Intermediate and Advanced Graphs in R

Jennifer Lin

2022-02-25

Welcome to the Intermediate and Advanced `ggplot` R Workshop! Today, we will explore, in greater detail, the inner workings of the `ggplot2` package in R. But before we do, I want to make clear my assumptions about participants' familiarity with `ggplot2` and lay out the goals for the workshop.

This workshop assumes familiarity with the basics of `ggplot2`. Participants should know how to install and load `ggplot2`, either in isolation or with other tidyverse packages. Participants should be familiar with the basic structure of a `ggplot2` code chunk such that they can identify the lines of code that import the data, create the graph, insert labels and change colors, among others[1].

[1] To review these basics, please refer to the materials for my Beginner's `ggplot` Workshop on GitHub: `https://github.com/NUpolisci/Fall-Module-3`

In the introductory graphs workshop, I conceptualized a basic `ggplot` object as one that consists of the following general structure:

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

The introductory sequence covered the fundamentals for each of these layers. In the workshop today, we add additional details to each of these layers to help us make more interesting graphs with more complex data.

The goals of this workshop are as follows:

1. Become familiar with different ways of integrating factor variables in graphs in R.
2. Become more comfortable in `ggplot2` by creating more complex graphs, including line plots and stacked bar charts.
3. Explore different ways to be creative with coding tools and streamline R code for making graphs

As a caveat, while this workshop covers `ggplot2` in greater detail than the introductory workshop, there are still lots to learn about this package. Possibilities are endless but here are just some of the many interesting things this package can do that this workshop will not address[2]:

[2] If you are interested in learning more about each of these examples here, please feel free to reach out!

1. Maps in R

2. Animated and Interactive plots
3. Social Network illustrations
4. Graphs from text analysis results

A Review of the Basics

The Data

For the workshop today, we will use data cleaned from the 2020 ANES.

```
ANES <- read.csv("ANES_adv_ggplot.csv")
```

The ANES asks respondents a battery of items each year to assess where they would place themselves on a left-right scale for various issue positions. In my examples, I consider responses to the following variables:

- Spending and Services
- Defense Spending
- Government versus Private Medical Insurance
- Guaranteed Job/Income
- Environment-Business Tradeoff
- Government Assistance to Blacks

For the exercises, you are free to use these variables, but I also included a bunch of others in the prepared data set that you can use. Here are the recoded names, original names as they appear in the codebook and a short description[3].

Recode Name	Original Name	Description
`self_spending`	V201246	7pt scale spending & services: self-placement
`self_defense`	V201249	7pt scale defense spending: self-placement
`self_medical`	V201252	7pt scale gov-private medical insurance: self-placement
`self_job`	V201255	7pt scale guaranteed job/income: self-placement
`self_blacks`	V201258	7pt scale government assistance to blacks: self-placement

[3] There are other variables in the data that you are free to use, but I did not include them in the table mainly because they are easier to find. Variables that start with `FT_` are feeling thermometers, and the name contains a description of what the feeling thermometer is for. Variables that start with `rr_` are the racial resentment items. Variables that end with `_factor` are actually character strings for demographics that you should convert to a factor using the code I show later on for the exercises.

Recode Name	Original Name	Description
`self_business`	V201262	7pt scale environment-business tradeoff: self-placement
`fed_socSecurity`	V201302x	Federal Budget Spending: Social Security
`fed_publicSch`	V201305x	Federal Budget Spending: Public Schools
`fed_border`	V201308x	Federal Budget Spending: Tightening border security
`fed_crime`	V201311x	Federal Budget Spending: Dealing with crime
`fed_welfare`	V201314x	Federal Budget Spending: Welfare programs
`fed_highway`	V201317x	Federal Budget Spending: Building and repairing highways
`fed_povertyAid`	V201320x	Federal Budget Spending: Aid to the poor
`fed_environment`	V201323x	Federal Budget Spending: Protecting the environment
`vote_byMail`	V201356x	Favor/Oppose vote by mail
`vote_ID`	V201359x	Favor/Oppose requiring ID when voting
`vote_felons`	V201362x	Favor/Oppose allowing felons to vote

Graphs

Recall the `ggplot` allows you to make many graphs using rather intuitive functions. Some of these include:

