---
title: "Chapter 5: Data Summarization, Manipulation with dplyr "
author: "M Affouf"
date: "1/8/2018"
output:output: html_document

#part1

Data Summarization

* Basic statistical summarization
    * `mean(x)`: takes the mean of x
    * `sd(x)`: takes the standard deviation of x
    * `median(x)`: takes the median of x
    * `quantile(x)`: displays sample quantities of x. Default is min, IQR, max
    * `range(x)`: displays the range. Same as c(min(x), max(x))

Some examples

We can use the `mtcars` and Charm City Circulator datasets to explore different ways of summarizing data.

```{r}
head(mtcars)
```


Statistical summarization

```{r}
mean(mtcars$hp)
quantile(mtcars$hp)
```

Statistical summarization

```{r}
median(mtcars$wt)
quantile(mtcars$wt, probs = 0.6)
```

Statistical summarization

Note that many of these functions have additional inputs regarding missing data, typically requiring the `na.rm` argument.

```{r}
x = c(1,5,7,NA,4,2, 8,10,45,42)
mean(x)
mean(x,na.rm=TRUE)
quantile(x,na.rm=TRUE)
```

Data Summarization on matrices/data frames

* Basic statistical summarization
    * `rowMeans(x)`: takes the means of each row of x
    * `colMeans(x)`: takes the means of each column of x
    * `rowSums(x)`: takes the sum of each row of x
    * `colSums(x)`: takes the sum of each column of x
    * `summary(x)`: for data frames, displays the quantile information

Charm City Circulator data

Please download the Charm City Circulator data:


```{r}
circ = read.csv("Charm_City_Circulator_Ridership.csv",
            header=TRUE,as.is=TRUE)
```

Subsetting to specific columns

Let's just take columns that represent average ridership:

```{r}
library(dplyr,quietly = TRUE)
circ2 = select(circ, date, day, ends_with("Average"))
```

column and row means

```{r colMeans}
avgs = select(circ2, ends_with("Average"))
colMeans(avgs,na.rm=TRUE)
circ2$daily = rowMeans(avgs,na.rm=TRUE)
head(circ2$daily)
```

Summary

```{r summary1}
summary(circ2)
```

Apply statements

You can apply more general functions to the rows or columns of a matrix or data frame, beyond the mean and sum.

```
apply(X, MARGIN, FUN, ...)
```

> X : an array, including a matrix.
>
> MARGIN : a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.
>
> FUN   : the function to be applied: see 'Details'.
>
> ...   : optional arguments to FUN.

Apply statements

```{r apply1}
apply(avgs,2,mean,na.rm=TRUE) # column means
apply(avgs,2,sd,na.rm=TRUE) # columns sds
apply(avgs,2,max,na.rm=TRUE) # column maxs
```

Other Apply Statements

* `tapply()`: 'table' apply
* `lapply()`: 'list' apply
* `sapply()`: 'simple' apply
* Other less used ones...

See more details here: http://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/

`tapply()`

From the help file: "Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors."

```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

Simply put, you can apply function `FUN` to `X` within each categorical level of `INDEX`. It is very useful for assessing properties of continuous data by levels of categorical data.

`tapply()`

For example, we can estimate the highest average daily ridership for each day of the week in 1 line in the Circulator dataset.

```{r tapply1}
tapply(circ2$daily, circ2$day, max, na.rm=TRUE)
```

Data Summarization

* Basic summarization plots
    * `plot(x,y)`: scatterplot of x and y
    * `boxplot(y~x)`: boxplot of y against levels of x
    * `hist(x)`: histogram of x
    * `density(X)`: kernel density plot of x

Basic Plots

Plotting is an important component of exploratory data analysis. We will review some of the more useful and informative plots here. We will go over formatting and making plots look nicer in additional lectures.

Scatterplot

```{r scatter1}
plot(mtcars$mpg, mtcars$disp)
```

Histograms

```{r hist1}
hist(circ2$daily)
```

Density

```{r dens1,fig.width=5,fig.height=5}
## plot(density(circ2$daily))
plot(density(circ2$daily,na.rm=TRUE))
```

Boxplots

```{r box1}
boxplot(circ2$daily ~ circ2$day)
```

Boxplots

```{r box2}
boxplot(daily ~ day, data=circ2)
```

Data Summarization for data.frames

* Basic summarization plots
    * `matplot(x,y)`: scatterplot of two matrices, x and y
    * `pairs(x,y)`: plots pairwise scatter plots of matrices x and y, column by column

Matrix plot

