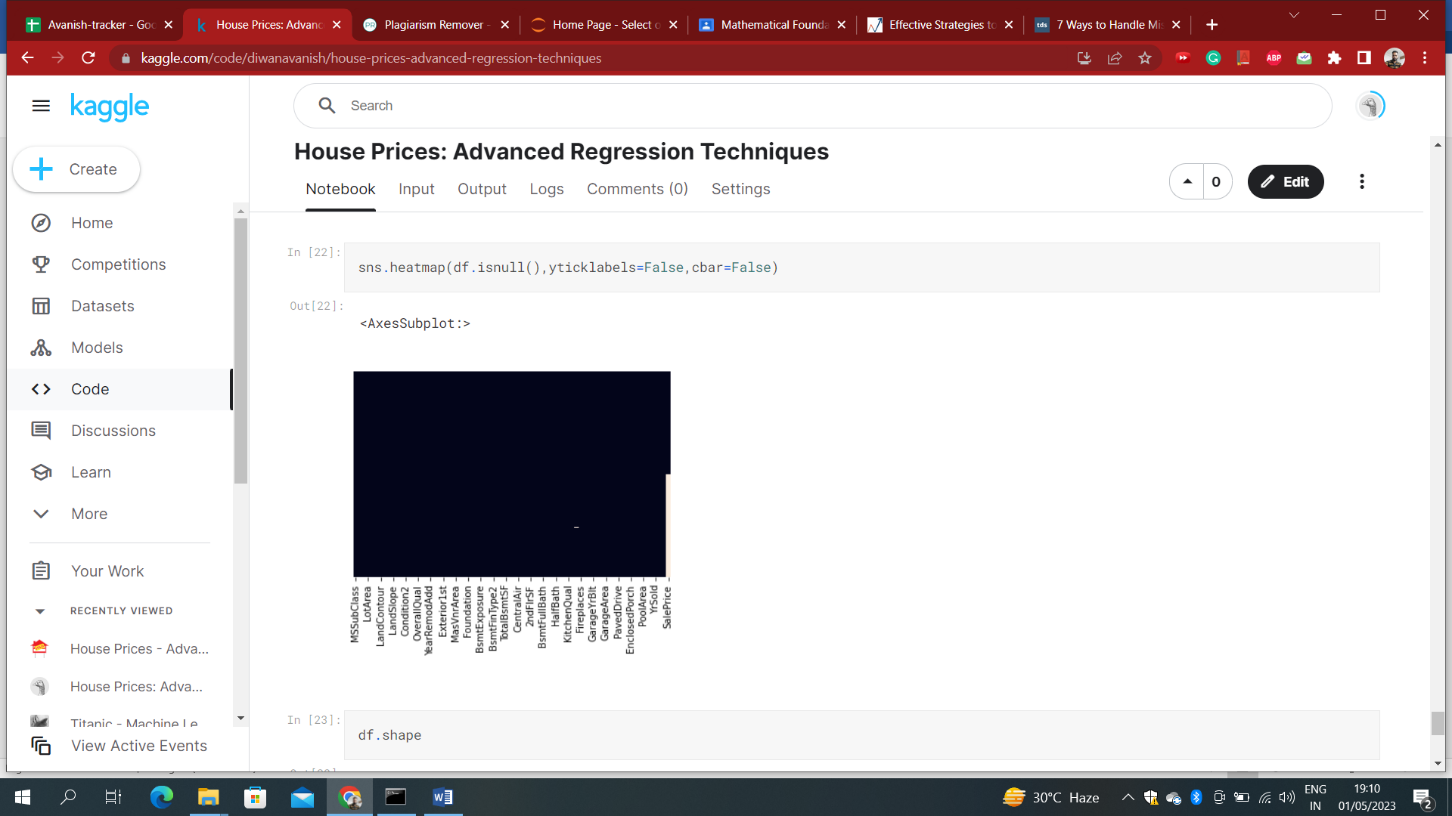
Mode is the most frequently occurring value. It is used in the case of categorical features.

*df['LotFrontage']=df['LotFrontage'].fillna(df['LotFrontage'].mean())*

*df['GarageYrBlt']=df['GarageYrBlt'].fillna(df['GarageYrBlt'].median())*

*df['BsmtCond']=df['BsmtCond'].fillna(df['BsmtCond'].mode()[0])*

*df['BsmtQual']=df['BsmtQual'].fillna(df['BsmtQual'].mode()[0])*



Above heatmap shows all the missing values are manipulated and thus not appear any white line or point.

