BA870 – Lecture 4 – Part B

Detecting and Handling Outliers with Pandas

Adapted from: https://hersanyagci.medium.com/detecting-and-handling-outliers-with-pandas-7adbfcd5cad8

Data analysis is a long process. There are some steps to do this. First of all, we need to recognize the data. We have to know every feature in the dataset. Then we must detect the missing values and clear our dataset from these NaN values. We can fill these NaN values with some values (mean, median, etc.) or we can create our function to fill these missing values. We can also drop some columns that are not helpful or have more NaN values than others.

This process can change. It depends on the data and target. But we must finally fight the outliers. We have to detect and handle them. Each data has different types of outliers, whether they are within 1.5 IQR or not. Sometimes these outliers aren't harmful, so we don't deal with them. But if we want to get good results in models or our analysis, we need to handle outliers. There are 3 commonly used methods to deal with outliers.

1. Dropping the outliers.

2. Winsorize method.

3. Log transformation.

Let's look at these methods with Python,

In this demo, we will use the Seaborn diamonds dataset.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:
```
import seaborn as sns
df = sns.load_dataset('diamonds')
df = df.dropna()
```

In [6]:
```
df.head()
```

Out[6]:

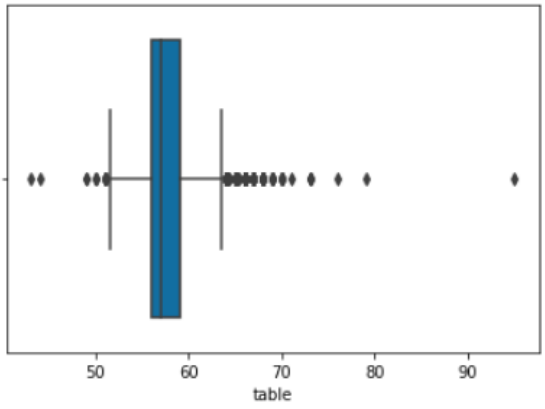
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [5]:
```
df['table']
```

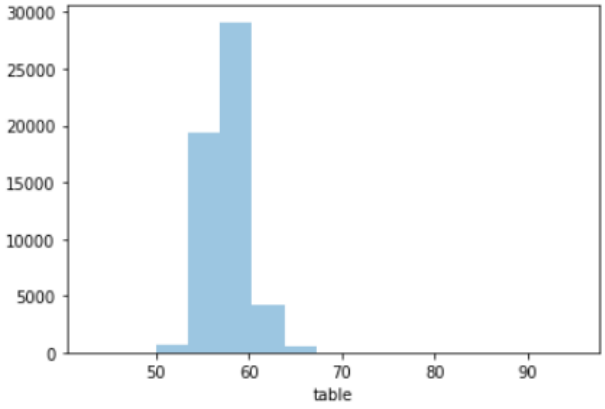
Out[5]:
```
0        55.0
1        61.0
2        65.0
3        58.0
4        58.0
         ...
53935    57.0
53936    55.0
53937    60.0
53938    58.0
53939    55.0
Name: table, Length: 53940, dtype: float64
```

We will handle the table feature of the diamonds dataset and assume all NaN values have been processed (we just dropped them). Let's look at the graphs boxplot and histogram.

In [7]:
```
1 sns.boxplot(x = df['table'])
2 plt.show()
```



In [8]:
```
1 sns.distplot(df['table'], bins = 15, kde = False)
2 plt.show()
```



As you can see this column has outliers (it is shown at boxplot) and it is right-skewed data(it is easily seen at histogram). Boxplot is the best way to see outliers.

Before handling outliers, we will detect them. We will use Tukey's rule to detect outliers. It is also known as the IQR rule. First, we will calculate the Interquartile Range of the data (IQR = Q3 — Q1). Later, we will determine our outlier boundaries with IQR.

We will get our lower boundary with this calculation Q1–1.5 * IQR. We will get our upper boundary with this calculation Q3 + 1.5 * IQR.