Operator	Description
`geom_line()`	line graph
`geom_point()`	scatterplot
`geom_bar()`	bar plot
`geom_histogram()`	histogram
`geom_boxplot()`	boxplot
`geom_violin()`	violin plot

The introductory workshop covers `geom_point()`, `geom_histogram()`, `geom_boxplot()` and `geom_bar()`. This workshop takes a deeper look at `geom_bar()` and looks at `geom_line()`

Theme Functions – A Quick Digression

Up to this point, you might have been copying and pasting the same many lines of code in the `theme()` function across all of your graphs in `ggplot2`. While this method works, it can get a bit annoying and take up too much space in your code. A convienent way to streamline your code on this front is to create a customized function with your favorite theme presets and use that function in your plots in place of `theme_*()` and `theme()` canned commands and preset arguments.

To do this, we leverage some facts that are core to the operation of R and `ggplot2`. In the Introductory workshop, I demonstrate how `ggplot2` objects are created in an additive fashion such that the order in which the layers were added did not affect how the graph was created unless contradictions occur. When this happens, the subsequent function will be given priority. On top of this, R is an object-oriented programming language, meaning that users can program their own objects and use them on top of canned commands. By creating a customized theme function, users can streamline code such that they can have a general theme going across the script but also have the freedom to customize for any specific graph just by adding another `theme()` layer to the plot object.

A customized theme function can take the following form. You can give the function any name you desire. The function takes no arguments but contains the presets for the various theme components that you might want for your graphs.

```
theme_DEMO <- function(){
  theme_classic()+
  theme(
    plot.title         = element_text(hjust = 0.5, size = 20),
    plot.subtitle      = element_text(colour="black", face = "bold"),
    legend.title       = element_text(hjust = 0.5, face = "bold"),
```

```
    ...
  )
}
```

Now, you can construct plots with a theme like so:

```
ggplot()+
  ...
  theme_DEMO()
```

For the purposes of this workshop, here is my theme function

```
theme_rworkshop <- function() {
  theme_bw()+
  theme(
    plot.title          = element_text(hjust = 0.5, size = 20, colour="black", face = "bold"),
    plot.subtitle       = element_text(hjust = 0.5, size = 16, colour="black", face = "bold"),
    legend.title        = element_text(hjust = 0.5, size = 14, colour="black", face = "bold"),
    plot.caption        = element_text(size = 10, colour="black"),
    axis.title          = element_text(size = 14, colour="black"),
    axis.text.x         = element_text(size = 12, colour="black", angle = 0, hjust = 0.5),
    axis.text.y         = element_text(size = 12, colour="black"),
    legend.position     = 'bottom',
    legend.direction    = "horizontal",
    legend.text         = element_text(size = 12, colour="black")
  )
}
```

Exercise 1

1. Look through the ANES data. PICK
   a. ONE FACTOR variable to use for your graph as a grouping variable
   b. ONE numeric variable for which to focus your graph
   c. ONE other variable – your pick – that would go well with the variable you picked in (a) and (b)
2. Generate a theme function for use for the rest of this workshop.
3. Generate a graph with everything you know about `ggplot`. Include proper labels, titles, colors, and themes (ideally using the function from (2)). Save as a PDF.

To help you out, here is some code that might be useful:

```
ANES %>%
  filter(!is.na(PARTY)) %>%
  ggplot(aes(x = FT_rural, y = FT_BLM, color = PARTY))+
```

```
    geom_point(position = position_jitter(1, 1), alpha = .5)+
  xlab("Feelings towards Rural Americans")+
  ylab("Feelings towards the BLM Movement")+
  labs(
    color = "Political Party",
    title = "Feelings towards Rural Americans and
    the BLM Movement",
    subtitle = "Analysis from the American National Elections Studies",
    caption = "Data: ANES 2020
    Author: Jennifer Lin"
  )+
  scale_color_manual(
    name = "Political\nParty",
    breaks = c("Democrat",  "Republican", "Independent"),
    values = c("Democrat" = "#3182bd", "Republican" = "#de2d26", "Independent" = "#636363")
  )+
  scale_x_continuous(
    breaks = seq(0, 100, 10),
    limits = c(0, 100)
  )+
  scale_y_continuous(
    breaks = seq(0, 100, 10),
    limits = c(0, 100)
  )+
  theme_rworkshop()
```

Stacked Bar Graphs

In the introductory sequence, we discussed bivariate bar graphs where there is traditionally a categorical variable on the x-axis and a count or average on the y-axis. However, what if we wanted to add a third variable that can help tell a more productive story?