Replacing with the previous value – forward fill

In some cases, imputing the values with the previous value instead of the mean, mode, or median is more appropriate. This is called forward fill. It is mostly used in time series data. You can use the ‘fillna’ function with the parameter ‘method = ffill’

IN:

import pandas as pd

import numpy as np

test = pd.Series(range(6))

test.loc[2:4] = np.nan

test

OUT:

0 0.0

1 1.0

2 Nan

3 Nan

4 Nan

5 5.0

dtype: float64

IN:

# Forward-Fill

test.fillna(method=‘ffill')

OUT:

0 0.0

1 1.0

2 1.0

3 1.0

4 1.0

5 5.0

dtype: float64

Replacing with the next value – backward fill

In backward fill, the missing value is imputed using the next value.

IN:

# Backward-Fill

test.fillna(method=‘bfill')

OUT:

0 0.0

1 1.0

2 5.0

3 5.0

4 5.0

5 5.0

dtype: float64

Interpolation

Missing values can also be imputed using interpolation. Pandas’ interpolate method can be used to replace the missing values with different interpolation methods like ‘polynomial,’ ‘linear,’ and ‘quadratic.’ The default method is ‘linear’.

IN:

test.interpolate()

OUT:

0 0.0

1 1.0

2 2.0

3 3.0

4 4.0

5 5.0

dtype: float64

## **How to Impute Missing Values for Categorical Features?**

There are two ways to impute missing values for categorical features as follows:

#### **Impute the Most Frequent Value**

We will use ‘SimpleImputer’ in this case, and as this is a non-numeric column, we can’t use mean or median, but we can use the most frequent value and constant.

IN:

import pandas as pd

import numpy as np

X = pd.DataFrame({'Shape':['square', 'square', 'oval', 'circle', np.nan]})

X

Shape

OUT:

0 square

1 square

2 oval

3 circle

4 NaN

IN:

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='most\_frequent')

imputer.fit\_transform(X)

OUT:

array([['square'],

['square'],

['oval'],

['circle'],

['square']], dtype=object)

As you can see, the missing value is imputed with the most frequent value, ’square.’

#### **Impute the Value “Missing”**

We can impute the value “missing,” which treats it as a separate category.

IN:

imputer = SimpleImputer(strategy='constant', fill\_value='missing')

imputer.fit\_transform(X)

OUT:

array([['square'],

['square'],

['oval'],

['circle'],

['missing']], dtype=object)

In any of the above approaches, you will still need to OneHotEncode the data (or you can also use another encoder of your choice). After One Hot Encoding, in case 1, instead of the values ‘square,’ ‘oval,’ and’ circle,’ you will get three feature columns. And in case 2, you will get four feature columns (4th one for the ‘missing’ category). So it’s like adding the missing indicator column in the data. There is another way to add a missing indicator column, which we will discuss further.

## **How to Impute Missing Values Using Sci-kit Learn Library?**

We can impute missing values using the sci-kit library by creating a model to predict the observed value of a variable based on another variable which is known as regression imputation.

#### **Univariate Approach**

In a Univariate approach, only a single feature is taken into consideration. You can use the class SimpleImputer and replace the missing values with mean, mode, median, or some constant value.

Let’s see an example:

IN:

import numpy as np

from sklearn.impute import SimpleImputer

imp = SimpleImputer(missing\_values=np.nan, strategy='mean')

imp.fit([[1, 2], [np.nan, 3], [7, 6]])

OUT: SimpleImputer()

IN:

X = [[np.nan, 2], [6, np.nan], [7, 6]]

print(imp.transform(X))

OUT:

[[4. 2. ]

[6. 3.666...]

[7. 6. ]]

#### **Multivariate Approach**

In a multivariate approach, more than one feature is taken into consideration. There are two ways to impute missing values considering the multivariate approach. Using KNNImputer or IterativeImputer classes.

Let’s take an example of a titanic dataset.

Suppose the feature ‘age’ is well correlated with the feature ‘Fare’ such that people with lower fares are also younger and people with higher fares are also older. In that case, it would make sense to impute low age for low fare values and high age for high fare values. So here, we are taking multiple features into account by following a multivariate approach.

