Introduction:

This is the second article on data quality, for the first part, please go to: http://laranikalranalytics.blogspot.com/2019/11/using-r-and-h2o-isolation-forest-for.html

Since Isolation Forest is building an ensemble of isolation trees, and these trees are created randomly, there is a lot of randomness in the isolation forest training, so, to have a more robust result, 3 isolation forest models will be trained for a better anomaly detection.

I will also use Apache Spark for data handling.

For a full example, testing data will be used after training the 3 IF(Isolation Forest) models.

This way of using Isolation Forest is kind of a general usage also for maintenance prediction.

I am working with data from file:
https://www.kaggle.com/bradklassen/pga-tour-20102018-data

```
# Set Java parameters, enough memory for Java.
options( java.parameters = c( "-Xmx40G" ) ) # 40GB Ram for Java

# Loading libraries
suppressWarnings(suppressMessages(library(sparklyr)))
suppressWarnings(suppressMessages(library(h2o)))
suppressWarnings(suppressMessages(library(dplyr)))
suppressWarnings(suppressMessages(library(xts)))
suppressWarnings(suppressMessages(library(rsparkling))) # Version 3.26.10-2.4
suppressWarnings(suppressMessages(library(DT)))
suppressWarnings(suppressMessages(library(dygraphs))) # For interactive plotting

Sys.setenv(TZ = "America/Chicago") # R environment time zone.

# Connecting to Spark, local mode.
# For reference go to: https://spark.rstudio.com/guides/connections/

# Set Spark Config Parameters
config <- spark_config()
config["sparklyr.shell.driver-memory"] = "40g" # I created more swap, I must have more memory avail
able.
config["sparklyr.cores.local"] = 4 # Using all cores on my Intel i5
config$sparklyr.cancellable = TRUE
config$spark.executor.cores = 4
config$spark.cores.max = 4
config$spark.ext.h2o.nthreads = -1 # Ensure all threads when using H2O

# Connecting to Spark.
sc = spark_connect(master = "local", version = "2.4.3", hadoop_version="2.7", config=config)

# Setting Java TimeZone to GMT after initializing spark allow us to have a better
# date time data handling.
sparklyr::invoke_static(sc, "java.util.TimeZone",  "getTimeZone", "GMT") %>%
  sparklyr::invoke_static(sc, "java.util.TimeZone", "setDefault", .)
```

```
## NULL
```

```
# Start importing data to Spark and doing some data cleaning
startTime = Sys.time()
# Start Time:
startTime
```

```
## [1] "2019-12-16 12:56:51 CST"
```

```
allDataF = spark_read_csv( sc, "allDataF"
                          , path = "/home/ckassab/Development/R/DataQuality/Data/PGA_Tour_Golf_Data
_2019_Kaggle.csv"
                          , memory = FALSE # Map the file, but not make a copy of it in memory, thi
s saves 1g ram.
                          , header = TRUE
                          , delimiter = ","
                          , quote = "\""
                          , infer_schema = TRUE
                          , null_value = NULL )
```

```
# Data cleaning
allData = allDataF %>%
  na.omit() %>% # Dropping all NAs from dataset
  mutate(Date = as.Date(substr(Date,1,10))) # Set date format as needed.
```

```
## * Dropped 174167 rows with 'na.omit' (9720529 => 9546362)
```

```
# End importing data to Spark and doing some data cleaning
# End Time:
Sys.time()
```

```
## [1] "2019-12-16 12:58:33 CST"
```

```
# Total time:
Sys.time() - startTime
```

```
## Time difference of 1.701981 mins
```

```
# Inspect the H2OContext for our Spark connection
# This will also start an H2O cluster
h2o_context(sc)
```

```
##
##   org.apache.spark.h2o.H2OContext
##
## Sparkling Water Context:
##   * Sparkling Water Version: 3.26.10-2.4
##   * H2O name: sparkling-water-ckassab_local-1576522610564
##   * cluster size: 1
##   * list of used nodes:
##    (executorId, host, port)
##    ------------------------
##    (driver,127.0.0.1,54321)
##    ------------------------
##
##   Open H2O Flow in browser: http://127.0.0.1:54321 (CMD + click in Mac OSX)
##
##
```

