Getting and Cleaning Data Course Project
Code Book
R Lockhart – 2014-08-23

This Code Book describes the creation and structure of a tidy data set of measurements from 30 test subjects performing 6 different activities multiple times. The data were supplied as multiple data sets and directories; the output is a single data set. This code book is organized into three sections after this introduction:

- Input Data for This Analysis
- Output Data from This Analysis
- Detailed Steps to Produce the Output Data Set

The final result of this exercise is a single data set that includes the mean for each measurement that was either a mean or a standard deviation, summarized by subject and activity. With 30 subjects performing 6 activities, the final data set has 180 rows (6x30) and 68 columns that are described below in detail.

Input Data for This Analysis

The input data contained a number of different files to describe the measurements take from accelerometer and gyroscope 3-axial raw signals from Samsung Galaxy S smartphones attached to study participants (the subjects). Each subject performed 6 different activities multiple times. 30% of the participants (that is, 9 subjects) were identified as a *test* group, while the remaining 70% (21 subjects) were identified as the *train* group. Results for the two groups were provided in separate subdirectories with identical structures. In each subdirectory, these three files in aggregate contained the results:

subject_test.txt	the anonymous subject number of the participant
y_test.txt	the activity number (1:6) being performed
x_test.txt	561 measurements taken from the smartphone or calculated

Those file names are from the *test* directory. Filenames in the *train* directory were similar: subject_train.txt, y_train.txt, and x_train.txt. The structure of the *test* data as originally presented can be envisioned as shown here, with each table corresponding to a separate file in the *test* directory (the *train* directory has an identical structure). These are sample values:

Subject	Activity	Meas1	Meas2	Meas3	Meas4
2	1	.0811	.0333	.0572	.611
4	1	.0644	.0211	.0444	.575
2	2	.0733	.0415	.0369	.751
…	…	…	…	…	…

In the *test* directory, each of these 3 files had 2,947 rows. In the *train* directory, each of these 3 files had 7,352 rows. There is no key to implicitly associate any row in one file with any row in the other two files. In absence of any implicit linkage, the analysis assumes that row 1 of the subject file corresponds to row 1 of the activity and measurement files, row #2 of each file corresponds, and so on.

Two other data sets provided key information needed for creation of a tidy date set:

features.txt	variable names for the 561 measurements recorded in the x_test.txt and x_train.txt files
activity_labels.txt	6x2 matrix relating the 6 activity numbers to activity names

The data also included a ReadMe file and a brief description of the measurements taken or calculated. At just over one page for 561 variables, this description was indeed brief.

Output Data from This Analysis

The output data set contains one row for each activity performed by each subject, so as noted above there are 180 rows in the data set: 30 subjects, each performing 6 activities. The output data set contains only variables that are either a mean or a standard deviation, plus subject and activity name. Further, the value presented for each variable is the mean of all measurements for that subject/activity/variable combination.

The output data set has been uploaded as both a text file, per the course project assumptions, and also as a comma-separated values (.csv) file. The .csv file was included because it is easier to read the output as compared to the text file. However it does require presence of spreadsheet software such as Microsoft Excel that is capable of reading .csv files and displaying them in a tabular format.

