Outlier

An **outlier** is defined as an observation which stands far away from the most of other observations. Often an outlier is present due to the measurements error. Therefore, one of the most important tasks in data analysis is to identify and only if it is necessary to remove the outlier.

The standard method to detect outlier is **Tukey's method.**

Suppose we have a variable assuming the values X_1, X_2, X_3,..., X_n. Now from the values we have to first determine the first quartile (Q1) and the third quartile (Q3) and the inter-quartile range (IQR = Q3 – Q1) based on the sample observations. Now the values outside the range $[(Q1 - 1.5 * IQR), (Q3 + 1.5 * IQR)]$ is considered as *outliers*. Now, the values outside the range $[(Q1 - 3 * IQR), (Q3 + 3 * IQR)]$ is known as *extreme* outliers and the values outside the range $[(Q1 - 1.5 * IQR), (Q3 + 1.5 * IQR)]$ but inside the range $[(Q1 - 3 * IQR), (Q3 + 3 * IQR)]$ is called *mild* outliers.

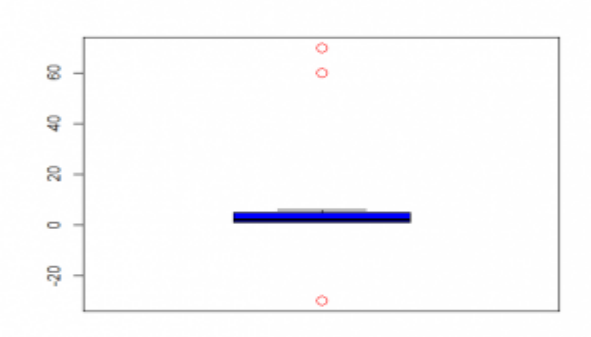
Suppose we have the values, 1, 60, 2, 1, 4, 4, 1, 1, 6, -30. Here Q1 = 1, Q3 = 4 and IQR = 3. So, the values outside the range (-3.5, 8.5) are the outliers. So, here the outliers are -30 and 60.

We have many functions in R. I prefer the function **boxplot.stats(x)$out** in the package **grDevices. boxplot.stats()$out** which use the Tukey's method to identify the outliers ranged above and below the 1.5*IQR.

```
1
2 library(grDevices)
3 x <- c(1, 60, 2, 1, 4, 4, 1, 1, 6, -30, 70)
4 x[which(x %in% boxplot.stats(x)$out)]
```

```
1
2 library(grDevices)
3 x <- c(1, 60, 2, 1, 4, 4, 1, 1, 6, -30, 70)
4 x[which(x %in% boxplot.stats(x)$out)]
5 [1]  60 -30  70
```

Now you may ask why we are using box-plot function here. Actually, box-plot is the visualization where we can see how the data is distributed along with if there are any outliers or not. Let's see the box-plot below to understand that.



The red circles are the outliers in this data.

We also have some test to detect outliers *like* **dixon.test(), chisq.out.test()** in **outliers** package. But I prefer a ***test* rosnerTest()** in ***EnvStats*** package in R. Let's see how it works.

Form the box-plot we got 3 outliers, so a parameter in **ronserTest()** is *k* which is the number you think that how many outliers are there. I will prefer to put, what you get from the box-plot adding with 1 or 2. Here we can see 3 outliers from the box-plot, so we are putting k = 4. Now see how the test performs,

```

rosnerTest(x, k = 4, warn = F)
OUTPUT:
> rosnerTest(x, k = 4, warn = F)

Results of Outlier Test
-------------------------

Test Method:                     Rosner's Test for Outliers

Hypothesized Distribution:       Normal

Data:                            x

Sample Size:                     11

Test Statistics:                 R.1 = 2.067720
                                 R.2 = 2.508654
                                 R.3 = 2.630493
                                 R.4 = 1.816061

Test Statistic Parameter:        k = 4

Alternative Hypothesis:          Up to 4 observations are not
                                 from the same Distribution.

Type I Error:                    5%

Number of Outliers Detected:     3

  i    Mean.i      SD.i Value Obs.Num    R.i+1 lambda.i+1 Outlier
1 0 10.909091 28.577804    70      11 2.067720   2.354730    TRUE
2 1  5.000000 21.924112    60       2 2.508654   2.289954    TRUE
3 2 -1.111111 10.982309   -30      10 2.630493   2.215004    TRUE
4 3  2.500000  1.927248     6       9 1.816061   2.126645   FALSE
```

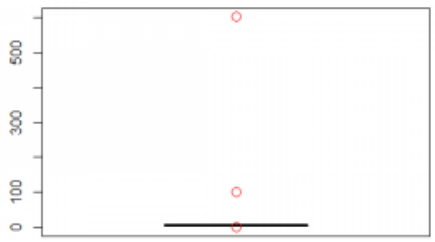
Here as you can see, 3 outliers are detected, and we also get the values corresponding to the TRUE values of Outlier from the last table. So, these are the methods to detect, visualize and test for outliers in data.

