Outliers

- An outlier is a data point in a data set that is distant from all other observations, which is significantly different from the remaining data.
- A data point that lies outside the overall distribution of the dataset.

What are the impacts of having outliers in a dataset?

1. It causes various problems during our statistical analysis
2. It may cause a significant impact on the mean and the standard deviation Statistics such as the mean and variance are very susceptible to outliers. In addition, **some Machine Learning models are sensitive to outliers** which may decrease their performance. Thus, depending on which algorithm we wish to train, we often remove outliers from our variables.

Reasons for Outliers

1. Data Entry Errors (Ex: Entering salary as 1,00,000 instead of 10,000)
2. Measurement Errors (Ex: Measuring in meters instead of KM)

Types of Outliers

1. Univariate Outliers --> Indentifing outlier for single variable
2. Bivariate Outliers --> Indentified as outlier by analyzing 2 variables at a time

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
dataset= [11,10,12,14,12,15,14,13,15,102,12,14,17,19,107, 10,13,12,14,12,108,12,1
```

Various ways of finding the outlier.

1. Z score
   - |Zscore| > 2
2. IQR
   - Value < Q1-1.5IQR
   - Value > Q3+1.5IQR
3. Visualization
   - Box plot, Histogram for univariate outliers
   - Scatter plots for bivariate outliers

1. Z score

- Z score = (X — μ) / σ

In [3]:
```
outliers=[]

def detect_outliers(data):
    for i in data:
        z_score= (i - np.mean(data))/np.std(data)
        if np.abs(z_score) > 2:
            outliers.append(i)
    return outliers
```

In [4]:
```
detect_outliers(dataset)
```

Out[4]:
```
[102, 107, 108]
```

In [5]:
```
l=[]

for x in dataset:
    z_score= (x - np.mean(dataset))/np.std(dataset)
    l.append(z_score)

print(l)
```

```
[-0.38587723217963826, -0.4237958041279264, -0.3479586602313501, -0.27212151633
477377, -0.3479586602313501, -0.23420294438648565, -0.27212151633477377, -0.310
04008828306195, -0.23420294438648565, 3.064712815114584, -0.3479586602313501, -
0.27212151633477377, -0.15836580048990934, -0.08252865659333301, 3.254305674856
025, -0.4237958041279264, -0.31004008828306195, -0.3479586602313501, -0.2721215
1633477377, -0.3479586602313501, 3.292224246804313, -0.3479586602313501, -0.385
87723217963826, -0.27212151633477377, -0.31004008828306195, -0.2342029443864856
5, -0.4237958041279264, -0.23420294438648565, -0.3479586602313501, -0.423795804
1279264, -0.27212151633477377, -0.31004008828306195, -0.23420294438648565, -0.4
237958041279264]
```

2. InterQuantile Range

Steps

- 1.Arrange the data in increasing order
- 2.Calculate first(q1) and third quartile(q3)
- 3.Find interquartile range (q3-q1)
- 4.Find lower bound q1*1.5
- 5.Find upper bound q3*1.5

Anything that lies outside of lower and upper bound is an outlier

In [6]:
```
## Perform all the steps of IQR
sorted(dataset)
```

Out[6]:
```
[10,
 10,
 10,
 10,
 10,
 11,
 11,
 12,
 12,
 12,
 12,
 12,
 12,
 12,
 12,
 13,
 13,
 13,
 13,
 14,
 14,
 14,
 14,
 14,
 14,
 14,
 15,
 15,
 15,
 15,
 15,
 15,
 17,
 19,
 102,
 107,
 108]
```

In [7]:
```
q1 = np.percentile(dataset,25)
q3 = np.percentile(dataset,75)
print(q1,q3)
```

```
12.0 15.0
```

In [8]:
```
## Find the IQR
iqr=q3-q1
print(iqr)
```

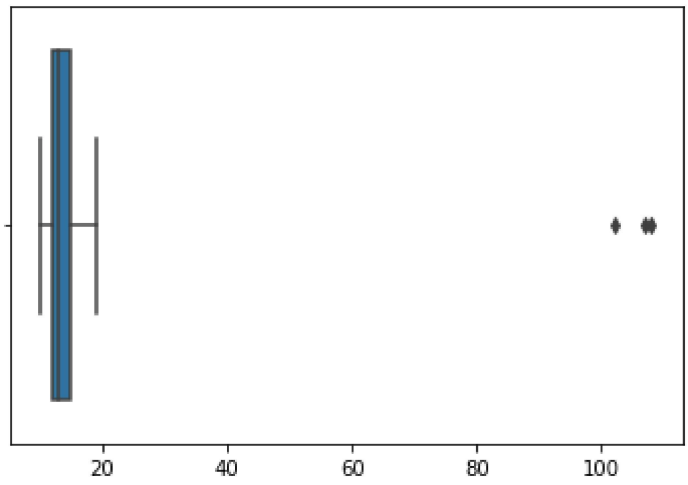
```
3.0
```

In [9]:
```
## Find the lower bound value and the higher bound value
lower_bound_val = q1 -(1.5 * (iqr))
upper_bound_val = q3 +(1.5 * (iqr))
print(lower_bound_val,upper_bound_val)
```