IN:

import pandas as pd

df = pd.read\_csv('http://bit.ly/kaggletrain', nrows=6)

cols = ['SibSp', 'Fare', 'Age']

X = df[cols]

X

|  |
| --- |
|  |
| **SibSp** | **Fare** | **Age** |
| **0** | 1 | 7.2500 | 22.0 |
| **1** | 1 | 71.2833 | 38.0 |
| **2** | 0 | 7.9250 | 26.0 |
| **3** | 1 | 53.1000 | 35.0 |
| **4** | 0 | 8.0500 | 35.0 |
| **5** | 0 | 8.4583 | NaN |
|  |  |  |  |

IN:

from sklearn.experimental import enable\_iterative\_imputer

from sklearn.impute import IterativeImputer

impute\_it = IterativeImputer()

impute\_it.fit\_transform(X)

OUT:

array([[ 1. , 7.25 , 22. ],

[ 1. , 71.2833 , 38. ],

[ 0. , 7.925 , 26. ],

[ 1. , 53.1 , 35. ],

[ 0. , 8.05 , 35. ],

[ 0. , 8.4583 , 28.50639495]])

Let’s see how IterativeImputer works. For all rows in which ‘Age’ is not missing, sci-kit learn runs a regression model. It uses ‘Sib sp’ and ‘Fare’ as the features and ‘Age’ as the target. And then, for all rows for which ‘Age’ is missing, it makes predictions for ‘Age’ by passing ‘Sib sp’ and ‘Fare’ to the training model. So it actually builds a regression model with two features and one target and then makes predictions on any places where there are missing values. And those predictions are the imputed values.

#### **Nearest Neighbors Imputations (KNNImputer)**

Missing values are imputed using the k-Nearest Neighbors approach, where a Euclidean distance is used to find the nearest neighbors. Let’s take the above example of the titanic dataset to see how it works.

IN:

from sklearn.impute import KNNImputer

impute\_knn = KNNImputer(n\_neighbors=2)

impute\_knn.fit\_transform(X)

OUT:

array([[ 1. , 7.25 , 22. ],

[ 1. , 71.2833, 38. ],

[ 0. , 7.925 , 26. ],

[ 1. , 53.1 , 35. ],

[ 0. , 8.05 , 35. ],

[ 0. , 8.4583, 30.5 ]])

In the above example, the n\_neighbors=2. So sci-kit learn finds the two most similar rows measured by how close the ‘Sib sp’ and ‘Fare’ values are to the row which has missing values. In this case, the last row has a missing value. And the third row and the fifth row have the closest values for the other two features. So the average of the ‘Age’ feature from these two rows is taken as the imputed value.

## **How to Use “Missingness” as a Feature?**

In some cases, while imputing missing values, you can preserve information about which values were missing and use that as a feature. This is because sometimes, there may be a relationship between the reason for missing values (also called the “missingness”) and the target variable you are trying to predict. In such cases, you can add a missing indicator to encode the “missingness” as a feature in the imputed data set.

**Where can we use this?**

Suppose you are predicting the presence of a disease. Now, imagine a scenario where a missing age is a good predictor of the disease because we don’t have records for people in poverty. The age values are not missing at random. They are missing for people in poverty, and poverty is a good predictor of disease. Thus, missing age or “missingness” is a good predictor of disease.

IN:

import pandas as pd

import numpy as np

X = pd.DataFrame({'Age':[20, 30, 10, np.nan, 10]})

X

Out:

|  |
| --- |
|  |
| **Age** |
| **0** | 20.0 |
| **1** | 30.0 |
| **2** | 10.0 |
| **3** | NaN |
| **4** | 10.0 |
|  |  |

IN:

from sklearn.impute

import SimpleImputer

# impute the mean

imputer = SimpleImputer()

imputer.fit\_transform(X)

OUT:

array([[20. ],

[30. ],

[10. ],

[17.5],

[10. ]])

IN:

imputer = SimpleImputer(add\_indicator=True)

imputer.fit\_transform(X)

OUT:

array([[20. , 0. ],

[30. , 0. ],

[10. , 0. ],

[17.5, 1. ],

[10. , 0. ]])

In the above example, the second column indicates whether the corresponding value in the first column was missing or not. ‘1’ indicates that the corresponding value was missing, and ‘0’ indicates that the corresponding value was not missing.

If you don’t want to impute missing values but only want to have the indicator matrix, then you can use the ‘MissingIndicator’ class from scikit learn.

**Handling Outliers:**

Outliers can be detected using various statistical techniques such as z-score, box plot, and scatter plot. Once identified, one can use the following techniques to handle them:

### **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/** Winsorization

In this technique, we cap our outliers data and make the limit i.e, above a particular value or less than that value, all the values will be considered as outliers, and the number of outliers in the dataset gives that capping number.

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.

Removal

Remove the outlier from the dataset.

Transformation

Transform the data using log or square root functions to reduce the impact of outliers.

### **Binning**

In this technique, by making the groups, we include the outliers in a particular group and force them to behave in the same manner as those of other points in that group.

One can learn binning in detail tapping the below link.