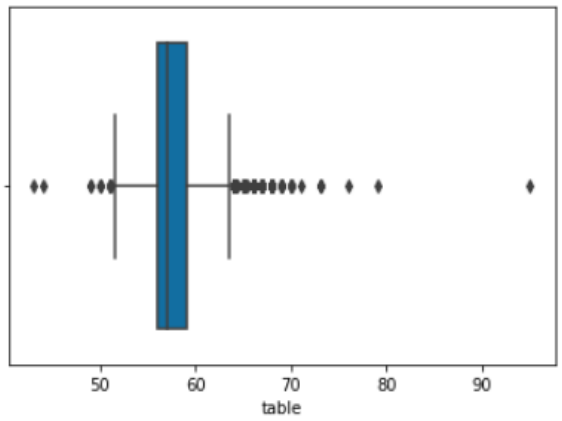
According to this rule, the data between boundaries are acceptable but the data outside of the between lower and upper boundaries are outliers. We can use 2.5 or 2 to detect IQR. It depends on our data and analysis. But the most commonly used is 1.5 and we will use 1.5 IQR in this analysis.

In [11]:
```
df[['table']].describe()
```

Out[11]:

	table
count	53940.000000
mean	57.457184
std	2.234491
min	43.000000
25%	56.000000
50%	57.000000
75%	59.000000
max	95.000000

In [12]:
```
sns.boxplot(x = df['table'])
plt.show()
```



With the describe method of pandas, we can see our data's Q1 (%25) and Q3 (%75) percentiles. We can calculate our IQR point and boundaries (with 1.5).

```
In [15]: df[['table']].describe()
```

Out[15]:

	table
count	53940.000000
mean	57.457184
std	2.234491
min	43.000000
25%	56.000000
50%	57.000000
75%	59.000000
max	95.000000

```
In [14]: df['table'].quantile(0.25)
Out[14]: 56.0

In [16]: df['table'].quantile(0.75)
Out[16]: 59.0

In [17]: Q1 = df['table'].quantile(0.25)
         Q3 = df['table'].quantile(0.75)
         IQR = Q3 - Q1

In [18]: IQR
Out[18]: 3.0

In [19]: lower_lim = Q1 - 1.5 * IQR
         upper_lim = Q3 + 1.5 * IQR

In [20]: lower_lim
Out[20]: 51.5

In [21]: upper_lim
Out[21]: 63.5
```

Our upper boundary is 63.5 and our lower boundary is 51.5. This means that these values between 51.5 and 63.5 are acceptable but those outside mean there are outliers. So we need to handle them because they corrupt our data.

Let’s handle outliers.

1 — Dropping the outliers;

We can easily remove outliers, but this narrows our data. If we have a lot of rows, big data, maybe we can take risks. But remember, if we drop the value, we delete all records (row). If we have vulnerable records, they can get lost.

```
In [23]:  1  outliers_15_low = (df['table'] < lower_lim)

In [24]:  1  outliers_15_up = (df['table'] > upper_lim)

In [25]:  1  len(df['table']) - (len(df['table'][outliers_15_low]) + len(df['table'][outliers_15

Out[25]: 53335

In [27]:  1  df['table'][(outliers_15_low | outliers_15_up)]

Out[27]: 2        65.0
         91       69.0
         145      64.0
         219      64.0
         227      67.0
                  ...
         53695    65.0
         53697    65.0
         53756    64.0
         53757    64.0
         53785    65.0
         Name: table, Length: 605, dtype: float64

In [28]:  1  df['table'][~(outliers_15_low | outliers_15_up)]

Out[28]: 0        55.0
         1        61.0
         3        58.0
         4        58.0
         5        57.0
                  ...
         53935    57.0
         53936    55.0
         53937    60.0
         53938    58.0
         53939    55.0
         Name: table, Length: 53335, dtype: float64
```

We make some equations to reach outliers and index. As you can see, if we drop outliers, we will lose 605 records. Line 27 shows us the outliers. Line 28 shows us the data without outliers. Check the length of the tables. In the beginning, we had 53940 rows.

