ReadMe

Coleen Smith

4/2/2019

Assignment

This assignment will create one R script called run_analysis.R that does the following. 1. Merges the training and the test sets to create one data set. 2. Extracts only the measurements on the mean and standard deviation for each measurement. 3. Uses descriptive activity names to name the activities in the data set. 4. Appropriately labels the data set with descriptive variable names. 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.

Raw Data

Data collected from the accelerometers from the Samsung Galaxy S smartphone was made available to download. A full description is available at the site where the data was obtained:

http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones

Here are the data for the project:

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

UCI HAR Dataset

After downloading and unzipping files, I renamed the directory from "UCI HAR Dataset" to "UCI_HAR_Dataset" to avoid possible issues with spaces in a pathname.

```r
dir("UCI_HAR_Dataset")
```

```
## [1] "activity_labels.txt" "features_info.txt"   "features.txt"
## [4] "README.txt"          "test"                "train"
```

Parent Directory Files

- activity_labels.txt - six activities, two columns (number & name), all uppercase, no headers
- features.txt - 561 variables, two columns (number & name), mixed case, no headers
  - 46 occurrences of "mean"
  - 10 occurrences of "Mean", all as part of an ordered pair, in 7 rows (555-561)
  - 33 occurrences of "std", only single occurrence/row
- features_info.txt - information on the variables and calculations listed in features.txt
- README.txt - overview of project
- test (folder) - data from first 30 subjects

- train (folder)- data from remaining 70 subjects

Based on the information included in features_info.txt, the files in the test and train and train folders are formatted and organized the same way. Since the test folder files are smaller with data from just 30 subjects, they became the starting point to understand the data and prepare the files for merging.

Test Folder Files

```r
dir("UCI_HAR_Dataset/test")
```

```
## [1] "Inertial Signals" "subject_test.txt" "X_test.txt"
## [4] "y_test.txt"
```

To get a sense of how the data were arranged, I also looked at each of the files using atom.

test(folder):
- inertial signals (folder):
- each .txt - a set of 35 element vectors.
- subject_test.txt - a single column of numbers, no column headers.
- x_test.txt - rows of 561 element vectors, no column headers.
- y_test.txt - a single column of numbers, no column headers.
- "1" 496 occurrences
- "2" 471 occurrences
- "3" 420 occurrences
- "4" 491 occurrences
- "5" 532 occurrences
- "6" 537 occurrences
- Total = 2947

Train Folder Files

```r
dir("UCI_HAR_Dataset/train")
```

```
## [1] "Inertial Signals"  "subject_train.txt" "X_train.txt"
## [4] "y_train.txt"
```

Train (folder) mirrors file structure and organization seen in test files, only larger because files represent 70 subjects rather than 30.

1. Merge the training and the test sets to create one data set

Based on this assignment's requirements, I decided to follow Wickham approach to tidying data when there is one type of data in multiple tables (Ref: Wickham, Section 3.5).

```
1. Read the files into a list of tables.
2. For each table, add a new column that records the original file name.
3. Combine all tables into a single table.
```

In this case, test and train data are one type of data observed for separate groups of people. Thinking ahead to tidy data,

```
1. Each variable forms a column.
    - subject (100)
    - activity (6)
```

```
    - features (561)
2. Each observation forms a row.
    - An observation is of a given subject
    - performing a specific activity
    - represented by 561 measured features
3. Each type of observational unit forms a table.
    - Interim: test and train in separate tables
    - Final: test and train data in single table
```

Since the subject, activity and measurements are recorded in separate tables within test and train, the first step would be to create a test data file with subject, activity and measurement observations. Then, do the same for the train data. Since they have the same variables, combine the interim test and train data files using rbind().

