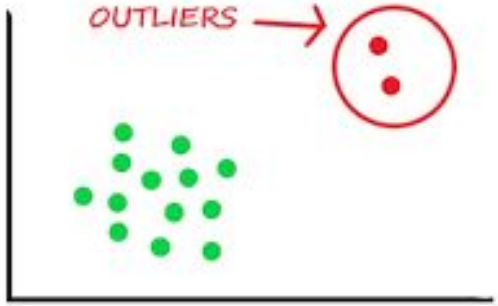
Big Data Analysis

How to Remove Outliers in Machine Learning

- Teaching Assistant
Yash More

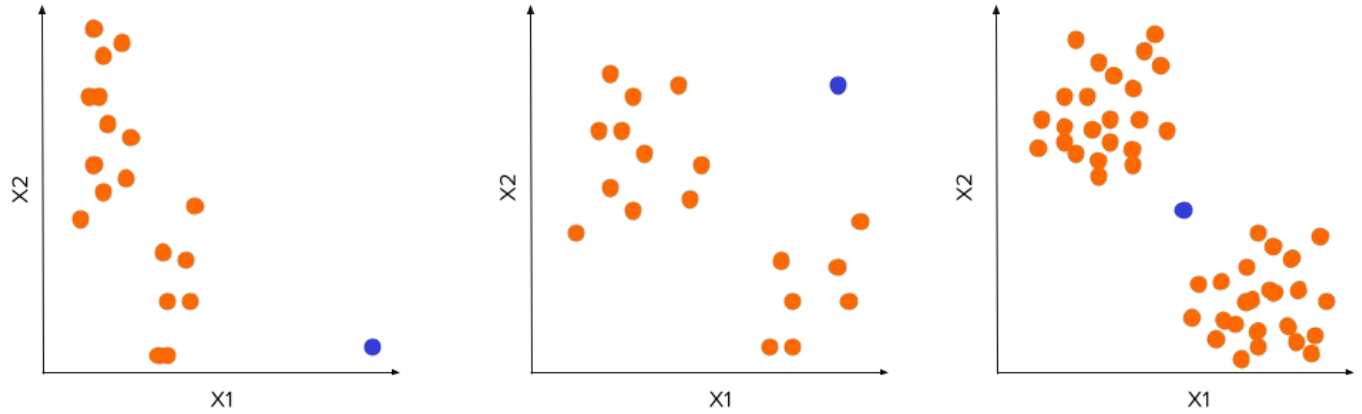
Outliers :

- Data points that stand out from the rest of the dataset are known as outliers. The data distribution is frequently skewed by these anomalous observations, which are frequently the result of inaccurate observations or incorrect data entry.
- It's crucial to find and eliminate outliers in order to guarantee that the trained model generalises well to the acceptable range of test inputs.`



Why Should You Detect Outliers?

- Data preparation and cleaning is a crucial stage in the machine learning pipeline since it makes the data easier to interpret. You deal with missing values, find outliers, and more in this step.
- The presence of outliers can frequently affect the findings of statistical studies performed on the dataset since they have highly distinct values—abnormally low or abnormally high. This can result in models that are less effective and useful.



a) Odd number for feature X1

b) Multivariate outlier

c) Outlier in a low density region

Why Should You Detect Outliers?

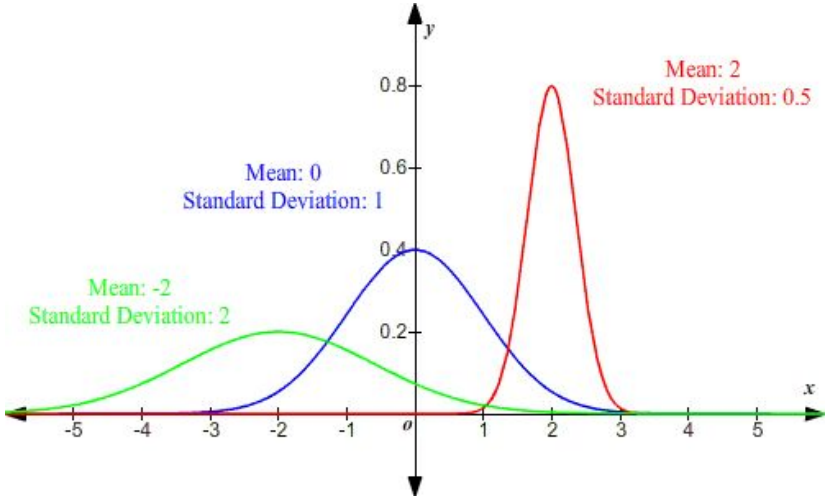
- However, handling outliers frequently calls for domain knowledge, and none of the outlier detection strategies should be used without first comprehending the data distribution and the use case.
- **Example :**

