Pipes (%>%) and ggplot

R Workshop

Jennifer Lin

2021-04-14

Overview of Today's Workshop

1. Introduce the data
2. Discuss `dplyr` and pipes (`%>%`)
3. Discuss `ggplot2` basics
4. If time permits, discuss `ggplot` features for time series and spatial data

The Data

Cooperative Congressional Elections Studies 2018

How do different groups of people vary in their approval of American Political Institutions?

The CCES 2018 data has some variables that can help us answer this question.

The dataset we use in this workshop is an oversimplification of the varaibles that are actually available in the data

For more information: https://cces.gov.harvard.edu/

Institutional Approval Variables

There are three branches of government in American politics and the variables that start with app signal institutional approval variables where 1 = approve and 2 = disapprove

> Do you approve or disapprove of the way each is doing their job?
>
> 1. President Trump
> 2. US Congress
> 3. US Supreme Court

Choose your Adventure

For this workshop, I will use the approval for Congress as my main dependent variable

All approval variables are coded the same way so feel free to explore as you wish!

Load the Data

If you downloaded the GitHub repository, this should run directly. Otherwise, please make sure that the data is in the same folder as your script or Markdown file.

```
library(readr)
CCES_2018 <- read_csv("CCES18_subset.csv")
```

Look at the Data

```
glimpse(CCES_2018)
```

```
head(CCES_2018)
```

```
dim(CCES_2018)
```

```
## [1] 60000    19
```

`dplyr` and Pipes

Common `dplyr` Commands

Imagine your data are a collection of cheese boards

Each type of meat or cheese is a variable

Each cheese board is an observation

`select()` = choosing the type of meat or type of cheese (or olives or whatever).

`filter()` = Exclude cheese boards with pickles

`mutate()` = Create an indicator for whether this board is vegetarian where `TRUE` there is no meat.

`rename()` = Name meats and cheese instead of `meat_1` or `cheese_1`

`group_by()` = group cheese boards that have olives to compare to those that do not have olives

`summarise()` = tell your friend about all the cheese boards you are looking at

Pipes (%>%)

Imagine doing a lot of different commands all in one chunk

You can with PIPES!

Funnels each stem into the next step without creating a new object

```
data %>%
  step 1 %>%
  step 2 %>%
  step 3 %>%
  steo 4
```

`filter()`

Putting Observations through a sifter where `var operator condition`

```
filter(.data, ...,)
```

```
data %>% filter(...)
```

We can use conditions for variables where the variables are joined by AND (&) or OR (|) operators

Operator	Description
==	exactly equal to
<	less than
<=	less than or equal to
`%in%`	is in this list
!=	not equal to
`is.na()`	NA values

```
CCES %>%
  select(
    pid3,
    votereg,
    appCongress
    ) %>%
  filter(
    pid3 == "Independent"
    ) %>%
  filter(
    votereg == "Yes" |
    appCongress == 2)
```

```
## # A tibble: 5 x 3
##   pid3        votereg appCongress
##   <chr>       <chr>          <dbl>
## 1 Independent Yes                2
## 2 Independent Yes                2
## 3 Independent No                 2
## 4 Independent Yes                1
## 5 Independent Yes                2
```

Exercise

We are interested in approval to a political institution. Think about the institution you want to focus on (and a second) and do the following:

1. Select the state, region, party, and two approval variables of choice (Congress, President, Supreme Court)

2. Filter out the NA values on state and party variables. Hint: use `is.na()` and `!`.

`mutate()`

Create new variables that are transformations of pre-existing ones.

```
data <- data %>%
  mutate(new = function(old))
```

where `function()` denotes classic functions like `sum()`, `mean()`, `length()`

Recoding variables

The `mutate()` command can also be useful to recode variables

Use `case_when()`, `recode()` or `if_else()` in the `dplyr` package

```
data <- data %>%
  mutate(new = case_when(
    old == [old] ~ [new code],
    # For example:
    voterreg == "yes" ~ 1,
    # For all else not previously defined
    TRUE ~ [new code]
  ))
```

summarise()

Reduce observations to simple statistics.

