**Introduction:**

We will identify anomalous patterns in data, this process is useful, not only to find inconsistencies and errors but also to find abnormal data behavior, being useful even to find cyber attacks on organizations.

On this article there is more information as reference:

[Data Quality and Anomaly Detection Thoughts For Web Analytics](https://www.bounteous.com/insights/2018/02/28/data-quality-and-anomaly-detection-thoughts-web-analytics/)

Before starting we need the next software installed and working:

– [R language installed.](https://www.r-project.org/)

– [H2O open source framework.](https://www.h2o.ai/download/#h2o)

– Java 8 ( For H2O ). Open JDK: <https://github.com/ojdkbuild/contrib_jdk8u-ci/releases>

– [R studio.](https://www.rstudio.com/products/rstudio/download/)

**About the data used in this article.**

*# I am using* [*https://www.kaggle.com/bradklassen/pga-tour-20102018-data*](https://www.kaggle.com/bradklassen/pga-tour-20102018-data)  
*# The version I have is not the most updated version but anyway, a new version*  
*# may be used.*  
 *# The file I am using is a csv 950 mb file with 9,720,530 records, including header.  
#  
# One very important thing is that we are going to see that instead to be lost in more  
 than 9 million records, we will just be looking at 158 records with anomalies for the  
 analysed variable, so, it is easier to inspect data in this way.*

**Let’s start coding:**

*# Loading libraries*  
suppressWarnings( suppressMessages( **library**( h2o ) ) )   
*# For interactive plotting*  
suppressWarnings( suppressMessages( **library**( dygraphs ) ) )  
suppressWarnings( suppressMessages( **library**( dplyr ) ) )  
suppressWarnings( suppressMessages( **library**( DT ) ) )  
  
*# Start a single-node instance of H2O using all available processor cores and reserve 5GB of memory*  
h2oServer = h2o.init( ip = "localhost", port = 54321, max\_mem\_size = "5g", nthreads = -1 )

##   
## H2O is not running yet, starting it now...  
##   
## Note: In case of errors look at the following log files:  
## /tmp/RtmpC1pHJS/h2o\_ckassab\_started\_from\_r.out  
## /tmp/RtmpC1pHJS/h2o\_ckassab\_started\_from\_r.err  
##   
##   
## Starting H2O JVM and connecting: . Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 2 seconds 395 milliseconds   
## H2O cluster timezone: America/Mexico\_City   
## H2O data parsing timezone: UTC   
## H2O cluster version: 3.26.0.6   
## H2O cluster version age: 1 month and 8 days   
## H2O cluster name: H2O\_started\_from\_R\_ckassab\_aat507   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 4.44 GB   
## H2O cluster total cores: 4   
## H2O cluster allowed cores: 4   
## H2O cluster healthy: TRUE   
## H2O Connection ip: localhost   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## H2O Internal Security: FALSE   
## H2O API Extensions: Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4   
## R Version: R version 3.6.1 (2019-07-05)

h2o.removeAll() *# Removes all data from h2o cluster, ensuring it is clean.*  
h2o.no\_progress() *# Turn off progress bars for notebook readability*  
  
*# Setting H2O timezone for proper date data type handling*  
*#h2o.getTimezone() ===>>> UTC*  
*#h2o.listTimezones() # We can see all H2O timezones*  
h2o.setTimezone("US/Central")

## [1] "US/Central"

*# Note. I am using Ubuntu 19.10, using /tmp directory*  
*# Every time I boot my computer, I need to copy the data file again to /tmp*  
*# directory.*  
  
*# Importing data file and setting data types accordingly.*  
allData = read.csv( "/tmp/PGA\_Tour\_Golf\_Data\_2019\_Kaggle.csv", sep = ",", header = T )  
  
*# When using as.Posixct H2O is not importing data, so we are using as.Date.*  
allData$Date = as.Date( allData$Date )  
allData$Value = as.numeric(allData$Value)  
  
*# Convert dataset to H2O format.*  
allData\_hex = as.h2o( allData )  
  
*# Build an Isolation forest model*  
startTime <- Sys.time()  
startTime

## [1] "2019-11-10 20:10:30 CST"

trainingModel = h2o.isolationForest( training\_frame = allData\_hex  
 , sample\_rate = 0.1  
 , max\_depth = 32  
 , ntrees = 100  
 )

## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Stopping tolerance is ignored for \_stopping\_rounds=0..

Sys.time()

## [1] "2019-11-10 20:20:15 CST"

Sys.time() - startTime

## Time difference of 9.756691 mins

*# According to H2O doc:*   
*#* [*http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/if.html*](http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/if.html)  
*#*  
*# Isolation Forest is similar in principle to Random Forest and is built on*   
*# the basis of decision trees.*   
  
*# Isolation Forest creates multiple decision trees to isolate observations.*  
*#*   
*# Trees are split randomly, The assumption is that:*  
*#*   
*# IF ONE UNIT MEASUREMENTS ARE SIMILAR TO OTHERS,*  
*# IT WILL TAKE MORE RANDOM SPLITS TO ISOLATE IT.*  
*#*   
*# The less splits needed, the unit is more likely to be anomalous.*  
*#*   
*# The average number of splits is then used as a score.*  
  
*# Calculate score for all data.*  
startTime <- Sys.time()  
startTime

## [1] "2019-11-10 20:20:15 CST"

score = h2o.predict( trainingModel, allData\_hex )  
result\_pred = as.vector( score$predict )  
Sys.time()

## [1] "2019-11-10 20:23:18 CST"

Sys.time() - startTime

## Time difference of 3.056829 mins

*################################################################################*  
*# Setting threshold value for anomaly detection.*  
*################################################################################*  
  
