Introduction to the Grammar of Graphics

R Tutorials for Applied Statistics

Note on required packages: The following code requires the packages `tidyverse`, `scales`, `stringr`, `Hmisc`, and `ggthemes`.

- The `tidyverse` package contains many packages that allow you to organize, summarize, and plot data.
- We use the `scales` library to customize the scales of our axes.
- The `stringr` package allows us to manipulate strings, which we use to manipulate string labels.
- The `Hmisc` provides mathematical and statistical functions to use with our plots.

If you have not already done so, download, install, and load the libraries with the following code:

```
# These lines only need to be executed once for your machine
install.packages("tidyverse")
install.packages("scales")
install.packages("stringr")
install.packages("Hmisc")

# These lines need to be executed every time you load R
library("tidyverse")
library("scales")
library("stringr")
library("Hmisc")
```

---

1 Introduction

The **grammar of graphics** is a way of thinking about how graphs are constructed that allows data analysts to move beyond thinking about a small number of graph types (like bar graphs, line graphs, scatter plots, etc).

Think about the grammar of graphics just like you would about the grammar of sentence structure in language. We think beyond understanding language as just sentence types, like declarative sentences (statements of fact), imperative sentences (statement of request), or exclamatory sentences (statements of excitement/emotion).

Grammar in language considers multiple structures within the sentence that are layered together. For example, at a minimum, the structures of noun and verb can be put together to form a most simple sentence of any type. We can optionally add more structures like conjunctions, pronouns, and propositions, and layer these together in simple or complex ways to form more complex sentences. We can layer these structures to produce sentences that have infinite possibilities for communicating.

So too with the grammar of graphics. There are seven *structures* or *layers* in the grammar of graphics. We can layer on just a minimum number of structures (just like a minimal sentence can have just one noun and one verb) or make more complicated graphs by using multiple types of layers and multiple layers of some of the types. This is just like a sentence can include multiple structures like nouns, verbs, prepositions, and also include more than one type of structure, like have more than one noun).

The following three layers are the minimum necessary for any type of plot:

1. Data: A data frame with one or more variables, each with one or more observations.

2. Aesthetic: A mapping of one or more variables to one or more visual elements on the graph. For example, you could map a variable to the x-axis, another variable to the y-axis, and a categorical variable to color so that different categories get plotted with different colors.

3. Geometry: The type or shape of the visual elements on the graph. For example, this could be a *point* in the case of a scatter plot, a *bar* in the case of a bar plot, or a *line* in the case of a line plot.

2 Data Set

The following example uses a sample from the 2016 Current Population Survey which is a monthly survey conducted by the U.S. Census Bureau and Bureau of Labor Statistics. The data is used, among other things, to compute employment statistics related to earnings, hours employed, and unemployment. This particular sample includes 1,552 observations and includes only head-of-households.

The line below downloads and loads the data set.

```
load(url("https://murraylax.org/datasets/cps2016.RData"))
```

The data is in a `data.frame` object called `df` and a description of the variables is given in another `data.frame` object called `df.desc`. You can familiarize yourself with the data set by opening these data frames in *Rstudio*. Alternatively, you can get a short description of the data frame `df` with a call to the `str()` function.

```
str(df)
```

```
## 'data.frame':    1552 obs. of  17 variables:
##  $ age        : num  46 37 35 38 66 28 50 49 63 43 ...
##  $ incwage    : num  24000 86000 22000 35000 50000 24000 75000 52000 40000 30000 ...
##  $ sex        : Factor w/ 2 levels "Female","Male": 1 2 1 2 2 1 1 2 2 2 ...
##  $ race       : Factor w/ 5 levels "Asian/Pacific Islander",..: 5 5 2 5 5 5 5 5 5 5 ...
##  $ empstat    : Factor w/ 4 levels "Armed Forces",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ inlf       : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ unempl     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ industry   : Factor w/ 7 levels "Agriculture, Forestry, and Fishing",..: 7 6 5 3 1 5 5
##  $ usualhrs   : num  32 40 NA 40 60 40 45 40 45 40 ...
##  $ ureason    : Factor w/ 4 levels "Job Leaver","Job Loser",..: NA NA NA NA NA NA NA NA NA
##  $ veteran    : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ usualhrearn: num  14.4 41.2 NA 16.8 16 ...
##  $ edu        : Ord.factor w/ 5 levels "Less than high school"<..: 2 4 3 1 3 3 5 3 3 3 ...
##  $ medoop     : num  1000 3300 3570 1500 1730 ...
##  $ insprem    : num  0 1000 1800 0 480 6500 0 2000 16000 4000 ...
##  $ totmed     : num  1000 4300 5370 1500 2210 ...
##  $ college    : num  0 1 0 0 0 0 1 0 0 0 ...
```