Is usually preceded by `group_by()` which groups the data by a grouping variable

```
CCES_2018 %>%
  group_by(pid3) %>%
  summarise(
    mean    = mean(appCongress, na.rm = TRUE)
    .groups = 'keep')
```

Add `.groups = 'keep'` to maintain the groups or `.groups = 'drop'` to remove all groupings

Pull it all together

In the data frame that you previously created that took out region, state, party, and approval variables, create a summary table that describes the precent of people in each state-party-region combination that approves your institution of choice. Use mutate and `case_when()` to recode your region variable so that 1 = Northeast, 2 = Midwest, 3 = South and 4 = West.

HINT - You can put multiple variables in the `group_by()` command

HINT - To create a proportion variable, use can use the following function in the summarise environment. NOTE how the approval variables are coded (1 = approve, 2 = disapprove)

Solution

```
state_party_Congress <- CCES_2018 %>%
  select(state, pid3, region, appCongress, appTrump) %>%
  filter(!is.na(state)) %>%
  filter(!is.na(pid3)) %>%
  mutate(
    region = case_when(
      region == 1 ~ "Northeast",
      region == 2 ~ "Midwest",
      region == 3 ~ "South",
      region == 4 ~ "West"
    )
  ) %>%
  group_by(state, region, pid3) %>%
  summarise(
    mean         = mean(appCongress, na.rm = TRUE),
    pctCongAppv = sum(appCongress == 1, na.rm = TRUE)/length(appCor
```

`ggplot2`: The Grammar of Graphics

`ggplot2` Helps Us Make Graphs

Graphs are created by layers such that

```
ggplot(data layer)+
  graph layer +
  label layer +
  scale loayer +
  theme layer +
  others
```

where each layer (roughly) shows what should be there. Graphs do NOT need to follow a standard order so long as you get your desired outcome

Layers are joined with + operator

The Data Loyer

This usually goes in the `ggplot()` function that starts the graph

Include information about your data frame and the variables that you want in `aes()`.

```
ggplot(state_party_Congress,
       aes(x = pctCongAppv,
           y = pctPresAppv,
           color = "red")
       )
```

Difference between `fill` and `color`

`color` the lines (or dots)

`fill` the space

It it is a point or a line, use `color`. For polygons, use `fill`

The Graph Layer

`ggplot2` has many options of what graphs we can include

Operator	Description
`geom_line()`	line graph
`geom_bar()`	bar plot
`geom_histogram()`	histogram
`geom_boxplot()`	boxplot
`geom_violin()`	violin plot
`geom_sf()`	maps with shapefiles

Quick Demo of a Scatterplot

```
ggplot(state_party_Congress,
       aes(x = pctCongAppv, y = pctPresAppv, color = "red"))+
  geom_point()
```

