R Notebook

Argus Media - Conding Test

Ivan Berlim Gonçalves

The purpose of this exercise is to design and implement an entire data preparation pipeline in R. We would like you to implement a robust, extensible and generic framework for data preparation.

```
# Verify if the package is already installed, if not, install package
packages <- c("gamlss", "gamlss.add", "gamlss.dist", "roll", "dplyr", "tseries", "ggpubr", "magrittr",
install.packages(setdiff(packages, rownames(installed.packages())))

# Loading libraries
library(gamlss)
library(gamlss.add)
library(gamlss.dist)
# library(DT)
library(roll)
library(dplyr)
# library(stats)
library(ggpubr)
library(tseries)
# library(ggplot2)
# library(dataspice)
library(Hmisc)
library(magrittr)
library(labelVector)
library(corrplot)
```

Packages and Libraries

1) Take as raw inputs to the data preparation process, the oil data from the gamlss package.

```
# Raw data preparation
data_prep <- function(rawdata) {
  oil_data <- rawdata
  return(oil_data)
}
```

```r
oil_data <- data_prep(gamlss.data::oil)

# Printing data information
paste0("The data set has ", nrow(oil_data), " observations and  ", ncol(oil_data), " variables. ")
```

```
## [1] "The data set has 1000 observations and  25 variables. "
```

```r
cat('\n')
```

```r
oil_col_names <- colnames(oil_data)
cat('Columns in the data frame: \n\n')
```

```
## Columns in the data frame:
```

```r
for (i in (1:ncol(oil))) {print(paste0("Column number: ",i, ". Variable name: ", oil_col_names[i]))}
```

```
## [1] "Column number: 1. Variable name: OILPRICE"
## [1] "Column number: 2. Variable name: CL2_log"
## [1] "Column number: 3. Variable name: CL3_log"
## [1] "Column number: 4. Variable name: CL4_log"
## [1] "Column number: 5. Variable name: CL5_log"
## [1] "Column number: 6. Variable name: CL6_log"
## [1] "Column number: 7. Variable name: CL7_log"
## [1] "Column number: 8. Variable name: CL8_log"
## [1] "Column number: 9. Variable name: CL9_log"
## [1] "Column number: 10. Variable name: CL10_log"
## [1] "Column number: 11. Variable name: CL11_log"
## [1] "Column number: 12. Variable name: CL12_log"
## [1] "Column number: 13. Variable name: CL13_log"
## [1] "Column number: 14. Variable name: CL14_log"
## [1] "Column number: 15. Variable name: CL15_log"
## [1] "Column number: 16. Variable name: BDIY_log"
## [1] "Column number: 17. Variable name: SPX_log"
## [1] "Column number: 18. Variable name: DX1_log"
## [1] "Column number: 19. Variable name: GC1_log"
## [1] "Column number: 20. Variable name: HO1_log"
## [1] "Column number: 21. Variable name: USCI_log"
## [1] "Column number: 22. Variable name: GNR_log"
## [1] "Column number: 23. Variable name: SHCOMP_log"
## [1] "Column number: 24. Variable name: FTSE_log"
## [1] "Column number: 25. Variable name: respLAG"
```

---

2) Develop a process that allows us to add additional drivers which are transformations of the raw input timeseries. Include the following transformations:

a. Rolling standard deviation (of arbitrary window)
b. Rolling mean (of arbitrary window)
c. Lagging (of arbitrary order)

d. Leading (of arbitrary order)
e. Differencing
f. Spread (between two input drivers)
g. Ratio (between two input drivers)
h. Product (between two input drivers)

- Function `data_trans` includes:

roll_std_dev (Rolling standard deviation) - 7 days selected roll_mean (Rolling mean) - 7 days selected lag_1 (Lagging) - order 1 selected lead (Leading) - order 1 selected diff (Differencing)

```
data_trans <- function(raw_data_1) {

  # add input driver to dataframe
  df_1 <- as.data.frame(raw_data_1)
  oil_data_1 <- raw_data_1
  oil_data_1 <- as.matrix(oil_data_1)

  # Rolling standard deviation, window = 7
  roll_std_dev <- roll::roll_sd(oil_data_1, 7)
  df_1$roll_std_dev <- roll_std_dev

  # Rolling mean, window = 7
  roll_mean <- roll::roll_mean(oil_data_1, 7)
  df_1$roll_mean <- roll_mean

  # Lagging, order = 1
  df_1$lag_1 <- dplyr::lag(raw_data_1)

  # Leading, order = 1
  df_1$lead <- dplyr::lead(raw_data_1)

  # Differencing
  Diff <- raw_data_1 %>% diff()
  Diff[1000] <- NA
  df_1$diff <- Diff

  return(df_1)
}

oil_data_trans <- data_trans(oil_data$OILPRICE)
head(oil_data_trans, n =10)
```

```
##    raw_data_1 roll_std_dev roll_mean    lag_1     lead          diff
## 1    4.640923           NA        NA       NA 4.633077 -0.0078462165
## 2    4.633077           NA        NA 4.640923 4.634049  0.0009720063
## 3    4.634049           NA        NA 4.633077 4.646312  0.0122629838
## 4    4.646312           NA        NA 4.634049 4.631520 -0.0147921680
## 5    4.631520           NA        NA 4.646312 4.627616 -0.0039035865
## 6    4.627616           NA        NA 4.631520 4.635214  0.0075979325
## 7    4.635214  0.006223043  4.635530 4.627616 4.635796  0.0005820722
## 8    4.635796  0.005767563  4.634798 4.635214 4.640055  0.0042582083
## 9    4.640055  0.006018100  4.635795 4.635796 4.645544  0.0054894923
## 10   4.645544  0.006957400  4.637437 4.640055 4.649665  0.0041213457
```

You can pass any dataframe column to the function. It will calculate all the compositions

- Building another function called `data_trans_2` to deal with 2 drivers. It contains the Ration and Product. I couldn't solve the spread between two input drivers. Maybe I needed more time to invest in a satisfatory answer.

f. Spread (between two input drivers)
g. Ratio (between two input drivers)
h. Product (between two input drivers)

```
data_trans_2 <- function(raw_data_2){

  # add input driver to dataframe
  df_2 <- as.data.frame(raw_data_2)

  # Ratio bet
  df_2$Ratio <- df_2[,1]/df_2[,2]

  # Product
  df_2$Product <- df_2[,1] * df_2[,2]

  return(df_2)
}

df_2 <- data_trans_2(oil_data[, c("OILPRICE", "respLAG")])
head(df_2, n = 10)
```

```
##    OILPRICE  respLAG     Ratio  Product
## 1  4.640923 4.631812 1.0019671 21.49589
## 2  4.633077 4.640923 0.9983093 21.50176
## 3  4.634049 4.633077 1.0002098 21.46991
## 4  4.646312 4.634049 1.0026463 21.53124
## 5  4.631520 4.646312 0.9968164 21.51949
## 6  4.627616 4.631520 0.9991572 21.43290
## 7  4.635214 4.627616 1.0016419 21.44999
## 8  4.635796 4.635214 1.0001256 21.48791
## 9  4.640055 4.635796 1.0009185 21.51035
## 10 4.645544 4.640055 1.0011831 21.55558
```

You can pass any dataframe column to the function. It will calculate the ratio and the product between the two columns

3) We must be able to have composition of transformations. Example: First calculate the difference between OILPRICE and resp_LAG, and then calculate the rolling standard deviation.

- Building `data_composition` function. This also works with two columns.

```
data_composition <- function(raw_data_3){

  # add input driver to dataframe
  df_3 <- as.data.frame(raw_data_3)
```

```r
  # Difference
  difference <- (df_3$OILPRICE - df_3$respLAG)
  df_3$difference <- difference
  difference <- as.matrix(difference)

  # Rolling standard deviation, window = 7
  roll_std <- roll::roll_sd(difference, 7)
  df_3$roll_std <- roll_std

  return(df_3)
}

df_3 <- data_composition(oil_data[,c("OILPRICE", "respLAG")])

head(df_3, n = 10)
```

```
##    OILPRICE  respLAG    difference    roll_std
## 1  4.640923 4.631812  0.0091112388          NA
## 2  4.633077 4.640923 -0.0078462165          NA
## 3  4.634049 4.633077  0.0009720063          NA
## 4  4.646312 4.634049  0.0122629838          NA
## 5  4.631520 4.646312 -0.0147921680          NA
## 6  4.627616 4.631520 -0.0039035865          NA
## 7  4.635214 4.627616  0.0075979325 0.009882852
## 8  4.635796 4.635214  0.0005820722 0.009140087
## 9  4.640055 4.635796  0.0042582083 0.008704563
## 10 4.645544 4.640055  0.0054894923 0.008868343
```

- You can pass any dataframe column to the function. It will calculate the ratio and the product between the two columns

---

4) The sequence of transformations, and which drivers they act on must be specified by the user. One of the main purposes of this challenge is to develop a generic framework to allow this.

