Chapter 1

Data Preparation

Before you can visualize your data, you have to get it into R. This involves importing the data from an external source and massaging it into a useful format.

1.1 Importing data

R can import data from almost any source, including text files, excel spreadsheets, statistical packages, and database management systems. We'll illustrate these techniques using the Salaries dataset, containing the 9 month academic salaries of college professors at a single institution in 2008-2009.

1.1.1 Text files

The readr package provides functions for importing delimited text files into R data frames.

```
library(readr)

# import data from a comma delimited file
Salaries <- read_csv("salaries.csv")

# import data from a tab delimited file
Salaries <- read_tsv("salaries.txt")
```

These function assume that the first line of data contains the variable names, values are separated by commas or tabs respectively, and that missing data are represented by blanks. For example, the first few lines of the comma delimited file looks like this.

```
"rank","discipline","yrs.since.phd","yrs.service","sex","salary"
"Prof","B",19,18,"Male",139750
"Prof","B",20,16,"Male",173200
"AsstProf","B",4,3,"Male",79750
"Prof","B",45,39,"Male",115000
"Prof","B",40,41,"Male",141500
"AssocProf","B",6,6,"Male",97000
```

Options allow you to alter these assumptions. See the documentation for more details.

1.1.2 Excel spreadsheets

The readxl package can import data from Excel workbooks. Both xls and xlsx formats are supported.

```
library(readxl)

# import data from an Excel workbook
Salaries <- read_excel("salaries.xlsx", sheet=1)
```

Since workbooks can have more than one worksheet, you can specify the one you want with the sheet option. The default is sheet=1.

1.1.3 Statistical packages

The haven package provides functions for importing data from a variety of statistical packages.

```
library(haven)

# import data from Stata
Salaries <- read_dta("salaries.dta")

# import data from SPSS
Salaries <- read_sav("salaries.sav")

# import data from SAS
Salaries <- read_sas("salaries.sas7bdat")
```

1.1.4 Databases

Importing data from a database requires additional steps and is beyond the scope of this book. Depending on the database containing the data, the following packages can help: RODBC, RMySQL, ROracle, RPostgreSQL, RSQLite, and RMongo. In the newest versions of RStudio, you can use the Connections pane to quickly access the data stored in database management systems.

1.2 Cleaning data

The processes of cleaning your data can be the most time-consuming part of any data analysis. The most important steps are considered below. While there are many approaches, those using the dplyr and tidyr packages are some of the quickest and easiest to learn.

Package	Function	Use
dplyr	select	select variables/columns
dplyr	filter	select observations/rows
dplyr	mutate	transform or recode variables
dplyr	summarize	summarize data
dplyr	group_by	identify subgroups for further processing
tidyr	gather	convert wide format dataset to long format
tidyr	spread	convert long format dataset to wide format

Examples in this section will use the starwars dataset from the **dplyr** package. The dataset provides descriptions of 87 characters from the Starwars universe on 13 variables. (I actually prefer StarTrek, but we work with what we have.)

1.2.1 Selecting variables

The `select` function allows you to limit your dataset to specified variables (columns).

```
library(dplyr)

# keep the variables name, height, and gender
newdata <- select(starwars, name, height, gender)

# keep the variables name and all variables
# between mass and species inclusive
newdata <- select(starwars, name, mass:species)

# keep all variables except birth_year and gender
newdata <- select(starwars, -birth_year, -gender)
```

1.2.2 Selecting observations

The `filter` function allows you to limit your dataset to observations (rows) meeting a specific criteria. Multiple criteria can be combined with the `&` (AND) and `|` (OR) symbols.

```
library(dplyr)

# select females
newdata <- filter(starwars,
                  gender == "female")

# select females that are from Alderaan
newdata <- select(starwars,
                  gender == "female" &
                  homeworld == "Alderaan")


# select individuals that are from
# Alderaan, Coruscant, or Endor
newdata <- select(starwars,
                  homeworld == "Alderaan" |
                  homeworld == "Coruscant" |
                  homeworld == "Endor")

# this can be written more succinctly as
newdata <- select(starwars,
                  homeworld %in% c("Alderaan", "Coruscant", "Endor"))
```

USE FILTER NOT SELECT

1.2.3 Creating/Recoding variables

The `mutate` function allows you to create new variables or transform existing ones.

```
library(dplyr)

# convert height in centimeters to inches,
# and mass in kilograms to pounds
newdata <- mutate(starwars,
                  height = height * 0.394,
                  mass   = mass   * 2.205)
```

The ifelse function (part of base R) can be used for recoding data. The format is `ifelse(test, return if TRUE, return if FALSE)`.

```
library(dplyr)

# if height is greater than 180
# then heightcat = "tall",
# otherwise heightcat = "short"

newdata <- mutate(starwars,
                  heightcat = ifelse(height > 180,
                                     "tall",
                                     "short"))

# convert any eye color that is not
# black, blue or brown, to other
newdata <- mutate(starwars,
                  eye_color = ifelse(eye_color %in% c("black", "blue", "brown"),
                                     eye_color,
                                     "other"))

# set heights greater than 200 or
# less than 75 to missing
newdata <- mutate(starwars,
                  height = ifelse(height < 75 | height > 200,
                                     NA,
                                     height))
```