Quick Demo of a Boxplot

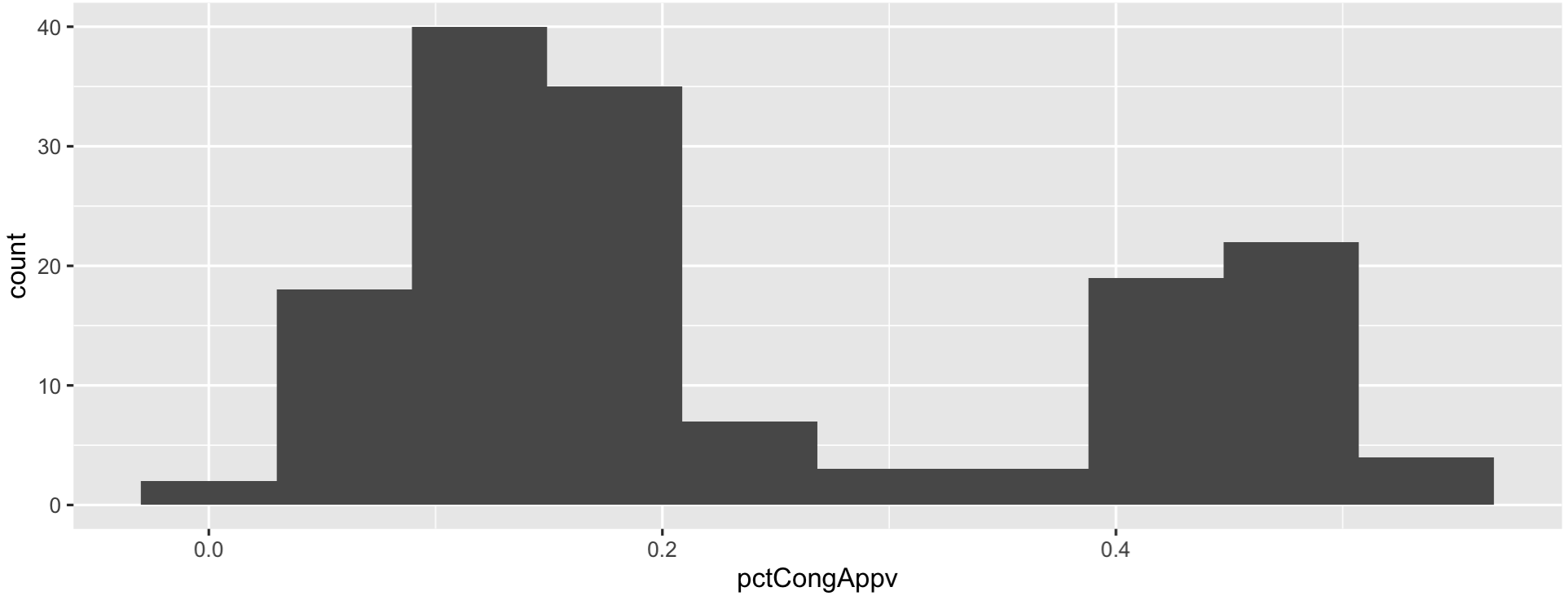
```
ggplot(state_party_Congress,
       aes(x = pid3, y = pctCongAppv,  fill = pid3))+
  geom_boxplot()
```

Quick Demo of a Bar chart

```
ggplot(state_party_Congress,
       aes(x = pid3, y = pctCongAppv ))+
  geom_bar(stat = "identity",
           fill = "white")
```

Quick Demo of a Histogram

```
ggplot(state_party_Congress, aes(x = pctCongAppv))+
  geom_histogram(bins = 10)
```



Exercise

Using the data frame that you created (or `state_party_Congress`), create a very basic graph like the demos above. Take some time to play around with the possible graphs in ggplot. The goal is to get you to feel comfortable with the data and the plot layers.

The Label Layer

For the x- and y-axes, use `xlab()` and `ylab()`

For the title and subtitle

```
labs(
  title    = "",
  subtitle = "",
  caption  = ""
)
```

The Scale Layer

Use scales to override defaults

Most `ggplot` scales come in the format of `scale_[SOMETHING]_[SOMEHOW]()`

- `[SOMETHING]` -- What do you want to scale? Color, the x-axis (x), the fill, y-axis (y)?
- `[SOMEHOW]` -- How do you want it to start the rescale process? Transformed, gradients, manual?

Arguments

Most scales take the following arguments

- `name` = Name the thing you are scaling
- `breaks` = Locate where you want to break it
- `labels` = Assign each break point a name
- `limits` = Set upper and lower bounds (if applicable)

Common Scale Uses

- `scale_x_continuous()`
  - `[SOMETHING]` = x-axis
  - `[SOMEHOW]` = continuously

- `scale_fill_manual()`
  - `[SOMETHING]` = shape fill
  - `[SOMEHOW]` = manually

- `scale_colour_brewer(palette = "[PALETTE NAME]")`
  - `[SOMETHING]` = color
  - `[SOMEHOW]` = Using the R Color Brewer palette