Read test files into a list of tables: subject_test, X_test, and Y_test data

From a bookkeeping perspective, a good place to start is with the number lines per file. The number of data lines in the source file should equal the number of lines in the destination file. [Back in the dark ages when file transfer protocol included a magnetic tape and FedEx Overnight, I spent a day looking for 13 "lost" lines out of 25,000 - lesson burned.]

```
## [1] 561
## attr(,"lastLineHasNewline")
## [1] TRUE
```

```
## [1] 2947
## attr(,"lastLineHasNewline")
## [1] TRUE
```

```
## [1] 2947
## attr(,"lastLineHasNewline")
## [1] TRUE
```

```
## [1] 2947
## attr(,"lastLineHasNewline")
## [1] TRUE
```

According to README.txt and features_info.txt, features.txt represents the names of the 561 variables recorded. Since the 561 variable list already contains the summary data (mean & std) required to comoplete the assignment, the merge for this assignment will not include the additional detail observations contained in the inertial signals test and train folders.

Using read.table, create a data frame from each file: subject_test.txt, X_test.txt and Y_test.txt.

```
subject_test <- read.table("UCI_HAR_Dataset/test/subject_test.txt", header = FALSE)
str(subject_test)
```

```
## 'data.frame':    2947 obs. of  1 variable:
##  $ V1: int  2 2 2 2 2 2 2 2 2 2 ...
```

```
X_test <- read.table("UCI_HAR_Dataset/test/X_test.txt", header = FALSE)
## This is a looooong output, just need highlights
str(X_test, list.len = 5)
```

```
## 'data.frame':    2947 obs. of  561 variables:
##  $ V1  : num  0.257 0.286 0.275 0.27 0.275 ...
##  $ V2  : num  -0.0233 -0.0132 -0.0261 -0.0326 -0.0278 ...
##  $ V3  : num  -0.0147 -0.1191 -0.1182 -0.1175 -0.1295 ...
```

```
##  $ V4  : num  -0.938 -0.975 -0.994 -0.995 -0.994 ...
##  $ V5  : num  -0.92 -0.967 -0.97 -0.973 -0.967 ...
##   [list output truncated]
```

```
Y_test <- read.table("UCI_HAR_Dataset/test/Y_test.txt", header = FALSE)
str(Y_test)
```

```
## 'data.frame':    2947 obs. of  1 variable:
##  $ V1: int  5 5 5 5 5 5 5 5 5 5 ...
```

Create a character vector from features.txt to use later in naming the Y_test data frame columns. Note, the features.txt file has some duplicate variable names, differing only by row/line number index. For example, "fBodyAcc-bandsEnergy()-1,8" appears at row 303, 317, and 331. There are three groups of 14 variables that start with "fBodyAcc-bandsEnergy()" and end with an ordered pair in rows 303-344. Use readLines to keep line number index with variable name.

```
features <- readLines("UCI_HAR_Dataset/features.txt")
str(features)
```

```
##  chr [1:561] "1 tBodyAcc-mean()-X" "2 tBodyAcc-mean()-Y" ...
```

```
print (features[303:316])
```

```
##  [1] "303 fBodyAcc-bandsEnergy()-1,8"   "304 fBodyAcc-bandsEnergy()-9,16"
##  [3] "305 fBodyAcc-bandsEnergy()-17,24" "306 fBodyAcc-bandsEnergy()-25,32"
##  [5] "307 fBodyAcc-bandsEnergy()-33,40" "308 fBodyAcc-bandsEnergy()-41,48"
##  [7] "309 fBodyAcc-bandsEnergy()-49,56" "310 fBodyAcc-bandsEnergy()-57,64"
##  [9] "311 fBodyAcc-bandsEnergy()-1,16"  "312 fBodyAcc-bandsEnergy()-17,32"
## [11] "313 fBodyAcc-bandsEnergy()-33,48" "314 fBodyAcc-bandsEnergy()-49,64"
## [13] "315 fBodyAcc-bandsEnergy()-1,24"  "316 fBodyAcc-bandsEnergy()-25,48"
```

For readibility, set column names before combining tables. Use conventions described in README.txt and features_info.txt files. This will make it easier later to filter "mean" and "std" variables.

```
names(subject_test) <- "subject"
names(Y_test) <- "activityNumber"
## Since features.txt contains two columns feature number and feature name
names(X_test) <- features
```

With the exception of adding column names, no data has been transformed or filtered. This is useful in case something goes wrong downstream and need to reset to original values. Each file (subject_test.txt, X_test.txt and Y_test.txt) has 2947 rows, and represent different measured variables of a single observation.