  In a dataset of home price data, for instance, if you spot a few homes with prices around $1.5 million, which is significantly more than the median home price, they are probably outliers. However, if the information includes a disproportionately large number of homes valued at $1 million or more, they might be a hint of an upward trend in home values. As a result, classifying them as all outliers would be erroneous. You need some familiarity with the real estate industry in this situation.

- To create a model that works effectively on test data that hasn't been seen before, outlier identification aims to eliminate the points that are actually outliers. We'll go over a few methods for finding outliers in data.

How to Detect Outliers Using Standard Deviation

- You can use the data's standard deviation or the corresponding z-score to identify outliers when the data or certain dataset attributes exhibit a normal distribution.
- Standard deviation, which analyses the distribution of data around the mean in statistics, essentially indicates how much the data points deviate from the mean.



Code for Outlier Detection Using Standard Deviation

```
import numpy as np
import pandas as pd
import seaborn as sns
```

Important libraries

Code for Outlier Detection Using Standard Deviation

- Seaborn is a data visualization library in Python that is built on top of the popular plotting library, Matplotlib. Seaborn provides a higher-level interface for creating statistical graphics and is particularly useful for creating informative and attractive visualizations of complex data.

- Seaborn offers many advantages over Matplotlib, including:

    - Better default styles and color palettes for plots.
    - Built-in support for many types of plots, including heatmaps, time series, and categorical plots.
    - Simplified and improved syntax for creating complex plots.
    - Integration with Pandas, making it easy to visualize data stored in dataframes.

In summary, Seaborn is used in data analysis and visualization tasks to create more visually appealing and informative plots and to simplify the process of creating complex visualizations.

Code for Outlier Detection Using Standard Deviation

- Let's now construct the function generate scores(), which produces a 200-record properly distributed collection of student scores.
- We'll make a call to the method and put the array that is returned in the scores data field.

```
def generate_scores(mean=60,std_dev=12,num_samples=200):
      np.random.seed(27)
      scores = np.random.normal(loc=mean,scale=std_dev,size=num_samples)
      scores = np.round(scores, decimals=0)
      return scores
scores_data = generate_scores()
```

```
def generate_scores(mean=60,std_dev=12,num_samples=200):
      np.random.seed(27)
      scores = np.random.normal(loc=mean,scale=std_dev,size=num_samples)
      scores = np.round(scores, decimals=0)
      return scores
scores_data = generate_scores()
```

- The given code defines a function called generate_scores which generates a set of normally distributed random scores for a given mean, standard deviation, and number of samples. The function takes three arguments:

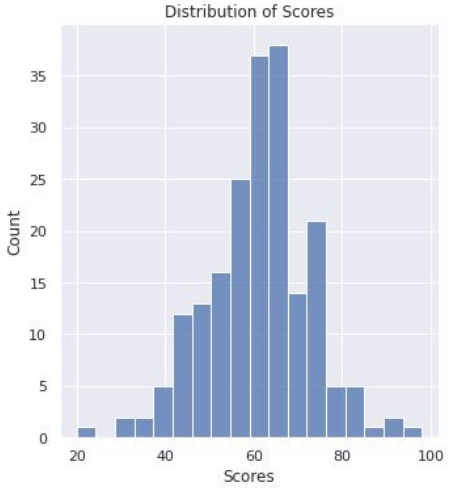
    - **mean (default value 60)** **:** the mean of the normal distribution used to generate the scores.
    - **std_dev (default value 12)** **:** the standard deviation of the normal distribution used to generate the scores.
    - **num_samples (default value 200)** **:** the number of scores to generate.

