week7: Exercise 3

Hakan Mehmetcik

```
# loaded packages
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.0     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

```
library(here)
```

```
here() starts at /Users/kobain/Library/CloudStorage/OneDrive-marmara.edu.tr/mac_projects 2/da
```

```
dpath <- "Lectures/week7_exercises"
```

Exercise 3: Data Visualization

Let's get the Data once again!

```
simd2020 <- read_csv(here(dpath, "simd2020.csv"), na="*")
```

Now, we can turn on examining the basics of doing a variety of data visualizations using tidyverse and more specifically the ggplot2 package. Although the ggplot2 functions will be the focus, we will utilize other packages and functions from tidyverse. Hence, this topic ties all the tidyverse materials together. Note: ggplot2 code tends to be extensive, but it follows a step-by-step approach.

ggplot: The Basics

Before we get into the nitty-gritty of producing various types of visualizations, it is necessary to discuss the basics of the ggplot2 package. The general concept of ggplot2 is to think of plots as a combination of layers. We start with a layer for our x and y axes, then we add a layer for the type of plot we want, and then we add a layer for the plot labels, etc. To get a basic x-y axis plot layer we use the ggplot() function where in parentheses we just leave it empty.

```
# library(ggplot2)
ggplot()
```


Next, we can add another layer to the base plot so that each part of the plot is in fact a distinct layer. To add the next layer, we simply include the + symbol after the parentheses. We specify the type of plot we want by using the geom functions, which simply stands for 'geometric object'. For example, we can use the functions geom_bar() for a bar plot, geom_histogram() for a

histogram, etc. There are numerous geoms available and we will use the most relevant ones for our needs; to see all of the options just Google it. Next, we use the mapping= argument to specify the x and y axes variables. The specific x and y variables are always inside the aes() argument, where aes stands for aesthetics and is the key code for telling R what is being done in the plot. (Note: we don't always need to specify mapping and aes(), but it is good practice.)

Bar Plot

We use bar plots to visually represent nominal- and ordinal-level categorical variables. Bar plots have spaces between the bars which indicate that the variable is not continuous and may or may not be ordered.

We will use data from the 2019 survey about perceptions of voter fraud in England. The name of the data file is VF England.csv and we will read it in using the read_csv() function.

```
vf_england <- read_csv(here(dpath, "VF England.csv"))
```

```
Rows: 2034 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (7): vfalter, vfsafe, vfproblem, gender, brexit_vote, vote2017_dum, pid
dbl (3): age, education, nation_weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We will start with a basic bar plot and then build the complexity.

Barplot with One Variable

When we are only looking at a single variable to plot using ggplot(), we do not specify the x and y variable; because by default there is only one variable. Let's create an initial bar plot using the variable vfproblem (the belief that voter fraud is a serious problem). Before we do that we need to re-order the values for vfproblem. Let's save the re-ordered version as vfproblem1.

```
vf_england |>
  count(vfproblem)
```

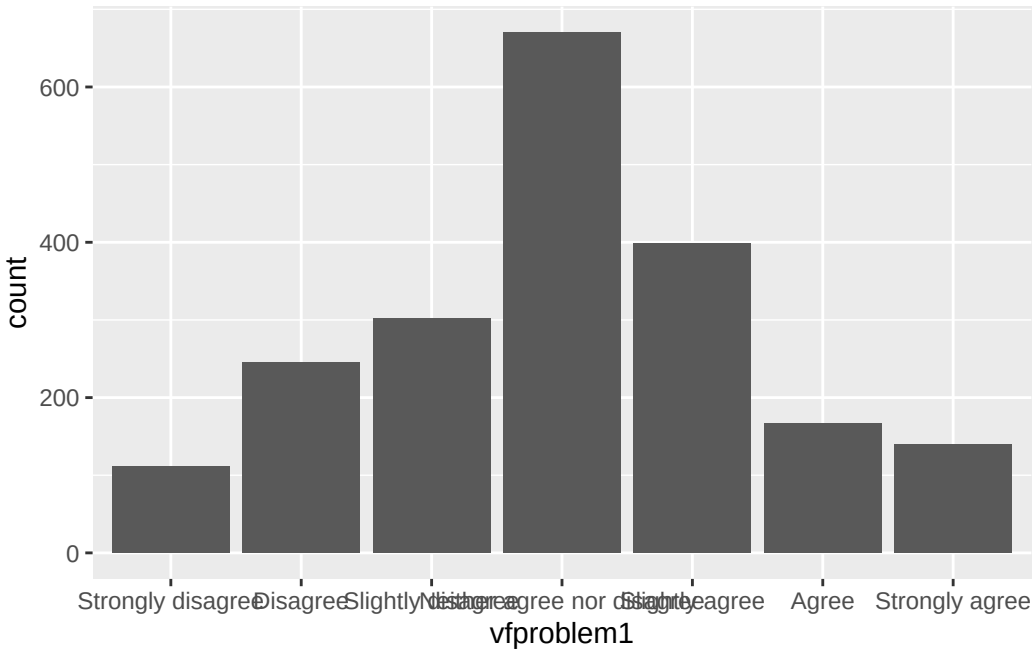
vfproblem	n
Agree	167
Disagree	245
Neither agree nor disagree	670
Slightly agree	398
Slightly disagree	302
Strongly agree	140
Strongly disagree	112

```
vf_england <- vf_england |>
      mutate(vfproblem1 = factor(vfproblem,
               levels=c("Strongly disagree","Disagree",
                      "Slightly disagree","Neither agree nor disagree",
                      "Slightly agree","Agree","Strongly agree")))
vf_england |>
  count(vfproblem1)
```

vfproblem1	n
Strongly disagree	112
Disagree	245
Slightly disagree	302
Neither agree nor disagree	670
Slightly agree	398
Agree	167
Strongly agree	140

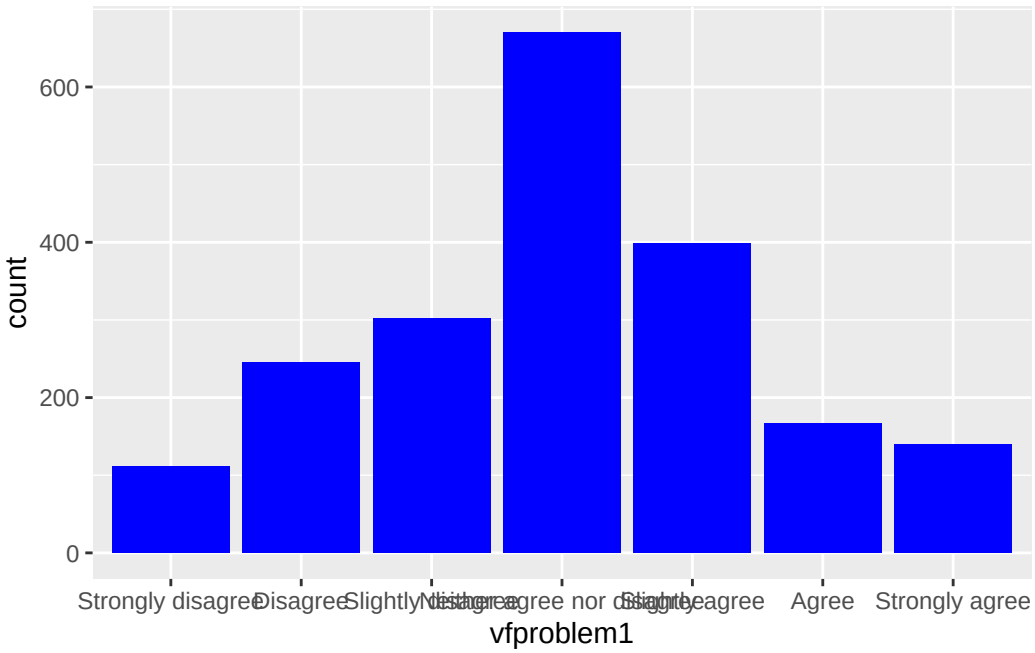
We will use the geom_bar() function to create the bar plot. We first specify the data we are using in the ggplot() function and then add the geom_bar() function. (When we run the code in the standard R Studio environment, where the plots are in the bottom right window, we will likely see that labels for vfproblem1 overlap one another. We could drag the window to make it larger or click on the zoom button. If we are creating RMarkdown documents, we just need to adjust sizing based on the rendered document.)

```
ggplot(data = vf_england) +
  geom_bar(mapping = aes(vfproblem1))
```



If we want to change the color of the bars, we simply include the option fill= in the geom_bar() argument and then whatever color we want to include.

```
ggplot(data = vf_england) +
  geom_bar(mapping = aes(vfproblem1), fill="blue")
```



Barplot with Two Variables

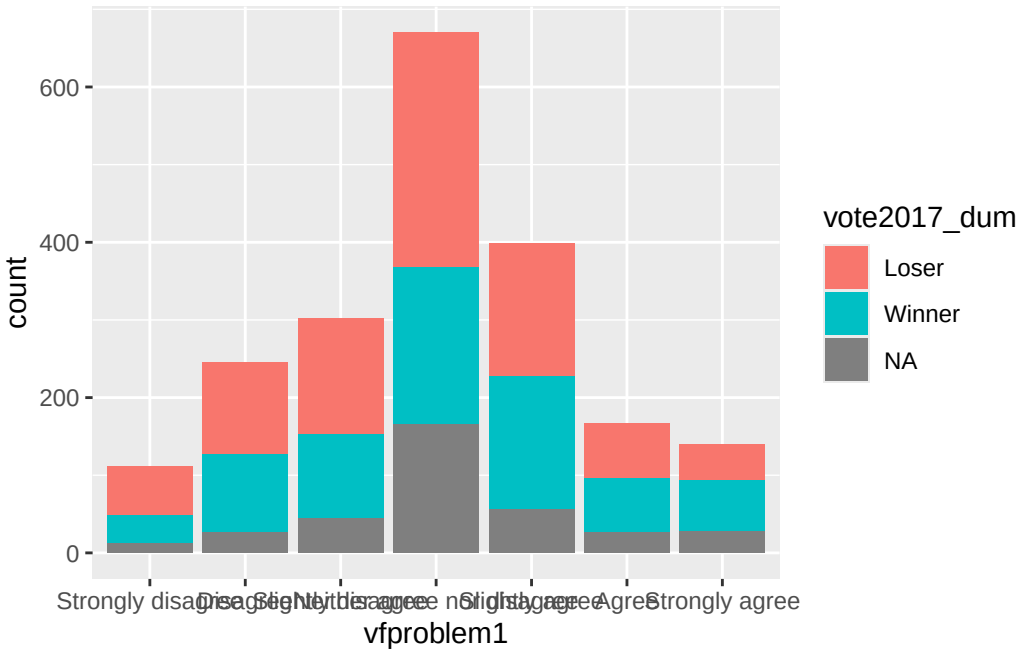
Using ggplot() also makes it easy to include a second variable into the bar plot. We will plot vfproblem1 and whether respondents voted for the winner (Conservative Party) of the 2017 UK Parliamentary elections or another party (vote2017_dum). Let's first look at vote2017_dum.

```
vf_england |>
  count(vote2017_dum)
```

