Data Manipulation

Objects in R R is an object-oriented language. Thus, every data element has a `class` and a `type`.

- Class: vector, `factor, matrix, data.frame, list`. Check the class of an R object with `class()`. `str()` also reveals much about the structure of an R object. To view just the first few rows of a `data.frame`, use `head()`.
  - A vector is a one-dimensional array of items of the same data type. A vector may be of class `numeric, character, integer, logical`, etc. Vectors have a `length()` but not a dimension. Vectors can have, but needn't have, `names()`. Vectors are the fundamental building blocks in R.
  - A `factor` is a special type of vector for categorical data. A factor has `level()`s. We can change the reference level of a factor with `relevel()`. *Do not confuse a factor with a character vector!* Factors are stored internally as integers that correspond to the id's of the factor levels.
  - A `matrix` is a two-dimensional array of items of the same data type. A matrix has a `length()` that is equal to `nrow()` times `ncol()`, or the product of `dim()`.
  - A `data.frame` is a `list` of vectors of the same length. This is like a matrix, except that columns can be of different data types. We will most often work with `data.frames`. Lots of information about data frames can be obtained via `summary()`. Data frames always have `names()` and often have `row.names()`.
  - A `list` is a collection of objects of any type. Lists are very flexible.

```
require(mosaic)
require(Lahman)
#str(Teams)
```

- Conversion from one class to another is called *casting*. Use `as.character()` to convert a factor to a character vector, `as.data.frame()` to convert a matrix to a data frame, etc. Note that R is also *case-sensitive*!
- Missing data: Watch out for `NA`'s. `NA` is the way that R stores a missing value – *do not confuse this with 0 or ""*. Use `is.na()` to check for missing values. Use the `na.rm=TRUE` argument to `mean` or `sum` to ignore missing values. Note that other languages use different symbols to denote a missing value (e.g. MySQL uses `NULL`). You can get the behavior you want by using the `na.strings` argument in `read.csv()`.

Common Operations in R In what follows we describe some common data manipulation operations in R. We will be using the `dplyr` syntax from Hadley Wickham's package of the same name. Note that `mosaic` now uses `dplyr` by default, so if you have loaded `mosaic`, then you have loaded `dplyr`.

`dplyr` defines five data manipulation *verbs* that act on data frames:

- `filter()`: take a subset of the rows
- `select()`: take a subset of the columns
- `mutate()`: modify existing columns or add new ones
- `arrange()`: sort the rows
- `summarise()`: aggregate the rows and apply functions to them

Hadley's assertion is that most of what you want to do can be accomplished using a combination of these five verbs.

- Add new variables to a data frame using `mutate()`, `with()` or `$`. Use `with()` to evaluate expressions within the scope of a specific `data.frame`.

```
Teams = mutate(Teams, SBPct = SB / (SB + CS))
favstats(~SBPct, data=Teams)

##  min        Q1    median        Q3       max      mean         sd    n
##    0 0.5856762 0.6517921 0.7056596 0.8789809 0.6426097 0.08811493 1914
##  missing
##      861
```

- If the value of the new variable depends on the value of one already defined, use `ifelse()`. Note that `ifelse()` operates and returns *vectors*, not individual values.

```
Teams = mutate(Teams, SBPct2 = ifelse(is.na(CS), 1, SBPct))
favstats(~SBPct2, data=Teams)

##  min        Q1 median Q3 max      mean        sd    n missing
##    0 0.6173913    0.7  1   1 0.7533195 0.1807706 2773       2
```

  Note that `mutate()` will also allow you to define multiple new variables at once.