```
7.5 19.5
```

3. Boxplot

In [10]:
```
sns.boxplot(dataset)
plt.show()
```



Real time Case Study - boston house dataset

In [11]:
```
boston= pd.read_csv('boston.csv')
boston.head()
```

Out[11]:

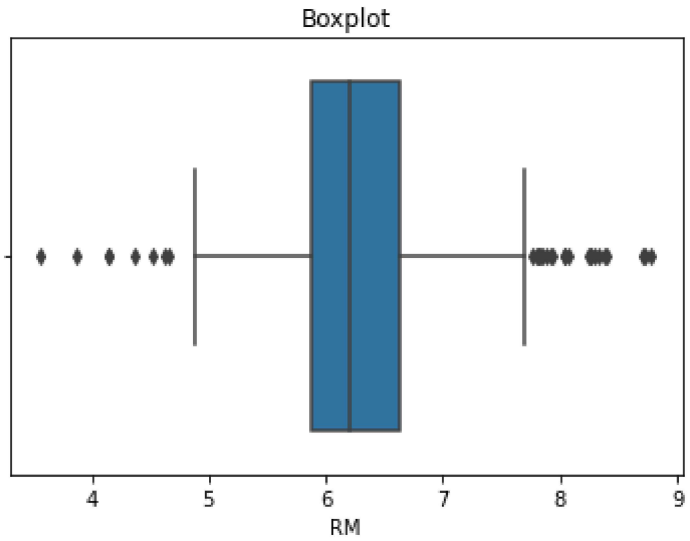
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33

In [12]:
```
boston.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  Price    506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

Detection of outliers of RM (based on Boxplot)

In [13]:
```
sns.boxplot(boston.RM)
plt.title('Boxplot')
plt.show()
```



There are outliers in both tails for "RM"

Detection of outliers of RM (based on IQR)

In [14]:
```
Q3=boston['RM'].quantile(0.75)
Q1=boston['RM'].quantile(0.25)

IQR = Q3 - Q1

lower_limit = Q1 - (IQR * 1.5)
upper_limit = Q3 + (IQR * 1.5)

lower_limit,upper_limit
```

Out[14]:
```
(4.778500000000001, 7.730499999999999)
```

Methods to deal the Outliers (3'R' Technique)

1. Remove (Trimming: remove the outliers from our dataset)
2. Rectify or Replace --> (data entry error) ---> Ask and confirm it from the Data Engineering team.
3. Retain (consider for analysis) ---> Treat them separately
4. Censoring: capping the variable distribution at a max and / or minimum value

Censoring is also known as:

- top and bottom coding
- winsorization
- capping

Important

Outliers should be detected AND **removed ONLY** from the training set, and NOT from the test set. So we should first divide our data set into train and tests, and remove outliers in the train set, but keep those in the test set, and measure how well our model is doing.

Remove (let's trimm the dataset)

In [15]:
```
boston_trimmed = boston[(boston['RM'] > lower_limit) & (boston['RM'] < upper_limi
boston_trimmed
```

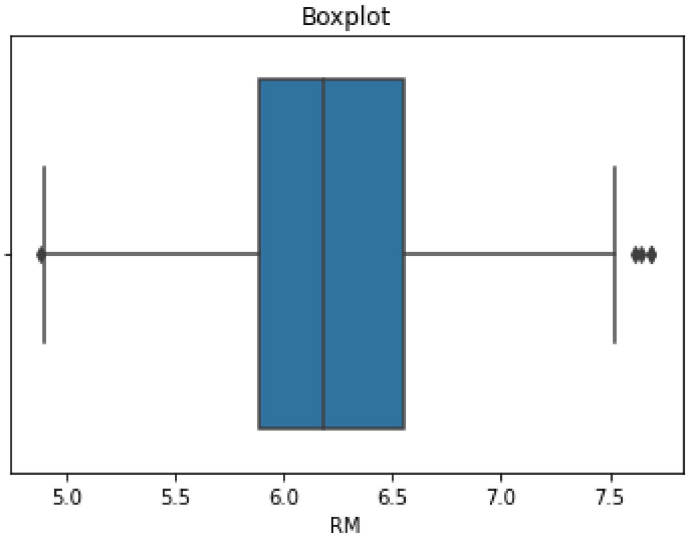
Out[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.9
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.0
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.9
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.3
...	...	...	...	...	...	...	...	...	...	...	...	...	.
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.6
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.0
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.6
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.4
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.8

476 rows × 14 columns

- We can see by using trimming, we removed 30 rows, from a dataset of 506 rows, this is about 6% of the data was removed.
- if I considered only 1 variable, I removed 30 rows and if I consider all variables then more than 100 rows we loose(20%)
- This is mostly why, we do not tend to use trimming much in machine learning.
- But if only a few variables present a tiny proportion of outliers, trimming could work.