vote2017_dum	n
Loser	919
Winner	757
NA	358

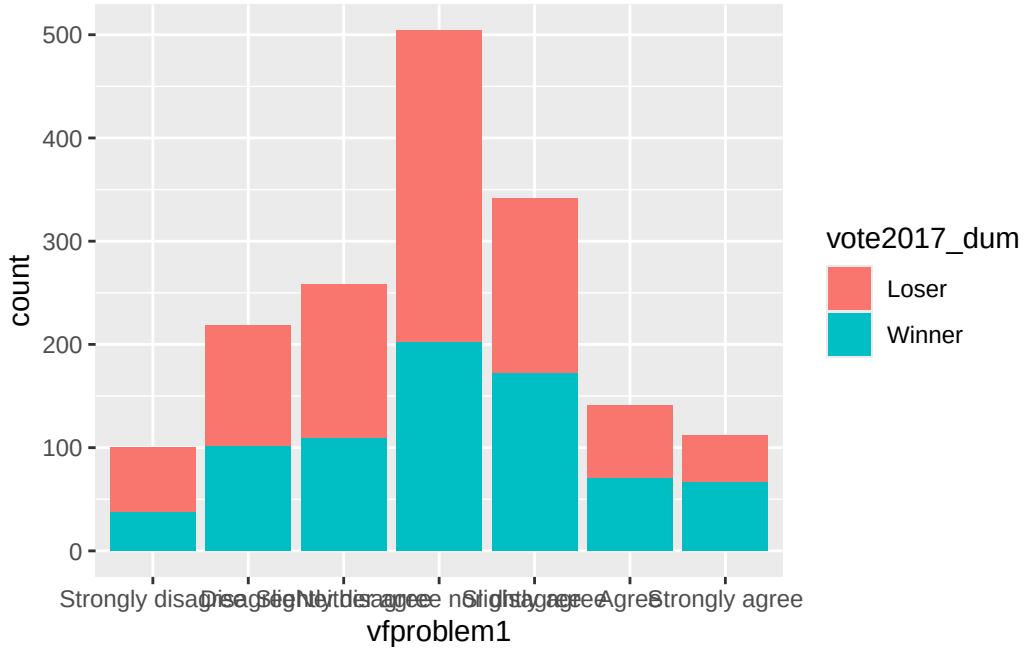
We see there are 2 categories for vote2017_dum and a number of missing values. Let's do a bar plot, where we specify the second variable with fill= within the aes() argument.

```
ggplot(data = vf_england) +
  geom_bar(mapping = aes(vfproblem1, fill=vote2017_dum))
```



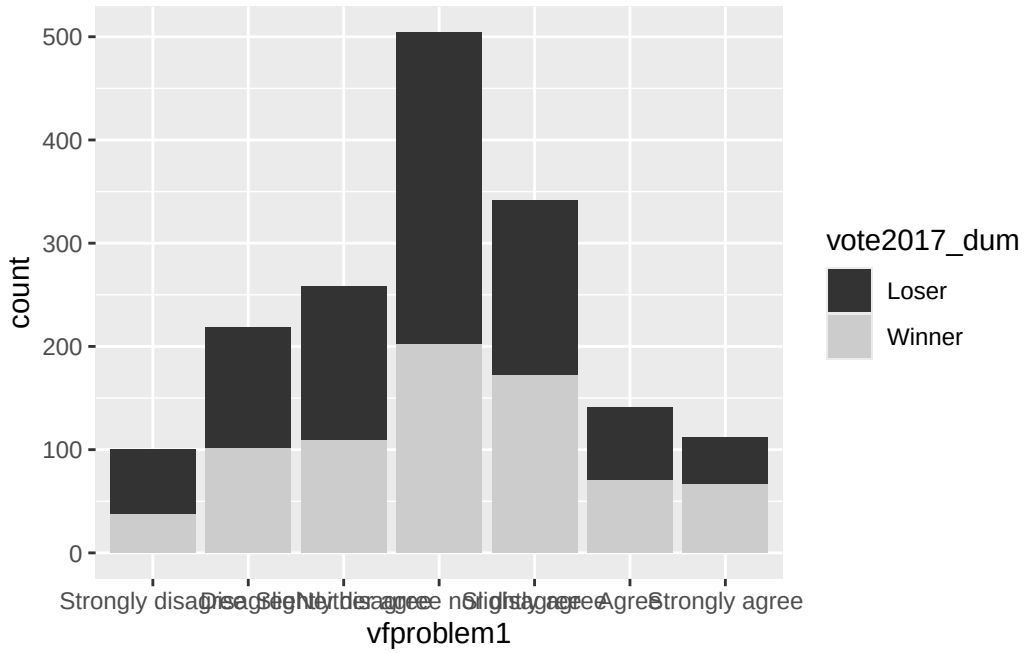
Before we go further, we see that the NAs were plotted, which ggplot2 does by default. Let's remove the missing values with the filter() function prior to the ggplot() specification and connect the code with the pipe operator |>.

```
vf_england |>
  filter(!is.na(vote2017_dum)) |>
ggplot() +
  geom_bar(mapping = aes(vfproblem1, fill=vote2017_dum))
```



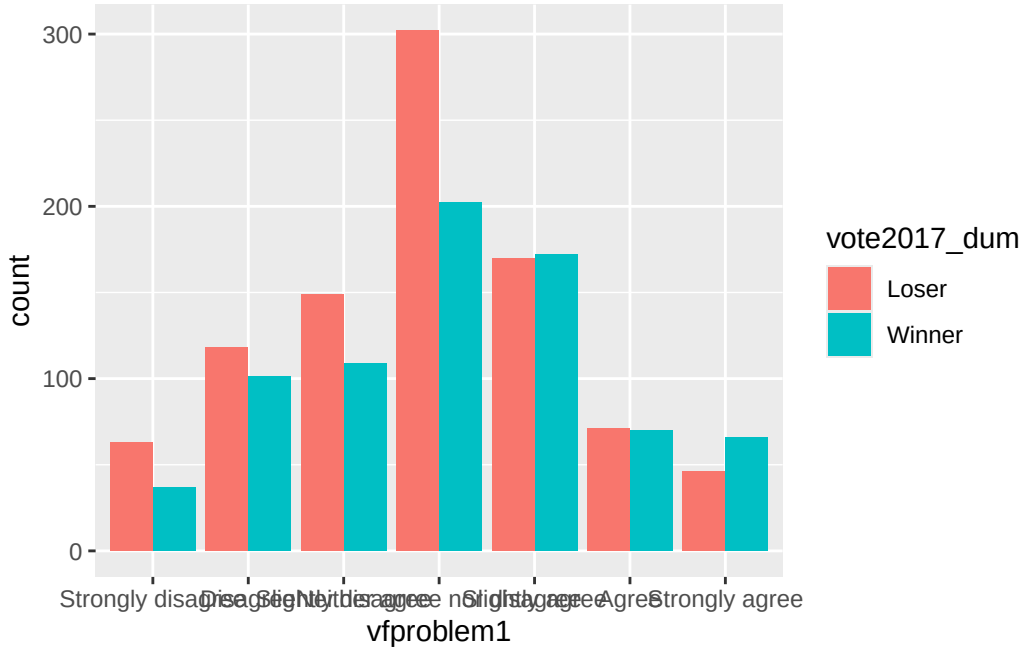
We now see that the NAs have been removed. An important thing to know when interpreting bar plots and histograms created with the default options in ggplot2 is that the values are stacked and not superimposed. We see that respondents who voted for the winning party (again, Conservatives) actually have a higher belief that voter fraud is a serious problem. The colors here are the default colors in ggplot2, which we are able to change. We can use shades of grey instead of colors in the plot. We do this by adding the function scale_fill_grey() to the plotting code.

```
vf_england |>
  filter(!is.na(vote2017_dum)) |>
ggplot() +
  geom_bar(mapping = aes(vfproblem1, fill=vote2017_dum)) +
  scale_fill_grey()
```



Another option we might want to use is to de-stack the colours and put them side-by-side for each x-axis category. We can do this by adding the option position="dodge" to the geom_bar() argument; notice the code is outside the aes() argument.

```
vf_england  |>
  filter(!is.na(vote2017_dum)) |>
ggplot() +
  geom_bar(mapping = aes(vfproblem1, fill=vote2017_dum), position="dodge")
```



This provides the same information as the previous bar plot, but perhaps in an easier to understand format. Let's now do a full-blown version of the bar plot. To make the bar plot presentable, we'll make the following changes:

- To add labels to the plot, we'll use the labs() function.
- To change the font sizes and faces of the labels, we'll use the theme() function. We will also use the theme_minimal() function, which mainly removes the default grey background of the plot.
- We'll also have the value labels on the x-axis slanted, so they fit better.
- To change the y-axis to percentages, we need to use the scales package. We will make use of the scales functions within the ggplot2 functions. This is a bit complicated: in the geom_bar() function, we specify that y=..prop.. for the proportions and we specify that group=vote2017_dum which tells R that all the categories on vfproblem1 for each vote2017_dum category add up to 100%. We also need to use the scale_y_continuous() function to get the percentages put onto the y-axis.
- We will also change the colors using the hexadecimal values for two of Notre Dame's main colors.
- We can also adjust the size of the bar plot in the rendered RMarkdown document by including fig.height=7, fig.width=8 in the code chunk header (i.e., the part that starts the code chunk).