```{r matplot1}
matplot(avgs)
```

#Part 2

##Agenda

- Summaries with the aggregate() function
- Standard graphics

##Getting started: birthwt data set
- We're going to start by operating on the `birthwt` dataset from the MASS library

- Let's get it loaded and see what we're working with

```{r}
library(MASS)
str(birthwt)
```

##Renaming the variables
- The dataset doesn't come with very descriptive variable names

- Let's get better column names (use `help(birthwt)` to understand the variables and come up with better names)

```{r}
colnames(birthwt)

# The default names are not very descriptive

colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
    "physician.visits", "birthwt.grams")

# Better names!
```

##Renaming the factors
- All the factors are currently represented as integers

- Let's use the `transform()` and `mapvalues()` functions to convert variables to factors and give the factors more meaningful levels
```{r}
library(plyr)
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                        c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                        c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                        c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                        c(0,1), c("no", "yes"))),
            birthwt.below.2500 = as.factor(mapvalues(birthwt.below.2500,
                        c(0,1), c("no", "yes")))
            )
```

##Summary of the data
- Now that things are coded correctly, we can look at an overall summary

```{r}
summary(birthwt)
```

##A simple table
- Let's use the `tapply()` function to see what the average birthweight looks like when broken down by race and smoking status

```{r}
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN = mean))
```

##What if we wanted nicer looking output?
- Let's use the header `{r, results='asis'}`, along with the `kable()` function from the `knitr` library


```{r, results='asis'}
library(knitr)
bwt.tbl <- with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN = mean))
kable(bwt.tbl, format = "markdown")
```


- `kable()` outputs the table in a way that Markdown can read and nicely display

- Note: changing the CSS changes the table appearance

##aggregate() function
- Let's first recall what `tapply()` does

- Command: `tapply(X, INDEX, FUN)`
    - Applies `FUN` to `X` grouped by factors in `INDEX`

- **`aggregate()`** performs a similar operation, but presents the results in a form that is at times more convenient

- There are many ways to call the `aggregate()` function

- Analog of `tapply` call: `aggregate(X, by, FUN)`
    - Here, `by` is exactly like `INDEX`

##Example: tapply vs aggregate
```{r}
library(MASS)
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN = mean)) # tapply
with(birthwt, aggregate(birthwt.grams, by = list(race, mother.smokes), FUN = mean)) # aggregate
```

##Example: different syntax
- Here's a convenient alternative way to call `aggregate`

- It uses the R `formula` syntax, which we'll learn more about when we discuss regression

```{r}
aggregate(birthwt.grams ~ race + mother.smokes, FUN=mean, data=birthwt)
```

- We'll see later that `aggregate` output can be more convenient for plotting

##A closer look at low birth weight

```{r}
weight.smoke.tbl <- with(birthwt, table(birthwt.below.2500, mother.smokes))
weight.smoke.tbl
```

- The odds of low bwt among non-smoking mothers is

```{r}
or.smoke.bwt <- (weight.smoke.tbl[2,2] / weight.smoke.tbl[1,2]) / (weight.smoke.tbl[2,1] /
weight.smoke.tbl[1,1])
or.smoke.bwt
```

- So the odds of low birth weight are `r round(or.smoke.bwt, 1)` times higher when the mother smokes

##continuted...

- Is the mother's age correlated with birth weight?

```{r}
with(birthwt, cor(birthwt.grams, mother.age))  # Calculate correlation
```

- Does this change when we account for smoking status?

```{r}
with(birthwt, cor(birthwt.grams[mother.smokes == "yes"], mother.age[mother.smokes == "yes"]))
```

```{r}
with(birthwt, cor(birthwt.grams[mother.smokes == "no"], mother.age[mother.smokes == "no"]))
```

##Faster way: by() function
- Think of the `by(data, INDICES, FUN)` function as a `tapply()` function that operates on data frames instead of just vectors

- When using `tapply(X, INDEX, FUN)`, `X` is generally a numeric vector

- To calculate correlations, we need to allow `X` to be a data frame or matrix

```{r}
by(data = birthwt[c("birthwt.grams", "mother.age")],
   INDICES = birthwt["mother.smokes"],
   FUN = function(x) {cor(x[,1], x[,2])})
```

##Standard graphics in R

Single-variable plots

Let's continue with the `birthwt` data from the `MASS` library.

Here are some basic single-variable plots.