*# Setting desired threshold percentage.*  
threshold = .999 *# Let's say we want the .001% data different than the rest.*  
  
*# Using this threshold to get score limit to filter data anomalies.*  
scoreLimit = round( quantile( result\_pred, threshold ), 4 )  
  
*# Add row score at the beginning of dataset*  
allData = cbind( RowScore = round( result\_pred, 4 ), allData )  
  
*# Get data anomalies by filtering all data.*  
anomalies = allData[ allData$RowScore > scoreLimit, ]  
  
*# As we can see in the summary:*  
summary(anomalies)

## RowScore Player.Name Date   
## Min. :0.9540 Jonas Blixt : 231 Min. :2019-07-07   
## 1st Qu.:0.9565 Jordan Spieth : 231 1st Qu.:2019-08-25   
## Median :0.9614 Julian Etulain: 221 Median :2019-08-25   
## Mean :0.9640 Johnson Wagner: 213 Mean :2019-08-24   
## 3rd Qu.:0.9701 John Chin : 209 3rd Qu.:2019-08-25   
## Max. :1.0000 Keegan Bradley: 209 Max. :2019-08-25   
## (Other) :8325   
## Statistic   
## Club Head Speed : 234   
## Driving Pct. 300-320 (Measured): 193   
## Carry Efficiency : 163   
## First Tee Early Lowest Round : 161   
## First Tee Late Lowest Round : 160   
## GIR Percentage - 100+ yards : 158   
## (Other) :8570   
## Variable   
## First Tee Early Lowest Round - (LOW RND) : 103   
## First Tee Late Lowest Round - (LOW RND) : 96   
## First Tee Late Lowest Round - (ROUNDS) : 64   
## Driving Pct. 300-320 (Measured) - (TOTAL DRVS - OVERALL): 61   
## GIR Percentage - 175-200 yards - (%) : 61   
## First Tee Early Lowest Round - (ROUNDS) : 58   
## (Other) :9196   
## Value   
## Min. : 1268   
## 1st Qu.: 53058   
## Median : 87088   
## Mean :111716   
## 3rd Qu.:184278   
## Max. :220583   
##

*# The Statistic: GIR Percentage - 100+ yards is one of the most important values*  
*# Filtering all anomalies within this Statistic value*  
statisticFilter = "GIR Percentage - 100+ yards"  
  
specificVar = anomalies %>%  
 filter(Statistic==statisticFilter)  
  
cat( statisticFilter,": ", dim(specificVar)[1] )

## GIR Percentage - 100+ yards : 158

**if**( dim(specificVar)[1] > 0 ) {  
  
 *# We want to know the relation between Players and "Approaches from 200-225 yards"*  
 *# So, in order to get a chart, we assign a code to each player*  
 *# Since factors in R are really integer values, we do this to get the codes:*  
 specificVar$PlayerCode = as.integer(specificVar$Player.Name)   
   
 *# To sort our dataset we convert the date to numeric*   
 specificVar$DateAsNum = as.numeric( paste0( substr(specificVar$Date,1,4)  
 , substr(specificVar$Date,6,7)  
 , substr(specificVar$Date,9,10) ) )  
 *# And sort the data frame.*  
 specificVar = specificVar[order(specificVar$DateAsNum),]  
 *# Set records num using a sequence.*  
 rownames(specificVar) = seq(1:dim(specificVar)[1])  
   
 colNamesFinalTable = c( "PlayerCode", "Player.Name", "Date", "Variable", "Value" )  
 specificVar = specificVar[, colNamesFinalTable]  
 specificVar$PlayerCode = as.factor(specificVar$PlayerCode)  
   
 *# Creating our final dataframe for our chart.*  
 specificVarChartData = data.frame( SeqNum = as.integer( rownames(specificVar) )  
 , PlayerCode = specificVar$PlayerCode  
 , Value = specificVar$Value  
 )  
   
  
   
 AnomaliesGraph = dygraph( specificVarChartData, main = ''  
 , xlab = paste(statisticFilter,"Anomaly Number."), ylab = "Player Code." ) %>%  
 dyAxis("y", label = "Player Code.") %>%  
 dyAxis("y2", label = "Value.", independentTicks = TRUE) %>%  
 dySeries( name = "PlayerCode", label = "Player Code.", drawPoints = TRUE, pointShape = "dot"  
 , color = "blue", pointSize = 2 ) %>%  
 dySeries( name = "Value", label = "Value.", drawPoints = TRUE, pointShape = "dot"  
 , color = "green", pointSize = 2, axis = 'y2' ) %>%  
 dyRangeSelector()  
 dyOptions( AnomaliesGraph, digitsAfterDecimal = 0 )  
}

## Registered S3 method overwritten by 'xts':  
## method from  
## as.zoo.xts zoo

**Sample chart with the anomalies found:**

**Sample data table with the anomalies found:**

Show

 entries

Search:

|  | **Player Code** | **Player Name** | **Date** | **Variable** | **Value** |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 1 | 686 | Josh Teater | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 198471 |
| 2 | 655 | Johnson Wagner | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 186658 |
| 3 | 618 | Jim Furyk | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 198471 |
| 4 | 723 | Keegan Bradley | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 211362 |
| 5 | 213 | Cameron Tringale | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 198471 |
| 6 | 712 | Justin Thomas | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 199671 |
| 7 | 520 | Hunter Mahan | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 178096 |
| 8 | 587 | Jason Day | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 189755 |
| 9 | 539 | J.J. Henry | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 177431 |
| 10 | 657 | Jon Rahm | 2019-08-25 | GIR Percentage – 100+ yards – (ROUNDS) | 199671 |