We can leverage stacked bar graphs to add an additional variable and have the picture tell a more detailed story about the data that we are interested in analyzing.

For this exercise, I will generate a stacked bar chart for the number of respondents by political party who answer on each level of the spending and services scale.

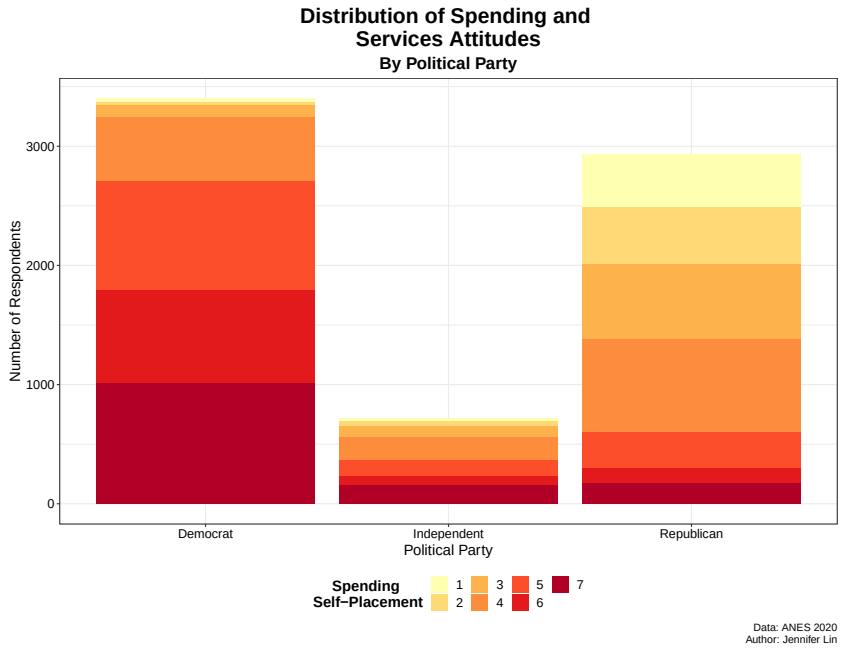
To do this, like the bar graph discussion in the introductory sequence, we need to construct a data frame that generates summary statistics to plot.

```
Spending <- ANES %>%
  group_by(PARTY, self_spending) %>%
  summarise(
```

```
    n = n(),
    .groups = 'keep'
    ) %>%
  filter(!is.na(PARTY) & self_spending != "NaN")
```

Now, we can make the graph

```
ggplot(Spending, aes(x = PARTY, y = n, fill = factor(self_spending)))+
  geom_bar(stat = 'identity', position=position_stack())+
  scale_fill_brewer(palette = 'YlOrRd')+
  labs(
    fill = "Spending \nSelf-Placement",
    title = "Distribution of Spending and \nServices Attitudes",
    subtitle = "By Political Party",
    caption = "Data: ANES 2020
    Author: Jennifer Lin"
  )+
  xlab("Political Party")+
  ylab("Number of Respondents")+
  theme_rworkshop()
```



Looks great! Now, let's decode each line in the graph.

```
ggplot(Spending, aes(x = PARTY, y = n, fill = factor(self_spending)))+
```

Looking at the first line, we know that this is the data layer. I am calling in the Spending data frame that I created earlier and I am using the partisanship variable on the x-axis and the number of respondents on the y-axis. Since this is a bar graph, I use the `fill` argument

and since I am interested in plotting each level of the responses to the Spending and Services question, I use `factor()` to declare the `self_spending` variable a factor.

```
geom_bar(stat = 'identity', position=position_stack())+
```

This line creates the bar plot, with the number plotted being that of the variable we previously declared for the y-axis. This is what `stat = 'identity` does. It tells R to do no implicit math to plot the data. To make a stacked bar chart, the key is `position=position_stack()`. This tells R to stack the bars into one bar per level on the x-axis[4].

[4] Recall that `position=position_dodge()` is the position argument for the standard bivariate bar graphs

```
scale_fill_brewer(palette = 'YlOrRd')+
```

This line tells R what color we want to assign to the fill argument for the bar chart. Since there are 7 levels at hand, I can manually indicate which color I would like to use. However, this can be a bit time consuming so, instead, I leverage the R Color Brewer palette series to automate this process[5].

