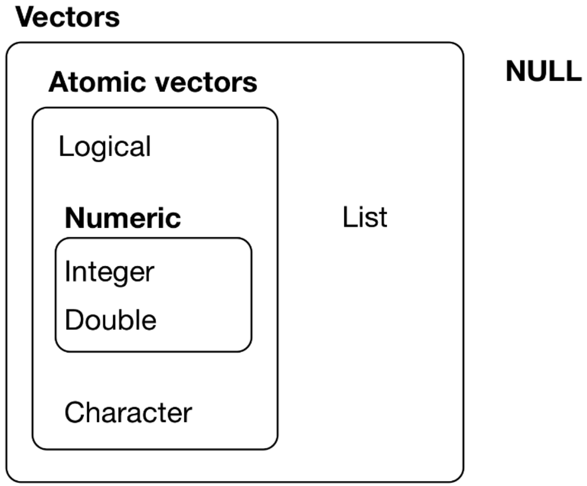
Data Manipulation with dplyr

36-600

Lists



- As can be seen above, a list is *not* an atomic entity; it may be thought of as a (perhaps heterogeneous) collection of atomic vectors:

```
(x <- list(1:5,c("a","b")))
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "a" "b"
```

Naming List "Columns"

- When you initialize a list, you can name each of the vectors and then use the names to access list elements

```
x <- list(u=1:5,v=c("a","b"))
x
```

```
## $u
## [1] 1 2 3 4 5
##
## $v
## [1] "a" "b"
```

```
x$v
```

```
## [1] "a" "b"
```

```
x$u[3]
```

```
## [1] 3
```

- Note the use of the dollar sign…the name of the list goes to the left of the dollar sign, while the name of the list element goes to the right

Data Frames

- **A data frame is simply a list whose entries all have the same number of elements!**

```
(x <- data.frame(u=1:2,v=c("a","b"),w=c(TRUE,FALSE)))
```

```
##   u v     w
## 1 1 a  TRUE
## 2 2 b FALSE
```

- Note that when we output a data frame, row numbers are added
- Are we sure this is a list?...we can check the variable type:

```
typeof(x)
```

```
## [1] "list"
```

```
class(x)
```

```
## [1] "data.frame"
```

Dataset: Galaxy Properties

- Let's read in data that are stored on disk in an ASCII (i.e., human-readable) file in csv format:

```
df <- read.csv("http://www.stat.cmu.edu/~pfreeman/GalaxyMass.csv",stringsAsFactors=TRUE)
```

- `df` stands for "data frame," but you can use whatever variable name you wish
- The data contain 3456 galaxies (in rows), with 10 measurements for each (in columns):

```
dim(df)
```

```
## [1] 3456   10
```

- We can use the `head()` function to display the first (in this case, 2) rows of the data frame:

```
head(df,2)
```

```
##    field      Gini       M20        C         A      size      n      q z.mode
## 1 COSMOS 0.4132853 -1.132330 2.257530 0.1211518 0.7554244 0.3398 0.3579   2.04
## 2 COSMOS 0.4177935 -1.603147 2.970555 0.1002479 0.6537786 0.6722 0.8124   1.79
##       mass
## 1 10.55120
## 2  9.87608
```

Dataset: Galaxy Properties

- To see the names associated with each variable/column, use the `names()` function:

```
names(df)
```

```
##  [1] "field"  "Gini"   "M20"    "C"      "A"      "size"   "n"      "q"
##  [9] "z.mode" "mass"
```

- The first column represents a sector of the sky in which the galaxy lies
  - this is a *factor* variable, because it takes on a few discrete values that don't have a specific ordering

```
unique(df$field)
```

```
## [1] COSMOS EGS    GOODSN GOODSS UDS
## Levels: COSMOS EGS GOODSN GOODSS UDS
```

dplyr

- `dplyr` is a package within the larger `tidyverse` that one uses to manipulate data frames

```
suppressMessages(library(tidyverse)) # suppressMessages() makes for cleaner output
```

- It may be helpful to think of data frames as nouns and `dplyr` functions as verbs, actions that you apply to the data frames
- The following are the most basic `dplyr` verbs:
    - `slice()`: choose particular rows based on integer indexing
    - `filter()`: choose particular rows based on logical criteria
    - `group_by()`: split the data frame into groups of rows based on factor variable values
    - `select()`: choose particular columns
    - `arrange()`: order rows by value of a column
    - `mutate()`: create new columns
- **NOTE**: calling `dplyr` verbs always outputs a new data frame…it *does not alter* the existing data frame