```{r, fig.height = 7, fig.align='center'}
par(mfrow = c(2,2)) # Display plots in a single 2 x 2 figure
plot(birthwt$mother.age)
with(birthwt, hist(mother.age))
plot(birthwt$mother.smokes)
plot(birthwt$birthwt.grams)
```

Note that the result of calling `plot(x, ...)` varies depending on what `x` is.
    - When `x` is *numeric*, you get a plot showing the value of `x` at every index.
    - When `x` is a *factor*, you get a bar plot of counts for every level

Let's add more information to the smoking bar plot, and also change the color by setting the `col` option.

```{r, fig.height=5, fig.width=5, fig.align='center'}
par(mfrow = c(1,1))
plot(birthwt$mother.smokes,
     main = "Mothers Who Smoked In Pregnancy",
     xlab = "Smoking during pregnancy",
     ylab = "Count of Mothers",
     col = "lightgrey")
```

(much) better graphics with ggplot2

Introduction to ggplot2

ggplot2 has a slightly steeper learning curve than the base graphics functions, but it also generally produces far better and more easily customizable graphics.

There are two basic calls in ggplot:

- `qplot(x, y, ..., data)`: a "quick-plot" routine, which essentially replaces the base `plot()`
- `ggplot(data, aes(x, y, ...), ...)`: defines a graphics object from which plots can be generated, along with *aesthetic mappings* that specify how variables are mapped to visual properties.

```{r}
library(ggplot2)
```

plot vs qplot

Here's how the default scatterplots look in ggplot compared to the base graphics. We'll illustrate things by continuing to use the birthwt data from the `MASS` library.

```{r, fig.align='center', fig.height=3, fig.width=4}
with(birthwt, plot(mother.age, birthwt.grams))  # Base graphics
qplot(x=mother.age, y=birthwt.grams, data=birthwt)  # using qplot from ggplot2
```

Remember how it took us some effort last time to add color coding, use different plotting characters, and add a legend? Here's the `qplot` call that does it all in one simple line.

```{r, fig.align='center', fig.height=4, fig.width=5}
qplot(x=mother.age, y=birthwt.grams, data=birthwt,
      color = mother.smokes,
      shape = mother.smokes,
```

```
      xlab = "Mother's age (years)",
      ylab = "Baby's birthweight (grams)"
      )
```

This way you won't run into problems of accidentally producing the wrong legend. The legend is produced based on the `colour` and `shape` argument that you pass in. (Note: `color` and `colour` have the same effect.)

ggplot function

The `ggplot2` library comes with a dataset called `diamonds`. Let's look at it

```{r}
dim(diamonds)
head(diamonds)
```

It is a data frame of 53,940 diamonds, recording their attributes such as carat, cut, color, clarity, and price.

We will make a scatterplot showing the price as a function of the carat (size). (The data set is large so the plot may take a few moments to generate.)

```{r fig.width=10, fig.height=4, dpi=70, cache=TRUE}
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price))
diamond.plot + geom_point()
```

The data set looks a little weird because a lot of diamonds are concentrated on the 1, 1.5 and 2 carat mark.

Let's take a step back and try to understand the ggplot syntax.

1) The first thing we did was to define a graphics object, `diamond.plot`. This definition told R that we're using the `diamonds` data, and that we want to display `carat` on the x-axis, and `price` on the y-axis.

2) We then called `diamond.plot + geom_point()` to get a scatterplot.

The arguments passed to `aes()` are called **mappings**. Mappings specify what variables are used for what purpose. When you use `geom_point()` in the second line, it pulls `x`, `y`, `colour`, `size`, etc., from the **mappings** specified in the `ggplot()` command.

You can also specify some arguments to `geom_point` directly if you want to specify them for each plot separately instead of pre-specifying a default.

Here we shrink the points to a smaller size, and use the `alpha` argument to make the points transparent.

```{r fig.width=10, fig.height=4, dpi=70, cache=TRUE}
diamond.plot + geom_point(size = 0.7, alpha = 0.3)
```

If we wanted to let point color depend on the color indicator of the diamond, we could do so in the following way.

```{r fig.width=10, fig.height=6, dpi=70, cache=TRUE}
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point()
```

If we didn't know anything about diamonds going in, this plot would indicate to us that **D** is likely the highest diamond grade, while **J** is the lowest grade.

We can change colors by specifying a different color palette. Here's how we can switch to the `cbPalette` we saw last class.