For ease of viewing, here is a cropped screen shot of the resulting data set in .csv format as displayed in Excel. As required, the data set is sorted by activity within subject number. The text file uploaded as part of the exercise contains the same data – as shown below in the analysis of the source code, both files are written from the same data table.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Subject	Activity	tBodyAcc-	tBodyAcc-	tBodyAcc-	tBodyAcc-	tBodyAcc-	tBodyAcc-	tGravityA	tGravityA	tGravityA	tGravityA
2	1	LAYING	0.221598	-0.04051	-0.1132	-0.92806	-0.83683	-0.82606	-0.24888	0.70555	0.445818	-0.89683
3	1	SITTING	0.261238	-0.00131	-0.10454	-0.97723	-0.92262	-0.93959	0.83151	0.204412	0.332044	-0.96846
4	1	STANDING	0.278918	-0.01614	-0.1106	-0.99576	-0.97319	-0.97978	0.942952	-0.27298	0.013491	-0.99376
5	1	WALKING	0.277331	-0.01738	-0.11115	-0.28374	0.114461	-0.26003	0.935223	-0.28217	-0.0681	-0.97661
6	1	WALKING_DOWNSTAIRS	0.289188	-0.00992	-0.10757	0.030035	-0.03194	-0.23043	0.931874	-0.26661	-0.06212	-0.95056
7	1	WALKING_UPSTAIRS	0.255462	-0.02395	-0.0973	-0.35471	-0.00232	-0.01948	0.893351	-0.36215	-0.0754	-0.95637
8	2	LAYING	0.281373	-0.01816	-0.10725	-0.97406	-0.98028	-0.98423	-0.50975	0.752537	0.646835	-0.95901
9	2	SITTING	0.277087	-0.01569	-0.10922	-0.98682	-0.9507	-0.95983	0.940477	-0.10563	0.198727	-0.97999
10	2	STANDING	0.277911	-0.01842	-0.10591	-0.98727	-0.9573	-0.94974	0.896929	-0.37006	0.129747	-0.98669
11	2	WALKING	0.276427	-0.01859	-0.1055	-0.42364	-0.07809	-0.42526	0.913017	-0.34661	0.084727	-0.97269
12	2	WALKING_DOWNSTAIRS	0.277615	-0.02266	-0.11681	0.046367	0.262882	-0.10284	0.861831	-0.32578	-0.04389	-0.94036
13	2	WALKING_UPSTAIRS	0.247165	-0.02141	-0.15251	-0.30438	0.108027	-0.11212	0.790717	-0.41621	-0.19589	-0.93441
14	3	LAYING	0.275517	-0.01896	-0.1013	-0.98278	-0.96206	-0.96369	-0.24176	0.837032	0.488703	-0.98251
15	3	SITTING	0.257198	-0.0035	-0.09836	-0.97101	-0.85662	-0.87511	0.901099	0.127303	0.139021	-0.95732
16	3	STANDING	0.280047	-0.01434	-0.10162	-0.96674	-0.89345	-0.91142	0.935031	-0.30174	0.024763	-0.98202
17	3	WALKING	0.275567	-0.01718	-0.11267	-0.36036	-0.06991	-0.38741	0.936507	-0.26199	-0.13811	-0.97777
18	3	WALKING_DOWNSTAIRS	0.292423	-0.01936	-0.11614	-0.05741	-0.03315	-0.36224	0.939058	-0.22883	-0.10235	-0.95006
19	3	WALKING_UPSTAIRS	0.26082	-0.03241	-0.11006	-0.31312	0.011628	-0.36975	0.883533	-0.38285	-0.16294	-0.94404
20	4	LAYING	0.263559	-0.015	-0.11069	-0.95419	-0.94171	-0.96267	-0.42066	0.915165	0.341531	-0.9212
21	4	SITTING	0.271538	-0.00716	-0.10587	-0.98031	-0.89022	-0.9322	0.869303	0.211623	0.11012	-0.98141
22	4	STANDING	0.2805	-0.00949	-0.09616	-0.97692	-0.8616	-0.89688	0.956198	-0.0759	0.166894	-0.97291
23	4	WALKING	0.278582	-0.01484	-0.1114	-0.44083	-0.07883	-0.58625	0.964	-0.08585	0.127764	-0.98383
24	4	WALKING_DOWNSTAIRS	0.279965	-0.0098	-0.10678	0.011194	-0.2186	-0.47919	0.947732	-0.06209	0.148715	-0.95526
25	4	WALKING_UPSTAIRS	0.270877	-0.03198	-0.11422	-0.20493	-0.06669	-0.37214	0.946264	-0.23294	0.084168	-0.9585
26	5	LAYING	0.278334	-0.0183	-0.10794	-0.96593	-0.9693	-0.96856	-0.48347	0.95489	0.263645	-0.9457

