**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.

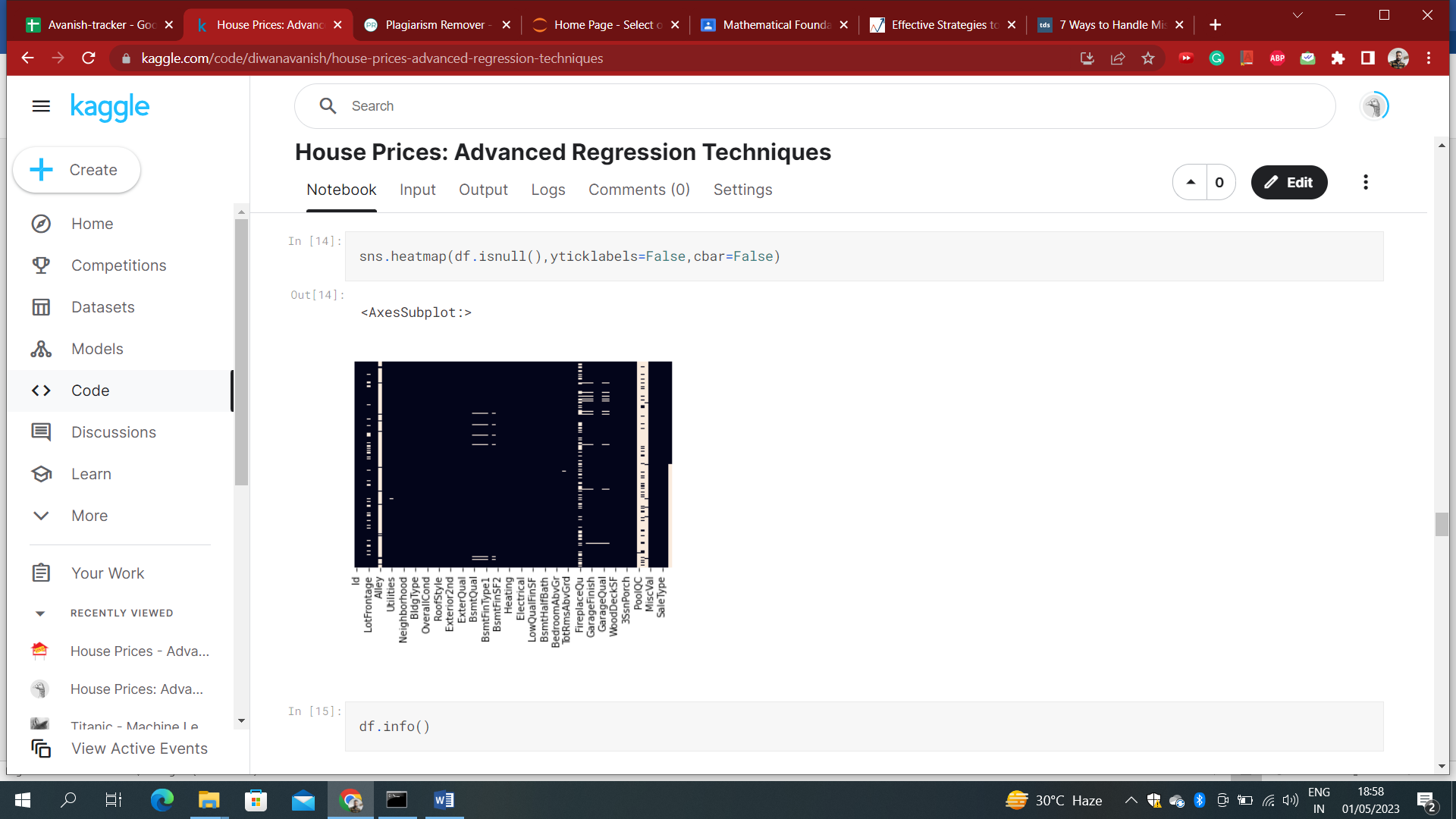
**Handling Missing Values:**

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:

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()*

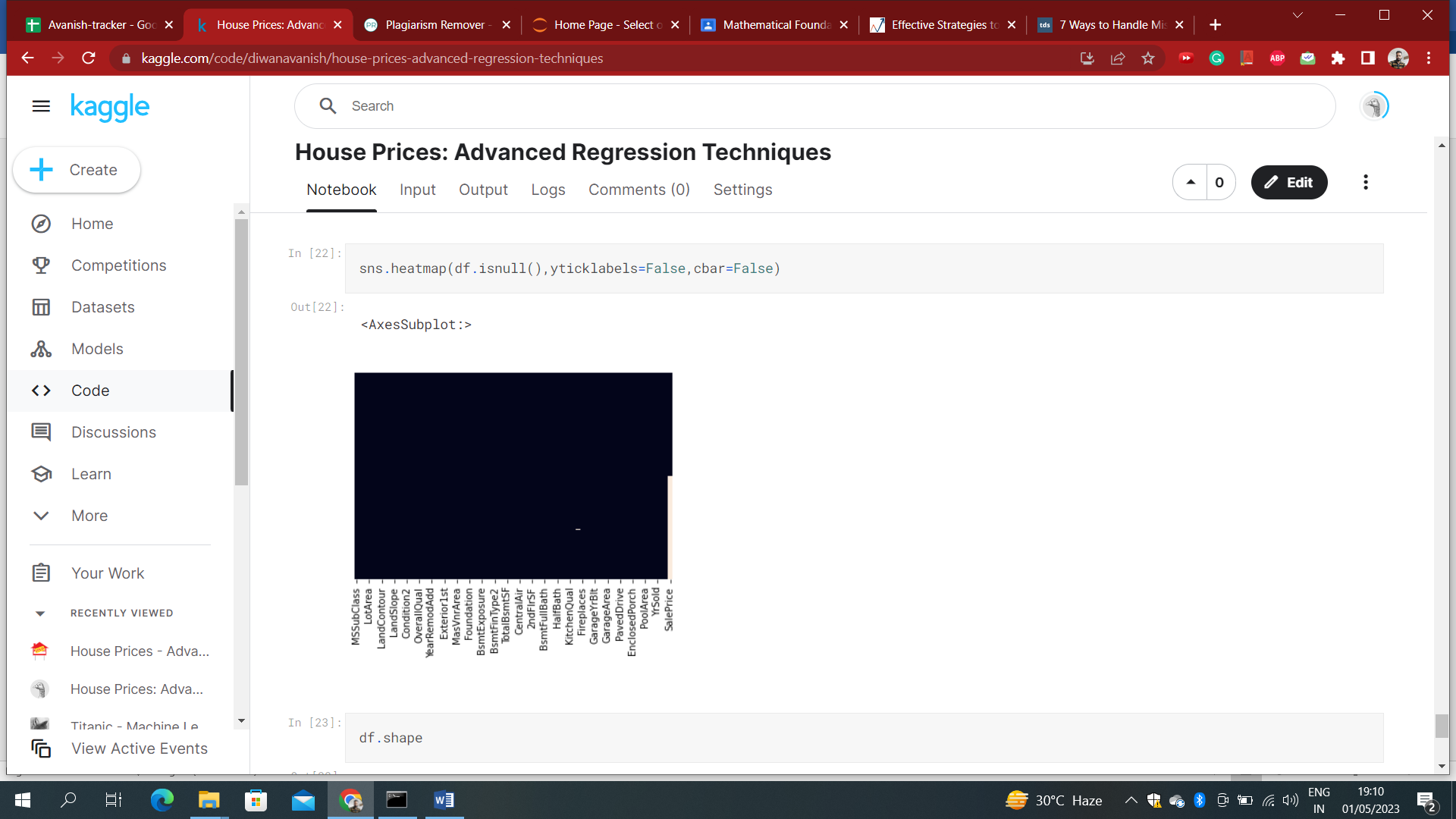
Mean or Median Imputation: Replace missing values with the mean or median value of the remaining data. Forward or Backward Fill: Fill the missing values with the previous or next value in the dataset. Interpolation: Predict missing values using a regression algorithm based on the remaining data.

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

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

*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.

**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:

Removal: Remove the outlier from the dataset.

Winsorization: Replace the outlier with the highest or lowest value in the dataset.