<https://www.analyticsvidhya.com/blog/2021/05/complete-guide-on-encode-numerical-features-in-machine-learning/>

## **How to Detect Outliers?**

### **For Normal Distributions**

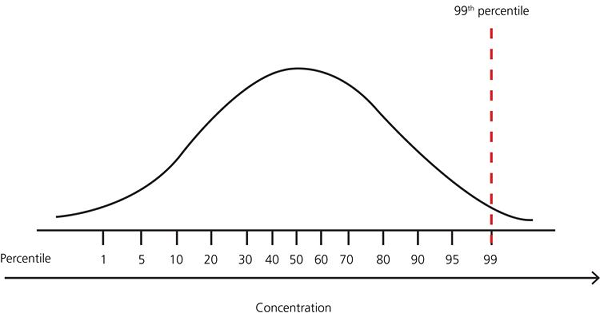
* Use empirical relations of Normal distribution.
* The data points that fall below **mean-3\*(sigma)**or above **mean+3\*(sigma)**are outliers, where mean and sigma are the **average value** and **standard deviation** of a particular column.

**For Skewed Distributions**

* Use Inter-Quartile Range (IQR) proximity rule.
* The data points that fall below **Q1 – 1.5 IQR**or above the third quartile **Q3 + 1.5 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.



### **1. Using Standard Deviation:**

When the data, or certain features in the dataset, follow a [normal distribution](https://mathworld.wolfram.com/NormalDistribution.html), you can use the standard deviation of the data, or the equivalent z-score to detect outliers.

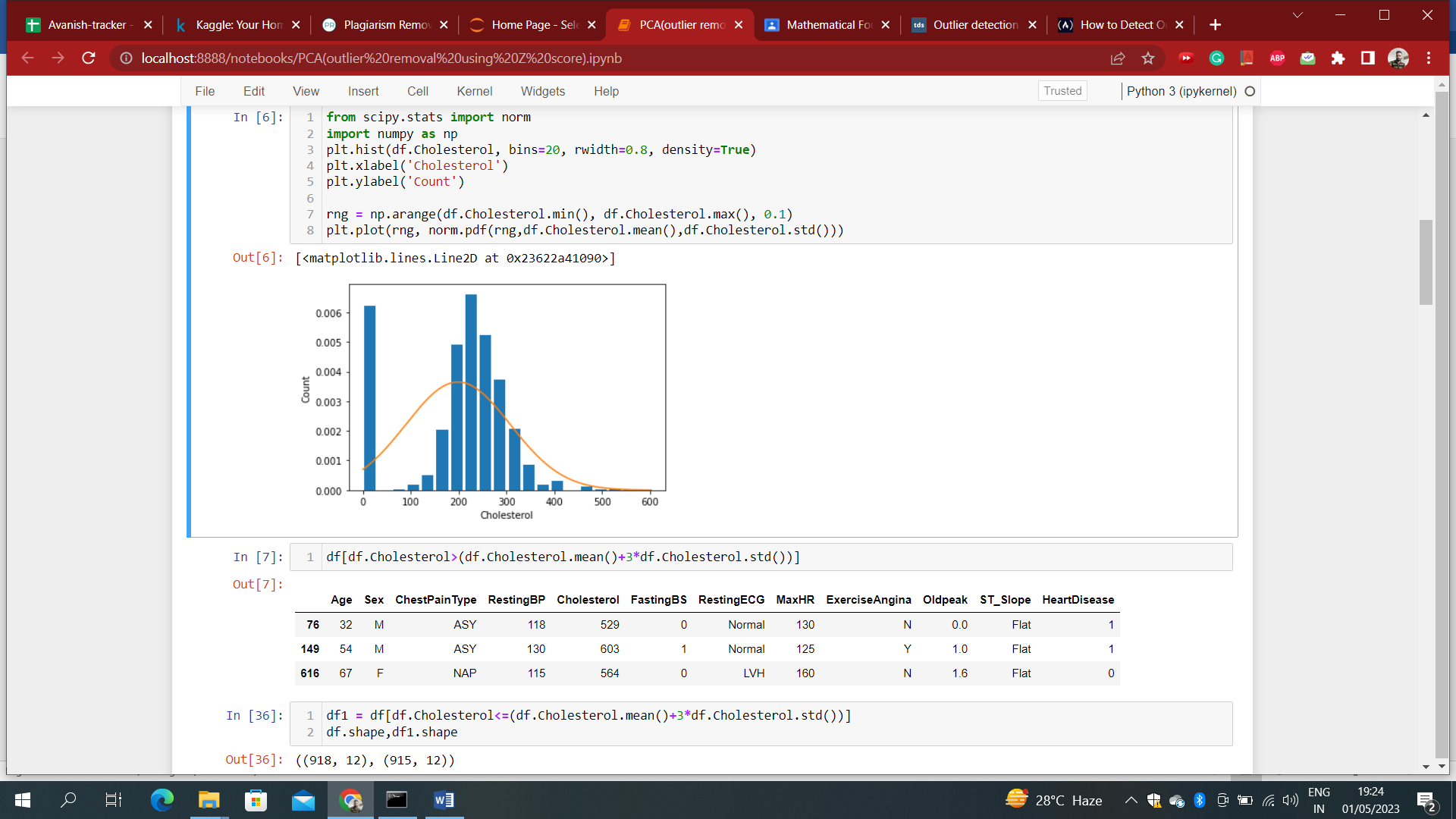
In statistics, standard deviation measures the spread of data around the mean, and in essence, it captures how far away from the mean the data points are.