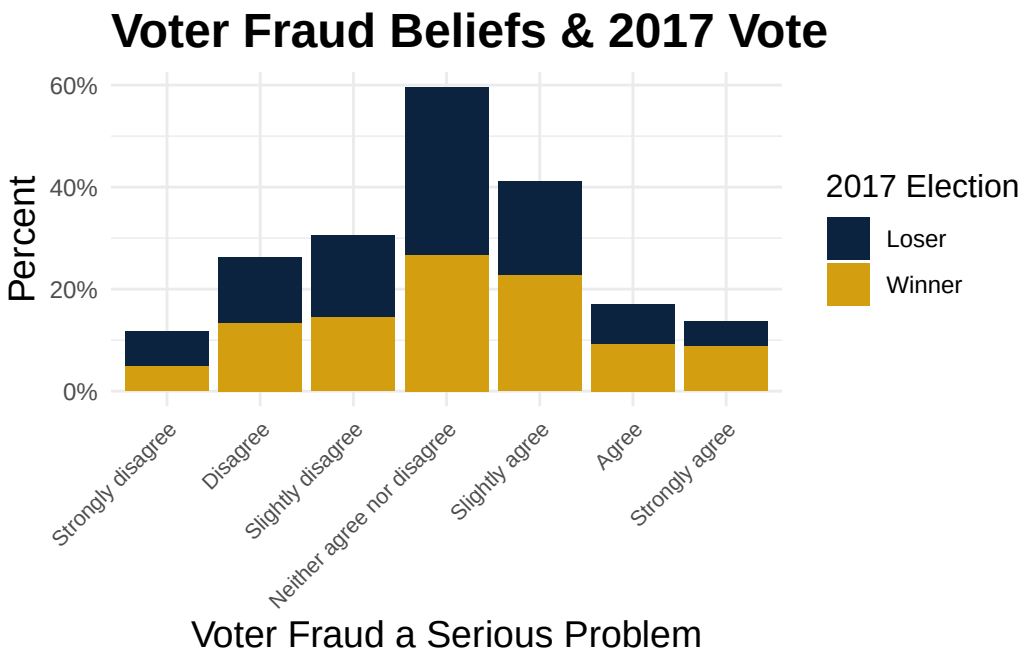
> **Note**
>
> (Note - it often will take several re-dos to get the plot exactly as we want.)

```
library(scales) # install.packages("scales") if you don't have the package
```

```
Attaching package: 'scales'

The following object is masked from 'package:purrr':

    discard

The following object is masked from 'package:readr':

    col_factor
```

```
vf_england %>%
  filter(!is.na(vote2017_dum)) %>%
ggplot() +
  geom_bar(mapping = aes(x=vfproblem1, y=..prop..,group=vote2017_dum,fill=vote2017_dum),
           stat="count") +
theme_minimal() +
scale_fill_manual(values=c("#0c2340", "#d39f10")) +
labs(x="Voter Fraud a Serious Problem", title="Voter Fraud Beliefs & 2017 Vote",
     fill="2017 Election", y="Percent") +
scale_x_discrete(guide = guide_axis(angle = 45)) +
scale_y_continuous(labels = percent_format()) +
theme(
  plot.title = element_text(size=18, face="bold"),
  axis.title.x = element_text(size=14),
  axis.title.y = element_text(size=14),
  legend.title = element_text(size=12),
  axis.text.x.bottom = element_text(size=8)
)
```

```
Warning: The dot-dot notation (`..prop..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(prop)` instead.
```



Again, this is a lot of code, but it shouldn't be overly complicated. Now that we've gone through a full-blown version of a bar plot in ggplot2, you should have a basic understanding for creating any plot using ggplot2.

Histogram

We use histograms for plotting interval-/ratio-level variables. One key aesthetic choice with histograms is choosing the binwidth. The binwidth tells R how wide we want our bars to be. The height of the bar is dependent on the number of observations that fall in that bin. Below we will explore the aesthetic effects of changing the binwidth.

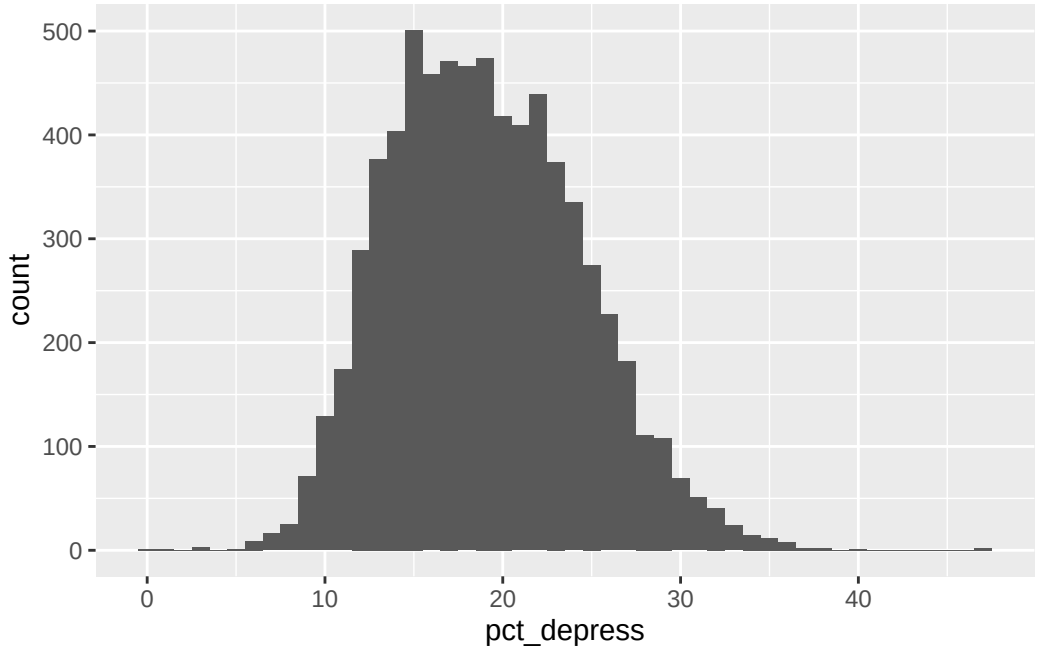
Histograms Using One Variable

We will create a histogram using the percentage of psychiatric prescriptions variable from the simd2020 data. We will use the geom_histogram() function and set the binwidth, but the rest of the code is similar to what we did with bar plots. As before, we need to filter out missing values. We need to first read-in the data and then we'll do some manipulation.

```
simd <- read_csv(here(dpath, "simd2020.csv"), na="*")
```

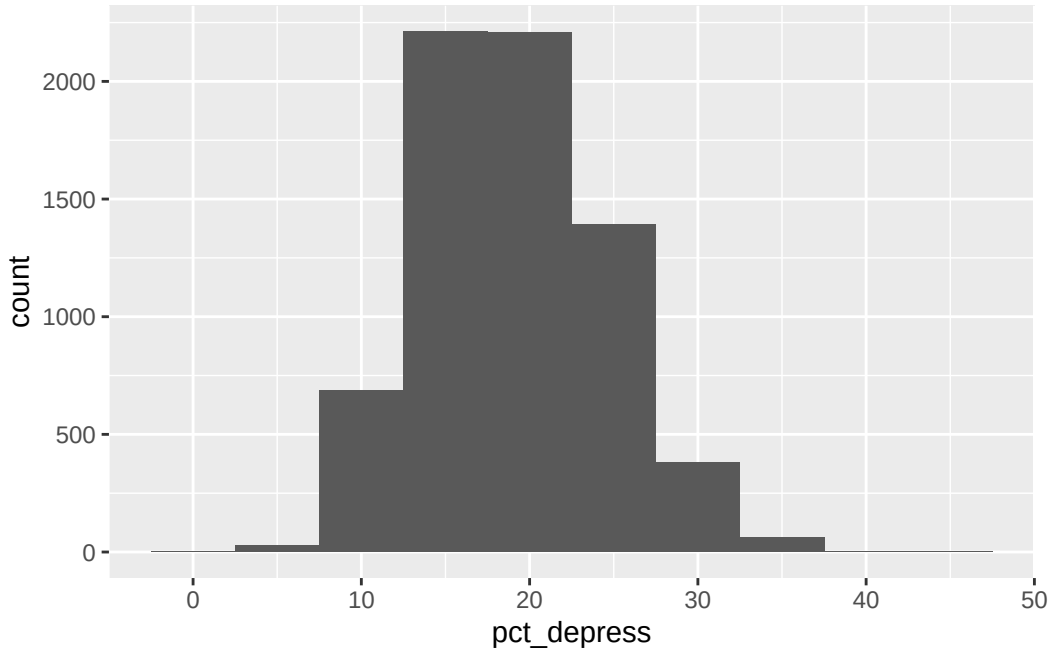
```
Rows: 6976 Columns: 38
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (3): Data_Zone, Intermediate_Zone, Council_area
dbl (35): Total_population, Working_age_population, Income_rate, Income_coun...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
simd <- simd |>
          rename_with(tolower) |>
          mutate(pct_depress = depress*100) |>
          mutate(urban_fct = recode(urban, `1`="Urban", `0`="Rural"))
simd |>
  filter(!is.na(pct_depress)) |>
ggplot() +
  geom_histogram(mapping = aes(pct_depress), binwidth=1)
```



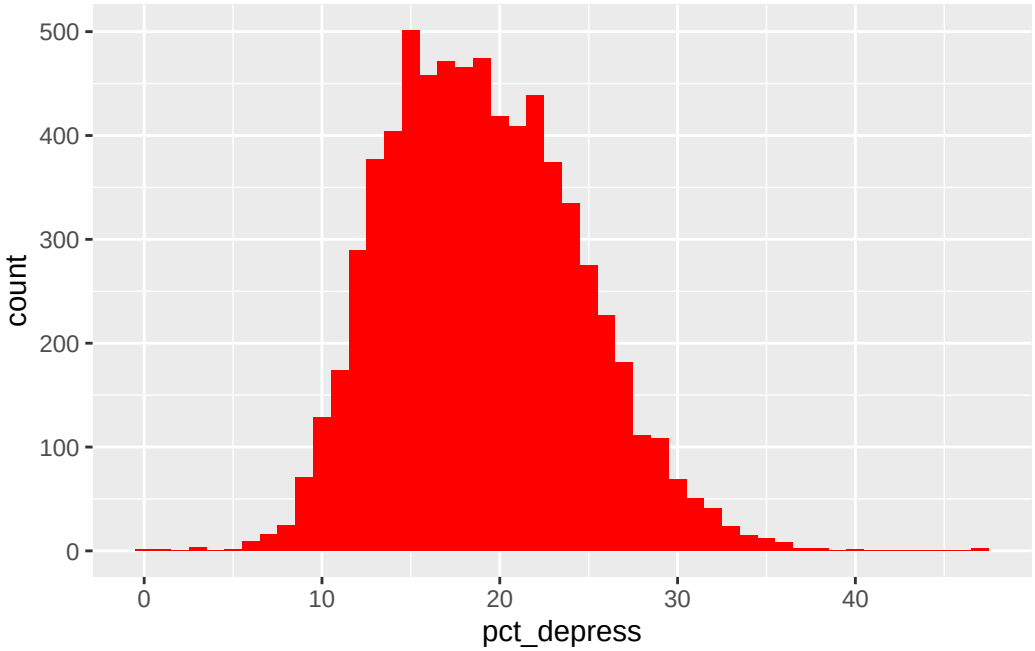
We see that the values are clustered around 15-20% and somewhat normally distributed. For the most basic, we can change the binwidths by simply changing the number in the geom_histogram() function.

```r
simd |>
  filter(!is.na(pct_depress)) |>
ggplot() +
  geom_histogram(mapping = aes(pct_depress), binwidth=5)
```



We can also fill the bins with a color using the option fill= in the geom_histogram() argument.