- The function uses np.random.normal function from the NumPy library to generate num_samples random scores. The loc parameter is used to specify the mean of the normal distribution, while the scale parameter is used to specify the standard deviation. The np.round function is used to round the generated scores to the nearest integer.
- After defining the generate_scores function, the code calls the function without any arguments and assigns the returned scores to a variable called scores_data. This means that scores_data will contain an array of 200 randomly generated scores with a mean of 60 and a standard deviation of 12 (unless the default values were overridden by passing different arguments to generate_scores).

Code for Outlier Detection Using Standard Deviation

- To see how the data is distributed, use Seaborn's displot() method. As can be seen in the image below, the dataset in this instance has a normal distribution.

```
sns.set_theme()
sns.displot(data=scores_data).set(title="Distribution of Scores", xlabel="Scores")
```



Code for Outlier Detection Using Standard Deviation

- Next, let's load the data into a Pandas dataframe for further analysis.

```
df_scores = pd.DataFrame(scores_data,columns=['score'])
```

- To obtain the mean and standard deviation of the data in the dataframe df_scores, you can use the .mean() and the .std() methods, respectively.

```
df_scores.mean()

# Output
score    61.005
dtype: float64

df_scores.std()

# Output
score    11.854434
dtype: float64
```

Code for Outlier Detection Using Standard Deviation

- As discussed earlier, set the lower limit (`lower_limit`) to be three standard deviations below the mean, and the upper limit (`upper_limit`) to be three standard deviations above the mean.

```
lower_limit = df_scores.mean() - 3*df_scores.std()
upper_limit = df_scores.mean() + 3*df_scores.std()
print(lower_limit)
print(upper_limit)

#=============================================================================================
# Output
25.530716709142666
96.47928329085734
```

Code for Outlier Detection Using Standard Deviation

```
df_scores_filtered=df_scores[(df_scores[['score']]>lower_limit)&(df_scores[['score']]<upper_limit)]
print(df_scores_filtered)

# ============================================================
# Output
score
0     75.0
1     56.0
2     67.0
3     65.0
4     63.0
..     ...
194   42.0
195   76.0
196   67.0
197   74.0
199   53.0
[198 rows x 1 columns]
```

Code for Outlier Detection Using Standard Deviation

- Now that you've defined the lower and upper limits, you may filter the dataframe df_scores to only retain the data points in the interval [lower_limit, upper_limit], as shown below.

```
df_scores_filtered=df_scores[(df_scores['score']>lower_limit)&(df_scores['score']<upper_limit)]
print(df_scores_filtered)

# ============================================================
# Output
score
0     75.0
1     56.0
2     67.0
3     65.0
4     63.0
..     ...
194   42.0
195   76.0
196   67.0
197   74.0
199   53.0
[198 rows x 1 columns]
```

- From the output above, you can see that two records have been removed, and df_scores_filtered contains 198 records.

How to Detect Outliers Using the 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:

$$z = (x - \mu)/\sigma$$

Code for Outlier Detection Using Z-Score

- Let's compute z-scores for all points in the dataset, and add z_score as a column to the dataframe df_scores.

```
df_scores['z_score']=(df_scores['score'] - df_scores['score'].mean())/df_scores['score'].std()
df_scores.head()

# ================================================
# Output
score z_score
0     75.0  1.180571
1     56.0  -0.422205
2     67.0  0.505718
3     65.0  0.337005
4     63.0  0.168291
```

- You can filter the dataframe df_scores to retain points whose z-scores are in the range [-3, 3], as shown below.
- The filtered dataframe contains 198 records, as expected.

Code for Outlier Detection Using Z-Score

- Let's compute z-scores for all points in the dataset, and add z_score as a column to the dataframe df_scores.

```
df_scores_filtered= df_scores[(df_scores['z_score']>-3) & (df_scores['z_score']<3)]

print(df_scores_filtered)
```

- 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.