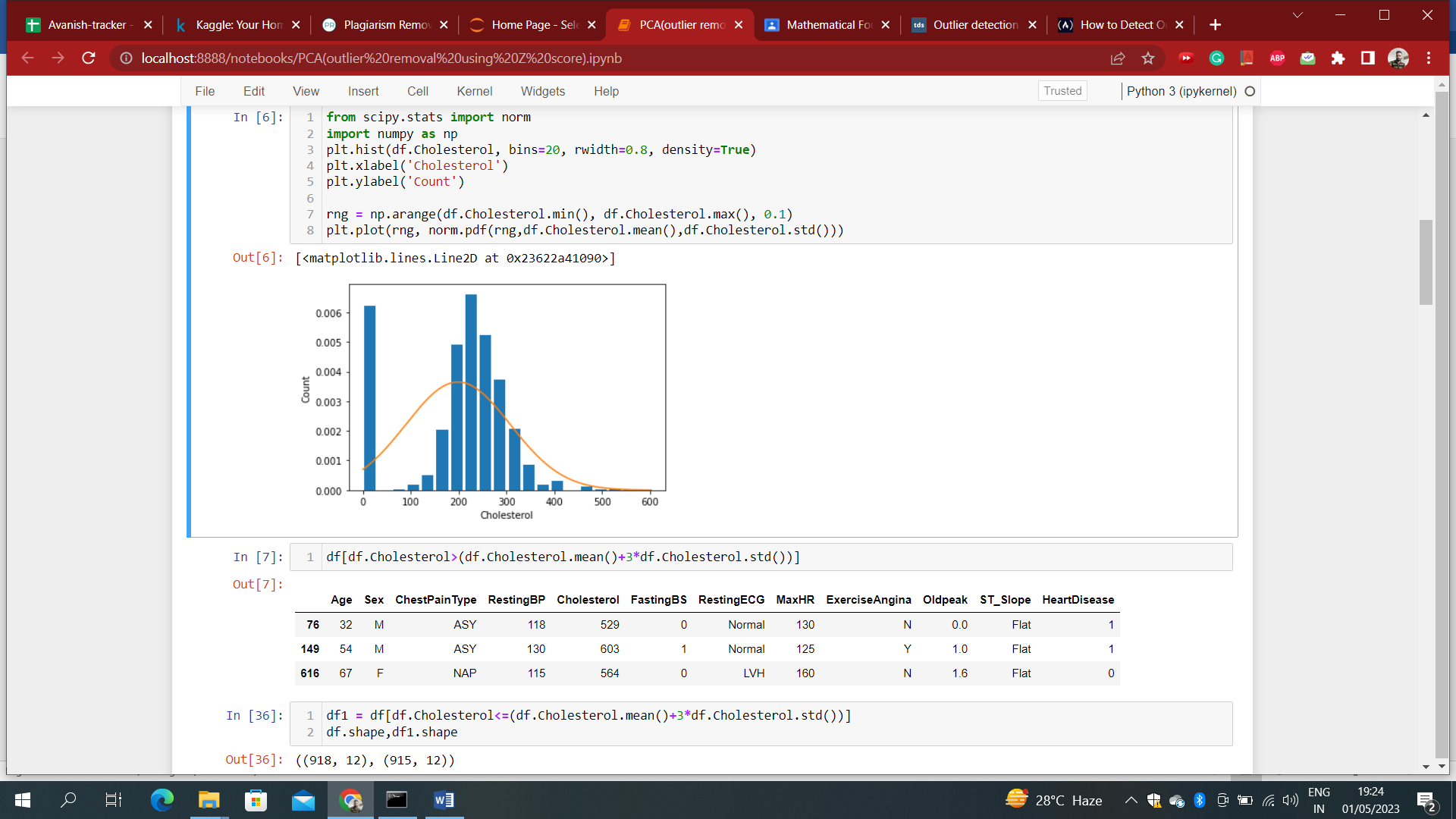
For data that is normally distributed, around 68.2% of the data will lie within one standard deviation from the mean. Close to 95.4% and 99.7% of the data lie within two and three standard deviations from the mean, respectively.