```
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.setTimezone("US/Central")
```

```
## [1] "US/Central"
```

```
# Convert dataset to H2O format.
allData_hex = as_h2o_frame( sc, allData )

# Converting certain columns to factor.
allData_hex[,1] = as.factor(allData_hex[,1])
allData_hex[,3] = as.factor(allData_hex[,3])
allData_hex[,4] = as.factor(allData_hex[,4])
allData_hex[,5] = as.factor(allData_hex[,5])

# Getting numeric codes from factors, so we can use them to build
# IF(Isolation Forest) models, I am doing this because data has no codes
# In a real model, the best is to have data with integer IDs.
# Getting the codes using H2O is easier, becuase Spark does not have factor data type.
allData_hex$Player_Code = as.numeric(allData_hex[,1])
allData_hex$Statistic_Code = as.numeric(allData_hex[,3])
allData_hex$Variable_Code = as.numeric(allData_hex[,4])
allData_hex$Value_Code = as.numeric(allData_hex[,5])


# split into train and validation sets
allData_hex_split = h2o.splitFrame(data = allData_hex, ratios = 0.9, seed = 1234)
trainData_hex = allData_hex_split[[1]]
testData_hex = allData_hex_split[[2]]
```

```
# Save training and testing datasets to kepp coded data backup.
h2o.exportFile(trainData_hex
               , force = TRUE
               , sep = "|"
               , path = "/home/ckassab/Development/R/DataQuality/Data/PGA_Tour_trainData_hex.csv" )

h2o.exportFile(testData_hex
               , force = TRUE
               , sep = "|"
               , path = "/home/ckassab/Development/R/DataQuality/Data/PGA_Tour_testData_hex.csv" )

# Variable names to be used when creating models.
featureNames = c( "Player_Code", "Statistic_Code", "Variable_Code", "Value_Code" )

#################################################################################
# Building 3 Isolation forest models:
# http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/if.html
# Parameter values set:
  # sample_rate:
  # Specify the row sampling rate (x-axis). (Note that this method is sample without replacement.)
  # Without replacement meaning:
  # Each sample unit of the population has only one chance to be selected in the sample.
  # I understand you take a sample of the population and then take a new sample
  # without putting the first sample on the population, this means without replacement.
  # in this way you avoid taking the same individual(record) more than once.
  # Reference:
  # https://methods.sagepub.com/reference/encyclopedia-of-survey-research-methods/n516.xml
  # https://stats.stackexchange.com/questions/69744/why-at-all-consider-sampling-without-replacement-in-a-practical-application
  # The sample_rate range is 0.0 to 1.0. Higher values may improve training accuracy.
  # Test accuracy improves when either columns or rows are sampled.
  # For details, refer to "Stochastic Gradient Boosting" (Friedman, 1999).
  # If set to -1 (default), then sample_size parameter will be used instead.
  #
  # For this analysis I am setting up sample_rate=.8
  #
  # From H2O docs:http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/sample_rate.html
  #
  # In GBM and XGBoost, this value defaults to 1; in DRF, this value defaults to 0.6320000291.
  # Row and column sampling (sample_rate and col_sample_rate) can improve generalization
  # and lead to lower validation and test set errors.
  # Good general values for large datasets are around 0.7 to 0.8 (sampling 70-80 percent of the dat
a)