Combine test tables into a single table

Each table represents a unique subset of variables, and are linked by the order of the rows rather than a specific, shared variable. Use cbind() rather than one of the dplyr join() commands to merge the three train data files. . Bind in order of subject (subject_test), activity (Y_test), then recorded measurements (X_test). Review the result.

```
test_merged <- cbind(subject_test, Y_test, X_test)
str(test_merged, list.len = 5)
```

```
## 'data.frame':    2947 obs. of  563 variables:
##  $ subject                            : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ activityNumber                     : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ 1 tBodyAcc-mean()-X                : num  0.257 0.286 0.275 0.27 0.275 ...
##  $ 2 tBodyAcc-mean()-Y                : num  -0.0233 -0.0132 -0.0261 -0.0326 -0.0278 ...
```

```
##  $ 3 tBodyAcc-mean()-Z                     : num  -0.0147 -0.1191 -0.1182 -0.1175 -0.1295 ...
##   [list output truncated]
```

With test_merged's 2947 observations, no rows lost. 563 column variables represent subject and activityNumber columns added to 561 feature measurements.

Read train files into a list of tables: subject_train, X_train, and Y_train data

As mentioned before, the test and train files have identical structure and organization. Although larger, they can be processed in the same way: count lines, name the columns, create a data frame for each.

```
## [1] 7352
## attr(,"lastLineHasNewline")
## [1] TRUE
```

```
## [1] 7352
## attr(,"lastLineHasNewline")
## [1] TRUE
```

```
## [1] 7352
## attr(,"lastLineHasNewline")
## [1] TRUE
```

Each train data files has 7352 lines.

Using read.table, create a data frame from each file: subject_train.txt, X_train.txt and Y_train.txt.

```
subject_train <- read.table("UCI_HAR_Dataset/train/subject_train.txt", header = FALSE)
str(subject_train)
```

```
## 'data.frame':    7352 obs. of  1 variable:
##  $ V1: int  1 1 1 1 1 1 1 1 1 1 ...
```

```
X_train <- read.table("UCI_HAR_Dataset/train/X_train.txt", header = FALSE)
## This is a looooong output, just need highlights
str(X_train, list.len = 5)
```

```
## 'data.frame':    7352 obs. of  561 variables:
##  $ V1  : num  0.289 0.278 0.28 0.279 0.277 ...
##  $ V2  : num  -0.0203 -0.0164 -0.0195 -0.0262 -0.0166 ...
##  $ V3  : num  -0.133 -0.124 -0.113 -0.123 -0.115 ...
##  $ V4  : num  -0.995 -0.998 -0.995 -0.996 -0.998 ...
##  $ V5  : num  -0.983 -0.975 -0.967 -0.983 -0.981 ...
##   [list output truncated]
```

```
Y_train <- read.table("UCI_HAR_Dataset/train/Y_train.txt", header = FALSE)
str(Y_train)
```

```
## 'data.frame':    7352 obs. of  1 variable:
##  $ V1: int  5 5 5 5 5 5 5 5 5 5 ...
```

For readibility, set column names to match subject_test, X_test, Y_test. Since X_test and X_train have identical structure, use the features vector created earlier to name columns.

```
names(subject_train) <- "subject"
names(Y_train) <- "activityNumber"
## Since features,txt contains two columns feature number and feature name
names(X_train) <- features
```

With the exception of adding column names, no data has been transformed or filtered. This is useful in case something goes wrong downstream and need to reset to original values. Each file (subject_train.txt, X_train.txt and Y_train.txt) has 7352 rows, and represent different measured variables of a single observation.