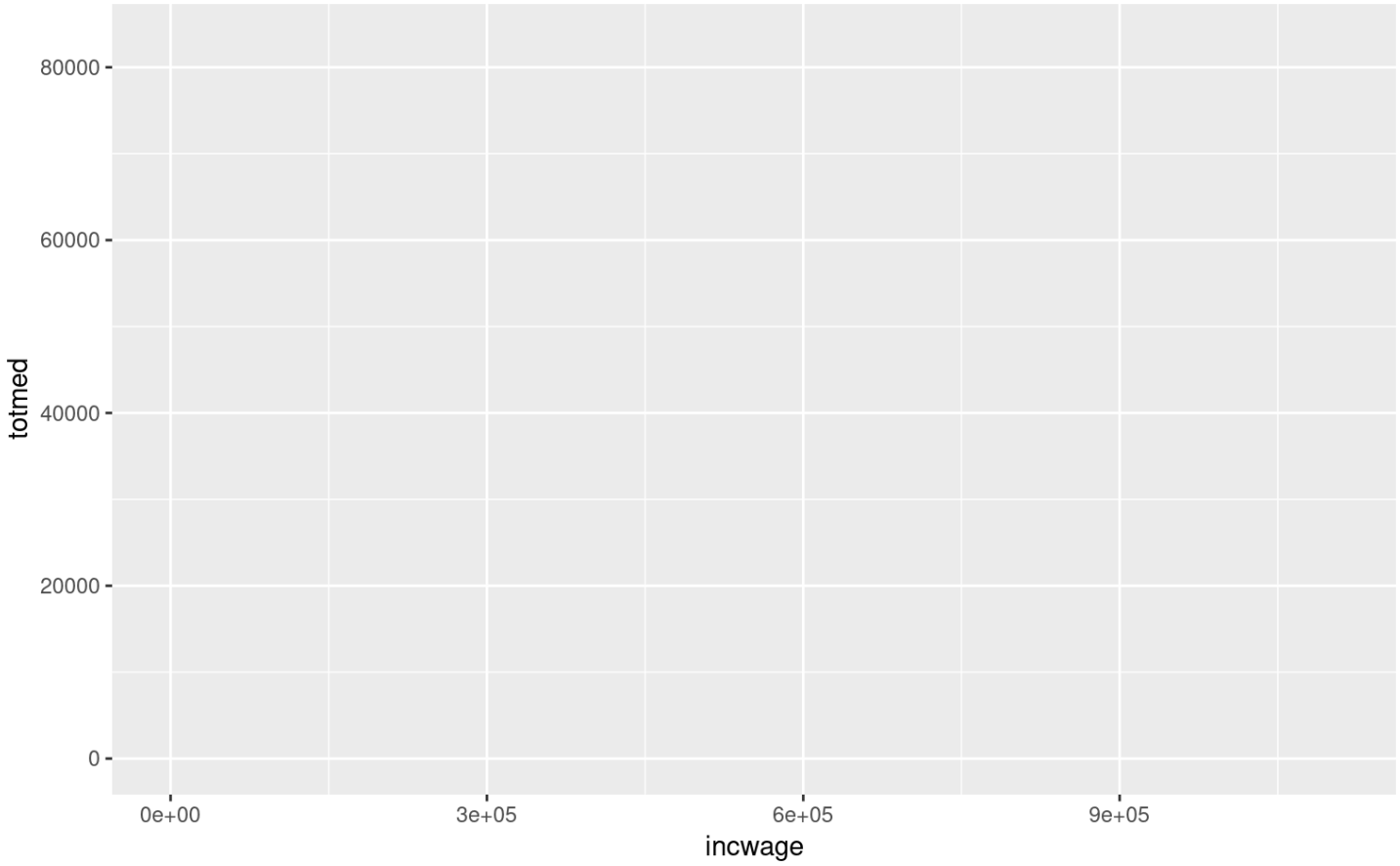
3 Example: Scatter Plot

We will use the three layers of the grammar of graphics above to create a basic scatter plot of the data. We have two numerical variables in the data set, the survey respondents' annual income from wages and salaries (`incwage`) and their annual total health care expenditures, including health insurance premiums and out-of-pocket expenses for health care services (`totmed`). We will create a scatter plot with `incwage` on the horizontal axis and `totmed` on the vertical axis.