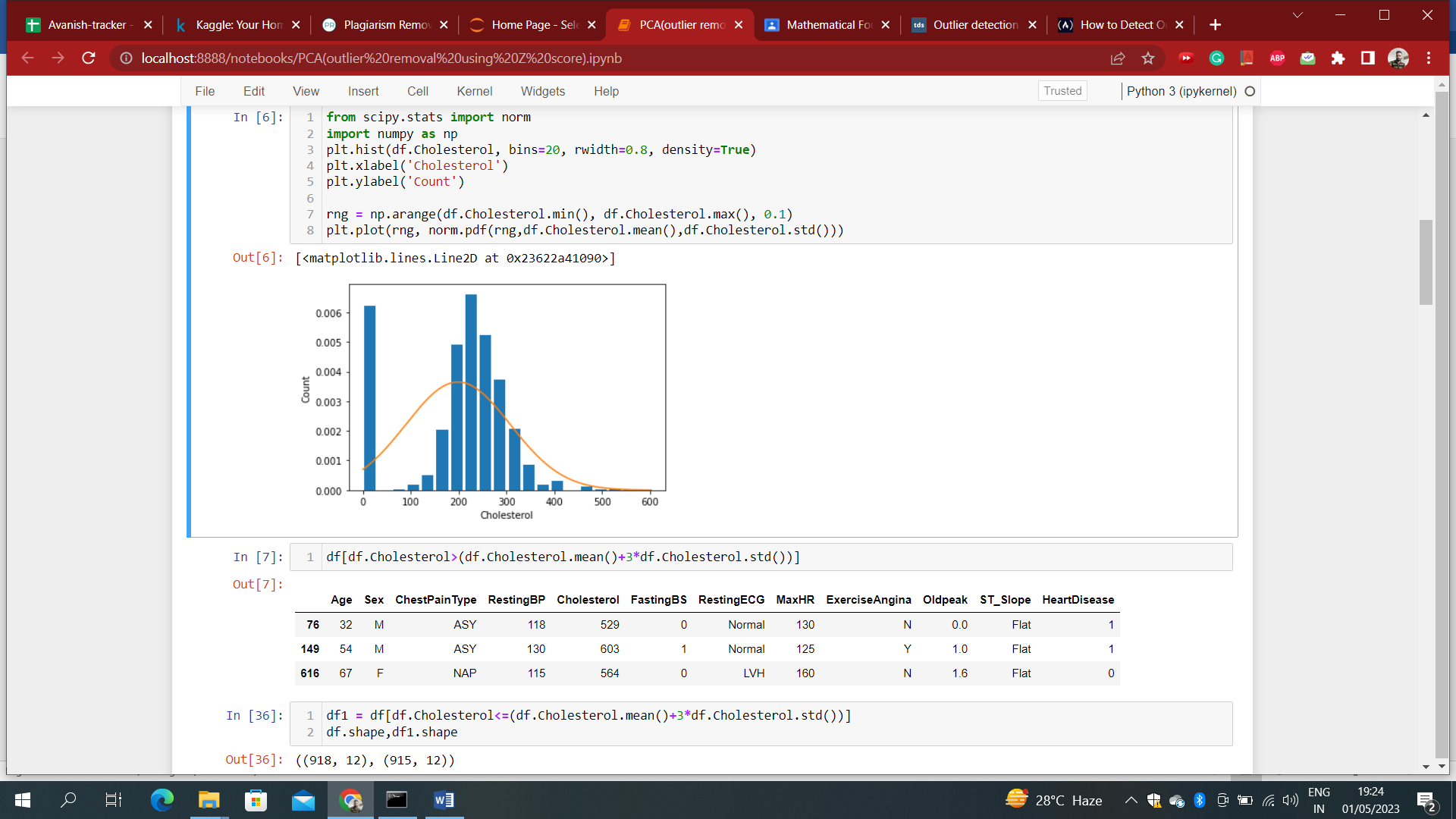
Let’s denote the standard deviation of the distribution by σ, and the mean by μ.

One approach to outlier detection is to set the lower limit to three standard deviations below the mean (μ - 3\*σ), and the upper limit to three standard deviations above the mean (μ + 3\*σ). Any data point that falls outside this range is detected as an outlier.

As 99.7% of the data typically lies within three standard deviations, the number of outliers will be close to 0.3% of the size of the dataset.