In [30]:
```
1 df
```

Out[30]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [33]:
```
1 df = df[~(outliers_15_low | outliers_15_up)]
```

In [34]:
```
1 df
```

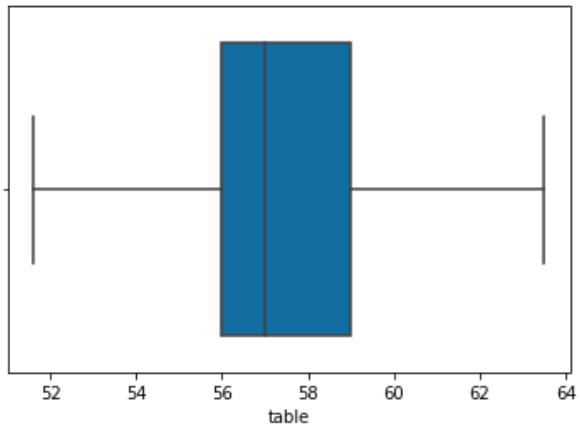
Out[34]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	0.23	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	0.21	326	3.89	3.84	2.31
3	0.29	Premium	I	VS2	62.4	0.29	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	0.31	335	4.34	4.35	2.75
5	0.24	Very Good	J	VVS2	62.8	0.24	336	3.94	3.96	2.48
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	0.72	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	0.72	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	0.7	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	0.86	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	0.75	2757	5.83	5.87	3.64

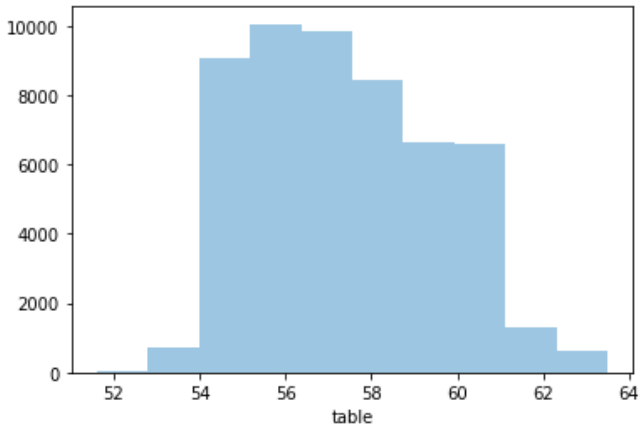
53335 rows × 10 columns

We removed the outliers and our data rows drop to 53335. After dropping outliers, let's check the boxplot and histogram of our data.

In [35]:
```
sns.boxplot(x = df['table'])
plt.show()
```



In [36]:
```
sns.distplot(df['table'], bins = 10, kde = False)
plt.show()
```



Now, we don’t have any outliers.

2 — Winsorize Method;

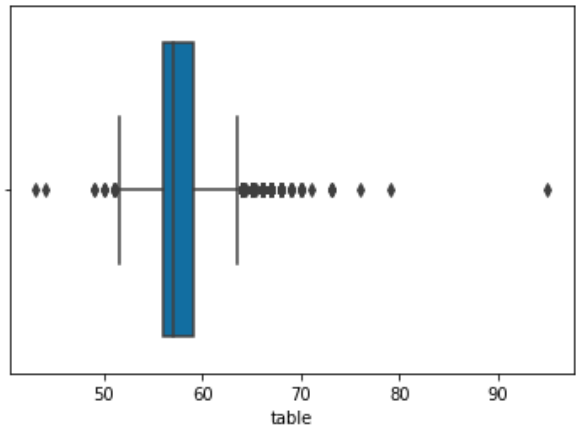
Our second method is the Winsorize Method. In the Winsorize Method, we limit outliers with an upper and lower limit. We will set the limits. We will make our upper and lower limits for data our new maximum and minimum points.