3.1 Data and Aesthetics Layers

We will first create a ggplot object with the first two layers, **data** and **aesthetics**.

```
ggplot(data=df, mapping=aes(x=incwage, y=totmed))
```



The function call `ggplot()` above sets two parameters. The first parameter, `data`, is set equal to the data frame `df` which ggplot will look to for the data.

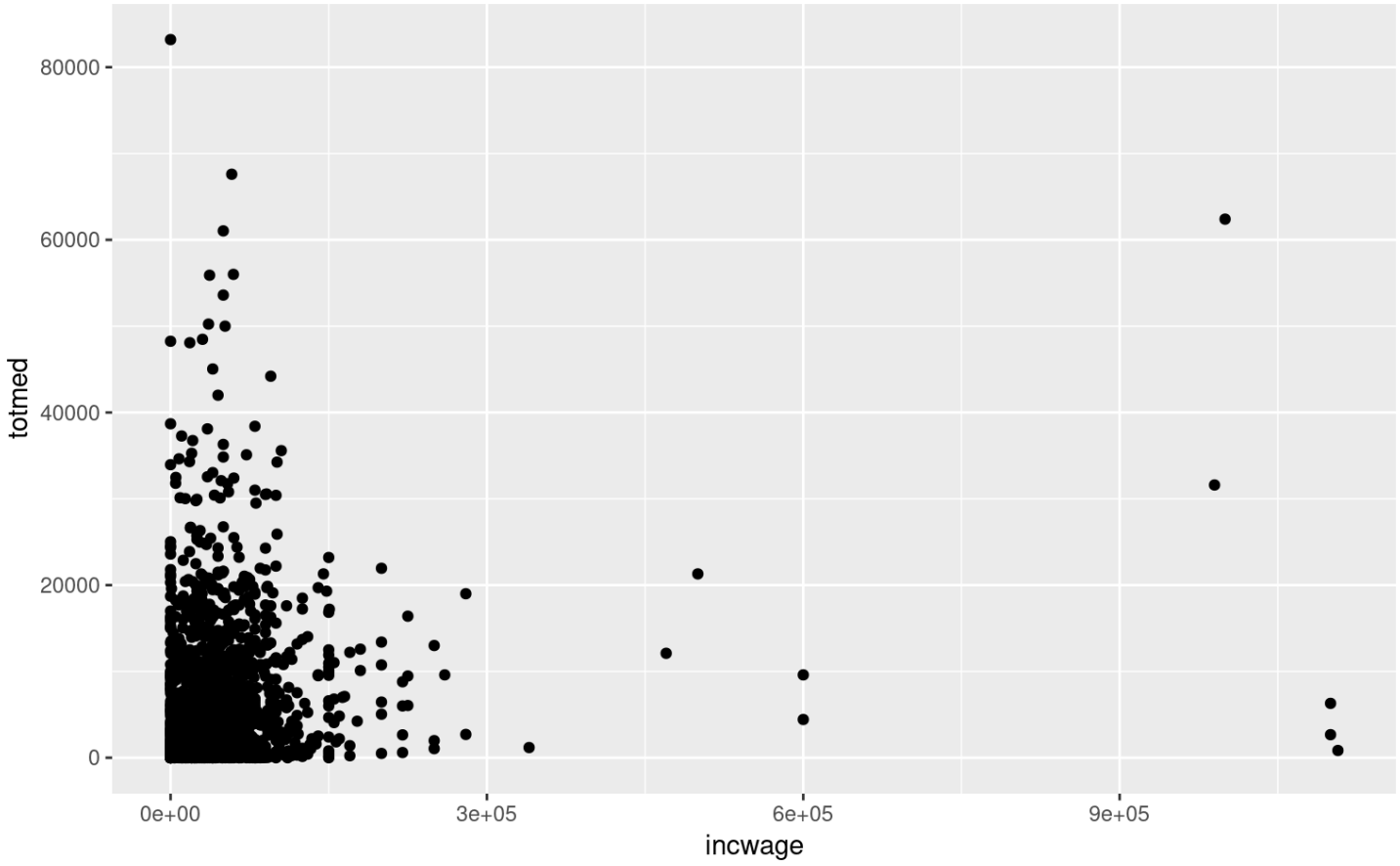
The second parameter, `mapping`, is the aesthetics layer. Here we map variable names from the data frame to visuals in our plot. We call another function here called `aes()` to create the mapping. The parameters we can set in our call to `aes()` include all the possible visual elements of the graph that can be mapped to variables. We use two visual elements, the x-axis and y-axis. We map the x-axis to income with `x=incwage` and the y-axis to total out-of-pocket expenses using `y=totmed`.