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

### **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.

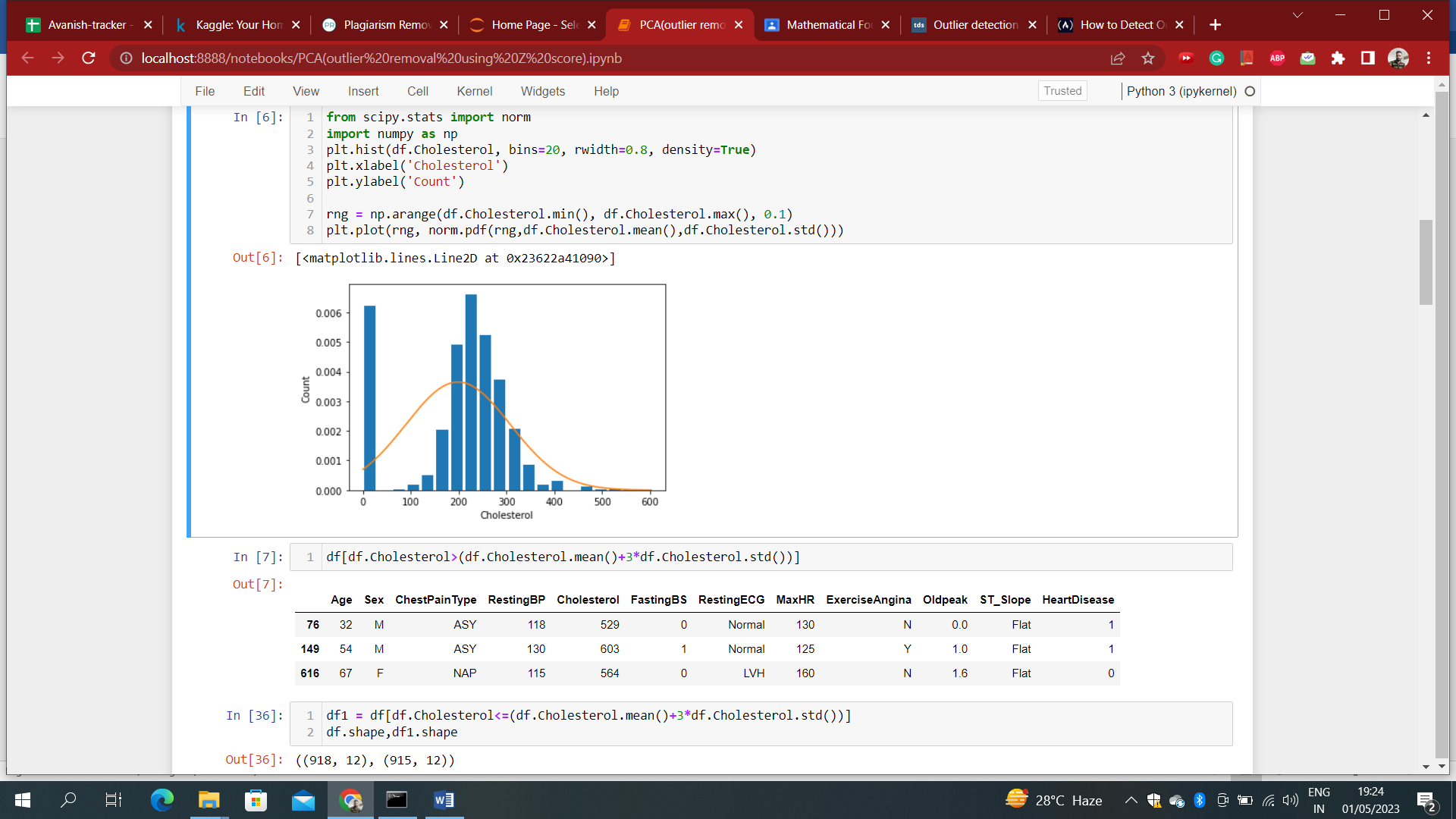
In statistics, standard deviation measures the spread of data around the mean, and in essence, it captures how far away from the mean the data points are.

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.

### **3. Using Interquartile Range (IQR):**

You can use the box plot, or the box and whisker plot, to explore the dataset and visualize the presence of outliers. The points that lie beyond the whiskers are detected as outliers.You can generate box plots in Seaborn using the boxplot function.

sns.boxplot(data=scores\_data).set(title="Box Plot of Scores")

df\_scores.describe()

# Output

score

count 200.000000

mean 61.005000

std 11.854434

min 20.000000

25% 54.000000

50% 62.000000

75% 67.000000

max 98.000000

IQR = 67-54

lower\_limit = 54 - 1.5\*IQR

upper\_limit = 67 + 1.5\*IQR

print(upper\_limit)

print(lower\_limit)

# Output

86.5

34.5

df\_scores\_filtered = df\_scores[(df\_scores['score']>lower\_limit) & (df\_scores['score']<upper\_limit)]

print(df\_scores\_filtered)

### **4. Using Percentile:**

Let’s define a custom range that accommodates all data points that lie anywhere between 0.5 and 99.5 percentile of the dataset. To do this, set q = [0.5, 99.5] in the percentile function, as shown below.

lower\_limit, upper\_limit = np.percentile(a=df\_scores,q=[0.5,99.5])

print(upper\_limit)

print(lower\_limit)

df\_scores\_filtered = df\_scores[(df\_scores['score']>lower\_limit) & (df\_scores['score']<upper\_limit)]

print(df\_scores\_filtered)