We will use the table column of the diamonds dataset again. Let’s check the boxplot again.

```
In [39]:  1  df['table']
```

```
Out[39]: 0        55.0
         1        61.0
         2        65.0
         3        58.0
         4        58.0
                  ...
         53935    57.0
         53936    55.0
         53937    60.0
         53938    58.0
         53939    55.0
         Name: table, Length: 53940, dtype: float64
```

```
In [40]:  1  sns.boxplot(x = df['table'])
          2  plt.show()
```



We have outliers, we detected them at the beginning. For outliers, our upper limit is 63.5 and our lower limit is 51.5.

For the Winsorize Method, we have to import winsorize from Scipy. We need boundaries to apply winsorize. We will limit our data between 53 and 63. These values are within limits for outliers. We need precise points of these values in the percentile and we can use the quantile method of Pandas.

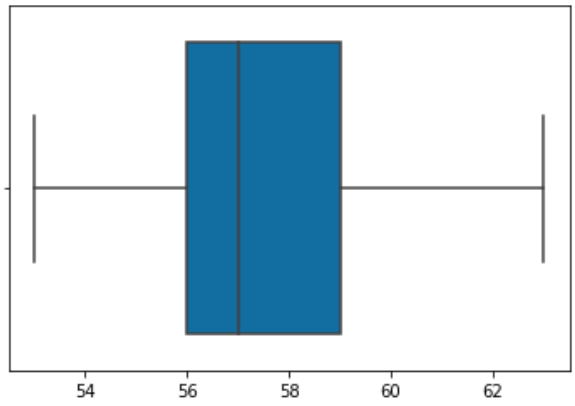
```
In [41]:
1 from scipy.stats.mstats import winsorize

In [42]:
1 upper_lim
Out[42]: 63.5

In [43]:
1 lower_lim
Out[43]: 51.5

In [44]:
1 df['table'].quantile(0.01)
Out[44]: 53.0

In [45]:
1 df['table'].quantile(0.98)
Out[45]: 63.0
```

We will create a new variable with Winsorize Method. To implement the Winsorize Method, we write the exact boundary points as a tuple on the percentile. For example, we will write (0.01, 0.02). This means we want to apply quantile(0.01) and quantile(0.98) as a boundary. The first one is the exact point on percentile from the beginning, the second one is exact point on percentile from the end.

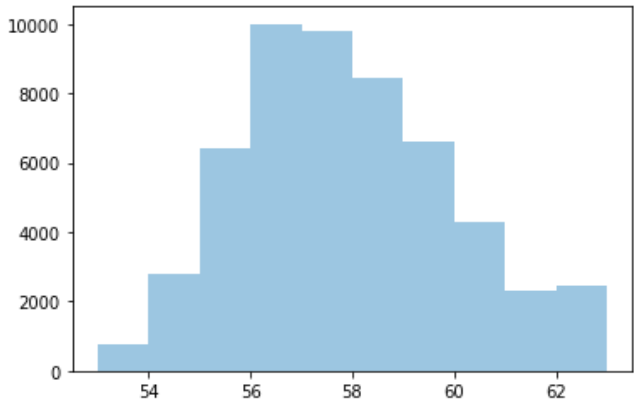
```
In [46]:
1 df_table_win = winsorize(df_table, (0.01, 0.02))

In [47]:
1 df_table_win
Out[47]: masked_array(data=[55., 61., 63., ..., 60., 58., 55.],
                      mask=False,
                fill_value=1e+20)
```

We applied Winsorize Method, let’s check data on graphs.

```
In [48]: 1 sns.boxplot(x = df_table_win)
Out[48]: <AxesSubplot:>
```




```
In [49]: 1 sns.distplot(df_table_win, bins = 10, kde =False)
Out[49]: <AxesSubplot:>
```