The Theme Layer

There are 2 different types of theme layer settings

1. Global Themes
2. Specific options

Global Themes

Classic Options

- `theme_bw()`
- `theme_classic()`
- `theme_light()`
- `theme_linedraw()`

from `ggthemes`

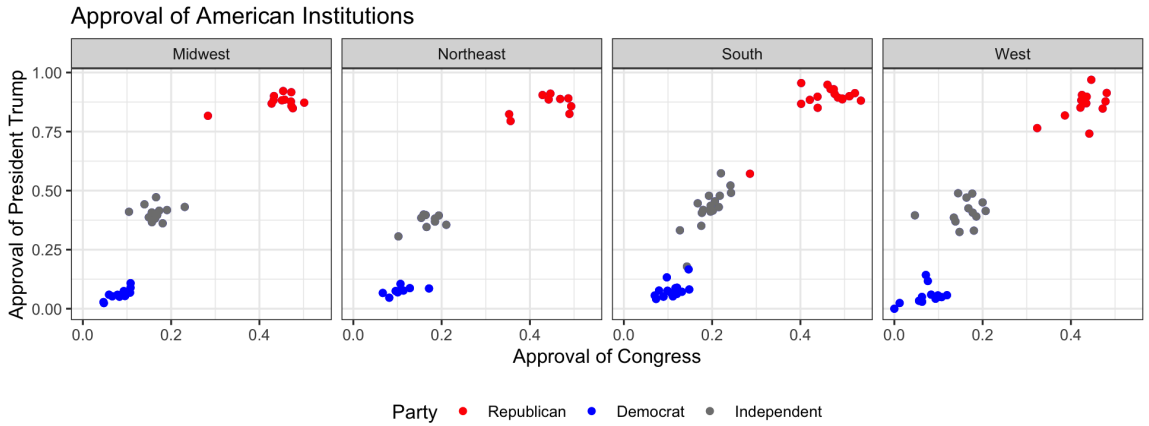
- `theme_tufte()`
- `theme_gdocs()`
- `theme_calc()`

Themes Options

On top of these presets, you can customize your graph using arguments in `theme()`

```
theme(
  plot.title       = element_text(hjust  = 0.5),
  axis.title       = element_text(colour ="black"),
  axis.title.y     = element_text(size   = 12),
  legend.position = 'none'
)
```

```
ggplot(state_party_Congre
       aes(
         x = pctCongAppv,
         y = pctPresAppv,
         group = pid3))+
  geom_point(color = 'blu
  geom_point(aes(color =
  xlab("Approval of Congr
  ylab("Approval of Presi
  ggtitle("Approval of Am
  scale_color_manual(
    name = "Party",
    breaks = c("Republica
    values = c("Republica
  )+
  theme_bw()+
  theme(
    legend.position = 'bo
  )+
```



Pull it Together

Using the plot that you created in the previous exercise, do the following

1. Add a title, x and y axis labels
2. Edit the legend to fit your graph
3. Change the theme

Dates

`lubridate` Rundown

We know `20210304`, `04mar2021` and `March 4, 2021` all refer to the same date.

But R doesn't

How can this be a problem?

Save all the dates and look at the way R treats it.

```
date_1 <- 20210304
class(date_1)
```

```
## [1] "numeric"
```

```
date_2 <- "04mar2021"
class(date_2)
```

```
## [1] "character"
```

```
date_3 <- "March 4, 2021"
class(date_3)
```

```
## [1] "character"
```

The Problem

There are many, but here are some

- Opens the opportunity to mistreatment of variables -- what might be meant as a date is instead a number
- Lack of standardization in date formats -- imagine having `mar042021` and `March 04 2021` and `03 04 2021` and `03-04-2021` all in the same variable.

Here comes `lubridate`!

`lubridate` makes it easy to parse variables

All we need to know are the positions of the day, month and year and it will handle the rest.

```
library(lubridate)
```

Examples

`20210304` -- **y**ear **m**onth **d**ay

```
ymd("20210304")
```

```
## [1] "2021-03-04"
```

`04mar2021` -- **d**ay **m**onth **y**ear

```
dmy("04mar2021")
```

```
## [1] "2021-03-04"
```