- Find subsets of a `data.frame` using `filter()`, or by appealing directly to the indices.

```
mets = filter(Teams, teamID == "NYN")
```

- Use `select()` to retrieve only those columns that you want.

```
Teams2=select(Teams, yearID, teamID, W, L)
```

  Recall that `<-` and `=` are (usually) interchangeable assignment operators in R, but that `==` is a *test for equality*. Thus, `var = 7` means "set `var` equal to 7", whereas `var == 7` asks, "is `var` equal to 7"? You can use these operators to find subsets of `data.frames`. If you want to test for multiple values in a `factor` or string, use the `%in%` operator.

```
ny = filter(Teams, teamID %in% c("NYN", "NYA"))
unique(ny$teamID)

## [1] NYA NYN
## 149 Levels: ALT ANA ARI ATL BAL BFN BFP BL1 BL2 BL3 BL4 BLA BLF BLN ... WSU
```

  Note that while `unique` returns only the unique values present in a vector, the full factor levels remain!

- Often you will have two `data.frames`, and you will want to stick them together. If they have the same number of rows, and you want to append one to the right of the other, use `cbind()`.

```
offense = select(Teams, yearID, teamID, R)
defense = select(Teams, yearID, teamID, RA)
both = cbind(offense, defense)
```

If they have the same columns (including the column names) and you want to append one below the other, use `rbind()`.

```
yankees = filter(Teams, teamID == "NYA")
ny = rbind(mets, yankees)
```

- R is designed for statistical analysis, not so much tabular presentation. So displaying data nicely in tabular form is not a strength. The function `arrange()` can be used for this purpose. Note that you can sort by more than one column, and you can sort in either direction.

```
#You can order the data in a way that is useful to read
arrange(mets, desc(W), desc(R))
#Perhaps you only want to see some relevant columns
select(arrange(mets, desc(W), desc(R)), yearID, W, L, R)
```

```
##   yearID   W  L   R
## 1   1986 108 54 783
## 2   1988 100 60 703
## 3   1969 100 62 632
## 4   1985  98 64 695
## 5   1999  97 66 853
## 6   2006  97 65 834
```

If you want to export this table to LaTeXor HTML, you can do so using the `xtable()` package.

- Sometimes you will have a `data.frame` in *wide* format (n observations of k variables) that you will want to translate into a *long* format (nk observations of a single variable). The `long` format is usually more convenient for data analysis, but data summarized in spreadsheets is often in `wide` format. There are several ways to do this, but the most straight forward way is with the `reshape2` package using the function `melt`.

```
require(reshape2)
mets.wide = select(mets, yearID, R, RA)
mets.long=melt(mets.wide, id="yearID")
```

- Often you will want to discretize a continuous variable (that is, break it into non-overlapping intervals). We do this with `cut()`. Note that the result is a `factor`!

```
mets = mutate(mets, era = cut(yearID, 3))
favstats(W ~ era, data=mets)

##                      era min    Q1 median    Q3 max     mean        sd  n
## 1 (1.96e+03,1.98e+03]  40 61.50   68.5 82.75 100 69.83333 15.545947 18
## 2    (1.98e+03,2e+03]  41 67.00   72.0 91.00 108 77.05882 17.984266 17
## 3    (2e+03,2.01e+03]  66 74.25   80.5 88.00  97 81.72222  9.367061 18
##   missing
## 1       0
## 2       0
## 3       0

mets = mutate(mets, decade = cut(yearID, breaks = seq(from=1960, to=2020,
                                                     by=10)))
favstats(W ~ decade, data=mets)
```

```
##                decade min    Q1 median    Q3 max     mean        sd  n
## 1 (1.96e+03,1.97e+03]  40 51.00   61.0 73.00 100 64.11111 18.751296  9
## 2 (1.97e+03,1.98e+03]  63 66.25   76.5 82.75  86 74.70000  9.262229 10
## 3 (1.98e+03,1.99e+03]  41 72.75   90.5 96.50 108 84.00000 20.143927 10
## 4    (1.99e+03,2e+03]  55 69.50   74.5 88.00  97 77.00000 14.391355 10
## 5    (2e+03,2.01e+03]  66 72.00   80.5 86.75  97 80.00000  9.718253 10
## 6 (2.01e+03,2.02e+03]  74 74.00   75.5 77.50  79 76.00000  2.449490  4
##   missing
## 1       0
## 2       0
## 3       0
## 4       0
## 5       0
## 6       0
```