- While calling the previously created functions, the user need to select the correct input drivers. Then select the sequence of transformation on the `final_drivers` variable below.

```r
final_drivers <- cbind(oil_data_trans, df_2, df_3)
final_drivers <-
  final_drivers[, c("raw_data_1", "roll_std_dev", "roll_mean",
                        "lag_1", "lead", "diff", "Ratio", "Product",
                        "roll_std")]

head(final_drivers, n = 10)
```

```
##    raw_data_1 roll_std_dev roll_mean    lag_1     lead          diff     Ratio
## 1    4.640923           NA        NA       NA 4.633077 -0.0078462165 1.0019671
## 2    4.633077           NA        NA 4.640923 4.634049  0.0009720063 0.9983093
## 3    4.634049           NA        NA 4.633077 4.646312  0.0122629838 1.0002098
```

```
## 4    4.646312           NA        NA 4.634049 4.631520 -0.0147921680 1.0026463
## 5    4.631520           NA        NA 4.646312 4.627616 -0.0039035865 0.9968164
## 6    4.627616           NA        NA 4.631520 4.635214  0.0075979325 0.9991572
## 7    4.635214  0.006223043  4.635530 4.627616 4.635796  0.0005820722 1.0016419
## 8    4.635796  0.005767563  4.634798 4.635214 4.640055  0.0042582083 1.0001256
## 9    4.640055  0.006018100  4.635795 4.635796 4.645544  0.0054894923 1.0009185
## 10   4.645544  0.006957400  4.637437 4.640055 4.649665  0.0041213457 1.0011831
##     Product    roll_std
## 1  21.49589          NA
## 2  21.50176          NA
## 3  21.46991          NA
## 4  21.53124          NA
## 5  21.51949          NA
## 6  21.43290          NA
## 7  21.44999 0.009882852
## 8  21.48791 0.009140087
## 9  21.51035 0.008704563
## 10 21.55558 0.008868343
```

---

5) For all drivers, either in their raw form or those that results from the application of one or several transformations, we must keep a meta data object where the sequence of transformations is stored. This will allow us to keep track of the meaning of each new driver. Combine all the drivers from their raw form or those that result from the application of one or several transformations using cbind(), named dataset as final_drivers.