In [16]:
```
# let's explore outliers in the trimmed dataset
sns.boxplot(boston_trimmed.RM)
plt.title('Boxplot')
plt.show()
```



- We can still see some outliers...
- When we remove data points from our dataset,all the parameters of the distribution are re-calculated,
- those are the mean, quantiles and inter-quantile range,
- therefore, in the new -trimmed- variable, values that before were not considered outliers
- This is an unwanted characteristic of this way of coping with outliers.

Replace

1. Replace with upper limit & lower limit (calculated based on IQR)

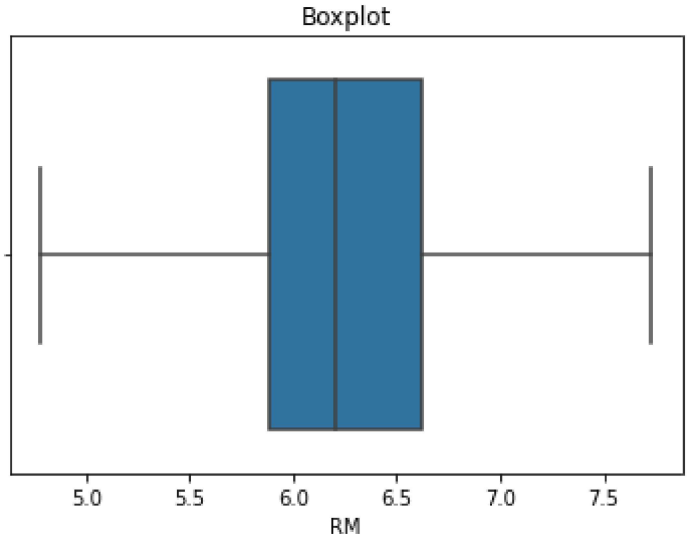
In [17]:
```
#pip install feature_engine
```

In [18]:
```
from feature_engine.outliers import Winsorizer
win = Winsorizer(capping_method='iqr', tail='both',fold=1.5,variables=['RM'])
boston_t = win.fit_transform(boston[['RM']])

# we can inspect the minimum caps and maximum caps
print(win.left_tail_caps_, win.right_tail_caps_)

# lets see boxplot
sns.boxplot(boston_t.RM)
plt.title('Boxplot')
plt.show()
```

```
{'RM': 4.778500000000001} {'RM': 7.730499999999999}
```



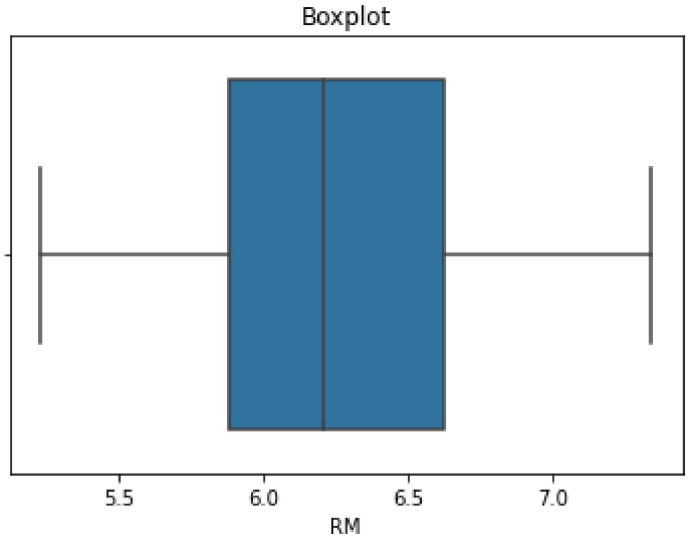
2. Replace using Winsorization (the minimum and maximum values by a value automatically taken based on guassian distribution)

In [19]:
```
from feature_engine.outliers import Winsorizer
win = Winsorizer(capping_method='gaussian', tail='both',fold=1.5,variables=['RM']
boston_t = win.fit_transform(boston[['RM']])

# we can inspect the minimum caps and maximum caps
print(win.left_tail_caps_, win.right_tail_caps_)

# lets see boxplot
sns.boxplot(boston_t.RM)
plt.title('Boxplot')
plt.show()
```

```
{'RM': 5.230708672228802} {'RM': 7.338560102474773}
```



3. Replace Arbitrary Outlier Capper (the minimum and maximum values by a value determined by the user)

- We get the min and max values based on domain expertise

In [20]:
```
from feature_engine.outliers import ArbitraryOutlierCapper
capper = ArbitraryOutlierCapper(max_capping_dict = {'RM':7.5}, min_capping_dict =
boston_c = capper.fit_transform(boston[['RM']])

# we can inspect the minimum caps and maximum caps
print(capper.right_tail_caps_,capper.left_tail_caps_)

# lets see boxplot
sns.boxplot(boston_c.RM)
plt.title('Boxplot')
plt.show()
```

```
{'RM': 7.5} {'RM': 4.8}
```