  # for both parameters, as higher values generally improve training accuracy.


  # max_depth: Specify the maximum tree depth. Higher values will make the model
  # more complex and can lead to overfitting. Setting this value to 0 specifies no limit.
  # This value defaults to 8.
  # http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/max_depth.html

  # seed: Specify the random number generator (RNG) seed for algorithm components
  # dependent on randomization. The seed is consistent for each H2O instance so
  # that you can create models with the same starting conditions in alternative configurations.
  # The meaning is fix a random number generator seed for reproducibility.
  # here I am creating 9 different models with 9 different seeds on the same data.

  # x: Specify a vector containing the names or indices of the predictor variables to use when buil
ding the model.
#################################################################################
startTime = Sys.time()
# Start Time:
startTime
```

```
## [1] "2019-12-16 13:05:50 CST"
```

```
trainingModel1 = h2o.isolationForest( training_frame = trainData_hex
                                      , x = featureNames
                                      , model_id = "trainingIFModel1"
                                      , sample_rate = 0.8
                                      , max_depth = 32
                                      , ntrees = 100
                                      , seed = 1260 )
```

```
## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Stopping tolerance is ignored fo
r _stopping_rounds=0..
```

```
trainingModel2 = h2o.isolationForest( training_frame = trainData_hex
                                    , x = featureNames
                                    , model_id = "trainingIFModel2"
                                    , sample_rate = 0.8
                                    , max_depth = 32
                                    , ntrees = 100
                                    , seed = 1634 )
```

```
## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Stopping tolerance is ignored fo
r _stopping_rounds=0..
```

```
trainingModel3 = h2o.isolationForest( training_frame = trainData_hex
                                    , x = featureNames
                                    , model_id = "trainingIFModel3"
                                    , sample_rate = 0.8
                                    , max_depth = 32
                                    , ntrees = 100
                                    , seed = 1235 )
```

```
## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Stopping tolerance is ignored fo
r _stopping_rounds=0..
```

```
# End Time:
Sys.time()
```

```
## [1] "2019-12-16 21:15:34 CST"
```

```
# Total time to train IF(Isolation Forest) models:
Sys.time() - startTime
```

```
## Time difference of 8.162204 hours
```

```
# Saving models for possible use with some future testing data.
h2o.saveModel( trainingModel1
             , "/home/ckassab/Development/R/DataQuality/Models"
             , force = TRUE )
```

```
## [1] "/home/ckassab/Development/R/DataQuality/Models/trainingIFModel1"
```

```
h2o.saveModel( trainingModel2
             , "/home/ckassab/Development/R/DataQuality/Models"
             , force = TRUE )
```

```
## [1] "/home/ckassab/Development/R/DataQuality/Models/trainingIFModel2"
```

```
h2o.saveModel( trainingModel3
             , "/home/ckassab/Development/R/DataQuality/Models"
             , force = TRUE )
```

```
## [1] "/home/ckassab/Development/R/DataQuality/Models/trainingIFModel3"
```

```
################################################################################
# Calculate scores.
startTime = Sys.time()
# Start Time:
startTime
```

```
## [1] "2019-12-16 21:16:05 CST"
```

```
score1 = h2o.predict( trainingModel1, trainData_hex )
score2 = h2o.predict( trainingModel2, trainData_hex )
```

```
score3 = h2o.predict( trainingModel3, trainData_hex )
# End Time:
Sys.time()
```

```
## [1] "2019-12-16 22:33:48 CST"
```

```
# Total time to get IF(Isolation Forest) models scores:
Sys.time() - startTime
```

```
## Time difference of 1.295181 hours
```

