CookBook - [**Getting-and-Cleaning-Data-Course-Project**](https://github.com/Filareth2015/Getting-and-Cleaning-Data-Course-Project)

**Introduction**

This is an attempt to solve the important Coursera Cleaning Data Project. Basically, this is more than a cook book. I have designed it as a culmination of my notes and the steps that any student will take to solve this project. The code is based upon algorithms and the mind-set. Many of the current solutions only have the code but the mindset and the thinking process is completely missing. Here it is a humble attempt to create a solution along with the thinking process and the logic behind the steps.

**Assumptions**

* Understanding of R language
* Taken Coursera R language Online Courses
* Enjoy the R programming better than Food =)
* RStudio Application Understanding

## 1 – Download the Data – Mechanism / Commands / MindSet

In this particular case, the data is in zip format. Zip Format is a bit different from the csv files. The file is available at:

https://d396qusza40orc.cloudfront.net/getdata%2Fprojectfiles%2FUCI%20HAR%20Dataset.zip

Since the data will be unzipped and multiple files will be under the zip file it is important to have a dedicated directory for this activity. In fact, many of times, data tables have to be created. Best Bet is to create a directory for it.

if(!file.exists(“./midtermdata”)){dir.create(“./midtermdata”)}

The important part is that even in the windows file system a midtermdata folder is now also created. This is where the zip file will be downloaded shortly.

download.file(fileUrl,destfile=”./midtermdata/Dataset.zip”, mode = “wb”)

wb mode is used so that the file can be transferred as a binary file

Once the file is saved you will see the zip file in the folder that was recently created

[1] "D:/rlang"

> setwd("D:/rlang/midtermdata")

> list.files()

[1] "Dataset.zip"

It is clearly shown that the folder midtermdata has the file safely downloaded. But it is still in zip mode. Now need to unzip the file.

unzip(zipfile="./midtermdata/Dataset.zip",exdir="./midtermdata")

Once it is downloaded you will get a zip file and also the new unzipped folder

> list.files("D:/rlang/midtermdata")

[1] "Dataset.zip" "UCI HAR Dataset"

#define the path where the new folder has been unzipped

pathdata = file.path("./midtermdata", "UCI HAR Dataset")

pathdata is another way to easily show the new folder - pathway output is as follows

> pathdata

[1] "./midtermdata/UCI HAR Dataset"

files = list.files(pathdata, recursive=TRUE)

this will show the outputs of the 28 files that are unzipped in the new folder

> files

[1] "activity\_labels.txt" "features.txt"

[3] "features\_info.txt" "README.txt"

[5] "test/Inertial Signals/body\_acc\_x\_test.txt" "test/Inertial Signals/body\_acc\_y\_test.txt"

[7] "test/Inertial Signals/body\_acc\_z\_test.txt" "test/Inertial Signals/body\_gyro\_x\_test.txt"

[9] "test/Inertial Signals/body\_gyro\_y\_test.txt" "test/Inertial Signals/body\_gyro\_z\_test.txt"

[11] "test/Inertial Signals/total\_acc\_x\_test.txt" "test/Inertial Signals/total\_acc\_y\_test.txt"

[13] "test/Inertial Signals/total\_acc\_z\_test.txt" "test/subject\_test.txt"

[15] "test/X\_test.txt" "test/y\_test.txt"

[17] "train/Inertial Signals/body\_acc\_x\_train.txt" "train/Inertial Signals/body\_acc\_y\_train.txt"

[19] "train/Inertial Signals/body\_acc\_z\_train.txt" "train/Inertial Signals/body\_gyro\_x\_train.txt"

[21] "train/Inertial Signals/body\_gyro\_y\_train.txt" "train/Inertial Signals/body\_gyro\_z\_train.txt"

[23] "train/Inertial Signals/total\_acc\_x\_train.txt" "train/Inertial Signals/total\_acc\_y\_train.txt"

[25] "train/Inertial Signals/total\_acc\_z\_train.txt" "train/subject\_train.txt"

[27] "train/X\_train.txt" "train/y\_train.txt"

At this stage, the data has been successfully downloaded and now ready for interpretation.