Presence of outliers may lead us to a bad result. So, either we have to remove them, or we have to replace them with some representative values. There are several statistical methods to deal with outliers. Let's discuss about the methods.

Trimming:

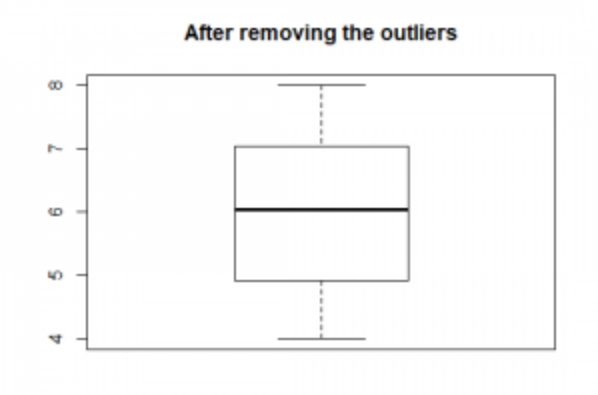
Suppose we have 1003 data on height (in ft) of adult people of a village. We are to find the mean height of the adult people.

```
height <- c((sample(seq(4,8,0.001),1000, replace = T)),101.51,-0.2346,601)
boxplot(height,outcol = "red", outcex = 1.5)
height <- c((sample(seq(4,8,0.001),1000, replace = T)),101.51,-0.2346,601)
boxplot(height,outcol = "red", outcex = 1.5)
```



Here we can see from the box-plot that there are three outliers. Now we will check which values are those outliers. Now, as I have 1003 data if we remove 3 data from it, it doesn't affect too much to the data. So, we remove/trim them from the data. You can notice that one is usual value which is negative we must have to remove it first. Then we perform trimming.

```
height <- height[-which(height < 0)]
rosnerTest(height,k = 4)
height <- height[-c(1001,1002)]
boxplot(height,main = "After removing the outliers")

height <- height[-which(height < 0)]
rosnerTest(height,k = 4)

Results of Outlier Test
-------------------------

Test Method:                     Rosner's Test for Outliers

Hypothesized Distribution:       Normal

Data:                            height

Sample Size:                     1002

Test Statistics:                 R.1 = 31.163267
                                 R.2 = 29.440518
                                 R.3 =  1.694513
                                 R.4 =  1.696956

Test Statistic Parameter:        k = 4

Alternative Hypothesis:          Up to 4 observations are not
                                 from the same Distribution.

Type I Error:                    5%

Number of Outliers Detected:     2

  i   Mean.i      SD.i   Value Obs.Num     R.i+1 lambda.i+1 Outlier
1 0 6.687901 19.070918 601.000    1002 31.163267   4.040471    TRUE
2 1 6.094183  3.240969 101.510    1001 29.440518   4.040225    TRUE
3 2 5.998767  1.179827   7.998     838  1.694513   4.039978   FALSE
4 3 5.996766  1.178719   7.997     919  1.696956   4.039731   FALSE


> height <- height[-c(1001,1002)]
> boxplot(height,main = "After removing the outliers")
```



Replacing with some representative values:

Now, if there are few numbers of observations where each of the data point is as important as others, in that case we don't use trimming. What we do here, we replace the outliers with some representative values like mean, median, minimum or maximum values etc.

Let's replace the outliers with minimum and maximum value. This method is also known as Tukey's method. Here what we do is, we first remove the outliers. Now, after removing outliers we have minimum and maximum value.

Now, replacing with minimum and maximum values results heavy tailed. If you replace with mean or median it will increase the number of observations near the center of the dataset. So now if your data is bell shaped you can use the mean or median to concentrate to the center. If your data is "U" shaped, you can use maximum and minimum value. So, it's your choice according to how data is distributed.