The slice() Function

- Use the `slice()` function when you want to retain certain rows:

```
df %>% slice(.,c(7,14)) # output rows 7 and 14 of the data frame
```

```
##    field      Gini      M20        C          A      size      n      q z.mode
## 1 COSMOS 0.3319734 -1.29408 2.975167 0.09361421 0.8088991 0.7442 0.2343   1.96
## 2 COSMOS 0.4839426 -1.60984 3.005040 0.17674453 0.4727087 1.3563 0.4967   1.69
##      mass
## 1 10.6824
## 2 10.1826
```

- The `%>%` is a *pipe*
  - a pipe takes the output of one `R` command and uses it as input to a following command
  - here, we pipe the output of `df` (which is simply the whole data frame) to the `slice()` function
  - the period in the `slice()` function call shows where the data frame is supposed to go among the functional arguments
- Slicing can be done "negatively":

```
df %>% slice(.,-c(1:2,19:23)) %>% nrow(.) # how many rows are left after removing rows 1-2 and 19-23?
```

```
## [1] 3449
```

The filter() Function

- Use the `filter()` function when you want to choose rows based on logical conditions:

```
df %>% filter(.,field=="COSMOS") %>% head(.,2)
```

```
##    field      Gini       M20        C         A      size      n      q z.mode
## 1 COSMOS 0.4132853 -1.132330 2.257530 0.1211518 0.7554244 0.3398 0.3579   2.04
## 2 COSMOS 0.4177935 -1.603147 2.970555 0.1002479 0.6537786 0.6722 0.8124   1.79
##       mass
## 1 10.55120
## 2  9.87608
```

```
df %>% filter(.,mass>10) %>% nrow(.)
```

```
## [1] 1862
```

```
df %>% filter(.,(field=="GOODSS" & mass<10)) %>% nrow(.)
```

```
## [1] 276
```

- Note the use of `&` in the third example: this combines conditions via the logical `and` (where `|` would be the analogous symbol for `or`)

The group_by() Function

- Use the `group_by()` function to split a data frame into groups of rows based on the values of a factor variable
    - note that `group_by()` in and of itself is only useful when its output is piped its to another function, e.g., `summarize()`:

```
df %>% group_by(.,field) %>% summarize(.,Mean=mean(mass),Number=n())
```

```
## # A tibble: 5 × 3
##   field   Mean Number
##   <fct>  <dbl>  <int>
## 1 COSMOS  10.2    905
## 2 EGS     10.0    750
## 3 GOODSN  10.1    464
## 4 GOODSS  10.1    588
## 5 UDS     10.2    749
```

- `Mean` and `Number` are column names in the *output* data frame
    - what is shown are the average value of the galaxy masses in each field, along with the number of galaxies in each field
- Note that the output of `summarize()` is a "tibble", which is a `tidyverse`-specific alternative to a base-`R` data frame

The select() Function

- Use the `select()` function when you want to choose certain columns:

```
df %>% select(.,Gini,q,z.mode) %>% slice(.,c(1:4)) # here we choose columns *and* rows
```

```
##        Gini      q z.mode
## 1 0.4132853 0.3579   2.04
## 2 0.4177935 0.8124   1.79
## 3 0.4212831 0.3585   2.02
## 4 0.4725081 0.4927   1.94
```

- `select()` can also be used negatively:

```
df %>% select(.,-Gini,-q,-z.mode) %>% ncol(.) # how many variables/columns are left?
```

```
## [1] 7
```

The arrange() Function

- Use the `arrange()` function to order rows by values of a column:

```
df %>% arrange(.,desc(mass)) %>% select(.,C,A,z.mode) %>% head(.,2)
```

```
##          C         A z.mode
## 1 3.594274 0.1711388   1.68
## 2 3.655988 0.2116704   1.78
```

- If one uses `arrange(.,mass)` then the masses will be ordered in *ascending* order, by default

The mutate() Function

- Use the `mutate()` function when you want to create one or several columns:

```
df %>% mutate(.,mass.linear=10^mass) %>% select(.,mass,mass.linear) %>% arrange(.,mass) %>% head(.,3)
```

```
##    mass mass.linear
## 1 8.285   192752491
## 2 8.540   346736850
## 3 8.550   354813389
```

Saving the New Data Frame

- As stated above, the original data frame is not altered in piping operations
- To save the result of a series of piping operations, use `<-` or `->`:

```
df.new <- df %>% filter(.,field=="EGS") %>% select(.,z.mode,mass)
nrow(df.new)
```

```
## [1] 750
```

```
df %>% filter(.,field=="COSMOS") %>% select(.,Gini,M20,C,A) -> df.new
nrow(df.new)
```

```
## [1] 905
```