Combine train tables into a single table

Like the test data, each train table represents a unique subset of variables, and are linked by the order of the rows rather than a specific, shared variable. Use cbind() rather than one of the dplyr join() commands to merge the three train data files. Bind in order of subject (subject_train), activityNumber (Y_train), then recorded measurements (X_train). Review the result.

```
train_merged <- cbind(subject_train, Y_train, X_train)
str(train_merged, list.len = 5)
```

```
## 'data.frame':    7352 obs. of  563 variables:
##  $ subject                               : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ activityNumber                        : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ 1 tBodyAcc-mean()-X                   : num  0.289 0.278 0.28 0.279 0.277 ...
##  $ 2 tBodyAcc-mean()-Y                   : num  -0.0203 -0.0164 -0.0195 -0.0262 -0.0166 ...
##  $ 3 tBodyAcc-mean()-Z                   : num  -0.133 -0.124 -0.113 -0.123 -0.115 ...
##   [list output truncated]
```

With train_merged's 7352 observations, no rows lost. 563 variables represent subject and activityNumber columns added to 561 feature measurements.

For each test_merge and train_merge table, add a new column that records the original file name.

Install and load dplyr packages. Use mutate() to create a new variable to identify the original file (test or train).

```
test_add <- mutate(test_merged, original_file = "test")
## sanity check ... expecting 564
ncol(test_add)
```

```
## [1] 564
```

```
tail(test_add$original_file)
```

```
## [1] "test" "test" "test" "test" "test" "test"
```

```
train_add <- mutate(train_merged, original_file = "train")
## sanity check ... expecting 564
ncol(test_add)
```

```
## [1] 564
```

```
tail(train_add$original_file)
```

```
## [1] "train" "train" "train" "train" "train" "train"
```

Using ncol confirms that each new table now has 564 columns. Explicitly specifying new column by name in tail operation, demonstrates original_name was added to test_merged and train_merged and was filled.

Combine test_add and train_add tables into a single table

Since the test_add and train_add tables have the same column organization, use rbind to combine into a single table.

```
test_train <- rbind(test_add, train_add)
## 2947 from test + 7352 from train = 10299 in test_train
## matches the 10299 instances recorded in raw data
nrow(test_train)
```

```
## [1] 10299
```

Row binding test_add and train_add tables creates a single data frame with 10299 rows: 2947 from test plus 7352 from train. Total rows bound matches the 10299 instances recorded in raw data source's webpage (http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones)

Tidy data

The test_train data set is tidy, if not pretty.

Each variable forms a column.

The final test_train data frame includes 564 variables for subject, activity, origin file plus 561 measured features. While some of the features represent computed values such as mean, they are a single observed variable.

It might be tempting to split the columns by parsing some of the variable names. For example, the first 40 rows could be reshaped based on feature (tBodyAcc), calculation (mean(), std(), min(), max(), . . .) and axis (X, Y, Z). With 561 measurements split into columns based on feature, calculationa and axis, the data set would become very wide. Furthermore, not every variable includes an axis designation. In David Hood's blog, he writes, "[W]ithin the broad set theory background it becomes what is the best tidy form of data to answer a specific question." The next part of the assignnment is to extract only the mean() and std() observations. The rows can be extracted using simple grep commnads that do not reshaping or risk changing the data.

For this application the variables are tidy.

Each observation forms a row.

For this data set, an observation is of a given subject, performing a specific activity, described by 561 measured features, and labeled by origin (test or train).

Each type of observational unit forms a table.

The collection of variables, although from different samples, represent the same type of observations (subject, activity, measured features) and includes a variable that identifies the original source of the observation.