Getting and Cleaning Data Course Project – Code Book

Each row of the output data set contains 68 variables; there would be 68 columns in the above screen shot if they could all fit on the screen. The variables are (FFT = Fast Fourier Transform):

Variable Name	Description
Subject	Subject # (1-30); anonymous identifier of each participant in the test
Activity	Factor variable indicating the activity being performed. One of: • WALKING • WALKING_UPSTAIRS • WALKING_DOWNSTAIRS • SITTING • STANDING • LAYING
tBodyAcc-mean()-X	Time signal: Body Acceleration mean on the X axis
tBodyAcc-mean()-Y	Time signal: Body Acceleration mean on the Y axis
tBodyAcc-mean()-Z	Time signal: Body Acceleration mean on the Z axis
tBodyAcc-std()-X	Time signal: Body Acceleration standard deviation on the X axis
tBodyAcc-std()-Y	Time signal: Body Acceleration standard deviation on the Y axis
tBodyAcc-std()-Z	Time signal: Body Acceleration standard deviation on the Z axis
tGravityAcc-mean()-X	Time signal: Gravity Acceleration mean on the X axis
tGravityAcc-mean()-Y	Time signal: Gravity Acceleration mean on the Y axis
tGravityAcc-mean()-Z	Time signal: Gravity Acceleration mean on the Z axis
tGravityAcc-std()-X	Time signal: Gravity Acceleration standard deviation on the X axis
tGravityAcc-std()-Y	Time signal: Gravity Acceleration standard deviation on the Y axis
tGravityAcc-std()-Z	Time signal: Gravity Acceleration standard deviation on the Z axis
tBodyAccJerk-mean()-X	Time signal: Body Acceleration Jerk mean on the X axis
tBodyAccJerk-mean()-Y	Time signal: Body Acceleration Jerk mean on the Y axis
tBodyAccJerk-mean()-Z	Time signal: Body Acceleration Jerk mean on the Z axis
tBodyAccJerk-std()-X	Time signal: Body Acceleration Jerk standard deviation on the X axis
tBodyAccJerk-std()-Y	Time signal: Body Acceleration Jerk standard deviation on the Y axis
tBodyAccJerk-std()-Z	Time signal: Body Acceleration Jerk standard deviation on the Z axis
tBodyGyro-mean()-X	Time signal: Body Angular Velocity mean on the X axis
tBodyGyro-mean()-Y	Time signal: Body Angular Velocity mean on the Y axis
tBodyGyro-mean()-Z	Time signal: Body Angular Velocity mean on the Z axis
tBodyGyro-std()-X	Time signal: Body Angular Velocity standard deviation on the X axis
tBodyGyro-std()-Y	Time signal: Body Angular Velocity standard deviation on the Y axis
tBodyGyro-std()-Z	Time signal: Body Angular Velocity standard deviation on the Z axis
tBodyGyroJerk-mean()-X	Time signal: Body Angular Velocity Jerk mean on the X axis
tBodyGyroJerk-mean()-Y	Time signal: Body Angular Velocity Jerk mean on the Y axis
tBodyGyroJerk-mean()-Z	Time signal: Body Angular Velocity Jerk mean on the Z axis
tBodyGyroJerk-std()-X	Time signal: Body Angular Velocity Jerk standard deviation on the X axis
tBodyGyroJerk-std()-Y	Time signal: Body Angular Velocity Jerk standard deviation on the Y axis
tBodyGyroJerk-std()-Z	Time signal: Body Angular Velocity Jerk standard deviation on the Z axis
tBodyAccMag-mean()	Time signal: Body Acceleration Magnitude mean
tBodyAccMag-std()	Time signal: Body Acceleration Magnitude standard deviation
tGravityAccMag-mean()	Time signal: Gravity Acceleration Magnitude mean
tGravityAccMag-std()	Time signal: Gravity Acceleration Magnitude standard deviation
tBodyAccJerkMag-mean()	Time signal: Body Acceleration Jerk Magnitude mean
tBodyAccJerkMag-std()	Time signal: Body Acceleration Jerk Magnitude standard deviation
tBodyGyroMag-mean()	Time signal: Body Angular Velocity Magnitude mean
tBodyGyroMag-std()	Time signal: Body Angular Velocity Magnitude standard deviation