```{r fig.width=10, fig.height=6, dpi=70, cache=TRUE}
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
"#CC79A7")
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))
diamond.plot + geom_point() + scale_colour_manual(values=cbPalette)
```

To make the scatterplot look more typical, we can switch to logarithmic coordinate axis spacing.

```{r, eval = FALSE}
diamond.plot + geom_point() +
  coord_trans(x = "log10", y = "log10")
```

Conditional plots

We can create plots showing the relationship between variables across different values of a factor. For instance, here's a scatterplot showing how diamond price varies with carat size, conditioned on color. It's created using the `facet_wrap(~ factor1 + factor2 + ... + factorn)` command.

```{r, fig.width=12, fig.height=6, dpi=70, cache=TRUE}
diamond.plot <- ggplot(data=diamonds, aes(x=carat, y=price, colour = color))

diamond.plot + geom_point() + facet_wrap(~ cut)
```

You can also use `facet_grid()` to produce this type of output.

```{r, fig.width=11, fig.height=4.5, dpi=70, cache=TRUE}
diamond.plot + geom_point() + facet_grid(. ~ cut)
```

```{r, fig.width = 8, fig.height = 10, dpi = 70, cache = TRUE}
diamond.plot + geom_point() + facet_grid(cut ~ .)
```

`ggplot` can create a lot of different kinds of plots, just like lattice. Here are some examples.

Function	Description
`geom_point(...)`	Points, i.e., scatterplot
`geom_bar(...)`	Bar chart
`geom_line(...)`	Line chart
`geom_boxplot(...)`	Boxplot
`geom_violin(...)`	Violin plot
`geom_density(...)`	Density plot with one variable
`geom_density2d(...)`	Density plot with two variables
`geom_histogram(...)`	Histogram

A bar chart

```{r}
qplot(x = race, data = birthwt, geom = "bar")
```

Histograms and density plots

```{r}
base.plot <- ggplot(birthwt, aes(x = mother.age)) +
  xlab("Mother's age")
base.plot + geom_histogram()
base.plot + geom_histogram(aes(fill = race))
base.plot + geom_density()
base.plot + geom_density(aes(fill = race), alpha = 0.5)
```

Box plots and violin plots

```{r}
base.plot <- ggplot(birthwt, aes(x = as.factor(physician.visits), y = birthwt.grams)) +
  xlab("Number of first trimester physician visits") +
  ylab("Baby's birthweight (grams)")

# Box plot
base.plot + geom_boxplot()

# Violin plot
base.plot + geom_violin()
```

Visualizing means

Previously we calculated the following table:

```{r}
bwt.summary <- aggregate(birthwt.grams ~ race + mother.smokes, data = birthwt, FUN = mean) #
aggregate
bwt.summary
```

We can plot this table in a nice bar chart as follows:

```{r}
# Define basic aesthetic parameters
p.bwt <- ggplot(data = bwt.summary, aes(y = birthwt.grams, x = race, fill = mother.smokes))

# Pick colors for the bars
bwt.colors <- c("#009E73", "#999999")

# Display barchart
p.bwt + geom_bar(stat = "identity", position = "dodge") +
  ylab("Average birthweight") +
  xlab("Mother's race") +
  guides(fill = guide_legend(title = "Mother's smoking status")) +
  scale_fill_manual(values=bwt.colors)
```

Does the association between birthweight and mother's age depend on smoking status?

We previously ran the following command to calculate the correlation between mother's ages and baby birthweights.

```{r}
by(data = birthwt[c("birthwt.grams", "mother.age")],
   INDICES = birthwt["mother.smokes"],
   FUN = function(x) {cor(x[,1], x[,2])})
```

Here's a visualization of our data that allows us to see what's going on.

```{r, fig.height=5, fig.width=6, fig.align='center'}
ggplot(birthwt, aes(x=mother.age, y=birthwt.grams, shape=mother.smokes, color=mother.smokes)) +
  geom_point() + # Adds points (scatterplot)
  geom_smooth(method = "lm") + # Adds regression lines
  ylab("Birth Weight (grams)") + # Changes y-axis label
  xlab("Mother's Age (years)") + # Changes x-axis label
  ggtitle("Birth Weight by Mother's Age") # Changes plot title
```

```{r}
 # library(knitr)
 # purl("chap5s_m.Rmd", documentation = 0)
```