[5] Here is a neat tool to help you visualize the colors before applying them to you graphs: `https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3`

```
labs(
  fill = "Spending \nSelf-Placement",
  title = "Distribution of Spending and \nServices Attitudes",
  subtitle = "By Political Party",
  caption = "Data: ANES 2020
  Author: Jennifer Lin"
)+
xlab("Political Party")+
ylab("Number of Respondents")+
```

These lines collectively add a legend label, title, subtitle, caption, x-axis label and y-axis label in that order. Notice that I can create a new line wither with the `\n` or by hitting ENTER in my labels arguments and that would generate a new line in the output[6].

[6] Note that this is a minor departure for the slides. I added the `\n` to demonstrate a new line here but not in the slides. This was mainly motivated by a compiling error for the notes.

```
theme_rworkshop()
```

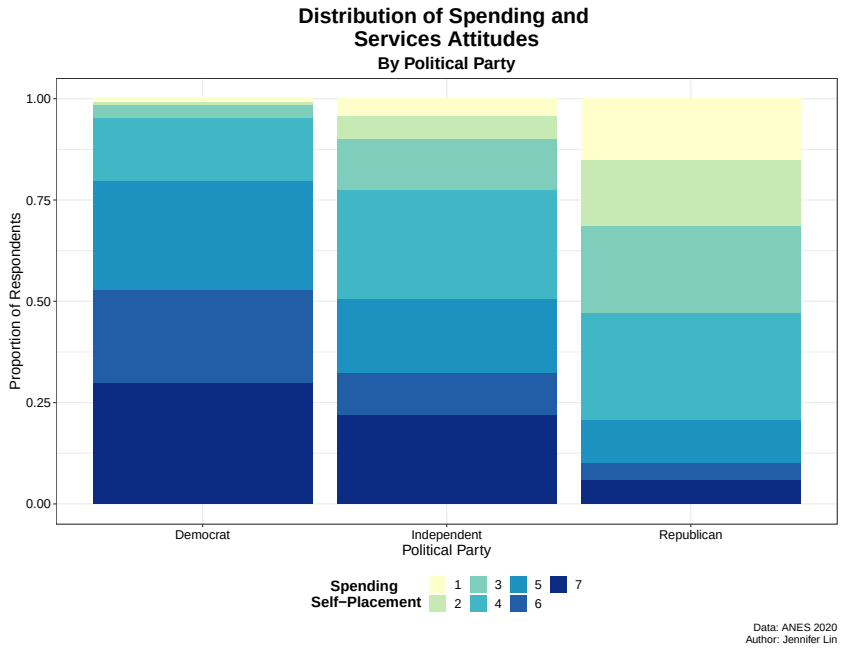
Finally, this is the theme object we created earlier.

Proportional Stacked Bar Charts

An additional layer to stacked bar graphs is the ability to display data in proportions rather than raw counts. We can leverage the same summary statistics data frame that we created above and do this quite easily.

```
ggplot(Spending, aes(x = PARTY, y = n, fill = factor(self_spending)))+
  geom_bar(stat = 'identity', position=position_fill())+
```

```
scale_fill_brewer(palette = 'YlGnBu')+
labs(
  fill = "Spending \nSelf-Placement",
  title = "Distribution of Spending and \nServices Attitudes",
  subtitle = "By Political Party",
  caption = "Data: ANES 2020
  Author: Jennifer Lin"
)+
xlab("Political Party")+
ylab("Proportion of Respondents")+
theme_rworkshop()
```



You might think that the code here looks largely the same as the one for raw counts, and for the most part, you are correct.

However, there is one key change that will turn a stacked bar graph from displaying raw counts to proportions. This change is located in the graph layer such that this line now reads:

```
geom_bar(stat = 'identity', position=position_fill())+
```

Recall that we used `postiion_stack()` in the previous segment for the stacked bar graphs. By changing this to `position_fill()`, we generate a proportion representation of the data in the graph[7].

[7] Oh and for demonstration purposes, I also used a different palette from R Color Brewer.