Getting and Cleaning Data Course Project – Code Book

Variable Name	Description
tBodyGyroJerkMag-mean()	Time signal: Body Angular Velocity Jerk Magnitude mean
tBodyGyroJerkMag-std()	Time signal: Body Angular Velocity Jerk Magnitude standard deviation
fBodyAcc-mean()-X	FFT: Body Acceleration mean on the X axis
fBodyAcc-mean()-Y	FFT: Body Acceleration mean on the Y axis
fBodyAcc-mean()-Z	FFT: Body Acceleration mean on the Z axis
fBodyAcc-std()-X	FFT: Body Acceleration standard deviation on the X axis
fBodyAcc-std()-Y	FFT: Body Acceleration standard deviation on the Y axis
fBodyAcc-std()-Z	FFT: Body Acceleration standard deviation on the Z axis
fBodyAccJerk-mean()-X	FFT: Body Acceleration Jerk mean on the X axis
fBodyAccJerk-mean()-Y	FFT: Body Acceleration Jerk mean on the Y axis
fBodyAccJerk-mean()-Z	FFT: Body Acceleration Jerk mean on the Z axis
fBodyAccJerk-std()-X	FFT: Body Acceleration Jerk standard deviation on the X axis
fBodyAccJerk-std()-Y	FFT: Body Acceleration Jerk standard deviation on the Y axis
fBodyAccJerk-std()-Z	FFT: Body Acceleration Jerk standard deviation on the Z axis
fBodyGyro-mean()-X	FFT: Body Angular Velocity mean on the X axis
fBodyGyro-mean()-Y	FFT: Body Angular Velocity mean on the Y axis
fBodyGyro-mean()-Z	FFT: Body Angular Velocity mean on the Z axis
fBodyGyro-std()-X	FFT: Body Angular Velocity standard deviation on the X axis
fBodyGyro-std()-Y	FFT: Body Angular Velocity standard deviation on the Y axis
fBodyGyro-std()-Z	FFT: Body Angular Velocity standard deviation on the Z axis
fBodyAccMag-mean()	FFT: Body Acceleration Magnitude mean
fBodyAccMag-std()	FFT: Body Acceleration Magnitude standard deviation
fBodyBodyAccJerkMag-mean()	FFT: Body Acceleration Jerk Magnitude mean
fBodyBodyAccJerkMag-std()	FFT: Body Acceleration Jerk Magnitude standard deviation
fBodyBodyGyroMag-mean()	FFT: Body Angular Velocity Magnitude mean
fBodyBodyGyroMag-std()	FFT: Body Angular Velocity Magnitude standard deviation
fBodyBodyGyroJerkMag-mean()	FFT: Body Angular Velocity Jerk Magnitude mean
fBodyBodyGyroJerkMag-std()	FFT: Body Angular Velocity Jerk Magnitude standard deviation

Detailed Steps to Produce the Output Data Set

Given the input data described above, the process to arrive at the required output can be summarized as follows:

1. Read data from *test* and *train* subdirectories: subjects, activities, and measurements
2. Read the names of variables from features.txt to create column names for the final data set
3. Set the column names in the *test* and *train* data sets so that those column names will be inherited when a single data set is created
4. Create a single *test* data set by combining its subject, activity, and measurement data then do same for the *train* data set.
5. Bind together the *test* and *train* data sets to create a single data set of all observations of subjects, activities, and measurements for each
6. Per the project instructions, extract from each row only those measurements that are either a mean or a standard deviation.
7. Replace the activity numbers (1:6) with a factor variable that describes the activity, as listed earlier in this Code Book
8. Calculate the mean of the extracted measurements by subject and activity, to create a new data table that has only one row for subject/activity combination, sorted by activity within subject.
9. Write the data table as a .txt file