As you can see, there are no outliers. Focus on this, our maximum and minimum values on the boxplot; 53 and 63. We applied them as boundaries. Now, we can look at descriptive statistical values of old and new data with the describe method.

```
In [50]: 1 df_table_win = pd.DataFrame(df_table_win)[0]

In [51]: 1 df_table_win.describe()

Out[51]: count    53940.000000
         mean        57.434607
         std          2.142774
         min         53.000000
         25%         56.000000
         50%         57.000000
         75%         59.000000
         max         63.000000
         Name: 0, dtype: float64

In [53]: 1 df['table'].describe()

Out[53]: count    53940.000000
         mean        57.457184
         std          2.234491
         min         43.000000
         25%         56.000000
         50%         57.000000
         75%         59.000000
         max         95.000000
         Name: table, dtype: float64
```

I changed df_table_win to a series to implement the describe method. Notice that, the mean and standard deviation of df_table_win have changed. Minimum and maximum points also changed but these changes do not affect median-logic descriptive statistical values. Therefore, we should apply the Winsorize Method carefully because as you can see, mean-logic descriptive statistical values can change. It can corrupt our data, damage our analysis, or have adverse effects on models.

3 — Log Transformation;

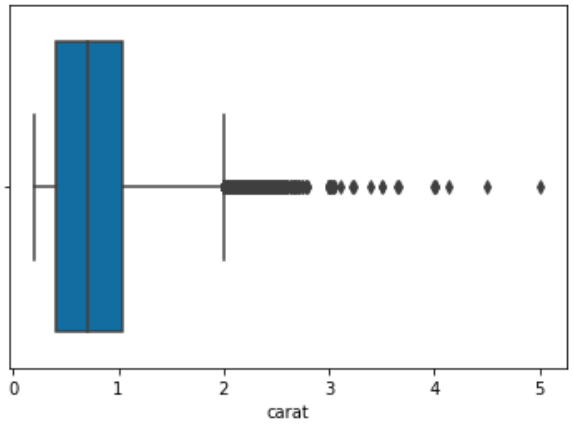
Our last method is Log Transformation. We use log transformation on skewed data. Log transformation reduces the skewness of data and tries to make it normal. Log transformation doesn't always make it normal, sometimes makes data more skewed. So it depends on the data. We have to apply transformation and control the result.

For this method, we will use the carat column of diamonds dataset. Let's check the data and graphs.

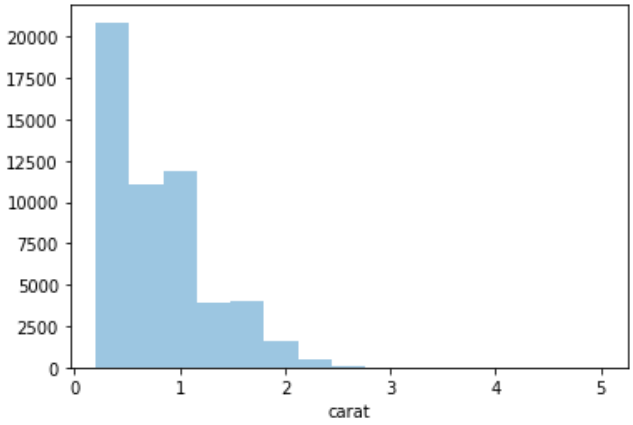
```
In [62]: 1 df['carat'].head()
```

```
Out[62]: 0    0.23
         1    0.21
         2    0.23
         3    0.29
         4    0.31
         Name: carat, dtype: float64
```

```
In [63]: 1 sns.boxplot(x = df['carat']);
         2
```