Exercise 2

From your variables in Exercise 1, generate a stacked bar chart with any combination of the three variables such that it tells a story about

the relationship of the variables. Include proper labels, titles, colors and themes

Line Plots with Factor Variables and Facets

Factor Variables: A Review

Before we dive into line plots, we start with a quick review on factor variables. For the line graph example, I want to plot attitudes on the Spending and Services item, along with other issue attitude variables, using the 7-category party ID variable, which is currently numeric. Therefore, I need to do some recoding to get this variable to be coded in the way that I need it for my graph. Like the stacked bar chart discussion, the line graph also would require a summary statistics table, so I will create my factor variable en route to generating this summary statistics table. In the code below, I do just that.

```
Spending_Line <- ANES %>%
  mutate(
    PARTY7          = case_when(
      pid7 == 1 ~ "Strong Democrat",
      pid7 == 2 ~ "Democrat",
      pid7 == 3 ~ "Lean Democrat",
      pid7 == 4 ~ "Independent",
      pid7 == 5 ~ "Lean Republican",
      pid7 == 6 ~ "Republican",
      pid7 == 7 ~ "Strong Republican"
    ),
    PARTY7          = factor(
      PARTY7,
      levels = c(
        "Strong Democrat",
        "Democrat","Lean Democrat",
        "Independent","Lean Republican",
        "Republican","Strong Republican"
      ),
      ordered = TRUE
    )
  )
```

Here, I use `mutate()` and `case_when()` to recode each level of the party ID variable. This makes the `PARTY7` variable a character variable. To make it a factor, I can not simply use `as.factor()` since R would generate a factor variable with the levels in alphabetical order. Therefore, I manually use the `factor()` command to generate my variable.

A Simple Line Graph

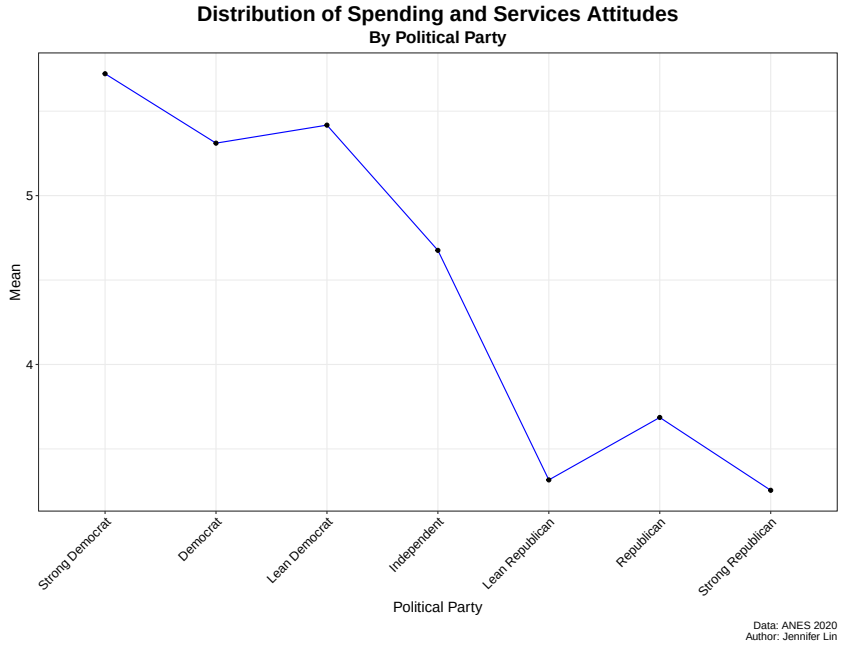
Now that we created this factor variable, we can generate a summary statistics table for the line graph as follows.

```
Spending_Line <- Spending_Line %>%
  group_by(PARTY7) %>%
  summarise(
    mean = mean(self_spending, na.rm = TRUE),
    .groups = 'keep'
  ) %>%
  filter(!is.na(PARTY7))
```

PARTY7	mean
Strong Democrat	5.721714
Democrat	5.310705
Lean Democrat	5.417421
Independent	4.675900
Lean Republican	3.316340
Republican	3.686303
Strong Republican	3.255047

Now that we have the data frame for the plot, we can proceed to make the graph

```
ggplot(Spending_Line, aes(x = PARTY7, y = mean, group = 1))+
  geom_line(color = "blue")+
  geom_point(color = "black")+
  labs(
    title = "Distribution of Spending and Services Attitudes",
    subtitle = "By Political Party",
    caption = "Data: ANES 2020
    Author: Jennifer Lin"
  )+
  xlab("Political Party")+
  ylab("Mean")+
  theme_rworkshop()+
  theme(
    axis.text.x     = element_text(angle = 45, hjust = 1)
  )
```