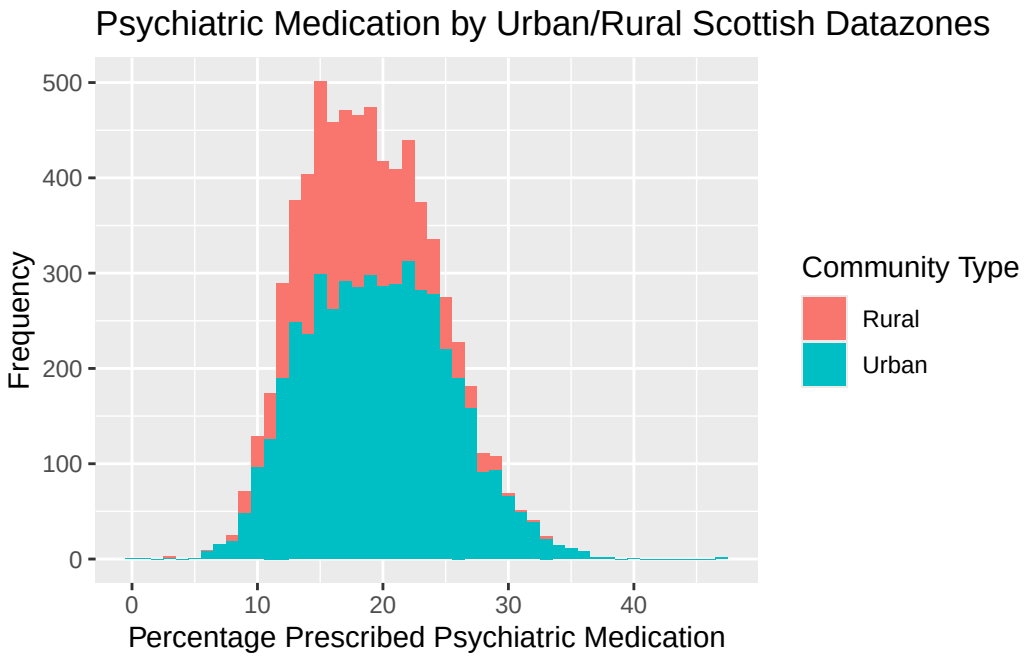
```r
simd |>
  filter(!is.na(pct_depress)) |>
ggplot() +
  geom_histogram(mapping = aes(pct_depress), binwidth=1, fill="red")
```



Histograms Using Two Variables

We frequently will want to plot two variables in a histogram. As with the single-variable histogram, we need an interval- or ratio-level variable, but the second variable is often nominal or ordinal. We'll use pct_depress and the urban_fct variable as our second variable. Hence, we are looking to see if urban datazones have different percentages of prescribed psychiatric medication than rural datazones? We include fill=urban_fct into the aes() argument to plot the second variable.

```
simd |>
  filter(!is.na(pct_depress) & !is.na(urban_fct)) |>
ggplot() +
  geom_histogram(mapping = aes(pct_depress,fill=urban_fct),
                 binwidth=1) +
  labs(x="Percentage Prescribed Psychiatric Medication",
       y="Frequency",
       title="Psychiatric Medication by Urban/Rural Scottish Datazones",
       fill="Community Type")
```



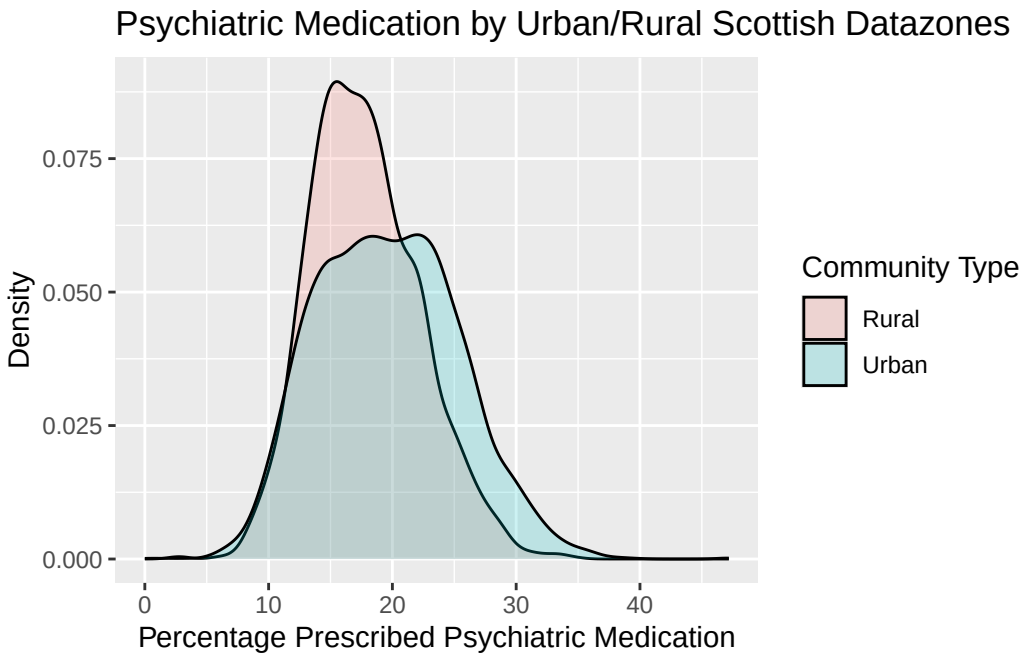
Each bar in the histogram represents the total number of observations, while the colors represent how many observations are rural or urban; the colors are stacked and not superimposed. We can see from this that there are slightly higher prescription percentages in urban areas than in rural areas.

Density Plots

Instead of using the histogram with stacked values, we can create a histogram using the smoothed density. Technically called a kernel density plot, these plots illustrate the smoothed distribution of continuous variables instead of grouping variables into specific bins for a histogram. Let's again look at pct_depress and urban_fct. To do so, we just need to change geom_histogram() to geom_density(), and we remove the binwidth argument because there are now no bars. To make the plot clearer, we can decrease the darkness of the color shading using the option alpha= in the geom_density() argument. An alpha of 1 is the same as the default point shading, while anything less than 1 is lighter. The smaller the alpha value, the lighter the points. Below we set alpha=1/5, change the default colors, and add labels.

```
simd |>
  filter(!is.na(pct_depress) & !is.na(urban_fct)) |>
ggplot() +
  geom_density(mapping = aes(pct_depress, fill=urban_fct), alpha=1/5) +
```

```
labs(x="Percentage Prescribed Psychiatric Medication",y="Density",
     title="Psychiatric Medication by Urban/Rural Scottish Datazones",
     fill="Community Type")
```



Now we can clearly make out both urban and rural datazones' densities. Again, we see that there are slightly higher prescription percentages in urban areas than in rural areas.

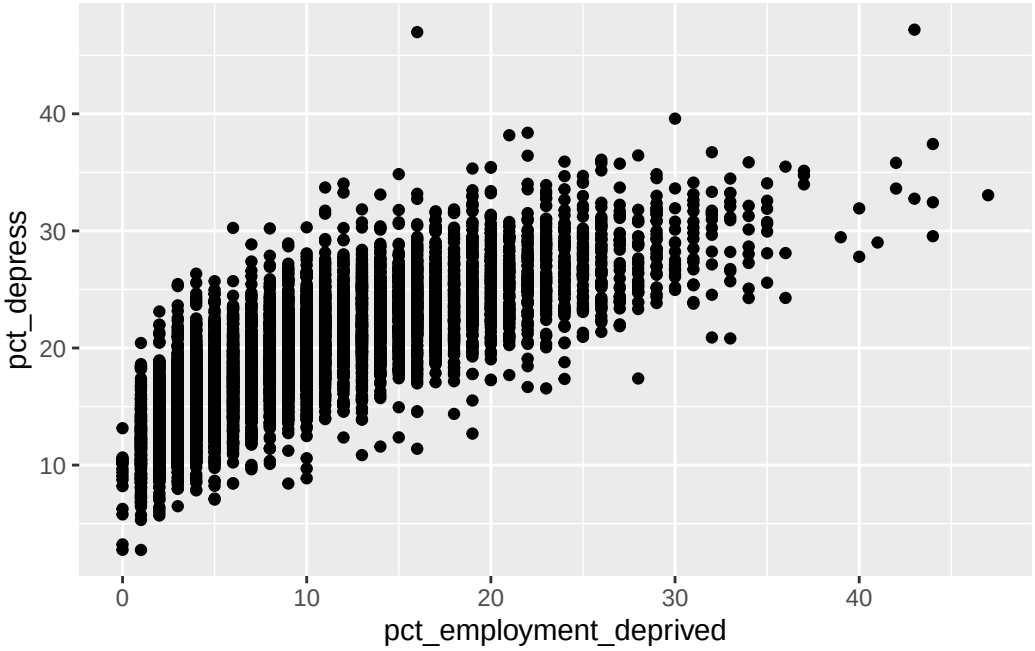
Scatterplots

If we have interval- and ratio-level x and y variables, scatterplots are the preferred visualization technique. In this section, we will do a series of scatterplots using ggplot. We will again use the 2020 Scottish Index of Multiple Deprivation data that we looked at previously. We will use the percentage of psychiatric prescriptions by datazone (pct_depress) as our y-axis variable and percentage of people in a datazone that are employment deprived (pct_employment_deprived) on the x-axis. The pct_employment_deprived variable measures the percentage of people in a datazone receiving incapacity benefits, employment and support allowance, or severe disablement allowance; essentially financial support from the government due to being unable to work.

Scatterplots with Two Variables

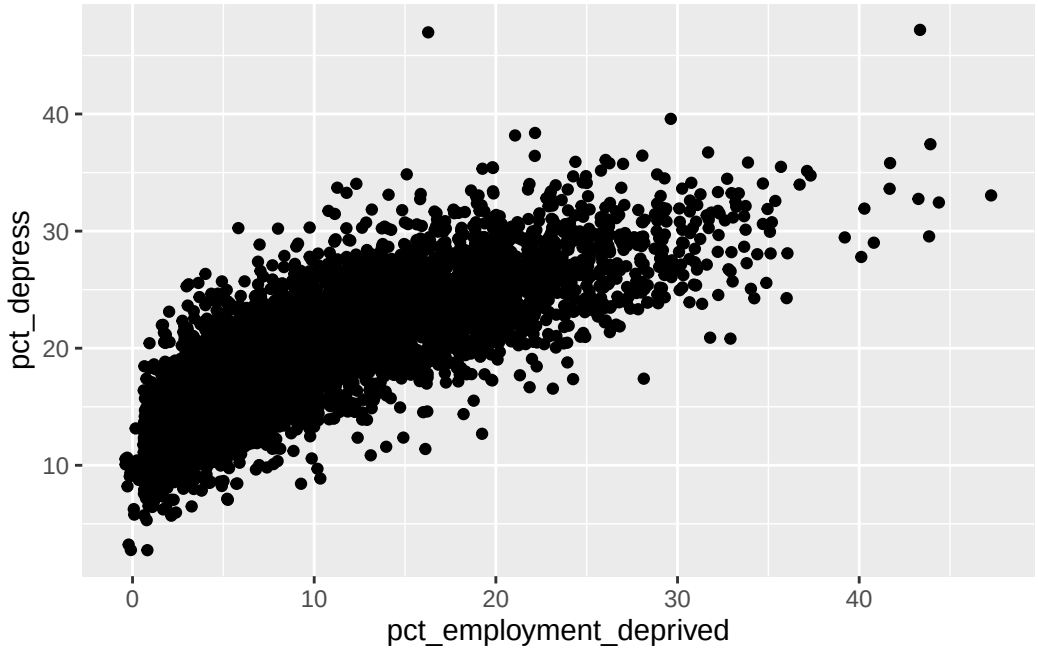
We will start with a basic scatterplot using ggplot(). Again, we first tell R that we are creating a plot using the ggplot() function and then we use the geom_point() function to create the scatterplot. We also need to specify what the x and y variables are in the aes() argument. Both pct_depress and pct_employment_deprived have missing values that we need to filter out. Instead of creating and saving the variable pct_employment_deprived, let's include it before the ggplot() code by using piping.