- Creating meta data object:

```r
# labeling the variables
print_with_label <- function(dframe){
  stopifnot(inherits(dframe, "data.frame"))
  labs <- labelVector::get_label(dframe, names(dframe))
  labs <- sprintf("%s: %s", names(dframe), labs)
  #print(dframe)
  cat("\n")
  cat(labs, sep = "\n")
}
final_drivers <-set_label(final_drivers,
                            raw_data_1 = "target variable",
                            roll_std_dev = "Rolling standard deviation(window = 7)",
                            roll_std = "Rolling standard deviation(window = 7)",
                            roll_mean = "Rolling mean (window = 7)",
                            lag_1 = "Lagging (order = 1)",
                            lead = "Leading (order = 1)",
                            diff = "Differencing (order = 1)",
                            Ratio = "Ration between two input drivers",
                            Product = "Multiplication between two input drivers"
                           )
```

```r
contents(final_drivers)
```

```
##
```

```
## Data frame:final_drivers 1000 observations and 9 variables    Maximum # NAs:6
##
##
##                                                  Labels   Class Storage NAs
## raw_data_1                              target variable numeric  double   0
## roll_std_dev   Rolling standard deviation(window = 7)  matrix  double   6
## roll_mean                   Rolling mean (window = 7)  matrix  double   6
## lag_1                             Lagging (order = 1) numeric  double   1
## lead                              Leading (order = 1) numeric  double   1
## diff                          Differencing (order = 1) numeric  double   1
## Ratio                 Ration between two input drivers numeric  double   0
## Product      Multiplication between two input drivers numeric  double   0
## roll_std       Rolling standard deviation(window = 7)  matrix  double   6
```

```
k <- contents(final_drivers, sort='names', prlevels=FALSE)
print(k)
```

```
##
## Data frame:final_drivers 1000 observations and 9 variables    Maximum # NAs:6
##
##
##                                                  Labels   Class Storage NAs
## raw_data_1                              target variable numeric  double   0
## roll_std_dev   Rolling standard deviation(window = 7)  matrix  double   6
## roll_mean                   Rolling mean (window = 7)  matrix  double   6
## lag_1                             Lagging (order = 1) numeric  double   1
## lead                              Leading (order = 1) numeric  double   1
## diff                          Differencing (order = 1) numeric  double   1
## Ratio                 Ration between two input drivers numeric  double   0
## Product      Multiplication between two input drivers numeric  double   0
## roll_std       Rolling standard deviation(window = 7)  matrix  double   6
```

```
# saving metadata.csv
lapply(k, function(x) write.table( data.frame(x), 'metadata.csv'  , append= T, sep=',' ))
```

```
## $contents
## NULL
##
## $dim
## NULL
##
## $maxnas
## NULL
##
## $id
## NULL
##
## $rangevar
## NULL
##
## $valuesvar
## NULL
##
```

```
## $unique.ids
## NULL
##
## $range
## NULL
##
## $values
## NULL
##
## $dfname
## NULL
##
## $Levels
## NULL
##
## $longLabels
## NULL
```

