R

Packages and Graphics

TidyData

- There are many ways to represent data in a data frame, and due to the history of R, almost all of them are use
- Recently there has been a push to create commonsense conventions, known as having "Tidy Data"
- Hadley Wickham (Major player in R and the tidy data movement) defines tidy data as
  - Each variable is in a column.
  - Each observation is a row.
  - Each value is a cell.

TidyR

- To promote and enable this, the package TidyR was released
- It was spawned an entire family of packages, collectively known as the tidyverse
  - You can install just tidyR by using install.packages('tidyR')
  - The entire family can be installed with install.packages('tidyverse')
- It contains many functions meant to manipulate data into a tidy form

The Pipe Operator

- `TidyR` is commonly presented using the operator `%>%`, which comes from an earlier package, `magrittr`
  - It is very similar to the pipe in bash, passing the output of one function as the first argument to the next function
  - The following are eqiuvalent

```
apply(data,1,function)

data %>% apply(1,function)
```

Spreading

- The `spread` function converts from long data to wide data
- The syntax of the `spread` function is

```
spread(data,key,value)
```

  - Key is the column you want to use to form your new columns
  - Value is the column you want to use to fill the cells

In [ ]:

```
library(DSR)
long <- table2
extra_wide_cases <- table4
combined <- table5
print(table2)
```

In [ ]:

```
library(tidyr)
print(as.data.frame(spread(long,key,value)))
```

Gathering

- Gathering is the opposite of spread
  - While it is uncommon to need this, it is possible someone made a data frame where not every column is a variable, and you need to collapse things a bit

```
gather(data, COLUMN_NAME1, COLUMN_NAME2, cols_to_gather)
```

In [ ]:

```r
#print(extra_wide_cases)
gathered_cases <- extra_wide_cases %>% gather("Year","Cases",2:3)
print(gathered_cases)
```

Separating and Uniting

- Separating and Uniting allows us to create multiple columns from one, or bring together columns that should never has been separated

```
separate(data,col_to_separate,new_columns)
  unite(data,col_to_add, from_columns)
```

In [ ]:

```
print(table5)
all_good <- table5 %>% unite("year",c("century","year"),sep="") %>%
separate("rate",c("cases",'population'),sep="/")
print(all_good)
```

DplyR

- DplyR is another package in the tidyverse
  - Improves upon earlier packaged named `plyr`, which allowed easy manipulation of data
  - Specifically designed to use with data frames
- Just like TidyR, commonly uses pipes
- All functions are verbs

Selecting Data

- DplyR contains two functions to select data
  - Select selects columns/variables
  - Filter selects rows/observations
- Both of these can take a list of names, but they are more useful with built-in functions in DplyR
  - endsWith
  - startsWith
  - contains
  - one_of

In [ ]:

```r
library(dplyr)
starwars <- as.data.frame(starwars)
row.names(starwars) <- starwars$name
head(starwars)
```

In [ ]:
```
## Standard Boring Select
select(starwars,hair_color,skin_color, eye_color)
```

In [ ]:

```
##  Select with Pipes and Ends_with
starwars %>% select(ends_with('color'))
```

In [ ]:
```
starwars %>% select(-name)
```

In [ ]:
```
starwars %>% filter(species != "Human")
```

In [ ]:
```
starwars %>% filter(species %in% c('Wookiee','Ewok'))
```

Selection Practice

- Print the names and planets of all characters who have a birth year of less than 50

Adding or Changing Variables

- The `mutate` and `transmute` functions are used to add new variables as well as update existing ones
  - `mutate` does not drop old variables
  - `transmute` drops everything except those in the function call

In [ ]:
```
starwars %>% mutate( height_inches = height * 0.393701)
```

In [ ]:
```
starwars %>% transmute( height_inches = height * 0.393701)
```

In [ ]:

```
starwars %>% filter(species %in% c('Wookiee','Ewok')) %>%
mutate( height = height * 0.393701)
```

Summarizing and Counting

- In general, to perform an action over a dataframe, use the summarize function
  - `summarize` takes in as its parameters other functions that do the calculations
  - The parameters to these inner functions should be the columns you want summarized
  - Multiple summaries can be computed with one call to `summarize`
- If all you want to do is count the frequency of values in certain column, use the `count` function and pass a column to count

In [ ]:
```
print(starwars %>% summarize(n_distinct(species)))
```

In [ ]:
```
species_counts <- starwars %>% count(species)
print(as.data.frame(species_counts))
```

In [ ]:

```
species_counts <- starwars %>% count(species,sort=TRUE)
print(as.data.frame(species_counts))
```

In [ ]:

```
species_counts <- starwars %>% count(species,homeworld,sort=TRUE)
print(as.data.frame(species_counts))
```

Group By

- The `group_by` function allows rows to be grouped based on their values in the given columns or columns
- This makes finding averages and other summary data per group very easy

```
group_by(data,LIST_OF_COLUMNS)
```

In [ ]:
```
print(starwars %>% group_by(species,homeworld) %>%
      summarize(avg_height = mean(height)))
```

In [ ]:

```
print(starwars %>%
                  group_by(species,homeworld) %>%
                      summarize(avg_height = mean(height),
                                min_height=min(height)))
```

GroupBy Practice

- Find the number of species on each planet

Combining Data Tables

- The various `join` functions offer database like functionality
  - Matching rows are joined together with their columns
  - Matching is done by default on any common variables, but can be specified
- `bind_rows` and `bind_columns` offer a simpler concatenation style combination
  - Matches by position always