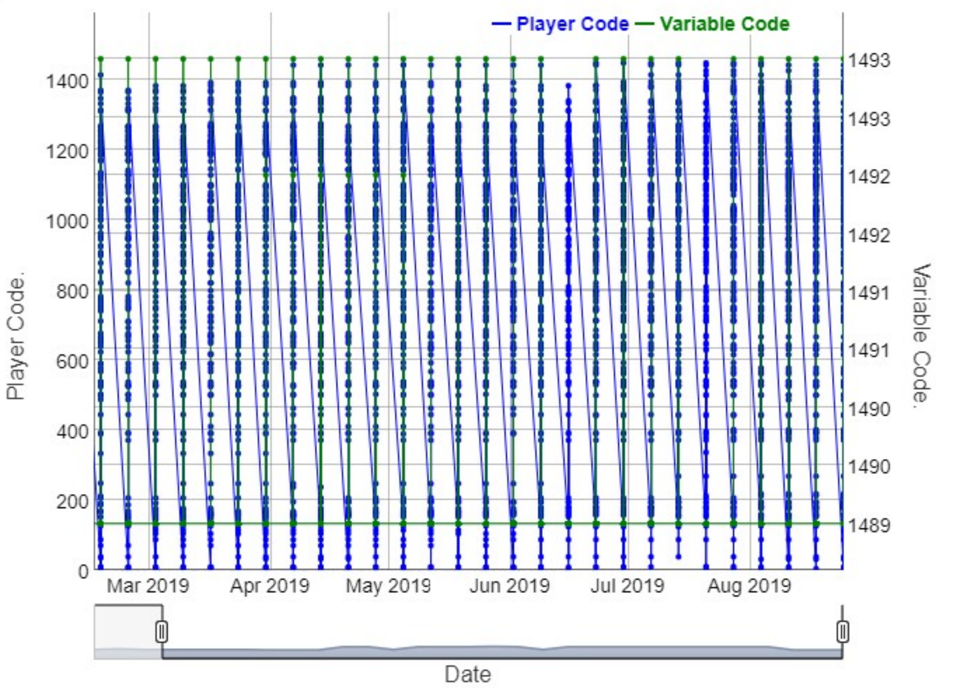
```
##################################################################################
# Setting desired threshold percentage.
threshold = .999 # Let's say we want the .001% data different than the rest.

# Using this threshold to get score limit to filter data anomalies.
# These score limits will be also used to get testing data anomalies.
scoreLimit1 = round( h2o.quantile( score1[,1], threshold ), 4 )
scoreLimit2 = round( h2o.quantile( score2[,1], threshold ), 4 )
scoreLimit3 = round( h2o.quantile( score3[,1], threshold ), 4 )

# Saving score limits to file.
scoreLimitNames = c( "scoreLimit1", "scoreLimit2", "scoreLimit3" )

scoreLimitValues = c( scoreLimit1, scoreLimit2, scoreLimit3 )

scoreLimits = data.frame(scoreLimitNames, scoreLimitValues)

write.table( scoreLimits
           , file = "/home/ckassab/Development/R/DataQuality/Data/scoreLimits.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )


##################################################################################
# Once we have our score limits, let's use them to get data anomalies.
##################################################################################

# Add row score at the beginning of dataset
trainData_hexScores = h2o.cbind( round( score1[,1], 4 )
                               , round( score2[,1], 4 )
                               , round( score3[,1], 4 )
                               , trainData_hex )

# Get data anomalies from training dataset.
anomalies1 = trainData_hexScores[ trainData_hexScores[,1] > scoreLimits[1,2], ]
anomalies2 = trainData_hexScores[ trainData_hexScores[,2] > scoreLimits[2,2], ]
anomalies3 = trainData_hexScores[ trainData_hexScores[,3] > scoreLimits[3,2], ]


##################################################################################
# All anomalies have been detected using 3 IF(Isolation Forest) models.

# As mentioned, using Spark for data handling, easier than H2O data handling

anomaliesS1 = as_spark_dataframe( sc, anomalies1, name = "anomaliesS1" )
anomaliesS2 = as_spark_dataframe( sc, anomalies2, name = "anomaliesS2" )
anomaliesS3 = as_spark_dataframe( sc, anomalies3, name = "anomaliesS3" )

# Grouping and counting anomalies
anomaliesS1 = anomaliesS1 %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(AnomCount = count()) %>%
  mutate(ModelNumber = "1")


anomaliesS2 = anomaliesS2 %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(AnomCount = count()) %>%
  mutate(ModelNumber = "2")


anomaliesS3 = anomaliesS3 %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
```

```
  select(Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(AnomCount = count()) %>%
  mutate(ModelNumber = "3")

anomaliesS = sdf_bind_rows( anomaliesS1, anomaliesS2, anomaliesS3 )
anomaliesS = sdf_sort(anomaliesS, c("Date", "Player_Code", "Variable_Code"))

anomsInAllModels = anomaliesS %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(ModelNumber, AnomCount, Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(TotalAnomalies = count()) %>%