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,y=pct_depress))
```



In the scatterplot, there are some observations that are on top of each other. When this occurs it can be difficult to ascertain the number of observations in certain points of the plot. To make the points clearer, we do something called 'jittering' that moves the data points a bit so they are not overlapping and can be seen on a plot. We need to just include position="jitter" into the geom_point() argument.simd |>.

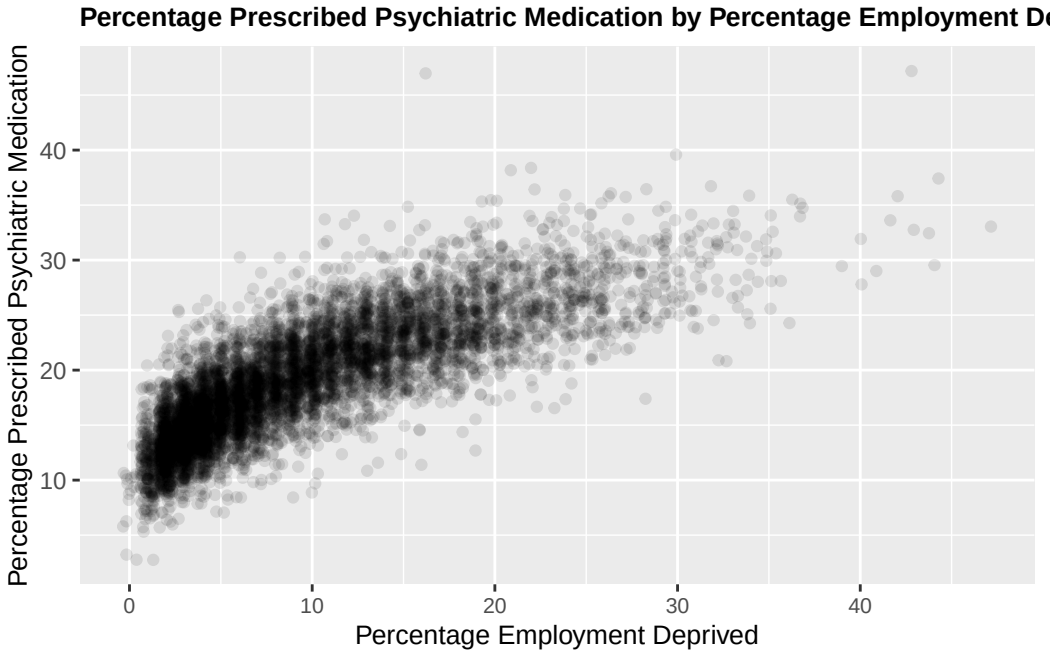
```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,y=pct_depress), position="jitter")
```



We can see that some of the points are just slightly moved, but it reveals a better representation of our data points. We will continue to include the jitter option for the rest of the scatterplots below. We can additionally lighten the points by including the alpha= option. We'll also use the labs() and theme() functions.

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
                         y=pct_depress),
             position="jitter", alpha=1/10) +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
```

```
theme(
  plot.title = element_text(size=10, face="bold"),
  plot.subtitle = element_text(size=10),
  axis.title.x = element_text(size=10),
  axis.title.y = element_text(size=10),
  legend.title = element_text(size=10),
  axis.text.x.bottom = element_text(size=8)
  )
```

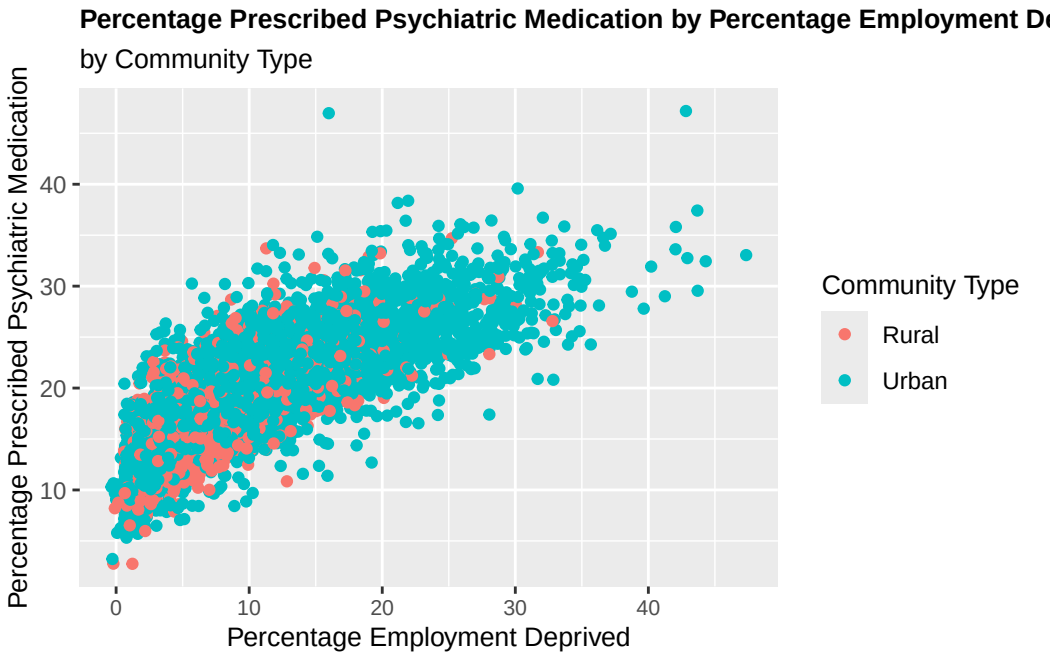


The darker shaded points represent where there are points on top of each other - in other words, points that have the same x and y values. It is now clear that the bulk of datazones have low percentages of employment deprivation and psychiatric prescription percentages between 10 and 20%.

Scatterplots with Three Variables

We can also include a third variable in scatterplots. The third variable will show up as differences in the points plotted. To show the third variable using colors, we add the color= option in the aes() argument. We will use urban_fct as the third variable.

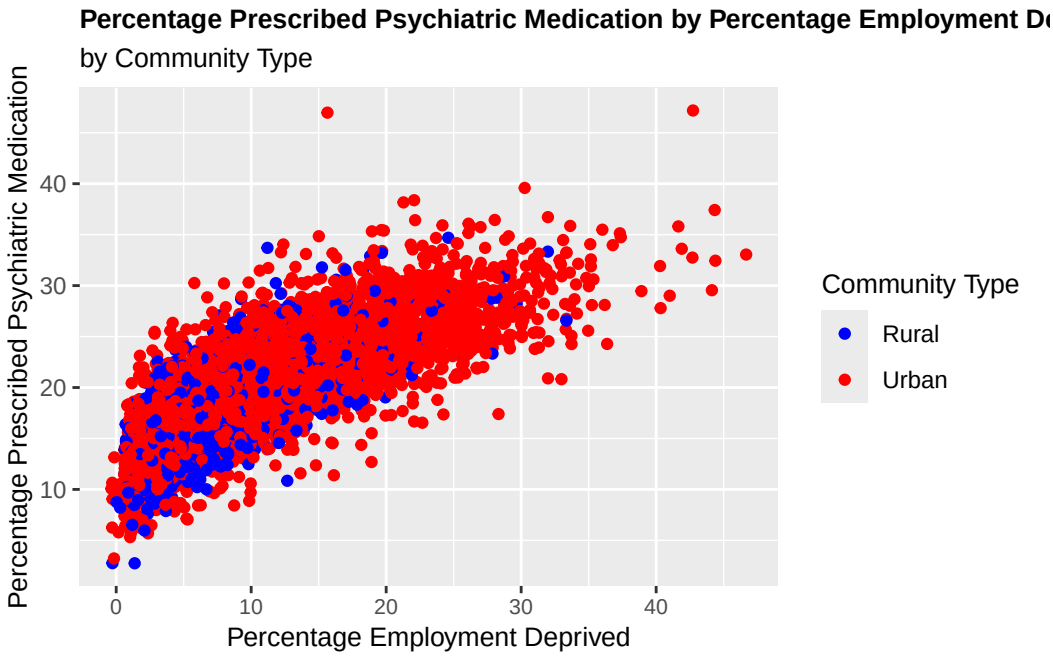
```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,y=pct_depress, color=urban_fct),
             position="jitter") +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type",color="Community Type") +
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
    legend.title = element_text(size=10),
    axis.text.x.bottom = element_text(size=8)
)
```



What do you see with urban/rural as the third variable? For the most part, urban and rural datazones have similar levels of employment deprivation and psychiatric prescriptions. However, it appears that datazones with the highest levels of employment deprivation and

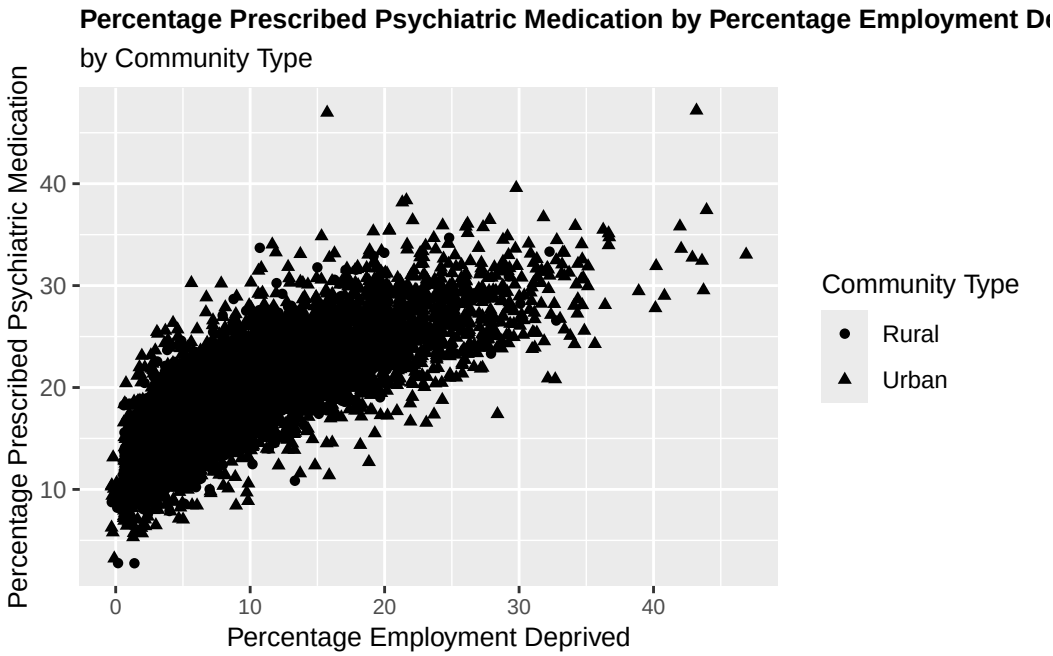
psychiatric prescriptions are all urban datazones. We can control the colors by using the scale_color_manual() function and specifying the values as the colors. We need to make sure we specify the same number of colors as categories in our third variable. Each color we specify links to a specific value in the third variable. For example, the first color (e.g., 'blue') will be for the first value of the third variable (e.g., 'Rural').