Each of these 9 steps is described below via the R source code that accomplishes that step.

```
## run_analysis.R - GCD Course Project - RPL - 2014-08-17
## assumes that working directory is "UCI HAR Dataset" with subdirectories "test" and "train"

run_analysis <- function () {
```

The follwing analysis presents the snippets of code used to perform each of the above ten steps. The full source of the script is also included in the repository that contains this code book. Throughout this script, data tables have been preferred to data frames for speed of processing and ease of coding for subsetting, data merges, and column calculations. As required for creation of a tidy data set, this function accepts no parameters.

Read data from *test* and *train* subdirectories: subjects, activities, and measurements

```
## read data from "test" subdirectory (use LaF package to save time with x_test.txt)
    x_test_read  <- laf_open_fwf("./test/x_test.txt", column_widths = rep(16,times=561),
                                 column_types=rep("numeric", times=561))

    x_test       <- x_test_read[,]
    y_test       <- fread ("./test/y_test.txt", stringsAsFactors=F)
    subject_test <- fread ("./test/subject_test.txt", stringsAsFactors=F)

## read data from "train" subdirectory (use LaF package to save time with x_train.txt)
    x_train_read  <- laf_open_fwf ("./train/x_train.txt", column_widths = rep(16,times=561),
                                   column_types=rep("numeric", times=561))

    x_train       <- x_train_read[,]
    y_train       <- fread ("./train/y_train.txt", stringsAsFactors=F)
    subject_train <- fread ("./train/subject_train.txt", stringsAsFactors=F)
```

The subject and activity files (e.g., subject_test.txt and y_test.txt) were easily read with fread. Given the size of the measurement files (x_test.txt and x_train.txt), initial attempts to read these files as tables proved extremely long in duration. Therefore the LaF package was used, reading each file as a fixed-width file (FWF) with 561 columns of exactly 16 characters each. LaF read each of the large files in less than one second.

Read the names of variables from features.txt to create column names for the final data set

```
## create vector of feature column names from features.txt (which is in current directory)
    feature_list <- read.table ("features.txt", sep=" ", stringsAsFactors=F)
    feature_colnames <- feature_list[,2]
```

This step simply reads the 561 variable names from the features.txt file and creates a character vector of length 561 for use as column names in the following step.

Set the column names in the *test* and *train* data sets so that those column names will be inherited when a single data set is created

```
## set column names in each table; will be inherited below when creating full tables
    setnames (subject_test,   colnames(subject_test),    "Subject")
    setnames (y_test,         colnames(y_test),          "Activity")
    setnames (x_test,         colnames(x_test),          feature_colnames)

    setnames (subject_train,  colnames(subject_train),   "Subject")
    setnames (y_train,        colnames(y_train),         "Activity")
    setnames (x_train,        colnames(x_train),         feature_colnames)
```

This code replaces default column names from reading the provided text files (V1, V2, V3, …) with desired column names from the features.txt file. The column names for subjects and activities have been hard-coded. The *setnames* function was used because it modifies column names in place, rather than creating another copy of the data table.

Create a single *test* data set by combining its subject, activity, and measurement data then do same for the *train* data set.

```
## create full train and test data tables, for each combining: subjects, activities, readings
    full_test  <- data.table (subject_test,  y_test, x_test)
    full_train <- data.table (subject_train, y_train,x_train)
```

These two lines of code create a *test* data table and a *train* data table by combining the three existing tables (subjects, activities, and measurements) into a single data table. Fortunately all three data sets have the same number of rows, so the column-oriented combination is straightforward.