The graph looks nice! Now, let's break down the various components of the code to generate this graph.

```
ggplot(Spending_Line, aes(x = PARTY7, y = mean, group = 1))+
```

We are using the descriptive stats table that we created for the line plot with partisanship as a factor variable on the x-axis and the mean score per party ID level as the y-axis. The key in this line that deviates from prior examples is the `group = 1` argument. For a line graph, especially those with factor variables on the x-axis, groups need to be specified manually. This can be with a variable (as I will demonstrate later) or, in this case, 1 signals to R to string all the points together.

```
geom_line(color = "blue")+
geom_point(color = "black")+
```

These two lines specify the graph as a line graph with blue lines and black points.

```
labs(
  title = "Distribution of Spending and \nServices Attitudes",
  subtitle = "By Political Party",
  caption = "Data: ANES 2020
  Author: Jennifer Lin"
)+
xlab("Political Party")+
ylab("Mean")+
```

Collectively, these lines of code specify the title, subtitle, caption, x-axis labels and y-axis labels.

```
theme_rworkshop()+
theme(
  axis.text.x     = element_text(angle = 45, hjust = 1)
)
```

Finally, we implement the theme, both using the function we defined at the top of the Workshop but also with an override. For this graph, it seems appropriate to tilt the x-axis labels a bit so that they are not on top of one another. However, since we did not make these presets in the function, we add them in a separate `theme()` function that overrides anything we did to the x-axis text in the function that defined `theme_rworkshop()`.

Line Plots with Facets

In this next example, I consider the suite of survey items that ask respondents to place themselves on a 1-7 scale on major political issues. To start, I will recreate the same factor variable out of the numeric party ID variable[8]

[8] I am only repeating this because I had not previously saved this to the cleaned version of the dataset. If you are cleaning your own data, this step should be included in the original cleaning of the data. In other words, this repetition is for teaching purposes only.

```
Personal_Placement <- ANES %>%
  mutate(
    PARTY7          = case_when(
      pid7 == 1 ~ "Strong Democrat",
      pid7 == 2 ~ "Democrat",
      pid7 == 3 ~ "Lean Democrat",
      pid7 == 4 ~ "Independent",
      pid7 == 5 ~ "Lean Republican",
      pid7 == 6 ~ "Republican",
      pid7 == 7 ~ "Strong Republican"
    ),
    PARTY7          = factor(
      PARTY7,
      levels = c(
        "Strong Democrat",
        "Democrat","Lean Democrat",
        "Independent","Lean Republican",
        "Republican","Strong Republican"
      ),
      ordered = TRUE
    )
  )
```

For this example, I am using the entire set of questions, so I generate summary statistics for each item.

```
Personal_Placement <- Personal_Placement %>%
    group_by(PARTY7) %>%
  summarise(
    spending   = mean(self_spending, na.rm = TRUE),
    defense    = mean(self_defense, na.rm = TRUE),
    medical    = mean(self_medical, na.rm = TRUE),
    job        = mean(self_job, na.rm = TRUE),
    blacks     = mean(self_blacks, na.rm = TRUE),
    business   = mean(self_business, na.rm = TRUE),
    .groups    = 'keep'
  ) %>%
  filter(!is.na(PARTY7))
```

PARTY7	spending	defense	medical	job	blacks	business
Strong Democrat	5.721714	3.643570	2.512263	2.889393	2.391257	1.808149
Democrat	5.310705	3.618858	2.767767	3.374680	2.981061	2.240320
Lean Democrat	5.417421	3.287383	2.478652	3.160183	2.648678	1.906321
Independent	4.675900	4.060140	3.491391	4.006658	3.885375	2.964817
Lean Republican	3.316340	4.734177	4.898515	5.349815	4.897114	4.230769
Republican	3.686303	4.757282	4.660787	5.028649	4.743555	3.786020