What you will see is the foundation for the plot, but nothing to plot yet. It does not yet specify a geometry layer, i.e. the geometrical shapes, to put on the graph.

3.2 Geometry Layer

Additional layers can be “added” to the ggplot object using the addition symbol (+). We next add a **geometry** layer. There are many types of geometries that are available for ggplot. We add *points* to the base plot above.

```
ggplot(data=df, mapping=aes(x=incwage, y=totmed)) +
  geom_point()
```



We have a plot, even if it is not too pretty, created with the minimum requirements for layers. Like a minimal sentence with just a noun and verb, it communicates something, but maybe not very much, and maybe not very effectively. We will next add on a few layers to make the plot not only more visually appealing, but change the visualization so it is more effectively communicates the relationship between our two variables.

3.3 Handling Over-plotting

Our plot suffers from a common problem called **over-plotting**. This is when a number of data points or statistical visuals land on top of one another which hides data and can obscure a visual relationship.

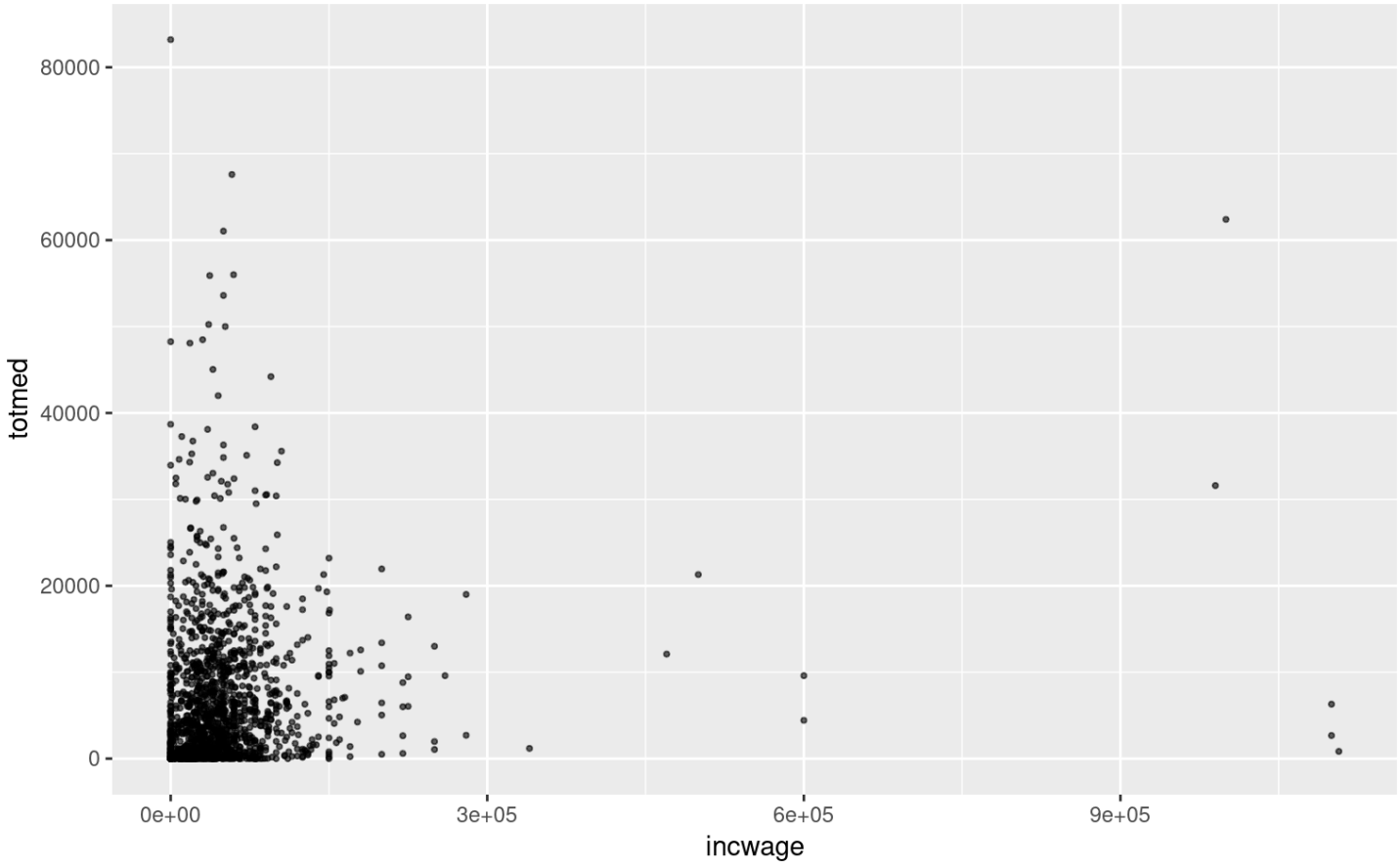
Here are some strategies we can employ with this plot to overcome our over-plotting problem:

1. Decrease size of geometries: We can make the points smaller.

2. Make geometries partially transparent: We can make it so that our points are not completely opaque so we can “see through” the points to find more points underneath. When a number of points land on top of one another, we will be better able to see the multiple points underneath. We will see darker areas where many points are clustered close together.
3. Zoom in: We can zoom in so that we visualize most of our sample, but not our outliers. The limits for annual income on our horizontal axis range from $0 per year to over $1,000,000 per year, yet most of our data includes individuals who earn less than $130,000 per year.

Let us first employ the first two strategies: making our points smaller and adding some transparency to them. To do this, we will change our call to the geometry layer. Consider the following plot:

```
ggplot(data=df, mapping=aes(x=incwage, y=totmed)) +
 geom_point(alpha=0.6, size=0.7)
```



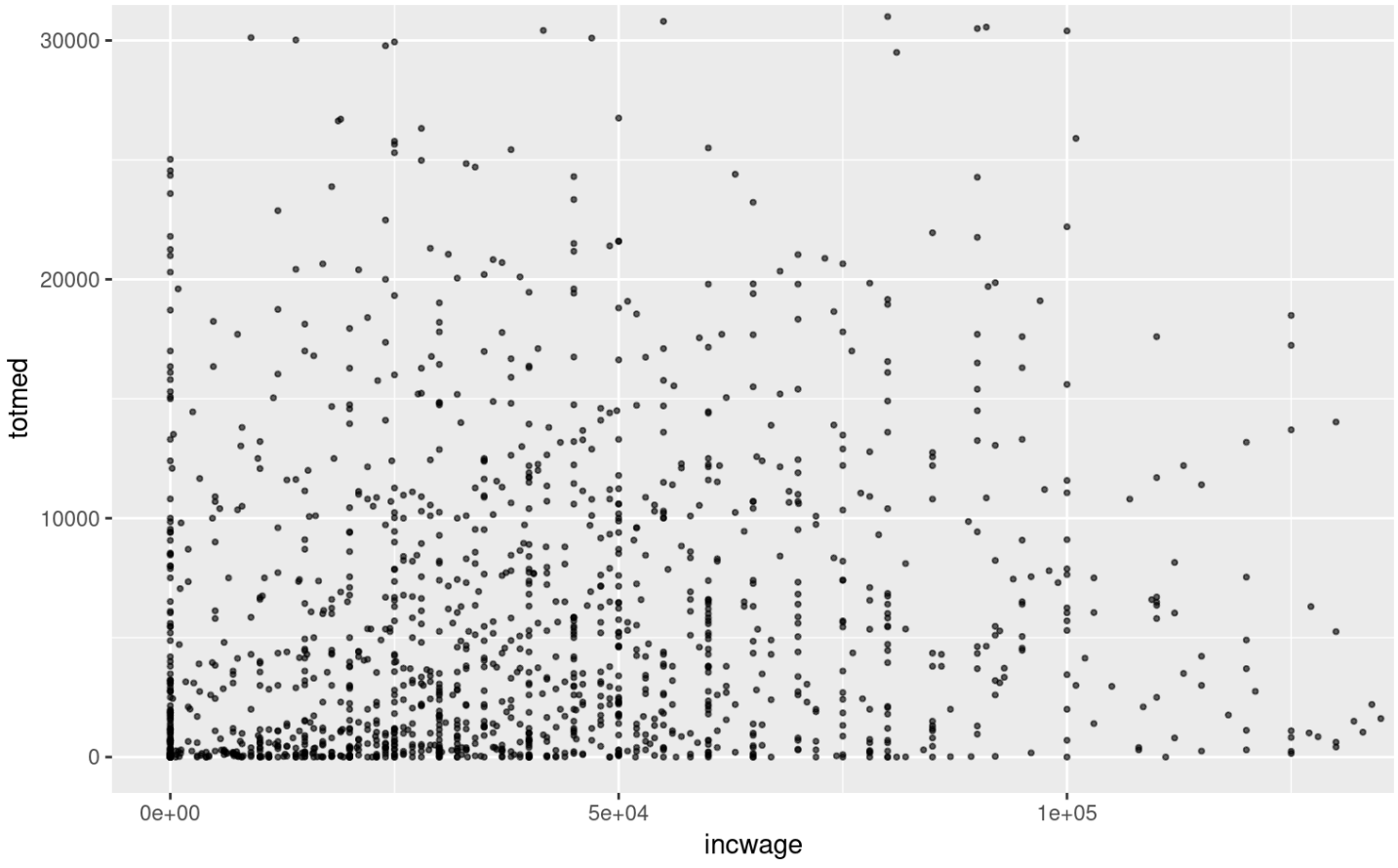
We passed two optional parameters to the function `geom_point()`. The first `alpha=0.6` sets the transparency. This is a number between 0 and 1, where 1 is completely opaque (the default) and 0 is completely invisible. A value for transparency of 0.6 lets us see the points, but gives us a lot of transparency to also see through them. The second