Tidy Data, plus

Based on Wickham's standards, the data frame test_train is tidy. Although not explicitly mentioned in the definition of tidy data, Wickham also discusses the order in which variables should appear in a data set:

> "Fixed variables describe the experimental design and are known in advance... Measured variables are what we actually measure in the study. Fixed variables should come first, followed by measured variables, each ordered so that related variables are contiguous."

By this standard, subject, acivity and original file would be fixed variables and should appear contiguously and first. Measured variables would be the items described by the feature measurements. This could be accomplished by moving the original_file column to the first column of the data frame.

```
## new dataframe with fixed variable first
tidyframe <- select(test_train, 564, 1:563)
```

Tidy data addresses the shape and organization of data that make it software readible rather than what makes it human readible: tidyframe is tidy data. Descriptive variable names and possible reshaping will be addressed later in the assignment (see sections 3 & 4).

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

Based on features_info.txt, observed features representing mean or standard deviation include the string "mean" or "std" in their name: mean, meanFreq and std. Features that included "Mean" as part of an ordered pair were excluded since they are inputs to the angle measurements and not discrete entities.

The tidyframe data includes variable names, use grep to extract a list of columns containing the target strings.

```
## use grep to find target strings
mean_std_columns <- grep("mean()|std()", names(tidyframe), value = TRUE)
mean_std_tidyframe <- select(tidyframe, original_file, activityNumber, subject, mean_std_columns)
str(mean_std_tidyframe,list.len = 5)
```

```
## 'data.frame':    10299 obs. of  82 variables:
##  $ original_file                     : chr  "test" "test" "test" "test" ...
##  $ activityNumber                    : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ subject                           : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ 1 tBodyAcc-mean()-X               : num  0.257 0.286 0.275 0.27 0.275 ...
##  $ 2 tBodyAcc-mean()-Y               : num  -0.0233 -0.0132 -0.0261 -0.0326 -0.0278 ...
##   [list output truncated]
```

There were 79 total occurrences of the strings "mean" and "std" in the original features.txt file. Including the three fixed variables (original_file, activityNumber, subject) the final data frame correctly includes 82 variables.

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

Six activity numbers and names are listed in activity_labels.txt. Read the activity_labels.txt file into a table to use with join to replace activity numbers with names in tidyframe.

```
activity_labels <- read.table("UCI_HAR_Dataset/activity_labels.txt", header = FALSE)
str(activity_labels)
```

```
## 'data.frame':    6 obs. of  2 variables:
##  $ V1: int  1 2 3 4 5 6
##  $ V2: Factor w/ 6 levels "LAYING","SITTING",..: 4 6 5 2 3 1
```

Check for missing values

Before doing any join based on a key, like activity, double check to make sure there will be a mapping for each. An outer join will substitute an NA for missed joins, but it would be better to know ahead of time about missing data.

```
uactivity <- unique(tidyframe$activityNumber)
str(uactivity)
```

```
## int [1:6] 5 4 6 1 3 2
```

```
uactivity <- unique(mean_std_tidyframe$activityNumber)
str(uactivity)
```

```
## int [1:6] 5 4 6 1 3 2
```

Add new descriptive variable activityName

From the above, there are 6 unique activities in tidyframe and mean_std_tidyframe, with values ranging from 1-6 and no NA. Add activity name to tidyframe via inner join, which is more descriptive that a number.

```
## An activity id/number and name is listed in activity_labels.txt for each unique activity
activity_labels <- read.table("UCI_HAR_Dataset/activity_labels.txt", header = FALSE)
## name the columns
names(activity_labels) <- c("activityNumber","activityName")

## apply names to mean_std_tidyframe using join
activity_tidyframe <- inner_join(mean_std_tidyframe, activity_labels, by = "activityNumber")
## move column to be contiguous with fixed variables
activity_tidyframe <- select (activity_tidyframe, original_file, activityName, subject, 4:82)

str(activity_tidyframe, list.len = 5)
```

```
## 'data.frame':    10299 obs. of  82 variables:
##  $ original_file                      : chr  "test" "test" "test" "test" ...
##  $ activityName                       : Factor w/ 6 levels "LAYING","SITTING",..: 3 3 3 3 3 3 3 3 3 3
##  $ subject                            : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ 1 tBodyAcc-mean()-X                : num  0.257 0.286 0.275 0.27 0.275 ...
##  $ 2 tBodyAcc-mean()-Y                : num  -0.0233 -0.0132 -0.0261 -0.0326 -0.0278 ...
##   [list output truncated]
```