The outcome from the above code needs to undergo some more data wrangling before we can use it for a line plot. The following code does this. First, `reshape2::melt()` turns the data from wideform to longform, creating two variables titled `variable` and `value` where `variable` is the name of the original variable and `value` is the value that this variable holds. R automatically picks a variable to do the pivot, and in this case, Party was use. For the purposes of graphing, I also employ `mutate()` to create a recoded variable that contains labels that reflect the topic of the question, which can be used for the plot.

```
Personal_Placement <- Personal_Placement %>%
  reshape2::melt() %>%
  mutate(
    question = case_when(
      variable == "spending" ~ "Spending & Services",
      variable == "defense" ~ "Defense Spending",
      variable == "medical" ~ "Government/Private Medical Insurance",
      variable == "job" ~ "Guaranteed job/income",
      variable == "blacks" ~ "Government Assistance to Blacks",
      variable == "business" ~ "Environment-Business Tradeoff"
    )
  )
```

	PARTY7	variable	value	question
1	Strong Democrat	spending	5.72171428571429	Spending & Services
2	Democrat	spending	5.31070496083551	Spending & Services
3	Lean Democrat	spending	5.41742081447964	Spending & Services
4	Independent	spending	4.67590027700831	Spending & Services
5	Lean Republican	spending	3.31633986928105	Spending & Services
6	Republican	spending	3.68630338733431	Spending & Services
7	...	...	...	...
37	Democrat	business	2.24032042723632	Environment-Business Tradeoff
38	Lean Democrat	business	1.90632054176072	Environment-Business Tradeoff
39	Independent	business	2.96481732070365	Environment-Business Tradeoff
40	Lean Republican	business	4.23076923076923	Environment-Business Tradeoff
41	Republican	business	3.78601997146933	Environment-Business Tradeoff
42	Strong Republican	business	4.84501347708895	Environment-Business Tradeoff

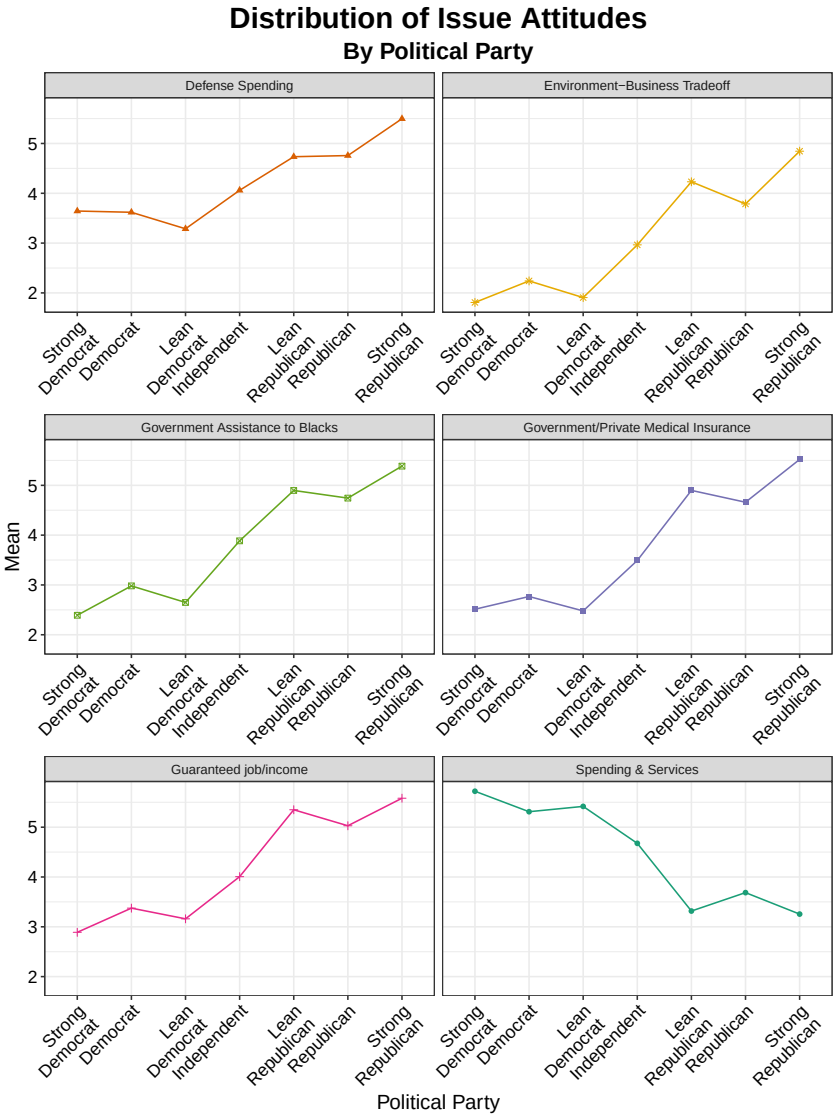
Taking a quick aside, I also want to use this example to show you how you can set axis labels using objects in R. In cases when you are making the same kind of graph for many times, you might want to use this method to streamline your code. The below code creates a vector for party ID labels intended to use for the x-axis. Notice that I also introduced some line break points so that it can produce nice labels without taking up as much apace as the labels did in the previous line plot example.

```
pid7_labels <- c(
  "Strong\nDemocrat", "Democrat",
  "Lean\nDemocrat", "Independent",
  "Lean\nRepublican", "Republican",
  "Strong\nRepublican")
```