In [ ]:
```
print(band_members)
```

In [ ]:
```
print(band_instruments)
```

In [ ]:
```
print(full_join(band_members,band_instruments))
```

In [ ]:
```
print(inner_join(band_members,band_instruments))
```

In [ ]:
```
print(left_join(band_members,band_instruments))
```

In [ ]:
```
print(right_join(band_members,band_instruments))
```

In [ ]:
```
print(band_instruments2)
```

In [ ]:

```
print(full_join(band_members,band_instruments2,
                by=c("name" = "artist")))
```

In [ ]:
```
print(bind_cols(band_members,band_members))
```

In [ ]:
```
print(bind_rows(band_members,band_instruments))
```

ggplot2

- R has long supported creating graphs from data, but the process was often messy and confusing
- `ggplot2` is a widely used package that standardizes how graphs are created
  - Based on the Grammar of Graphics, a language independent theory on how graphs should be created
  - A very large community with lots of extensions and enhancements available
  - Works directly on data frames

The `ggplot` function

- The `ggplot` function sets up the basics for our graph, including which data frame to use, and how to use it

```
ggplot(data_frame,aes(AESTHETICS))
```

- Aesthetics are what we see are the graph, and are defined using data frame columns
  - x and y position
  - color
  - shape

In [ ]:

```
library(ggplot2)
ggplot(starwars,aes(x=height,y=mass))
```

Geometries

- The base `ggplot` function sets up the graph and creates a ggplot object, but doesn't produce anything visually
- We need to specify how we want to display our data using geometries
  - geom_point
  - geom_boxplot
  - geom_histogram
  - geom_dist
- Geometries, and every other specification in ggplot2 is done by adding to the original ggplot call

In [ ]:
```
ggplot(starwars,aes(x=height,y=mass)) + geom_point()
```

In [ ]:
```
ggplot(starwars,aes(x=height,y=mass)) + geom_histogram()
```

In [ ]:
```
ggplot(starwars) + geom_histogram(aes(height)) +
geom_histogram(aes(mass))
```

In [ ]:
```
ggplot(starwars) + geom_density(aes(height),fill="blue",alpha=0.3) +
geom_density(aes(mass))
```

In [ ]:
```
ggplot(starwars,aes(x=height,y=mass,color=species)) +
geom_point()
```

GGplot 2 Basics Practice

- Draw a scatter plot that charts the number of species on a planet by the average age on that planet

In [ ]:

```
interesting <- (starwars %>%
         filter(!is.na(species)) %>%
             group_by(species) %>%
             summarize(count = n()) %>%
             filter(count > 2))$species
print(interesting)
to_vis <- starwars %>%
    filter(species %in% interesting)
```

In [ ]:
```
base_plot <- ggplot(to_vis,aes(x=species,fill=species,y=height))
base_plot + geom_violin()
```

Modifying Other Aspects

- `ggplot` has a function for almost every aspect of a graphs appearance
- To add titles, use the functions
  - xlabs, ylabs, ggtitle, labs
- To modify area shown, use
  - xlim, ylim, lims
- To modify colors use one of the `scale_` functions

In [ ]:

```
base_plot2 <- ggplot(to_vis,aes(x=mass,y=height,color=species))
scatter <- base_plot2 + geom_point()
plot(scatter)
```

In [ ]:
```
scatter + ggtitle("Height vs Mass of Starwars Characters")
```

In [ ]:
```
scatter + labs(title="Height vs Mass of Starwars Characters",
               x="Mass (kg)",y="Height (cm)")
```

In [ ]:

```
scatter + labs(title="Height vs Mass of Starwars Characters",
               x="Mass (kg)",y="Height (cm)") + xlim(0,175) +
ylim(0,240)
```

In [ ]:

```
scatter + labs(title="Height vs Mass of Starwars Characters",
               x="Mass (kg)",y="Height (cm)") + xlim(0,175) +
guides(color=guide_legend(title="Species"))
```

In [ ]:

```
scatter + labs(title="Height vs Mass of Starwars Characters",
               x="Mass (kg)",y="Height (cm)") + xlim(0,175) +
guides(color=guide_legend(title="Species")) +
scale_color_brewer(palette = "Set1")
```

Themes

- Themes allow you to control things like font, gridline color, etc.
- The elements of the theme can be modified by using the `theme` function and passing the appropriate parameters
- More common is to download or use an existing theme, and add it to your plot using `+ theme_NAME`

In [ ]:

```
library(ggthemes)
almost_finished <- scatter +
labs(title="Height vs Mass of Starwars Characters",
     x="Mass (kg)",y="Height (cm)") +
xlim(0,175) + guides(color=guide_legend(title="Species"))
almost_finished + theme_fivethirtyeight()
```

In [ ]:
```
almost_finished + theme_wsj()
```

In [ ]:
```
almost_finished + theme_economist()
```

In [ ]:
```
almost_finished + theme_tufte()
```

Facet Grids

- Facet Grids allow us to create "mini" plots, per categorical variable
- After setting up your plot as your normally would, you add in the facet_grid()

```
facet_grid(ROWS ~ COLUMNS)
```

In [ ]:
```
almost_finished + facet_grid(. ~ eye_color)
```

In [ ]:
```
almost_finished + facet_grid(hair_color ~ .)
```

In [ ]:
```
almost_finished + facet_grid(hair_color ~ eye_color)
```

Saving Plots

- While gpplot2 is very easy to use in a good R IDE, many times we want to share our plots
- The `ggsave` function by default will save the last plot to a given file location
- The type of file is guessed from the name, but if you want to specify it, use the device parameter

```
ggsave(file_name, plot = plot_var)
```

In [ ]:
```
my_final_plot <- almost_finished + theme_fivethirtyeight()
ggsave("final_plot.pdf",dpi=600,width=10)
```