1.2.4 Summarizing data

The summarize function can be used to reduce multiple values down to a single value (such as a mean). It is often used in conjunction with the by_group function, to calculate statistics by group. In the code below, the na.rm=TRUE option is used to drop missing values before calculating the means.

```
library(dplyr)

# calculate mean height and mass
newdata <- summarize(starwars,
                     mean_ht   = mean(height, na.rm=TRUE),
                     mean_mass = mean(mass, na.rm=TRUE))
newdata
```

```
## # A tibble: 1 x 2
##   mean_ht mean_mass
##     <dbl>     <dbl>
## 1    174.      97.3
```

```
# calculate mean height and weight by gender
newdata <- group_by(starwars, gender)
newdata <- summarize(newdata,
                     mean_ht = mean(height, na.rm=TRUE),
                     mean_wt = mean(mass, na.rm=TRUE))
newdata
```

```
## # A tibble: 5 x 3
##   gender        mean_ht mean_wt
##   <chr>           <dbl>   <dbl>
## 1 female           165.    54.0
## 2 hermaphrodite    175.  1358.
## 3 male             179.    81.0
## 4 none             200.   140.
## 5 <NA>             120.    46.3
```

1.2.5 Using pipes

Packages like dplyr and tidyr allow you to write your code in a compact format using the pipe %>% operator. Here is an example.

```
library(dplyr)

# calculate the mean height for women by species
newdata <- filter(starwars,
                  gender == "female")
newdata <- group_by(species)
newdata <- summarize(newdata,
                     mean_ht = mean(height, na.rm = TRUE))

# this can be written as
newdata <- starwars %>%
  filter(gender == "female") %>%
  group_by(species) %>%
  summarize(mean_ht = mean(height, na.rm = TRUE))
```

The %>% operator passes the result on the left to the first parameter of the function on the right.

1.2.6 Reshaping data

Some graphs require the data to be in wide format, while some graphs require the data to be in long format. You can convert a wide dataset to a long dataset using

```
library(tidyr)
long_data <- gather(wide_data,
                    key="variable",
                    value="value",
                    sex:income)
```

Table 1.2: Wide data

id	name	sex	age	income
01	Bill	Male	22	55000
02	Bob	Male	25	75000
03	Mary	Female	18	90000

Table 1.3: Long data

id	name	variable	value
01	Bill	sex	Male
02	Bob	sex	Male
03	Mary	sex	Female
01	Bill	age	22
02	Bob	age	25
03	Mary	age	18
01	Bill	income	55000
02	Bob	income	75000
03	Mary	income	90000

Conversely, you can convert a long dataset to a wide dataset using

```
library(tidyr)
wide_data <- spread(long_data, variable, value)
```

1.2.7 Missing data

Real data are likely to contain missing values. There are three basic approaches to dealing with missing data: feature selection, listwise deletion, and imputation. Let's see how each applies to the msleep dataset from the `ggplot2` package. The `msleep` dataset describes the sleep habits of mammals and contains missing values on several variables.

1.2.7.1 Feature selection

In feature selection, you delete variables (columns) that contain too many missing values.

```
data(msleep, package="ggplot2")

# what is the proportion of missing data for each variable?
pctmiss <- colSums(is.na(msleep))/nrow(msleep)
round(pctmiss, 2)
```

```
##          name        genus         vore        order conservation
##          0.00         0.00         0.08         0.00         0.35
##   sleep_total    sleep_rem  sleep_cycle        awake      brainwt
##          0.00         0.27         0.61         0.00         0.33
##        bodywt
##          0.00
```

Sixty-one percent of the sleep_cycle values are missing. You may decide to drop it.

1.2.7.2 Listwise deletion

Listwise deletion involves deleting observations (rows) that contain missing values on *any* of the variables of interest.

```
# Create a dataset containing genus, vore, and conservation.
# Delete any rows containing missing data.
newdata <- select(msleep, genus, vore, conservation)
newdata <- na.omit(newdata)
```

1.2.7.3 Imputation

Imputation involves replacing missing values with "reasonable" guesses about what the values would have been if they had not been missing. There are several approaches, as detailed in such packages as `VIM`, `mice`, `Amelia` and `missForest`. Here we will use the kNN function from the `VIM` package to replace missing values with imputed values.

```
# Impute missing values using the 5 nearest neighbors
library(VIM)
newdata <- kNN(msleep, k=5)
```

Basically, for each case with a missing value, the k most similar cases not having a missing value are selected. If the missing value is numeric, the mean of those k cases is used as the imputed value. If the missing value is categorical, the most frequent value from the k cases is used. The process iterates over cases and variables until the results converge (become stable). This is a bit of an oversimplification - see Imputation with R Package VIM for the actual details.

> Important caveate: Missing values can bias the results of studies (sometimes severely). If you have a significant amount of missing data, it is probably a good idea to consult a statistician or data scientist before deleting cases or imputing missing values.