Now we are finally ready to make the plot.

```
ggplot(Personal_Placement,
       aes(x = PARTY7, y = value,
           group = variable, color = variable, shape = variable))+
  geom_line()+
  geom_point()+
  facet_wrap(~question, scales = "free_x", ncol = 2)+
  scale_color_brewer(palette = 'Dark2')+
  scale_x_discrete(labels = pid7_labels)+
  labs(
    title = "Distribution of Issue Attitudes",
    subtitle = "By Political Party",
    caption = "Data: ANES 2020
    Author: Jennifer Lin"
```

```
)+
xlab("Political Party")+
ylab("Mean")+
theme_rworkshop()+
theme(
  axis.text.x     = element_text(angle = 45, hjust = 1),
  legend.position = "none"
)
```



There are a few notable deviations in this plot, code-wise, compared to the previous line graph. So I will walk through them below.

```
ggplot(Personal_Placement, aes(x = PARTY7, y = value,
  group = variable, color = variable, shape = variable))+
```

In the data layer, we use the `Personal_Placement` summary statistics data table with the factor version of the party ID variable as the x-axis and the mean as the y-axis. We are grouping by the question name variable and coloring by this same variable. This is to tell R to string points from the same question name together and to color the lines uniquely. In addition, I use the `shape=` argument to give each question a unique point shape.

```
geom_line()+
geom_point()+
```

These create the plots.

```
facet_wrap(~question, scales = "free_x")+
```

Here, we use the `facet_wrap()` function to make graph sections with each question. The question label was one that I generated as a character variable. R does not care whether you supply a character variable or factor variable into `facet*()`. The little `~` denotes the variable that you want to facet by. The `scales=` argument allows you to specify how you want the axes to appear. In this case, `free_x` denotes the need for each subplot to have its own x-axis[9].

[9] Note: For the purposes of handout display, I also added a `ncol = 2` argument in the handout version to tell R to wrap the figure such that there are 2 columns in the graph.

```
scale_color_brewer(palette = 'Dark2')+
scale_x_discrete(labels = pid7_labels)+
```

Here I am using the "Dark2" color palette to color the lines and the points[10]. Under `scale_x_discrete`, we implement the `pid7_labels` that we created earlier. This is a custom vector that we use to implement labels. This effectively prevents the need to retype the various levels that are included in the variable. Bear in mind that while a 7-point scale might seem easy to carry over and manually type, often, you might face larger variables with more levels. At that point, something like this will be more convienent.

[10] Recall that you use fill when there is a space to fill and color when it is a point or line to color. Since there is no space to fill on a line plot, color changes for points and lines can be addressed using `color`.

```
labs(
  title = "Distribution of Issue Attitudes",
  subtitle = "By Political Party",
  caption = "Data: ANES 2020
  Author: Jennifer Lin"
)+
xlab("Political Party")+
ylab("Mean")+
theme_rworkshop()+
theme(
  axis.text.x     = element_text(angle = 45, hjust = 1),
  legend.position = "none"
)
```

We close out with the labels for titles, subtitles and so forth, and the theme. Notice that this time, I added yet another edit to the `theme_rworkshop()` function such that we are deleting the legend, when we previously said to move it to the bottom of the plot. As such, the plot currently has no legend.

Exercise 3

From your variables in Exercise 1, generate a line graph with appropriate facets with any combination of the three variables such that it tells a story about the relationship of the variables. Include proper labels, titles, colors and themes