How to Detect Outliers Using the Interquartile Range (IQR)

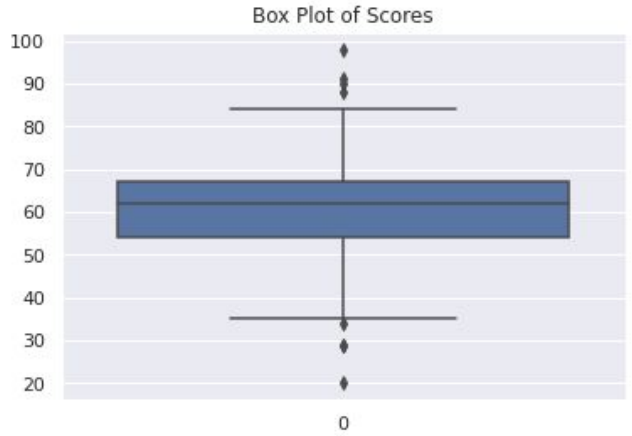
In statistics, interquartile range or IQR is a quantity that measures the difference between the first and the third quartiles in a given dataset.

- The first quartile is also called the one-fourth quartile, or the 25% quartile.
- If q25 is the first quartile, it means 25% of the points in the dataset have values less than q25.
- The third quartile is also called the three-fourth, or the 75% quartile.
- If q75 is the three-fourth quartile, 75% of the points have values less than q75.
- Using the above notations, IQR = q75 - q25.

Code for Outlier Detection 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")
```



Code for Outlier Detection Using Interquartile Range (IQR)

- Now, call the describe method on the dataframe df_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
```

Code for Outlier Detection Using Interquartile Range (IQR)

- We use the 25% and 75% quartile values from the above result to compute IQR, and subsequently set the lower and upper limits to filter df_scores.

```
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
```

Code for Outlier Detection Using Interquartile Range (IQR)

- As a next step, filter the dataframe df_scores to retain records that lie in the permissible range.

```
df_scores_filtered = df_scores[(df_scores['score']>lower_limit) & (df_scores['score']<upper_limit)]
print(df_scores_filtered)

# ============================================================
# Output
score
0     75.0
1     56.0
2     67.0
3     65.0
4     63.0
..     ...
194   42.0
195   76.0
196   67.0
197   74.0
199   53.0
[192 rows x 1 columns]
```

Code for Outlier Detection Using Interquartile Range (IQR)

- As seen in the output, this method labels eight points as outliers, and the filtered dataframe is 192 records long.
- You don't always have to call the describe method to identify the quartiles. You may instead use the percentile() function in NumPy. It takes in two arguments, a: an array or a dataframe and q: a list of quartiles.
- The code cell below shows how you can calculate the first and the third quartiles using the percentile function.

```
q25,q75 = np.percentile(a = df_scores,q=[25,75])
IQR = q75 - q25
print(IQR)

# ======================================================
# Output
13.0
```

How to Detect Outliers Using Percentile :

- In the previous section, we explored the concept of interquartile range, and its application to outlier detection. You can think of percentile as an extension to the interquartile range.
- As discussed earlier, the interquartile range works by dropping all points that are outside the range

  [q25 - 1.5*IQR, q75 + 1.5*IQR] as outliers.

- But removing outliers this way may not be the most optimal choice when your observations have a wide distribution. And you may be discarding more points—than you actually should—as outliers.
- Depending on the domain, you may want to widen the range of permissible values to estimate the outliers better. Next, let’s revisit the scores dataset, and use percentile to detect outliers.

Code for Outlier Detection 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)

# ============================================================
# Output
91.035
28.955
```

Code for Outlier Detection Using Percentile :

- Next, you may filter the dataframe using the lower and upper limits obtained from the previous step.

```
df_scores_filtered = df_scores[(df_scores['score']>lower_limit) & (df_scores['score']<upper_limit)]
print(df_scores_filtered)

# ================================================================
# Output
score
0     75.0
1     56.0
2     67.0
3     65.0
4     63.0
..     ...
194   42.0
195   76.0
196   67.0
197   74.0
199   53.0
[198 rows x 1 columns]
```

Code for Outlier Detection Using Percentile :

```
df_scores_filtered = df_scores[(df_scores['score']>lower_limit) & (df_scores['score']<upper_limit)]
print(df_scores_filtered)

# ================================================================
# Output
score
0     75.0
1     56.0
2     67.0
3     65.0
4     63.0
..     ...
194   42.0
195   76.0
196   67.0
197   74.0
199   53.0
[198 rows x 1 columns]
```

- From the code cell above, you can see that there are two outliers, and the filtered dataframe has 198 data records.

- Full code link :

  https://www.freecodecamp.org/news/how-to-detect-outliers-in-machine-learning/

Thank You!!