parameter, `size=0.7`, gives us a smaller point size. Picking the right level of transparency and size for a geometry takes some trial and error and will be different for every plot, and even different depending on your screen resolution and the resolution you choose should you save your plot.

Let us next zoom in so that our plot displays well the large majority of the data while omitting the outliers. We can zoom in by specifying another *layer* of the plot, the **coordinates layer**. The coordinates layer specifies the type of coordinate system to use and it allows you to customize the dimensions. The most common coordinate system for a scatter plot, and the one we have been using by default, is the Cartesian plane.

In the code below, we explicitly add the Cartesian coordinate system with a call to the function `coord_cartesian()` and we specify the largest and smallest values for each the x- and y-axes. In the code that follows, we start with the plot code we have already created and we add the coordinates layer.

```
ggplot(data=df, mapping=aes(x=incwage, y=totmed)) +
  geom_point(alpha=0.6, size=0.7) +
  coord_cartesian(xlim=c(0,130000), ylim=c(0,30000))
```

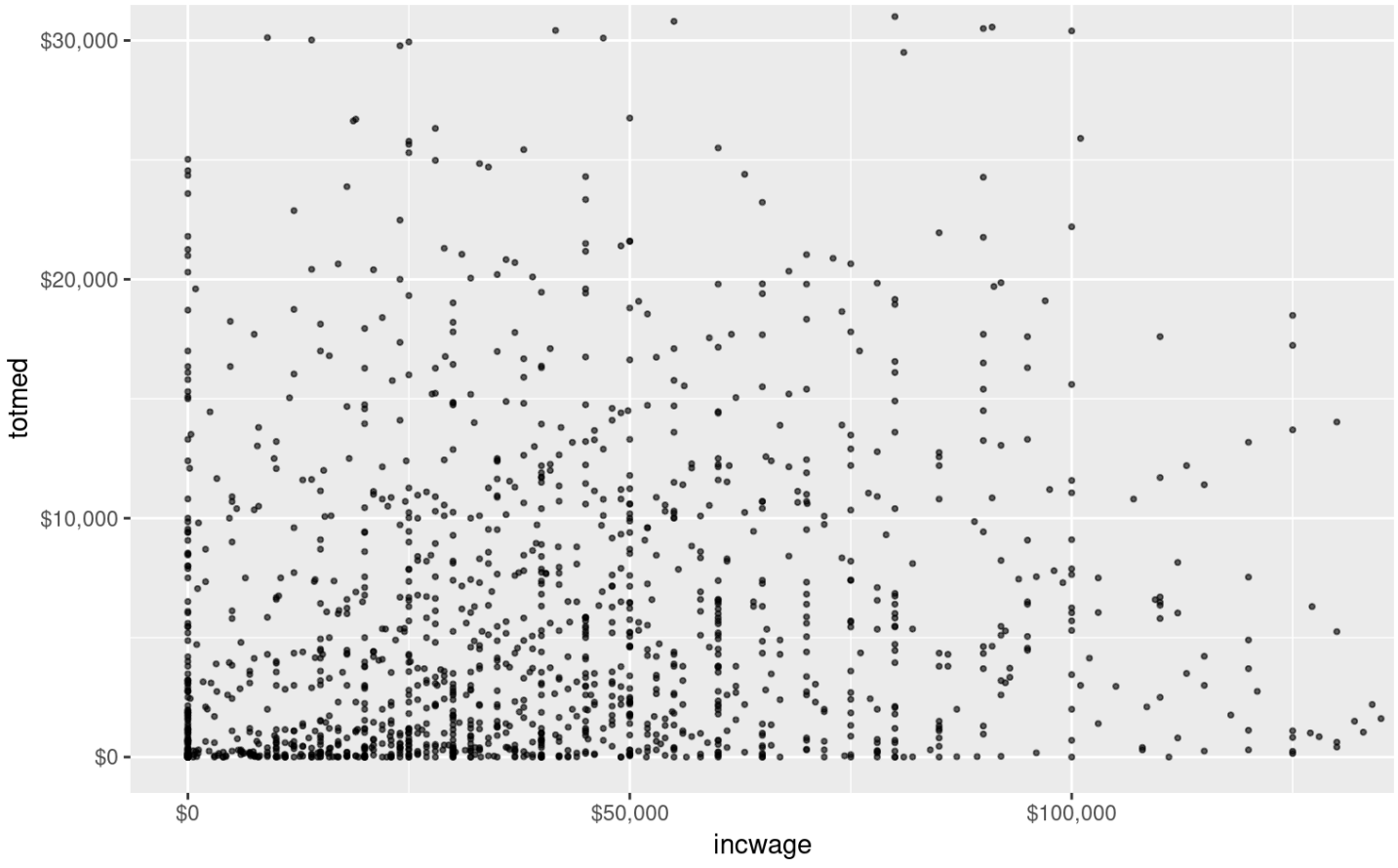


The plot itself looks much better. While the wide variability in the data still does not make it too clear whether there is a strong positive correlation, we can see the distribution of the data quite well. In the next subsection, we will fix up some of the non-data ink, including the axes scale labels, axes titles, and plot title.

3.4 Scales and Titles: Dressing up Non-data Ink