  filter(TotalAnomalies==(AnomCount*3)) %>% # Filtering anomalies found in 3 models.
  collect() # Copy to R to create chart.

# Save anomsInAllModels to pipe delimited file.
write.table( anomsInAllModels
           , file = "/home/ckassab/Development/R/DataQuality/Data/anomsInAllModels_PGA_Tour_Golf_Da
ta_2019_Kaggle.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )

# Just for reference and future study, getting anomalies not in all models.
# The consideration here is that if the anomaly is present in less than 3
# models, it is more possible not to be a "real" anomaly.
anomsNOtInAllModels = anomaliesS %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(ModelNumber, AnomCount, Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(TotalAnomalies = count()) %>%
  filter(TotalAnomalies<(AnomCount*3)) %>% # Filtering anomalies found in less than 3 models.
  collect() # Copy to R to create chart.

# Save anomsNOtInAllModels to pipe delimited file.
write.table( anomsNOtInAllModels
           , file = "/home/ckassab/Development/R/DataQuality/Data/anomsNOtInAllModels_PGA_Tour_Golf
_Data_2019_Kaggle.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )


# Since we have data processed with 3 models, it is needed to keep just unique values.
distinctAnomalies = anomsInAllModels %>%
  distinct(Date, Player_Code, Player_Name, Statistic, Variable_Code, Variable, Value)

write.table( distinctAnomalies
           , file = "/home/ckassab/Development/R/DataQuality/Data/distinctAnomalies_PGA_Tour_Golf_D
ata_2019_Kaggle.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )


cat( "Anomalies found in training dataset: ", dim(distinctAnomalies)[1] )
```

```
## Anomalies found in training dataset:  3895
```

```
################################################################################
# If anomalies found, create chart
################################################################################

if( dim(distinctAnomalies)[1] > 0 ) {

  # Creating a time series with player codes
  players_xts <- xts( distinctAnomalies$Player_Code, order.by=as.Date(distinctAnomalies$Date))

  # Creating a time series with variable codes
  variables_xts <- xts( distinctAnomalies[,5], order.by=as.Date(distinctAnomalies$Date))

  # Binding time series.
  allAnomalies_xts <- cbind(players_xts, variables_xts)

  # Displaying the chart.
  anomaliesGraph = dygraph( allAnomalies_xts, main = ''
                          , xlab = "Date", ylab = "Player Code." ) %>%
    dyAxis("y", label = "Player Code.") %>%
    dyAxis("y2", label = "Variable Code.", independentTicks = TRUE) %>%
    dySeries( name = "players_xts", label = "Player Code", drawPoints = TRUE, pointShape = "dot"
            , color = "blue", pointSize = 2 ) %>%
    dySeries( name = "Variable_Code", label = "Variable Code", drawPoints = TRUE, pointShape = "do
t"
            , color = "green", pointSize = 2, axis = 'y2' ) %>%
    dyRangeSelector()
```

```
  dyOptions( anomaliesGraph, digitsAfterDecimal = 0 )


}
```