## Step 2 - Analyze the Data

In the data the readme.me document, you will get a detailed perspective of what to expect and how to manipulate the data

|  |  |  |
| --- | --- | --- |
| Main | Test | Train |
| activity\_labels | Inertial Signals | Inertial Signals |
| features | subject\_test | subject\_train |
| features.info | X\_test | X\_train |
| README | y\_test | y\_train |

 ‘features\_info.txt’: Shows information about the variables used on the feature vector.

– ‘features.txt’: List of all features.

– ‘activity\_labels.txt’: Links the class labels with their activity name.

– ‘train/X\_train.txt’: Training set.

– ‘train/y\_train.txt’: Training labels.

– ‘test/X\_test.txt’: Test set.

– ‘test/y\_test.txt’: Test labels

Analysis shows that you can categorize the data into 4 segments

training set

test set

features

activity labels

Inertial Signal data is not required.

Additionally, features and activity label are more for tagging and descriptive than data sets.

#Reading training tables – xtrain / ytrain, subject train  
xtrain = read.table(file.path(pathdata, “train”, “X\_train.txt”),header = FALSE)  
ytrain = read.table(file.path(pathdata, “train”, “y\_train.txt”),header = FALSE)  
subject\_train = read.table(file.path(pathdata, “train”, “subject\_train.txt”),header = FALSE)

At this stage we need to evaluate the structure and names and dimensions of these new tables

> dim(xtrain)

[1] 7352 561

> dim(ytrain)

[1] 7352 1

> dim(subject\_train)

[1] 7352 1

> tail(ytrain)

V1

7347 2

7348 2

7349 2

7350 2

7351 2

7352 2

> tail(subject\_train)

V1

7347 30

7348 30

7349 30

7350 30

7351 30

7352 30

So the number of rows are identical but the difference is in the columns

Though the data set is crated, there needs to be approprioate tagging and labels. This is the outcome of 3

> names(xtrain)   [1] "V1"   "V2"   "V3"   "V4"   "V5"   "V6"   "V7"   "V8"   "V9"   "V10"  "V11"  "V12"  "V13"  "V14"

[15] "V15"  "V16"  "V17"  "V18"  "V19"  "V20"  "V21"  "V22"  "V23"  "V24"  "V25"  "V26"  "V27"  "V28"

so we cant tell what is really what value except that each row will have a sequential number

### Sanity to the Training Data Tables

here the 3 tables are without column values. we will again have to look at the readme folder and see what values correlate to the tables.

colnames(xtrain) <- features[,2]

colnames(ytrain) <-"activityId"

colnames(subject\_train) <- "subjectId"

The features list is a list of 561 unique parameters and we need to map with the train data.

> head(features[,2])

[1] tBodyAcc-mean()-X tBodyAcc-mean()-Y tBodyAcc-mean()-Z tBodyAcc-std()-X tBodyAcc-std()-Y

[6] tBodyAcc-std()-Z

477 Levels: angle(tBodyAccJerkMean),gravityMean) ... tGravityAccMag-std()

so lets evaluate the xtrain data

> tail(colnames(xtrain))

[1] "V556" "V557" "V558" "V559" "V560" "V561"

these are only col names of the xtrain data - there is only parameters that are sequential.