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,y=pct_depress, color=urban_fct),
             position="jitter") +
  scale_color_manual(values = c("blue", "red")) +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type",color="Community Type") +
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
    legend.title = element_text(size=10),
    axis.text.x.bottom = element_text(size=8)
)
```



We might choose to represent the third variable using shapes instead of colors. To do so, we include the option shape= into the aes() argument instead of the color= option.

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,y=pct_depress, shape=urban_fct),
             position="jitter") +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type",shape="Community Type") +
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
    legend.title = element_text(size=10),
    axis.text.x.bottom = element_text(size=8)
    )
```



One issue with the shapes option in ggplot() is that we are limited to only six types of shapes. So, using the default settings, our third variable can have at most six categories. We can manually add more than six shapes by adding the scale_shape_manual() function to the ggplot() code. However, the developers of ggplot2 contend that using more than six shapes can make it difficult to discern individual shapes in the scatterplot.

Scatterplots with Four Variables

We can include a fourth variable by specifying one variable as different shapes and the other shown as different colors. We'll use the variable university for our fourth variable, which is the portion of 17-21 year olds in a datazone going to university. Let's first take a quick look at university.

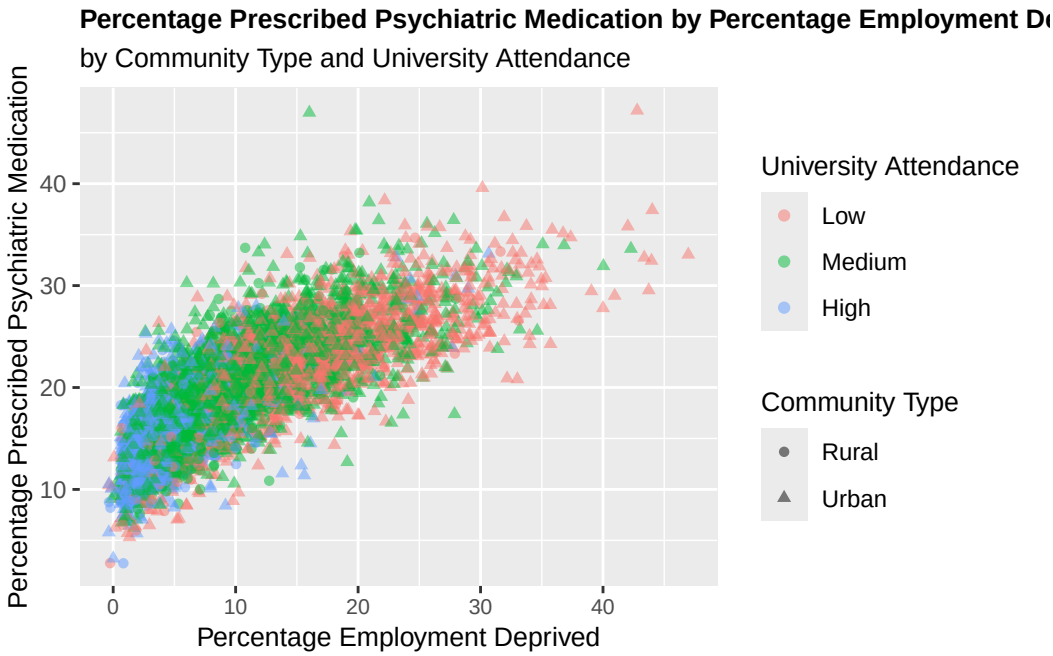
```
summary(simd$university)
```

Min. 1st Qu. Median Mean 3rd Qu. Max. NA's 0.00000 0.04688 0.07843 0.09188 0.12136 0.81907 2

We see that a majority of datazones have a fairly low portion of 17-21 year olds going to university as the median is .08 and the 3rd quartile is .12. (Note: university [0, 1].) First, we'll create and use a collapsed version of university that has 3 categories, roughly based on

the quartile values. Like pct_employment_deprived, we'll create this variable prior to the ggplot() code through piping.

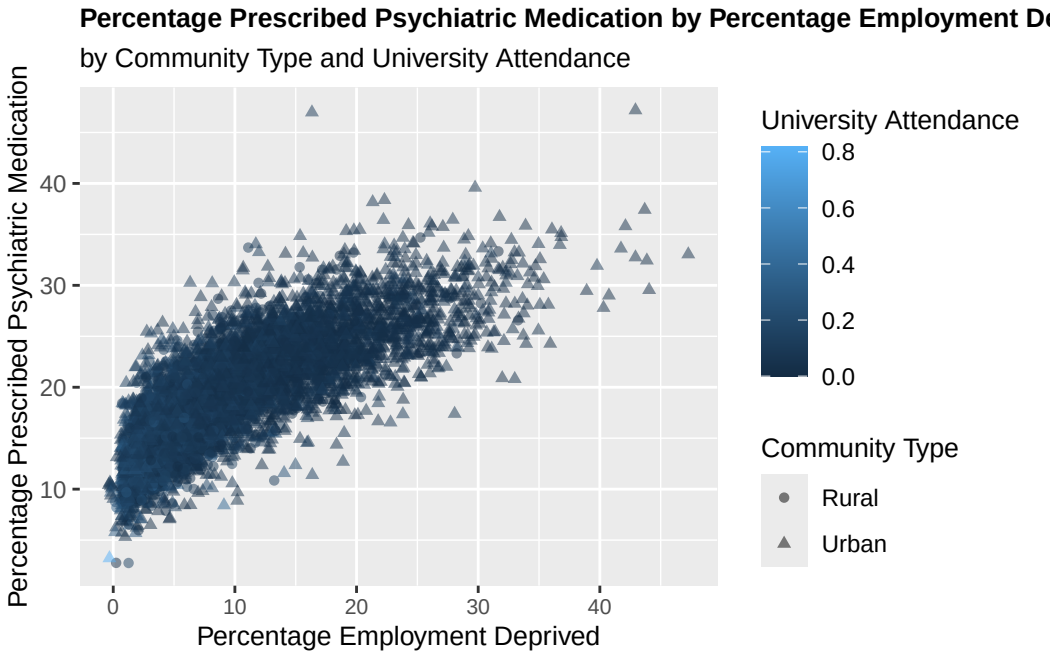
```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct) &
           !is.na(university)) |>
  mutate(uni_fct = case_when(
    university <= .05 ~ "Low",
    university > .05 & university <= .12 ~ "Medium",
    university > .12 ~ "High")) %>%
  mutate(uni_fct = factor(uni_fct,
                          levels=c("Low","Medium","High"))) %>%
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
             y=pct_depress, shape=urban_fct,
             color=uni_fct),
             position="jitter",
             alpha=1/2) +
  labs(x="Percentage Employment Deprived",
       y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type and University Attendance",
    shape="Community Type", color="University Attendance") +
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
    legend.title = element_text(size=10),
    axis.text.x.bottom = element_text(size=8)
    )
```



Now, let's do a version where we use the original numeric university variable.

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct) &
           !is.na(university)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
             y=pct_depress, shape=urban_fct,
             color=university),
             position="jitter",
             alpha=1/2) +
  labs(x="Percentage Employment Deprived",
       y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type and University Attendance",
    shape="Community Type", color="University Attendance") +
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
```