Ex:
outlier_values <- boxplot.stats(inputData$pressure_height)$out **# outlier values**
boxplot(inputData$pressure_height, main="Pressure Height", boxwex=0.1)
boxplot(ozone_reading ~ Month, data=ozone, main="Ozone reading across months")

Outliers Test

The function **outlierTest** from car package gives the most extreme observation based on the given model. Here's an example based on the mod linear model object we'd just created

```
car::outlierTest(mod)
#> No Studentized residuals with Bonferonni p < 0.05
#> Largest |rstudent|:
#>     rstudent unadjusted p-value Bonferonni p
#> 243 3.045756         0.0026525      0.53845
```

This output suggests that observation in row 243 is most extreme.

Outliers package

The outliers package provides a number of useful functions to systematically extract outliers. Some of these are convenient and come handy, especially the outlier() and scores() functions.

Outliers gets the extreme most observation from the mean. If you set the argument opposite=TRUE, it fetches from the other side.

```
set.seed(1234)
y=rnorm(100)
outlier(y)
#> [1] 2.548991
outlier(y,opposite=TRUE)
#> [1] -2.345698
dim(y) <- c(20,5)  # convert it to a matrix
outlier(y)
#> [1] 2.415835 1.102298 1.647817 2.548991 2.121117
outlier(y,opposite=TRUE)
#> [1] -2.345698 -2.180040 -1.806031 -1.390701 -1.372302
```

The **Tukey** fences can be modified to be "outlier" fences by changing the range flag in the boxplot call (it defaults to 1.5 times the IQR). If you set range = 3, then the **Tukey** fences are drawn at the last point inside three times the IQR instead.

```
# Boxplot with tukey fences at the last outlier point
(boxplot(pregnancy.df$GestationDays, range = 3))
```

rm.outlier

Remove the value(s) most differing from the mean

If the outlier is detected and confirmed by statistical tests, this function can remove it or replace by sample mean or median.

rm.outlier(x, fill = FALSE, median = FALSE, opposite = FALSE)

Arguments

X a dataset, most frequently a vector. If argument is a dataframe, then outlier is removed from each column by sapply. The same behavior is applied by apply when the matrix is given.

Fill If set to TRUE, the median or mean is placed instead of outlier. Otherwise, the outlier(s) is/are simply removed.

Median If set to TRUE, median is used instead of mean in outlier replacement.

Opposite if set to TRUE, gives opposite value (if largest value has maximum difference from the mean, it gives smallest and vice versa)

```
library(outliers)
# Dixon Tests for Outliers for y
dixon.test(y[,2],opposite = TRUE)
dixon.test(y[,2],opposite = FALSE)

# Dixon Tests for Outliers for ylarge
dixon.test(ylarge[,2],opposite = TRUE)
dixon.test(ylarge[,2],opposite = FALSE)


# Chi-Sq Tests for Outliers for y
chisq.out.test(y[,2],variance = var(y[,2]),opposite = TRUE)
chisq.out.test(y[,2],variance = var(y[,2]),opposite = FALSE)

# Chi-Sq Tests for Outliers for ylarge
chisq.out.test(ylarge[,2],variance = var(ylarge[,2]),opposite = TRUE)
chisq.out.test(ylarge[,2],variance = var(ylarge[,2]),opposite = FALSE)
```

In each of the Dixon and Chi-Squared tests for outliers above, we've chosen both options TRUE and FALSE in turn, for the argument `opposite`. This argument helps us choose between whether we're testing for the lowest extreme value, or the highest extreme value, since outliers can lie to both sides of the data set.

Bias and Variance

Error due to Bias:

The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict.

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Error due to Variance: The error due to variance is taken as the variability of a model prediction for a given data point.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on

the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

Bias Variance Tradeoff: If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time. An optimal balance of bias and variance would never overfit or underfit the model.

Trade-off*: a balance achieved between two desirable but incompatible features; a compromise.*

The ***bias variance trade-off*** is one of the most important aspect of error handling in ***supervised learning.***

Bias refers to the error due to overly-simplistic assumptions or faulty assumptions in the learning algorithm. Bias results in ***under-fitting*** the data. A high bias means our learning algorithm is missing important trends amongst the features.

Variance refers to the error due to an overly-complex that tries to fit the training data as closely as possible. In high variance cases the model's predicted values are extremely close to the actual values from the training set. The learning algorithm copies the training data's trends and this results in loss of generalisation capabilities. High Variance gives rise to **over-fitting**

A model having high variance when tested on unseen data will not yield satisfactory results. This is because the algorithm is highly sensitive to high degrees of variation in the training data, which results in it carrying too much noise from training data for the model to be useful for the test data.

Bias is the difference between the average of predicted values (also called the expected value) of an estimator and the actual (true) value at that point. **Variance** on the other hand quantifies how scattered or how much variation there is from the true value.