### **2. Using Z Score:**

Now let's explore the concept of the z-score. For a normal distribution with mean μ and standard deviation σ, the z-score for a value x in the dataset is given by:

**z = (x - μ)/σ**

From the above equation, we have the following:

* When x = μ, the value of z-score is 0.
* When x = μ ± 1, μ ± 2, or μ ± 3, the z-score is ± 1, ± 2, or ± 3, respectively.

Notice how this technique is equivalent to the scores based on standard deviation we had earlier. Under this transformation, all data points that lie below the lower limit, μ - 3\*σ, now map to points that are less than -3 on the z-score scale.

Similarly, all points that lie above the upper limit, μ + 3\*σ map to a value above 3 on the z-score scale. So [lower\_limit, upper\_limit] becomes [-3, 3].

df\_scores['z\_score']=(df\_scores['score'] - df\_scores['score'].mean())/df\_scores['score'].std()

df\_scores.head()

You can filter the dataframe to retain points whose z-scores are in the range [-3, 3], as shown below.

df\_scores\_filtered= df\_scores[(df\_scores['z\_score']>-3) & (df\_scores['z\_score']<3)]

print(df\_scores\_filtered)

**NOTE:** The methods involving standard deviation and z-scores can be used only when the data set, or the feature that you are examining, follows a normal distribution. Next, we’ll discuss two outlier detection techniques that can be used *independently* of the data distribution.

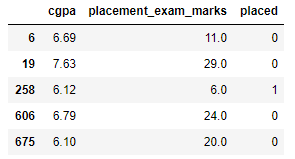
### **Z-score Treatment**

**Assumption:** The features are normally or approximately normally distributed.

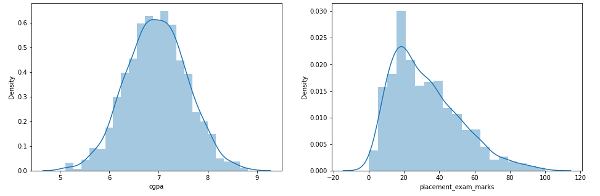
1. **Step 1: Importing necessary dependencies**

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

1. **Step 2: Read and load the dataset**

df = pd.read\_csv(‘placement.csv’)  
df.sample(5)  


1. **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()  


1. **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

1. **Step 5: Finding the outliers**

df[(df[‘cgpa’] > 8.80) | (df[‘cgpa’] < 5.11)]

1. **Step 6: Trimming of outliers**

new\_df = df[(df[‘cgpa’] < 8.80) & (df[‘cgpa’] > 5.11)]  
new\_df

1. **Step 7: Capping on outliers**

upper\_limit = df[‘cgpa’].mean() + 3\*df[‘cgpa’].std()  
lower\_limit = df[‘cgpa’].mean() – 3\*df[‘cgpa’].std()

1. **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’]))

1. **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'])

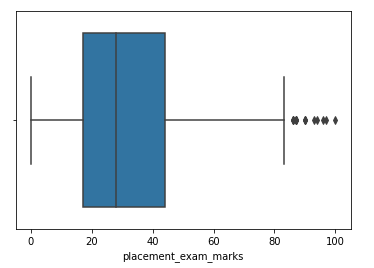
plt.subplot(1,2,2)

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'])

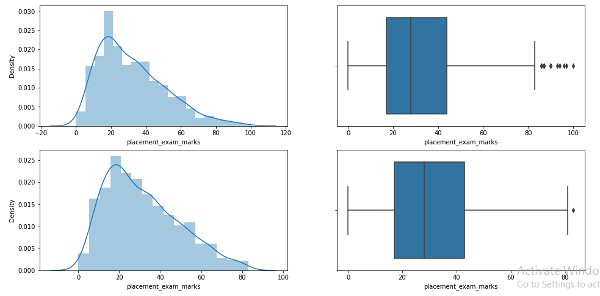
plt.subplot(2,2,3)

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'])

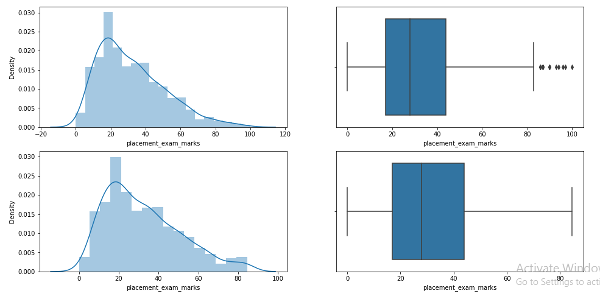
plt.subplot(2,2,3)

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%. We can customize the range by removing 0.5 % from right and left.