```
  legend.title = element_text(size=10),
  axis.text.x.bottom = element_text(size=8)
  )
```

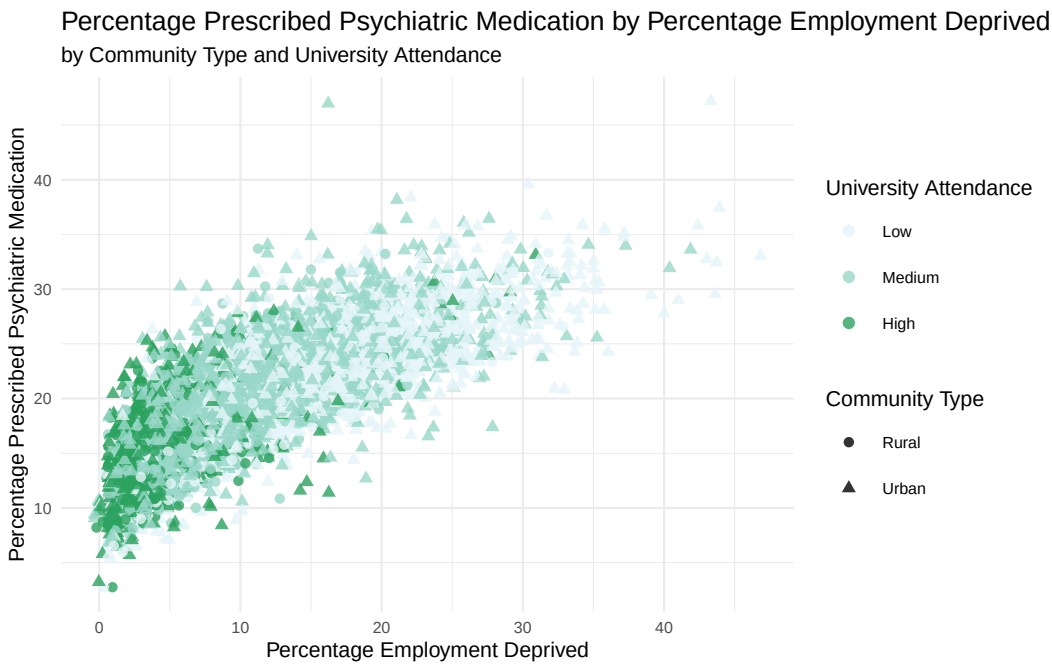


We see that with the numeric version of university we get a blue color scale instead of individual colors. The color scale is the default representation for numeric variables in ggplot.

Color Considerations

Although we have our own aesthetic color preferences when creating data visualizations, some people have difficult seeing differences in colors or are color-blind. This is particularly the case with visualizations, such as maps, where we are using a number of different colors. To ameliorate this problem, we can make use of the brewer scales in ggplot2. The brewer scales are based on the color palettes from http://colorbrewer2.org developed to allow all people to more easily differentiate colors in data visualizations. There are many possible color palette options available in ggplot2, but let's do an example using the scale_color_brewer() function and the palette titled "BuGn". We will add this to the end of the code for the previous scatterplot that combines urban_fct as shapes and uni_fct as colors. We'll also include theme_minimal() to help with the color rendering.

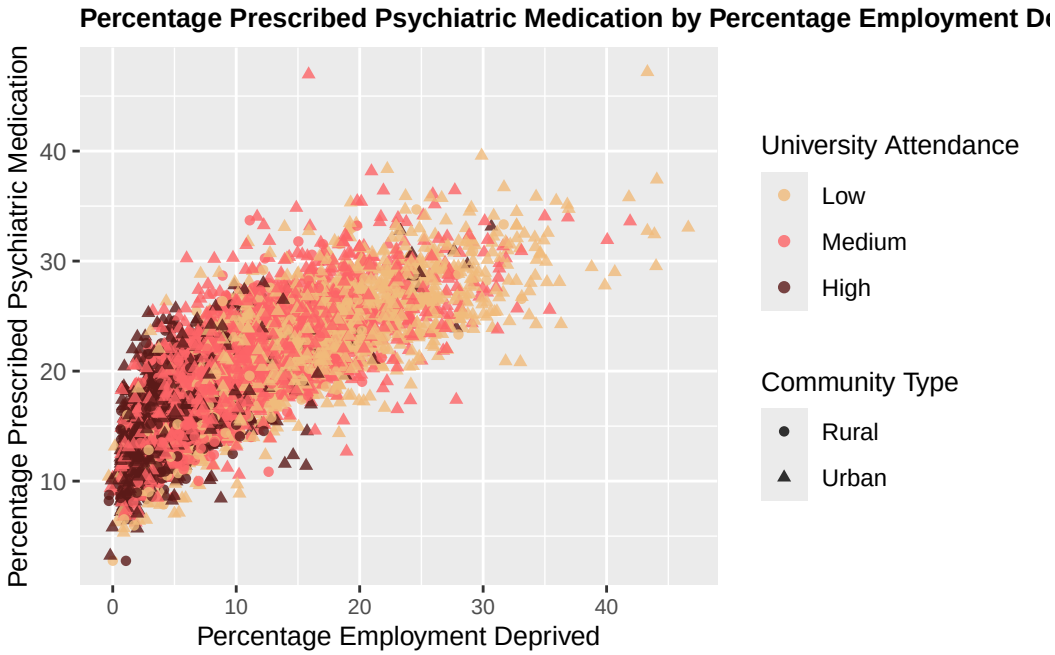
```r
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct) &
           !is.na(university)) |>
  mutate(uni_fct = case_when(
    university <= .05 ~ "Low",
    university > .05 & university <= .12 ~ "Medium",
    university > .12 ~ "High")) %>%
  mutate(uni_fct = factor(uni_fct,
                          levels=c("Low","Medium","High"))) %>%
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
             y=pct_depress, shape=urban_fct,
             color=uni_fct),
             position="jitter",
             alpha=1/1.25) +
  labs(x="Percentage Employment Deprived",
       y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived",
    subtitle="by Community Type and University Attendance",
    shape="Community Type", color="University Attendance") +
  #theme_minimal(
    #plot.title = element_text(size=10, face="bold"),
    #plot.subtitle = element_text(size=10),
    #axis.title.x = element_text(size=10),
    #axis.title.y = element_text(size=10),
    #legend.title = element_text(size=10),
    #axis.text.x.bottom = element_text(size=8)
    #) +
  scale_color_brewer(palette="BuGn") +
  theme_minimal(base_size = 8)
```



There are other color palettes we can use, including one based on Wes Anderson movies - from the wesanderson package. It looks like the wesanderson package hasn't been updated since 2018, but there are still some good options including a palette based on the film Grand Budapest Hotel. After loading the wesanderson package, we use the scale_color_manual() function and specify values=wes_palette("GrandBudapest1") to get the Grand Budapest Hotel palette.

```
library(wesanderson)

simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct) &
           !is.na(university)) |>
  mutate(uni_fct = case_when(
    university <= .05 ~ "Low",
    university > .05 & university <= .12 ~ "Medium",
    university > .12 ~ "High")) %>%
  mutate(uni_fct = factor(uni_fct,
                          levels=c("Low","Medium","High"))) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
            y=pct_depress, shape=urban_fct,
```

```
            color=uni_fct),
            position="jitter",
            alpha=1/1.25) +
  labs(x="Percentage Employment Deprived",
       y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
  shape="Community Type", color="University Attendance") +
theme(
  plot.title = element_text(size=10, face="bold"),
  plot.subtitle = element_text(size=10),
  axis.title.x = element_text(size=10),
  axis.title.y = element_text(size=10),
  legend.title = element_text(size=10),
  axis.text.x.bottom = element_text(size=8)
)+ scale_color_manual(values=wes_palette("GrandBudapest1"))
```



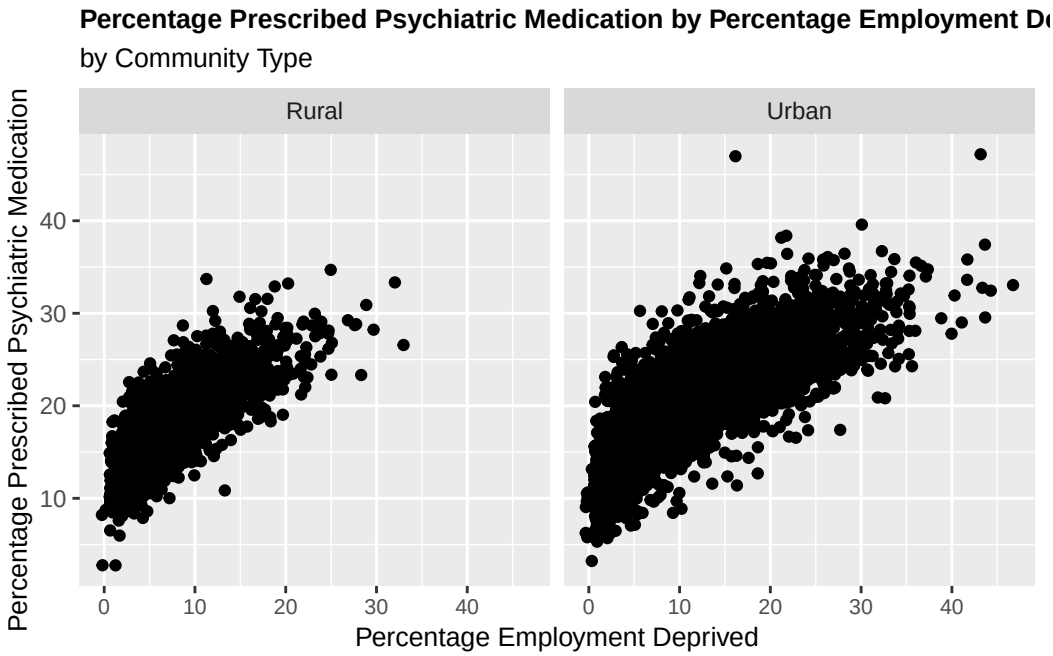
Multiple Plots

Often, we want to combine multiple plots into one plot or plot certain aspects of variables. There are a few ways to do this depending on the type of plot and what we want to achieve in our visualizations. Below we'll look at faceting and combining multiple plots into one plot.

Faceting

Faceting, in the ggplot2 context, involves creating separate plots for the different values of our variable(s). Ordinarily, this is done with a categorical (nominal- or ordinal-level) variable where each plot consists of a single value from the variable. Let's use faceting to create different scatterplots for urban_fct, where we use pct_employment_deprived on the x-axis and pct_depress on the y-axis. We do this by adding the facet_wrap() function to our code. To do subplots for a third variable, we specify the code as facet_wrap(~urban); ~ separates the column and row variables, but here we just use a column variable.

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
              y=pct_depress), position="jitter") +
  facet_wrap(~urban_fct) +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type") +
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
    legend.title = element_text(size=10),
    axis.text.x.bottom = element_text(size=8)
    )
```

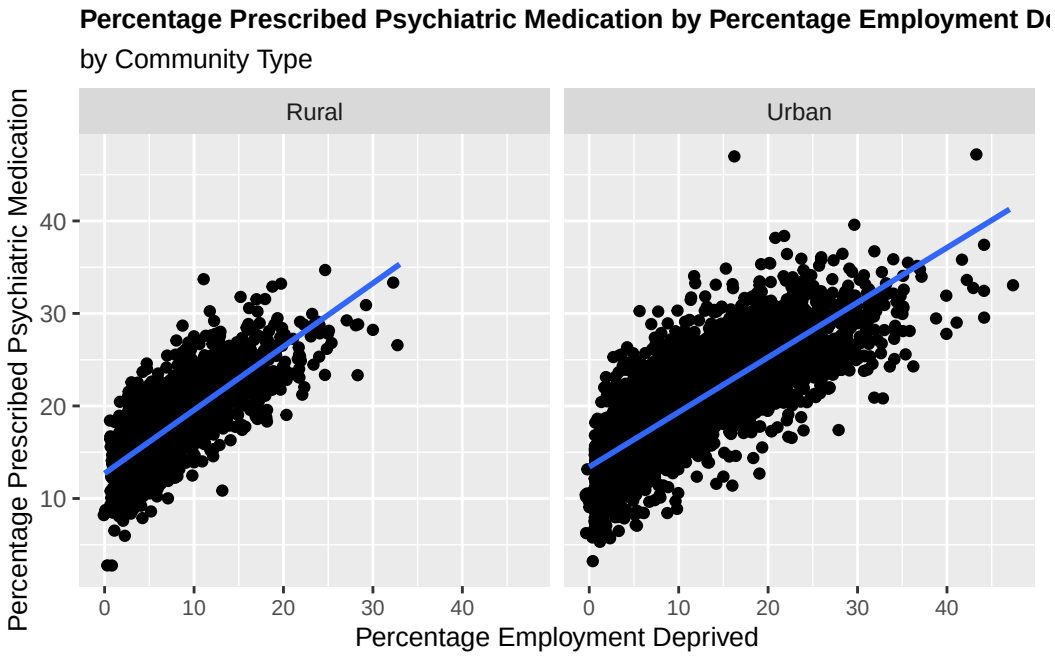


We see that we now have two scatterplots: the left-hand side shows the relationship between employment deprivation and psychiatric prescriptions for rural datazones and the right-hand side shows the relationship for urban datazones. As I mentioned in the workshop, sometimes we include a linear-fit line (i.e., a bivariate linear regression line) in scatterplots, generally, and when faceting, specifically, to clearly demonstrate relationships between variables. We'll do this below by including the geom_smooth() function, specifying method="lm" (for linear model), and specifying se=FALSE (don't plot standard errors). (Note: by default, we get an output message telling us the "formula" used with geom_smooth().)