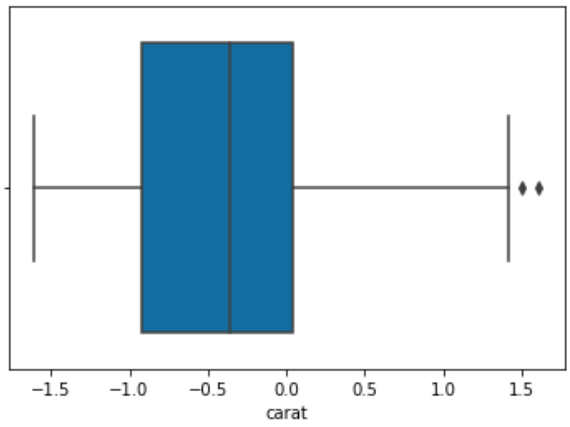



```
In [64]: 1 sns.distplot(df['carat'], bins = 15, kde = False);
```



There are many outliers and the data is right-skewed. Log transformation will transform data to normal or close to normal. Let's apply the log transformation to reduce the variability of data.

```
In [71]:  1  import numpy as np

In [72]:  1  df_carat_log = np.log(df['carat'])

In [73]:  1  df['carat'].head()

Out[73]: 0    0.23
         1    0.21
         2    0.23
         3    0.29
         4    0.31
         Name: carat, dtype: float64

In [74]:  1  df_carat_log.head()

Out[74]: 0   -1.469676
         1   -1.560648
         2   -1.469676
         3   -1.237874
         4   -1.171183
         Name: carat, dtype: float64

In [75]:  1  sns.boxplot(x = df_carat_log)

Out[75]: <AxesSubplot:xlabel='carat'>
```



We implemented log transformation from NumPy with np.log. It completely changed our data and it removed outliers, we can see this in the boxplot.

Log transformation is commonly used for machine learning algorithms. Be careful, it changes our values but removes outliers. It makes our model normal and the machine learning algorithm likes normal distribution data. There are some methods and features in machine learning algorithms such as scaling and normalization. We will talk about these terms in our next stories.

Conclusion

It takes a long time to deal with outliers but it's worth it. We have to choose the best method for the data. It is important to know the data very well. With this domain knowledge, we will decide

on the method of handling outliers. Apart from these methods, we can consider these outliers as missing values. We can use the methods of filling missing values to get rid of outliers.

More on finding outliers in dataset using python

Use z score and IQR -interquartile range to identify any outliers using python

What is an outlier?

An outlier is a data point in a data set that is distant from all other observations. A data point that lies outside the overall distribution of the dataset.

What are the criteria to identify an outlier?

- Data point that falls outside of 1.5 times of an interquartile range above the 3rd quartile and below the 1st quartile
- Data point that falls outside of 3 standard deviations. we can use a z score and if the z score falls outside of 2 standard deviation

What is the reason for an outlier to exists in a dataset?

An outlier could exist in a dataset due to

- Variability in the data
- An experimental measurement error

What is the impact of an outlier?