Steps to follow for the percentile method:

#### **Step-1: Import necessary dependencies**

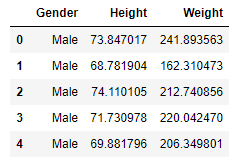
import numpy as np

import pandas as pd

#### **Step-2: Read and Load the dataset**

df = pd.read\_csv('weight-height.csv')

df.sample(5)

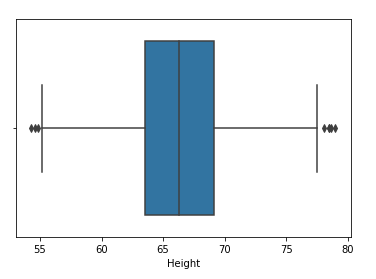


#### **Step-3: Plot the distribution plot of the “height” feature**

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)

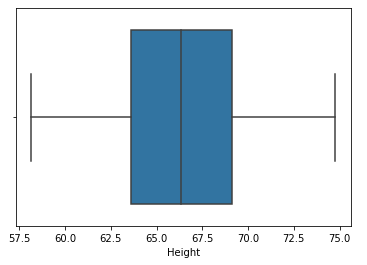
#### **Step-6: Apply trimming**

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'])



**Winsorization**

#### **Step-8: Apply Capping (Winsorization)**

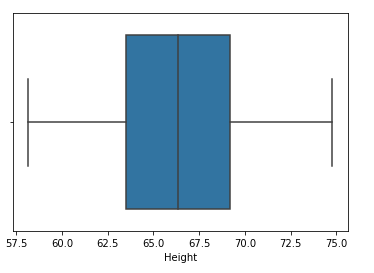
df['Height'] = np.where(df['Height'] >= upper\_limit, upper\_limit, np.where(df['Height'] <= lower\_limit,

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!

Note: Outlier detection and removal using ml models as per below:

<https://machinelearningmastery.com/model-based-outlier-detection-and-removal-in-python/>

<https://pieriantraining.com/dbscan-for-outlier-detection-in-python-and-scikit-learn-machine-learning-in-python/#:~:text=In%20summary%2C%20DBSCAN%20is%20a,to%20any%20cluster%20as%20outliers>.

## **Frequently Asked Questions**

**Q1. What are some of the most popular outlier detection techniques?**

A. Most popular outlier detection methods are Z-Score, IQR (Interquartile Range), Mahalanobis Distance, DBSCAN (Density-Based Spatial Clustering of Applications with Noise, Local Outlier Factor (LOF), and One-Class SVM (Support Vector Machine).

**Q2. What are the libraries and plots we can utilize to detect and remove outliers in a data set for a data science project?**

A. Libraries like SciPy and NumPy can be used to identify outliers. Also, plots like Box plot, Scatter plot, and Histogram are useful in visualizing the data and its distribution to identify outliers based on the values that fall outside the normal range.

**Q3. What is the advantage of removing outliers?**

A. The benefit of removing outliers is to enhance the accuracy and stability of statistical models and ML algorithms by reducing their impact on results. Outliers can distort statistical analyses and skew results as they are extreme values that differ from the rest of the data. Removing outliers makes the results more robust and accurate by eliminating their influence. It reduces overfitting in ML algorithms by avoiding fitting to extreme values instead of the underlying data pattern.

Doubt: Can DBSCAN help to remove outliers?

Probably not, but can detect them.

**Extranotes:**

Scikit-Learn provides a handy class to take care of missing values: Imputer. Here is how to use it. First, you need to create an Imputer instance, specifying that you want to replace each attribute’s missing values with the median of that attribute:

from sklearn.preprocessing import Imputer

imputer = Imputer(strategy="median")

Since the median can only be computed on numerical attributes, we need to create a copy of the data without the text attribute ocean\_proximity:

housing\_num = housing.drop("ocean\_proximity", axis=1)

Now you can fit the imputer instance to the training data using the fit() method:

imputer.fit(housing\_num)

The imputer has simply computed the median of each attribute and stored the result in its statistics\_ instance variable. Only the total\_bedrooms attribute had missing values, but we cannot be sure that there won’t be any missing values in new data after the system goes live, so it is safer to apply the imputer to all the numerical attributes:

>>> imputer.statistics\_ array([ -118.51 , 34.26 , 29. , 2119. , 433. , 1164. , 408. , 3.5414])

>>> housing\_num.median().values array([ -118.51 , 34.26 , 29. , 2119. , 433. , 1164. , 408. , 3.5414])

Now you can use this “trained” imputer to transform the training set by replacing missing values by the learned medians:

X = imputer.transform(housing\_num)

The result is a plain Numpy array containing the transformed features. If you want to put it back into a Pandas DataFrame, it’s simple:

housing\_tr = pd.DataFrame(X, columns=housing\_num.columns)

Note: Don’t forget to add the dropped categorical column in the dataset later.