The new data frame activity_tidyframe replaced activityNumber with activityName, to describe the activities in the data set.

4. Appropriately label the data set with descriptive variable names.

The column names for the subject, activity and measured variables were assigned earlier to the subject_, Y_, and X_ tables just before the three were merged to create the test_merged and train_merged data frames.

Naming the columns before the cbind() merges, allowed for better visual inspection of new data frames. Used str() to review the merge and make sure that columns were bound in the correct order. With subject and activityNumber both being numeric, it might be more challenging to scroll through the new data frames to make sure the columns had been bound in the correct order.

How descriptive?

The question remains, could the variable names be more descriptive? It depends on the consumer. To one who extracts and studies this type of data, the names are very descriptive. There is always a trade off between how descriptive to make a variable name, and how practical it would be to use a very long name for data exploration and study.

Assuming the consumer of the final data set is knowledgeable about the field, the variable names listed in features.txt would be satifactory. However, they might appreciate some standardization and fine tuning.

Possible to standardize variable names from features.txt?

The current version of activity_tidyframe has some quirks within the variable names:

- dashes and parentheses in the variable names
- leading digits leftover from earlier name assignment

Standardized format for variable names

There are several approaches to creating variable names. The UCI HAR Data Set authors chose the lowerCamelCase style for data frame variables and underscore_separated for text file names. Data frames and vectors used in run_analysis.R use underscore_separated.

Because special characters like dashes and parentheses can create problems, use gsub to replace or remove them.

```
names_dash_to_under <- gsub("-", "_", names(activity_tidyframe))
names_no_paren <- gsub("\\(\\)", "", names_dash_to_under)
```

Remove leading digits from measured variables

Earlier, it was discovered there are three groups of 14 identical variables that start with "fBodyAcc-bandsEnergy()" and end with an ordered pair in lines 303 through 344 of features.txt. The only way to make these variable names unique was to include the row number. Check to see if any of these variables are included in activity_tidyframe.

```
grep("fBodyAcc-bandsEnergy()", names(activity_tidyframe), value = TRUE)
```

```
## character(0)
```

None of these variables are included with the mean() or std() calculated measurements extracted to create activity_tidyframe. The leading digits can be removed. Use gsub to remove leading digits and space leftover from Part 1. Update activity_tidyframe variable names.

```
names_no_digits <- gsub("^[0-9]{1,3} ", "", names_no_paren)
names(activity_tidyframe) <- names_no_digits
```

The measured variable names are now standardized, descriptive text for a consumer familiar with the types of calculations used.

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

The data set submitted will be wide form, with each column representing a fixed or measured variable.

Not So Tidy - Do measured features represent more than one value?

Looking at the data frame extracted in Part 4, the mean() and std() variables contain three segments: feature, calculation and reference. For tBodyAcc_mean_X, tBodyAcc would be the feature variable, mean would be the calculation and X would be the reference. Summarized:

- features as measured variable (11): fBodyAcc, fBodyAccJerk, fBodyBodyAccJerk, fBodyGyro, fBodyBodyGyro, fBodyBodyGyroJerk, tBodyAcc, tBodyAccJerk, tBodyGyro, tBodyGyroJerk, tGravityAcc
- Calculation (3): mean, meanFreq, std
- Reference (4): X-axis, Y-axis, Z-Axis Mag

The data frame could be reshaped to allow feature variable grouping by subject and activity, as well as calculation and/or reference. Grouping by all four would yield summary data equivalent to the final_tidyframe above.