Bind together the test and train data sets to create a single data set of all observations of subjects, activities, and measurements for each

```
## combine test and train data into a single table
    test_train <- list (full_test, full_train)
    full_data  <- rbindlist (test_train)
```

Here the *rbindlist* function creates a new data set of both *test* and *train* data, with the *test* data first and then all of the *train* data following. At this point the data table is not sorted. Given the number of *test* and *train* rows provided as input, this data table has 10,299 rows. Given the 561 measurements, plus subject and activity codes, this data table has 563 rows. In all, the dimensions of this data table are 10299x563.

Per the project instructions, extract from each row only those measurements that are either a mean or a standard deviation.

```
## extract only the mean and standard deviation measurements
## first build logical vector of columns that have 'mean' and 'std' in their column names
## double backslash (\\) necessary to search for mean() and std()
    temp_cols <- grepl ("mean\\(\\)",colnames(full_data)) |
                 grepl ("std\\(\\)",colnames(full_data))

## add the Subject and Activity columns to selection
## then OR the two selections to get logical vector of all desired columns
    temp_cols2 <- grepl ("Subject", colnames(full_data)) | grepl ("Activity",colnames(full_data))
    keep_cols <- temp_cols | temp_cols2

## create new table with only the desired columns (requires null row argument)
    keep_table <- subset (full_data, , keep_cols)
```

This code builds a logical vector of length 563 (the number of columns in the data table) to determine which columns will be retained. The *grepl* function is used to create logical vectors indicating which columns have either "mean()" or "std()" in their variable names, and then a logical OR of those two *grepls*, along with a logical OR of "Subject" and "Activity", creates the desired logical vector (*keep_cols* in the source code) to indicate which columns will be retained.

Finally, a new data table (*keep_table*) is created as a subset of the full data table, retaining only the desired columns based upon the *keep_cols* logical vector. The row argument to the *subset* function is null so that all rows will be processed.

Replace the activity numbers (1:6) with a factor variable that describes the activity, as listed earlier in this Code Book

```
## replace activity code (from y_test & y_train) with activity name as factor in Activity column
## (LAYING should really be LYING unless test subjects were hens)
    activity_names <- read.table("activity_labels.txt", stringsAsFactors=T)
    acnames <- activity_names[,2]
    keep_table [, Activity:=acnames[Activity]]
```

This code reads the six activity names from the file activity_labels.txt as factors and then extracts them into a factor vector of length 6 (*acnames*). Then the := function of the data.table package is used to replace the existing activity code with the corresponding activity factor within the data table *keep_table*.

Calculate the mean of the extracted measurements by subject and activity, to create a new data table that has only one row for subject/activity combination, sorted by activity within subject.

```
## set Subject and Activity as keys to data table for summarization
    keycols <- c("Subject", "Activity")
    setkeyv (keep_table, keycols)

## create final tidy data set with mean of each column by key (subject, activity)
    final_tab <- keep_table[, lapply(.SD,mean), by = key(keep_table)]
```

This code sets up the Subject and Activity columns of the *keep_table* data table by which sorting and summarization will occur, using the *setkeyv* feature of the data.table package. The data table *final_tab* is created from *keep_table*, using *lapply* and the .SD notation to indicate a subset of the data table (excluding key columns) on which to calculate the mean with *lapply*.

The data table *final_tab* is the tidy data set required as output from the script.

Write the data table as a .txt file

```
## write tidy data set as csv in working directory to easily view results
## keep one old copy just in case
    check_csv <- "analysisresult.csv"
    if (file.exists ("oldtest.csv"))  {file.remove("oldtest.csv")}
    if (file.exists (check_csv))      {file.rename(check_csv,"oldtest.csv")}
    write.csv (final_tab, check_csv, row.names=F)

## write the text file required as upload for the course project
    if (file.exists ("final_table.txt"))  {file.remove("final_table.txt")}
    write.table (final_tab, "final_table.txt", sep=" ", quote=F, row.names=F)

## return file dimensions of new tidy data set to indicate completion
    dim (final_tab)

}
```

The R script ends by writing the data table *final_tab* as both a .txt file (required by the course project instructions) and a .csv file, which is easier to review for validating results. Finally the script returns the dimensions of the tidy data set to indicate completion of the script.