causes serious issues for statistical analysis

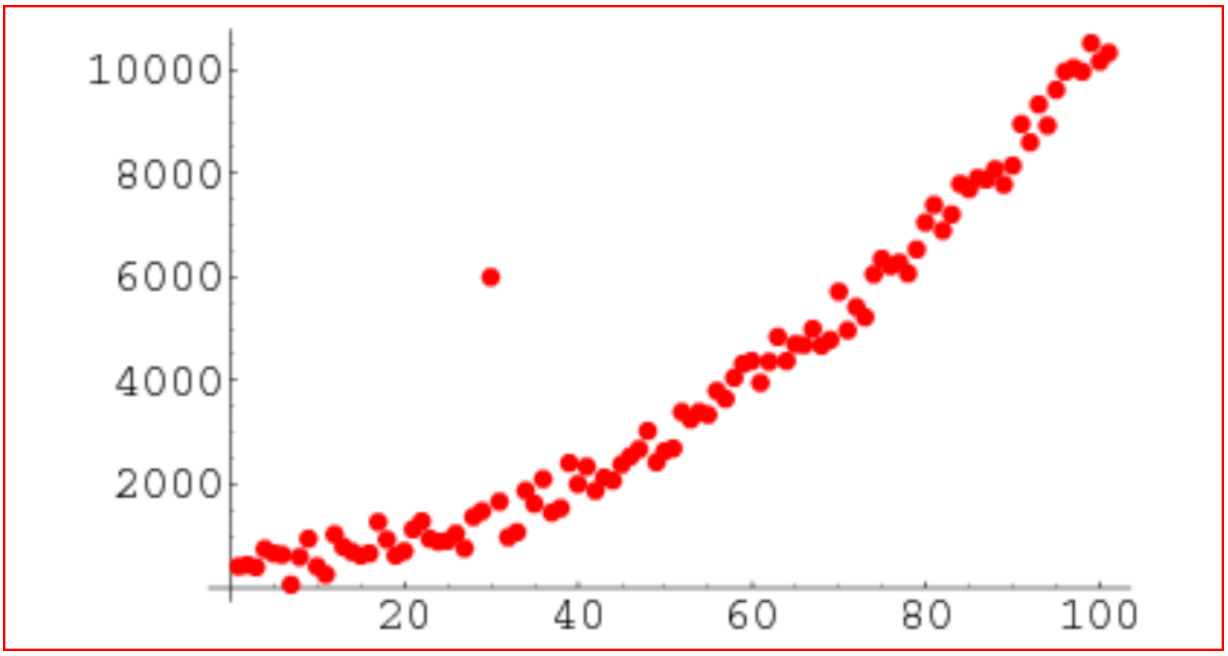
- skew the data,
- significant impact on mean
- significant impact on standard deviation.

How can we identify an outlier?

- using scatter plots
- using Z score
- using the IQR interquartile range

Using Scatter Plot

We can see the scatter plot and it shows us if a data point lies outside the overall distribution of the dataset



Scatter plot to identify an outlier

Using Z score

Formula for Z score = (Observation — Mean)/Standard Deviation

$z = (X — \mu) / \sigma$

we first import the libraries

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

we will use a list here

```
dataset= [10,12,12,13,12,11,14,13,15,10,10,10,100,12,14,13, 12,10,
10,11,12,15,12,13,12,11,14,13,15,10,15,12,10,14,13,15,10]
```

we write a function that takes numeric data as an input argument.

we find the mean and standard deviation of the all the data points

We find the z score for each of the data point in the dataset and if the z score is greater than 3 than we can classify that point as an outlier. Any point outside of 3 standard deviations would be an outlier.

```
import numpy as np
import pandas as pd
outliers=[]
def detect_outlier(data_1):

    threshold=3
    mean_1 = np.mean(data_1)
    std_1 =np.std(data_1)


    for y in data_1:
        z_score= (y - mean_1)/std_1
        if np.abs(z_score) > threshold:
            outliers.append(y)
    return outliers
```

we now pass dataset that we created earlier and pass that as an input argument to the detect_outlier function

```
outlier_datapoints = detect_outlier(dataset)
print(outlier_datapoints)
```

```
[100]
```

output of the outlier_datapoints

Using IQR

IQR tells how spread the middle values are. It can be used to tell when a value is too far from the middle.

An outlier is a point which falls more than 1.5 times the interquartile range above the third quartile or below the first quartile.

we will use the same dataset

step 1:

- Arrange the data in increasing order
- Calculate first(q1) and third quartile(q3)
- Find interquartile range (q3-q1)
- Find lower bound q1*1.5
- Find upper bound q3*1.5
- Anything that lies outside of lower and upper bound is an outlier

Fist sorting the dataset

```
sorted(dataset)
```

Finding first quartile and third quartile

```
q1, q3= np.percentile(dataset,[25,75])
```

q1 is 11 and q3 is 14

Find the IQR which is the difference between third and first quartile

```
iqr = q3 - q1
```

iqr is 3

Find lower and upper bound

```
lower_bound = q1 -(1.5 * iqr)
upper_bound = q3 +(1.5 * iqr)
```

lower_bound is 6.5 and upper bound is 18.5, so anything outside of 6.5 and 18.5 is an outlier.