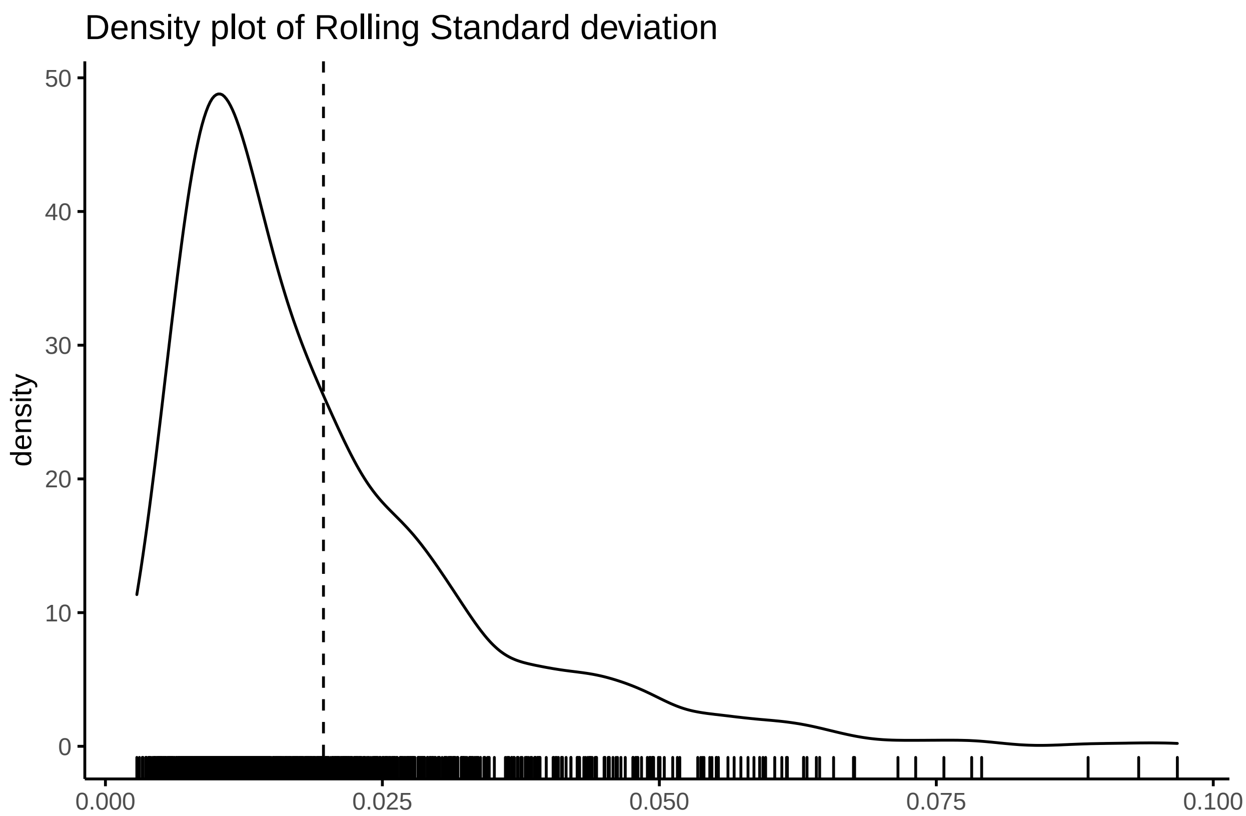
---

6) For each driver that results from the user-specified sequence of transformations, we need to assess a few statistics: Normality test Stationarity test Correlation coefficient with the target These statistics need to be stored in the meta data object. The purpose of this is, we may be interested in keeping in the final model only drivers that are normally distributed, or only drivers whose correlation with the target is above a given threshold, or another combination of such criteria.

- Normality test.

```
# normality graph
normality <- function(input_driver, p_value) {

  print(ggdensity(input_driver,
          main = "Density plot of Rolling Standard deviation",
          xlab = "",
          add = 'mean',
          ggtheme = theme_classic(),
          rug = TRUE))

  z <- shapiro.test(input_driver)
  print(shapiro.test(input_driver))

  if(z[2]>= p_value){
    print('Normally Distributed')
    x <<- sys.call()
    x <<- as.character(x)
    norm_lst <<- append(norm_lst, x)
  }
  else{
    print('Not normally distributed')
  }
}

# function call
normality(final_drivers$roll_std_dev, 0.05)
```

```
## Don't know how to automatically pick scale for object of type labelled/matrix/array. Defaulting to co
```




```
##
##  Shapiro-Wilk normality test
##
## data:  input_driver
## W = 0.84009, p-value < 2.2e-16
##
## [1] "Not normally distributed"
```

If the result of the p-value is higher or equal to the passed p-value, the name of the variable is saved on `norm_lst` and stored in a meta data object. Usually, p-value <= 0.05 means that the distribution is significantly different than normal distribution.

---

b. Stationarity test

- Augmented Dickey-Fuller (ADF) t-statistic is used to find if the series has a unit root (a series with a trend line will have a unit root and result in a large p-value). If the p-value < 0.05 then data is stationary if p-value > 0.05 then data is non-stationary.

Before the test, we remove NA values and replace them with 0.

```r
# Stationarity check

stationarity <- function(input_driver, p_value) {
  input_driver[is.na(input_driver)] <- 0
  tseries::adf.test(input_driver)
  sz <- tseries::adf.test(input_driver)

    if(sz[2]<= p_value){
    y <<- sys.call()
    y <<- as.character(y)
    stat_lst <<- append(norm_lst, y)
    print(sz)
    print('Stationary Data')
  }
  else{
    print(sz)
    print('Non-stationary Data')
  }

}

# function call
stationarity(final_drivers$roll_mean, 0.05)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  input_driver
## Dickey-Fuller = -1.7106, Lag order = 9, p-value = 0.7008
## alternative hypothesis: stationary
##
## [1] "Non-stationary Data"
```

- To test another drivers, just replace the input_driver. The data is also stored in a metadata object

```r
# saving normal data in metadata_normality.csv
try(lapply(norm_lst, function(x) write.table( data.frame(x), 'metadata_normality.csv'  , append= T, sep

# saving stationary data in stationary_normality.csv
try(lapply(stat_lst, function(x) write.table( data.frame(x), 'metadata_stationary.csv'  , append= T, se
```

---

c. Correlation coefficient with the target

```r
# Correlation coefficient
correlation <- function(input_drivers){

  input_drivers[is.na(input_drivers)] <- 0
  corr_mat=cor(input_drivers)
col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
  return(corrplot(corr_mat, method="color",
```

```
          type="upper", order="hclust",
          addCoef.col = "black",
          tl.col="black", tl.srt=45,
          # hide correlation coefficient on the principal diagonal
          diag=FALSE
          ))

}

correlation(final_drivers)
```