However, not all possible combination of feature, calculation and reference are included in the original data. Introducing NA creates the appearance the original researchers missed key calculations when perhaps that permutation of feature, calculation and frame was impractical or irrelevant.

That would not be tidy.

Tidy Data - Measured features represent a single value

The most direct approach for creating a second data set with the average of each activity and each subject would be to compute the column mean for each of the measured variables as listed in features.txt, grouped by subject and activity.

```
final_tidyframe <- activity_tidyframe %>%
    select (2:82) %>%
    group_by(subject, activityName) %>%
    summarise_all(mean)
```

With thirty subjects and six activities, final_tidyframe should have 180 rows. The original_file column was not required, thus not included in final_tidyframe, resulting in 81 columns.

This approach is tidy and appropriate if the data set consumer considers each of the 79 measured variables a single unit of data. Grouping this way by subject and then activity creates a data set that could be used as is, or to further summarize data by additional combinations of subject or activity.

Summary: A tidy data set

To meet the rubric, the data set may be narrow or wide, provided it is tidy. A tidy data set must meet all three of Wickham's conditions. The final_tidyframe data set meets these conditions.

```
1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.
```

Wickham and Hood also discussed that "tidy-ness" is a function of how well the data set fits the task. The task was, from the the data set in step 4, to create a second independent tidy data set with the average of each variable for each activity and each subject. The data set final_tidyframe presents the average of each variable, in a tudy, rectangular format that could be further summarised by subject or activity.

Resources

In addition to lecture notes, several resources were very helpful in strategizing and working with the data sets in the assignment.

Tidy Data

Henry, Lionel, and Hadley Wickham. "Tidy Evaluation." Tidy Evaluation, tidyeval.tidyverse.org/index.html.

Thoughtfulbloke. "Getting and Cleaning the Assignment." Thoughtfulbloke Aka David Hood, 26 Jan. 2016, thoughtfulbloke.wordpress.com/2015/09/09/getting-and-cleaning-the-assignment/.

Wickham, Hadley, and Garrett Grolemund. R For Data Science: Import, Tidy, Transform, Visualize and Model Data. O'Reilly, 2017.

Wickham, Hadley. Tidy Data. Journal of Statistical Software, 2014, vita.had.co.nz/papers/tidy-data.pdf.

General

R Markdown Cheatsheet at https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf

Afterword: Tidy Data, Tidy Pantry

As with creating descriptive varaibles, there is a trade off between using data as originally recorded and data as segmented. R offers powerful tools for reshaping data, but just because one can reshape the data, does not mean the reshaped version is meaningful. Reshaping data sets must be accompanied by a solid knowledge of the derivation of the data and its applications.

Consider an example, an observer reviews my pantry and records:

- shelf 1: square based, plastic bins, medium
- shelf 2: square based, plastic bins, large, medium, small
- shelf 3: glass jars and plastic bags, mixed inside square baskets
- shelf 4: canned goods, plastic bags
- floor: canned goods

It might be tempting to re-organize my pantry storage by storage type: bags, bins, baskets, cans, jars. It would be very tidy, but a total disaster for meal prep because it is not the container that is critical, but the contents and application.

A more valid recording of my pantry:

- shelf 1: sweeteners, chocolate, garnishes for desserts

- shelf 2: flours, rising agents for baking
- shelf 3: nuts, seeds, cereals and dried fruits for making granola & trail mix
- shelf 4: beans, pasta, grains for side dishes

- floor: dog food

I definitely would not want the dog food next to the refried beans, or the pumpkin seeds with the mung beans!

Returning to the submitted data set, segmenting the feature variables without a solid knowledge of how the feature variable were computed and used, may not add to the value of the data set.