`March 4, 2021` -- **m**onth **d**ay **y**ear

```
mdy("March 4, 2021")
```

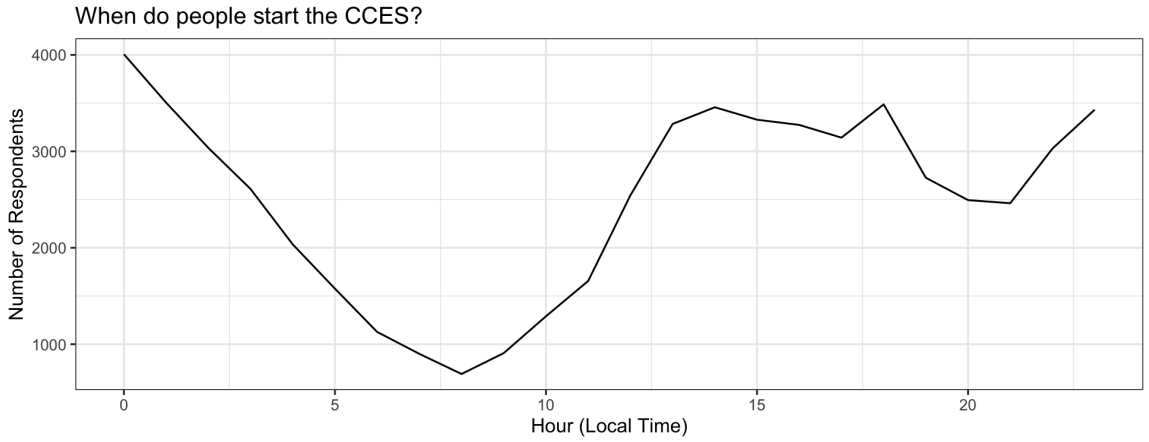
```
## [1] "2021-03-04"
```

`20210403` -- **y**ear **d**ay **m**onth

```
ydm("20210403")
```

```
## [1] "2021-03-04"
```

```
ggplot(Hour,
       aes(x = hour, y =
  geom_line()+
  xlab("Hour (Local Time)
  ylab("Number of Respond
  labs(
    title = "When do peop
  )+
  theme_bw()
```



Spatial Data

Overview

R contains simple and powerful tools to create maps using `ggplot` in conjunction with packages like `tigris` and `sf` (both are designed for spatial data)

This next section will quickly walk through how to draw a simple map in R using Census shapefiles and `ggplot`

But what is a Shapefile?

ArcGIS defines it as

> A shapefile is a simple, nontopological format for storing the geometric location and attribute information of geographic features. Geographic features in a shapefile can be represented by points, lines, or polygons (areas).

Basically a dataframe with data about a geographic boundary

Getting Shape Files

The `tigris` package contains commands that help us interact with the Census API to get the shapefiles.

```
library(tigris)
```

The `counties()` command gets us shape files for a state of choice

```
IL_county <- counties("IL", cb = TRUE)
```

Exercise

The CCES has data for respondents from nearly every state.

Pick a state (not Illinois) and do the following:

1. Get the state's county shape files with `tigris`
2. Adopt the following code to your state to find the number of people who respond to the CCES by county, identified with `countyfips`

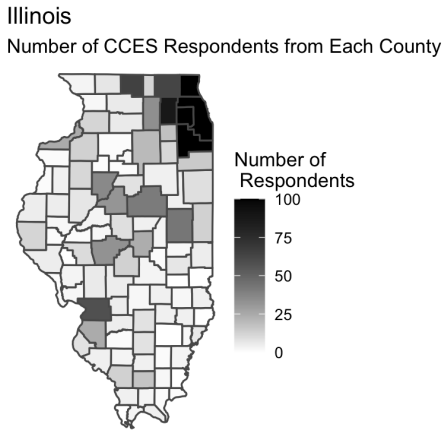
```
IL <- CCES_2018 %>% filter(state == "IL") %>%
  group_by(countyfips) %>%
  summarise(
    n = n(),
```

Prepare for mapping

Use the following code (change state abbreviation) to fix your data for mapping

```
IL_respondents <- left_join(
  IL_county,
  IL,
  by = c("GEOID" = "countyfips")
) %>%
  mutate(
    respondents = ifelse(is.na(n), 0, n)
  )
```

```
ggplot()+
  geom_sf(
    data = IL_respondents
    aes(fill = respondent
  scale_fill_gradient(
    low = "gray100",
    high = "gray0",
    na.value = "black",
    limits   = c(0, 100)
  )+
  labs(
    title    = "Illinois"
    subtitle = "Number of
    fill     = "Number of
  )+
  theme_void()
```



Pull it ALL Together

Take the mapping code and adjust it to your data

While you are at it, add a title and see what happens when you change the limits under `scale_fill_gradient()`