Show entries **Search:**

	Date	Player_Code	Player_Name	Statistic	Variable_Code	Variable	Value
1	2019-01-27	5	Abraham Ancer	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$1,203,506
2	2019-01-27	8	Adam Hadwin	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$1,182,380
3	2019-01-27	9	Adam Long	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$1,075,568
4	2019-01-27	11	Adam Scott	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$1,098,300
5	2019-01-27	35	Alexander Bjork	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$148,500
6	2019-01-27	38	Alexander Levy	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$129,667
7	2019-01-27	70	Andrew Putnam	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$1,318,184
8	2019-01-27	126	Berry Henson	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$11,970
9	2019-01-27	152	Braden Thornberry	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$16,557
10	2019-01-27	157	Branden Grace	Total Money (Official and	1489	Total Money (Official and Unofficial) –	$102,595

	Date	Player_Code	Player_Name	Statistic	Variable_Code	Variable	Value
				Unofficial)		(MONEY)	

Showing 1 to 10 of 3,895 entries

Previous 1 2 3 4 5 … 390 Next

****

******************Checking Testing Data Anomalies.

****

```
# Calculate scores
testScore1 = h2o.predict( trainingModel1, testData_hex )
testScore2 = h2o.predict( trainingModel2, testData_hex )
testScore3 = h2o.predict( trainingModel3, testData_hex )

# Add row scores at the beginning of dataset
testData_hexScores = h2o.cbind( round( testScore1[,1], 4 )
                                , round( testScore2[,1], 4 )
                                , round( testScore3[,1], 4 )
                                , testData_hex )

# Get data anomalies by filtering using scorelimits.
testAnomalies1 = testData_hexScores[ testData_hexScores[,1] > scoreLimits[1,2], ]
testAnomalies2 = testData_hexScores[ testData_hexScores[,2] > scoreLimits[2,2], ]
testAnomalies3 = testData_hexScores[ testData_hexScores[,3] > scoreLimits[3,2], ]

# Convert H2O dataframes to spark dataframes.
testAnomaliesS1 = as_spark_dataframe(sc, testAnomalies1)
testAnomaliesS2 = as_spark_dataframe(sc, testAnomalies2)
testAnomaliesS3 = as_spark_dataframe(sc, testAnomalies3)


# Grouping and counting anomalies
testAnomaliesS1 = testAnomaliesS1 %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(AnomCount = count()) %>%
  mutate(ModelNumber = "1")


testAnomaliesS2 = testAnomaliesS2 %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(AnomCount = count()) %>%
  mutate(ModelNumber = "2")


testAnomaliesS3 = testAnomaliesS3 %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(AnomCount = count()) %>%
  mutate(ModelNumber = "3")


testAnomaliesS = sdf_bind_rows( testAnomaliesS1, testAnomaliesS2, testAnomaliesS3 )
testAnomaliesS = sdf_sort(testAnomaliesS, c("Date", "Player_Code", "Variable_Code"))

testAnomsInAllModels = testAnomaliesS %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(ModelNumber, AnomCount, Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(TotalAnomalies = count()) %>%
  filter(TotalAnomalies==(AnomCount*3)) %>% # Filtering anomalies found in 3 models.
  collect() # Copy to R to create chart.

# Save anomsInAllModels to pipe delimited file.
write.table( testAnomsInAllModels
           , file = "/home/ckassab/Development/R/DataQuality/Data/testAnomsInAllModels_PGA_Tour_Gol
f_Data_2019_Kaggle.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )

# Just for reference and future study, getting anomalies not in all models.
# The consideration here is that if the anomaly is present in less than 3
# models, it is more possible not to be a "real" anomaly.
```