Execute colnames(xtrain) = features[,2]

The difference is now the data parameters have all changed

> tail(colnames(xtrain))

[1] "angle(tBodyAccJerkMean),gravityMean)" "angle(tBodyGyroMean,gravityMean)"

[3] "angle(tBodyGyroJerkMean,gravityMean)" "angle(X,gravityMean)"

[5] "angle(Y,gravityMean)" "angle(Z,gravityMean)"

Now lets try out for the ytrain data

> head(ytrain)

V1

1 5

2 5

3 5

4 5

5 5

6 5

colnames(ytrain) = "activityId"

again the only column is now changed to

> head(ytrain)

activityId

1 5

2 5

3 5

4 5

5 5

6 5

> head(subject\_train)

V1

1 1

2 1

3 1

4 1

5 1

6 1

head(subject\_train)

subjectId

1 1

2 1

3 1

4 1

5 1

6 1

now we would do the same sanity check for the test data.

the third element is the activity labels:

> head(activityLabels)

V1 V2

1 1 WALKING

2 2 WALKING\_UPSTAIRS

3 3 WALKING\_DOWNSTAIRS

4 4 SITTING

5 5 STANDING

6 6 LAYING

now when we give it the sanity check then through this command

colnames(activityLabels) <- c('activityId','activityType')

head(activityLabels)

activityId activityType

1 1 WALKING

2 2 WALKING\_UPSTAIRS

3 3 WALKING\_DOWNSTAIRS

4 4 SITTING

5 5 STANDING

6 6 LAYING

### Merging of all the data

mrg\_train <- cbind(yxtrain, subject\_train, xytrain)

with this command the dim of this particular data table is 7352 and 563

> head(mrg\_train)

activityId subjectId tBodyAcc-mean()-X tBodyAcc-mean()-Y tBodyAcc-mean()-Z tBodyAcc-std()-X

1 5 1 0.2885845 -0.02029417 -0.1329051 -0.9952786

tBodyAcc-std()-Y tBodyAcc-std()-Z tBodyAcc-mad()-X tBodyAcc-mad()-Y tBodyAcc-mad()-Z tBodyAcc-max()-X

this nomencluatre creates a database with the following elements

> names(mrg\_train)

[1] "activityId" "subjectId"

[3] "tBodyAcc-mean()-X" "tBodyAcc-mean()-Y"

[5] "tBodyAcc-mean()-Z" "tBodyAcc-std()-X"

this seems more reasonable to place the beginning and as compared as the end

now we have to do this again for the test data

#Merging the train and test data - important outcome of the project

mrg\_train = cbind(ytrain, subject\_train, xtrain)

mrg\_test = cbind(ytest, subject\_test, xtest)

#Create the main data table merging both table tables - this is the outcome of 1

alldatamerge = rbind(mrg\_train, mrg\_test)

This is the final output of the data merge requirement

## 2. Extracting only the measurements on the mean and standard deviation for each measurement

This technique is to look at the parameters and only find the ones that show mean and standard deviations. So basically a subset of the current data

> colnames(setAllInOne)

[1] "activityId" "subjectId" "tBodyAcc-mean()-X"

[4] "tBodyAcc-mean()-Y" "tBodyAcc-mean()-Z" "tBodyAcc-std()-X"

[7] "tBodyAcc-std()-Y" "tBodyAcc-std()-Z" "tBodyAcc-mad()-X"

[10] "tBodyAcc-mad()-Y" "tBodyAcc-mad()-Z" "tBodyAcc-max()-X"

[13] "tBodyAcc-max()-Y" "tBodyAcc-max()-Z" "tBodyAcc-min()-X"

[16] "tBodyAcc-min()-Y" "tBodyAcc-min()-Z" "tBodyAcc-sma()"

[19] "tBodyAcc-energy()-X" "tBodyAcc-energy()-Y" "tBodyAcc-energy()-Z"

[22] "tBodyAcc-iqr()-X" "tBodyAcc-iqr()-Y" "tBodyAcc-iqr()-Z"

We do not need all the data only the ones with the mean and the std values

> head(setForMeanAndStd)

activityId subjectId tBodyAcc-mean()-X tBodyAcc-mean()-Y tBodyAcc-mean()-Z tBodyAcc-std()-X tBodyAcc-std()-Y tBodyAcc-std()-Z