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot() +
  geom_point(mapping=aes(x=pct_employment_deprived,
              y=pct_depress), position="jitter") +
  geom_smooth(mapping=aes(x=pct_employment_deprived,
              y=pct_depress), method="lm", se=FALSE) +
  facet_wrap(~urban_fct) +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type") +
```

```
  theme(
    plot.title = element_text(size=10, face="bold"),
    plot.subtitle = element_text(size=10),
    axis.title.x = element_text(size=10),
    axis.title.y = element_text(size=10),
    legend.title = element_text(size=10),
axis.text.x.bottom = element_text(size=8)
)
```

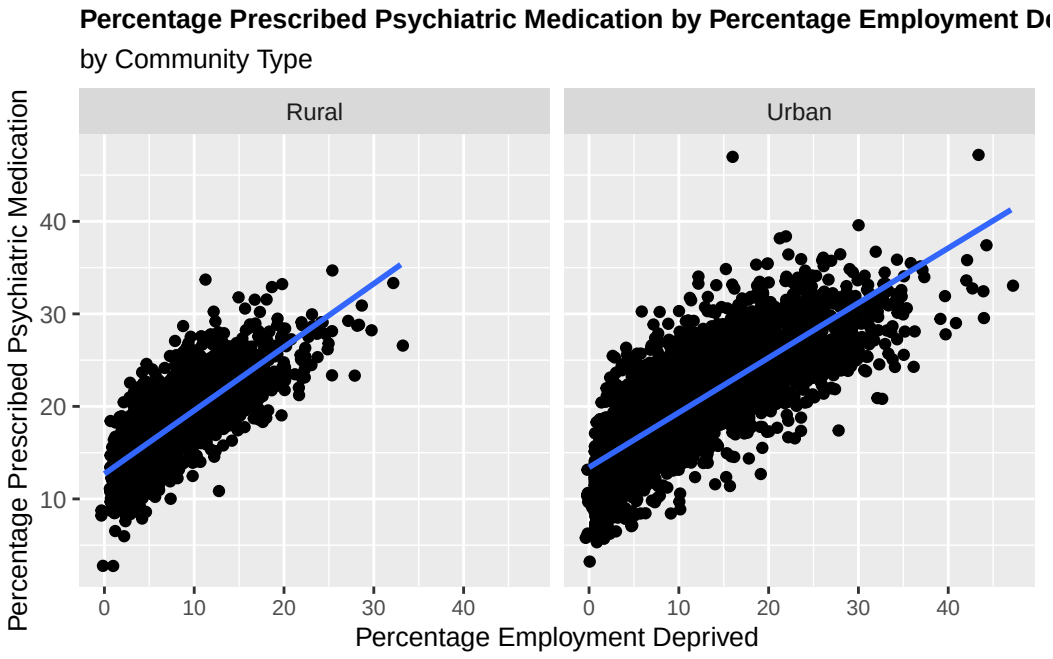
```
`geom_smooth()` using formula = 'y ~ x'
```



We see that the linear slope is roughly the same between rural and urban datazones. This suggests the relationship between employment deprivation and psychiatric prescriptions is similar in both types of datazones; even though urban datazones have more extreme values on both variables. In the workshop, a question was asked whether the mapping and aes can be put in the ggplot() function instead of the geom_() function(s)? In short, yes. If we specify the mapping and aes in the ggplot() function, it is used for all of the geom_() functions that follow. Hence, a good time to take this approach is if we are using several geom_() functions in our plot and we don't want to repeat the specification. To demonstrate, let's move the specification in geom_point() and geom_smooth() from the previous plot to the ggplot() function.

```r
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot(mapping=aes(x=pct_employment_deprived, y=pct_depress)) +
  geom_point(position="jitter") +
  geom_smooth(method="lm", se=FALSE) +
  facet_wrap(~urban_fct) +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type") +
  theme(
  plot.title = element_text(size=10, face="bold"),
  plot.subtitle = element_text(size=10),
  axis.title.x = element_text(size=10),
  axis.title.y = element_text(size=10),
  legend.title = element_text(size=10),
  axis.text.x.bottom = element_text(size=8)
  )
```

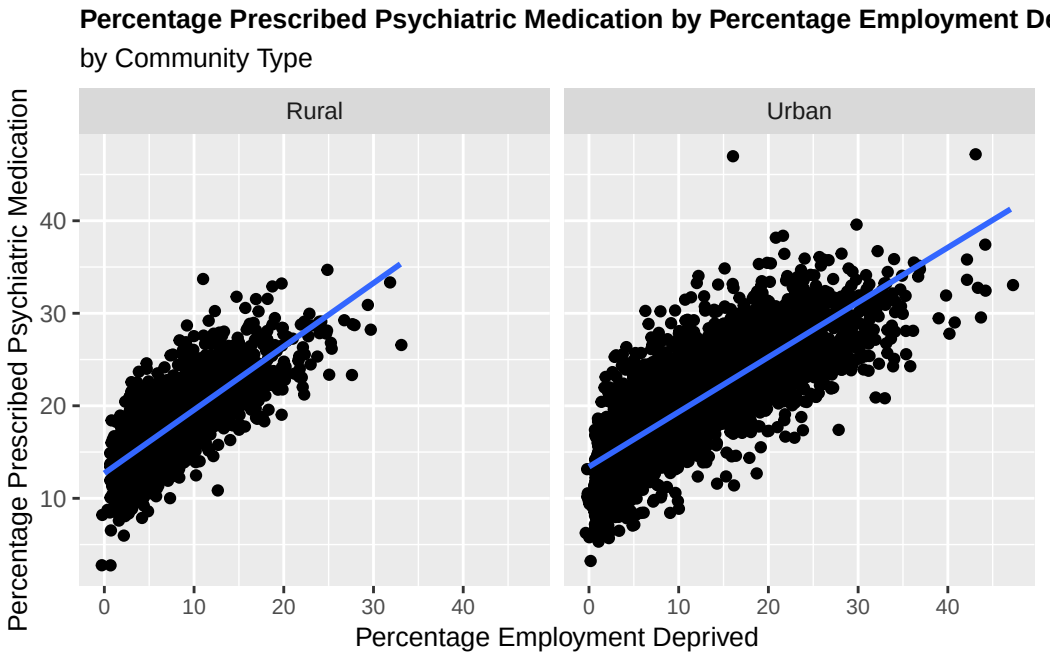
```
`geom_smooth()` using formula = 'y ~ x'
```



Lastly, if we include the specification in the ggplot() function, we can override it in any individual geom_() functions by including a (new) specification. We do this below by adding the specification to geom_smooth().

```
simd |>
  mutate(pct_employment_deprived = employment_rate*100) |>
  filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) & !is.na(urban_fct)) |>
ggplot(mapping=aes(x=pct_employment_deprived, y=pct_depress)) +
  geom_point(position="jitter") +
  geom_smooth(mapping=aes(x=pct_employment_deprived, y=pct_depress),
              method="lm", se=FALSE) +
  facet_wrap(~urban_fct) +
  labs(x="Percentage Employment Deprived",
    y="Percentage Prescribed Psychiatric Medication",
    title="Percentage Prescribed Psychiatric Medication by Percentage Employment Deprived"
    subtitle="by Community Type") +
theme(plot.title = element_text(size=10, face="bold"),
plot.subtitle = element_text(size=10),
axis.title.x = element_text(size=10),
axis.title.y = element_text(size=10),
legend.title = element_text(size=10),
axis.text.x.bottom = element_text(size=8)
)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Combining Individual Plots

Sometimes we have individual plots that we want to combine into one plot. The default way to do this in R is using the par() function. However, it can be tricky to get par() to combine multiple ggplot plots. There are a variety existing R packages we can use to combine multiple plots (e.g., gridExtra, ggpubr). Here, let's use the patchwork package for combining plots. Below, we will create two plots and combine them into a 2x1 array by simply separating the plot objects using /. To do so, we need to create plots with distinct names; specifically, we will save the plots as objects by naming each plot sequentially starting with p1. (Note: the below doesn't print the plots because we are just asking R to create an object, not show it.)

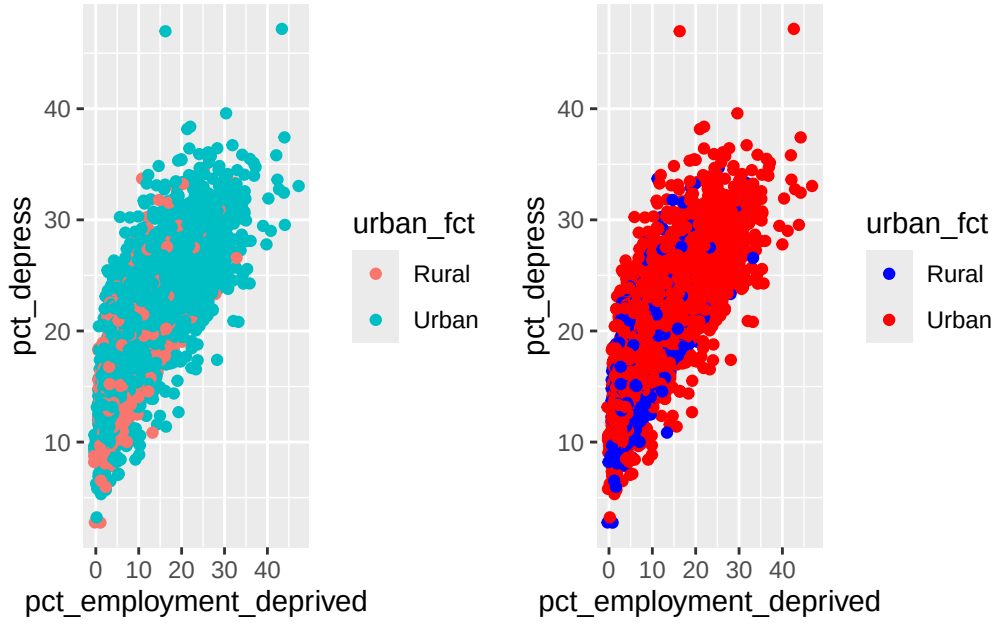
```
library(patchwork)

p1 <- simd |>
        mutate(pct_employment_deprived = employment_rate*100) |>
        filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) &
                  !is.na(urban_fct)) |>
      ggplot() +
        geom_point(mapping=aes(x=pct_employment_deprived,
              y=pct_depress, color=urban_fct),
              position="jitter")
```

```
p2 <- simd  |>
        mutate(pct_employment_deprived = employment_rate*100) |>
        filter(!is.na(pct_depress) & !is.na(pct_employment_deprived) &
                 !is.na(urban_fct)) |>
      ggplot() +
        geom_point(mapping=aes(x=pct_employment_deprived,
              y=pct_depress, color=urban_fct),
              position="jitter")  +
      scale_color_manual(values = c("blue", "red"))
patch <- p1 / p2
patch
```

As discussed in the workshop, combining plots often results in smooshed plots. If this occurs, we just need to play around with the sizing to get them to look better.

If we wanted to combine the plots into a 1x2 array, we can use +.

```
patch1 <- p1 + p2
patch1
```



The patchwork reference site demonstrates all the different possible customizations (https: //patchwork.data- imaginist.com/).