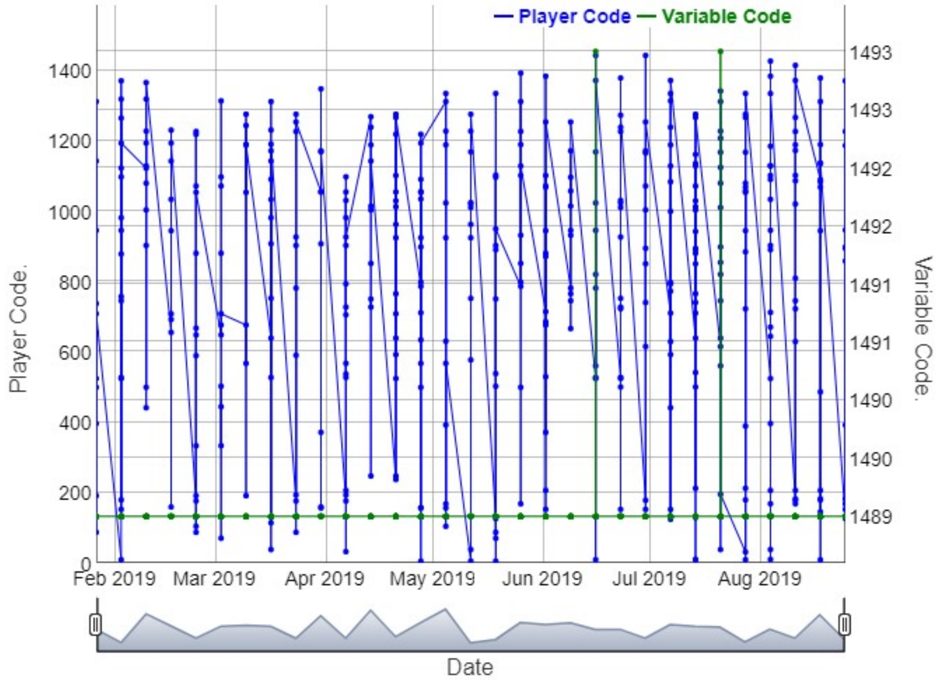
```
testAnomsNOtInAllModels = testAnomaliesS %>%
  group_by(Player_Code, Statistic_Code, Variable_Code, Value_Code) %>%
  select(ModelNumber, AnomCount, Date, Player_Name, Statistic, Variable, Value) %>%
  mutate(TotalAnomalies = count()) %>%
  filter(TotalAnomalies<(AnomCount*3)) %>% # Filtering anomalies found in less than 3 models.
  collect() # Copy to R to create chart.

# Save testAnomsNOtInAllModels to pipe delimited file.
write.table( testAnomsNOtInAllModels
           , file = "/home/ckassab/Development/R/DataQuality/Data/testAnomsNOtInAllModels_PGA_Tour_
Golf_Data_2019_Kaggle.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )

testDistinctAnomalies = testAnomsInAllModels %>%
  distinct(Date, Player_Code, Player_Name, Statistic, Variable_Code, Variable, Value)

write.table( testDistinctAnomalies
           , file = "/home/ckassab/Development/R/DataQuality/Data/testDistinctAnomalies_PGA_Tour_Go
lf_Data_2019_Kaggle.csv"
           , append = FALSE, quote = TRUE, sep = "|", row.names = FALSE )


cat( "Anomalies found in testing dataset: ", dim(testDistinctAnomalies)[1] )
```

```
## Anomalies found in testing dataset:  425
```

```
# Now we disconnect from Spark, this will result in the H2OContext being stopped as
# well since it's owned by the spark shell process used by our Spark connection:
spark_disconnect(sc)
```

```
## NULL
```

```
#################################################################################
# If anomalies found, create chart
#################################################################################

if( dim(testDistinctAnomalies)[1] > 0 ) {

  # Creating a time series with player codes
  testPlayers_xts <- xts( testDistinctAnomalies$Player_Code, order.by=as.Date(testDistinctAnomalies$D
ate))

  # Creating a time series with variable codes
  testVariables_xts <- xts( testDistinctAnomalies[,5], order.by=as.Date(testDistinctAnomalies$Date))

  # Binding time series.
  testAllAnomalies_xts <- cbind(testPlayers_xts, testVariables_xts)

  # Displaying the chart.
  anomaliesGraph = dygraph( testAllAnomalies_xts, main = ''
                          , xlab = "Date", ylab = "Player Code." ) %>%
    dyAxis("y", label = "Player Code.") %>%
    dyAxis("y2", label = "Variable Code.", independentTicks = TRUE) %>%
    dySeries( name = "testPlayers_xts", label = "Player Code", drawPoints = TRUE, pointShape = "do
t"
             , color = "blue", pointSize = 2 ) %>%
    dySeries( name = "Variable_Code", label = "Variable Code", drawPoints = TRUE, pointShape = "do
t"
             , color = "green", pointSize = 2, axis = 'y2' ) %>%
    dyRangeSelector()
  dyOptions( anomaliesGraph, digitsAfterDecimal = 0 )


}
```



Show entries **Search:**

	Date	Player_Code	Player_Name	Statistic	Variable_Code	Variable	Value
1	2019-01-27	87	Armando Favela	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$108,000
2	2019-01-27	191	Bryson DeChambeau	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$1,747,000
3	2019-01-27	396	Dylan Frittelli	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$143,040
4	2019-01-27	499	Henrik Norlander	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$14,592
5	2019-01-27	524	Hyun-woo Ryu	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$15,295
6	2019-01-27	655	Jon Curran	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$11,480
7	2019-01-27	708	Justin Rose	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$2,144,795
8	2019-01-27	737	Kevin Kisner	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$218,585
9	2019-01-27	945	Mikko Korhonen	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$32,000
10	2019-01-27	1142	Ryan Fox	Total Money (Official and Unofficial)	1489	Total Money (Official and Unofficial) – (MONEY)	$106,500