1 5 1 0.2885845 -0.02029417 -0.1329051 -0.9952786 -0.9831106 -0.9135264

2 5 1 0.2784188 -0.01641057 -0.1235202 -0.9982453 -0.9753002 -0.9603220

3 5 1 0.2796531 -0.01946716 -0.1134617 -0.9953796 -0.9671870 -0.9789440

4 5 1 0.2791739 -0.02620065 -0.1232826 -0.9960915 -0.9834027 -0.9906751

5 5 1 0.2766288 -0.01656965 -0.1153619 -0.9981386 -0.9808173 -0.9904816

6 5 1 0.2771988 -0.01009785 -0.1051373 -0.9973350 -0.9904868 -0.9954200

> head(setWithActivityNames)

activityId subjectId tBodyAcc-mean()-X tBodyAcc-mean()-Y tBodyAcc-mean()-Z tBodyAcc-std()-X tBodyAcc-std()-Y tBodyAcc-std()-Z

1 1 7 0.3016485 -0.026883636 -0.09579580 -0.3801243 -0.1913292 0.34055774

2 1 5 0.3433592 -0.003426473 -0.10154465 -0.2011536 0.1331536 -0.31817123

3 1 6 0.2696745 0.010907280 -0.07494859 -0.3366399 0.1462498 -0.44557070

4 1 23 0.2681938 -0.012730069 -0.09365263 -0.3836978 -0.2038974 0.14800031

5 1 7 0.3141912 -0.008695973 -0.12456099 -0.3558778 -0.1657995 0.40672936

6 1 7 0.2032763 -0.009764083 -0.15139663 -0.4286661 -0.2610000 0.07675962

fBodyBodyGyroJerkMag-meanFreq()

10294 0.15872752

10295 0.07447170

10296 0.10185944

10297 -0.06624872

10298 -0.04646651

10299 -0.01038585

> head(str(setForMeanAndStd))

'data.frame': 10299 obs. of 81 variables:

$ activityId : int 5 5 5 5 5 5 5 5 5 5 ...

$ subjectId : int 1 1 1 1 1 1 1 1 1 1 ...

$ tBodyAcc-mean()-X : num 0.289 0.278 0.28 0.279 0.277 ...

$ tBodyAcc-mean()-Y : num -0.0203 -0.0164 -0.0195 -0.0262 -0.0166 ...

$ tBodyAcc-mean()-Z : num -0.133 -0.124 -0.113 -0.123 -0.115 ...

$ tBodyAcc-std()-X : num -0.995 -0.998 -0.995 -0.996 -0.998 ...

fBodyBodyGyroJerkMag-meanFreq() activityType

10294 0.44505406 LAYING

10295 0.38090105 LAYING

10296 0.05219279 LAYING

10297 0.25038102 LAYING

10298 0.14113754 LAYING

10299 0.28656318 LAYING

Hence you can see that there is a need to get only the required nomenclature with the grepl function. This works well if you have the complete data.

# Need step is to read all the values that are available

colNames = colnames(setAllInOne)

#Need to get a subset of all the mean and standards and the correspondongin activityID and subjectID

mean\_and\_std = (grepl("activityId" , colNames) | grepl("subjectId" , colNames) | grepl("mean.." , colNames) | grepl("std.." , colNames))

#A subtset has to be created to get the required dataset

setForMeanAndStd <- setAllInOne[ , mean\_and\_std == TRUE]

# 3. Use descriptive activity names to name the activities in the data set

setWithActivityNames = merge(setForMeanAndStd, activityLabels, by='activityId', all.x=TRUE)

# 5. From the data set in step 4, creates a second, independent tidy data set with the average of each variable for each activity and each subject

# New tidy set has to be created

secTidySet <- aggregate(. ~subjectId + activityId, setWithActivityNames, mean)

secTidySet <- secTidySet[order(secTidySet$subjectId, secTidySet$activityId),]

#The last step is to write the ouput to a text file

write.table(secTidySet, "secTidySet.txt", row.name=FALSE)