In this section, we clean up some of the non-data ink to make our graph more visually appealing and more effectively communicate our variables and how they are measured. Below we give the axes more descriptive labels and give the plot a title. We also saw in the previous subsection that the scale labels on the horizontal axes was inconveniently expressed in scientific notation (i.e. `5e+4` was given as a label rather than `50,000`). The in code below, we add to our zoomed-in scatter plot calls to `scale_x_continuous()` and `scale_y_continuous()` which set the notation for the x- and y-axis scale labels, respectively.

```
ggplot(data=df, mapping=aes(x=incwage, y=totmed)) +
  geom_point(alpha=0.6, size=0.7) +
  coord_cartesian(xlim=c(0,130000), ylim=c(0,30000)) +
  scale_x_continuous(labels=dollar) +
  scale_y_continuous(labels=dollar)
```



The parameter `labels` in each of these functions is set equal to the *function* `dollar()`, which takes on a single numerical value and returns a string that expresses the same number with a dollar symbol and commas after every third digit from the decimal point. The `dollar()` function from the `scales` package.

We can set the titles for the x-axis, y-axis, and overall plot with a call to the `labs()` (short for labels) function. Here is our final scatter plot:

```
ggplot(data=df, mapping=aes(x=incwage, y=totmed)) +
  geom_point(alpha=0.6, size=0.7) +
  coord_cartesian(xlim=c(0,130000), ylim=c(0,30000)) +
  scale_x_continuous(labels=dollar) +
  scale_y_continuous(labels=dollar) +
  labs(title="Distribution of Labor Income and Medical Expenditures",
       x="Wage and Salary Income (dollars)", y="Total Medical Expenditures (dollars)")
```

4 Example: Bar Chart

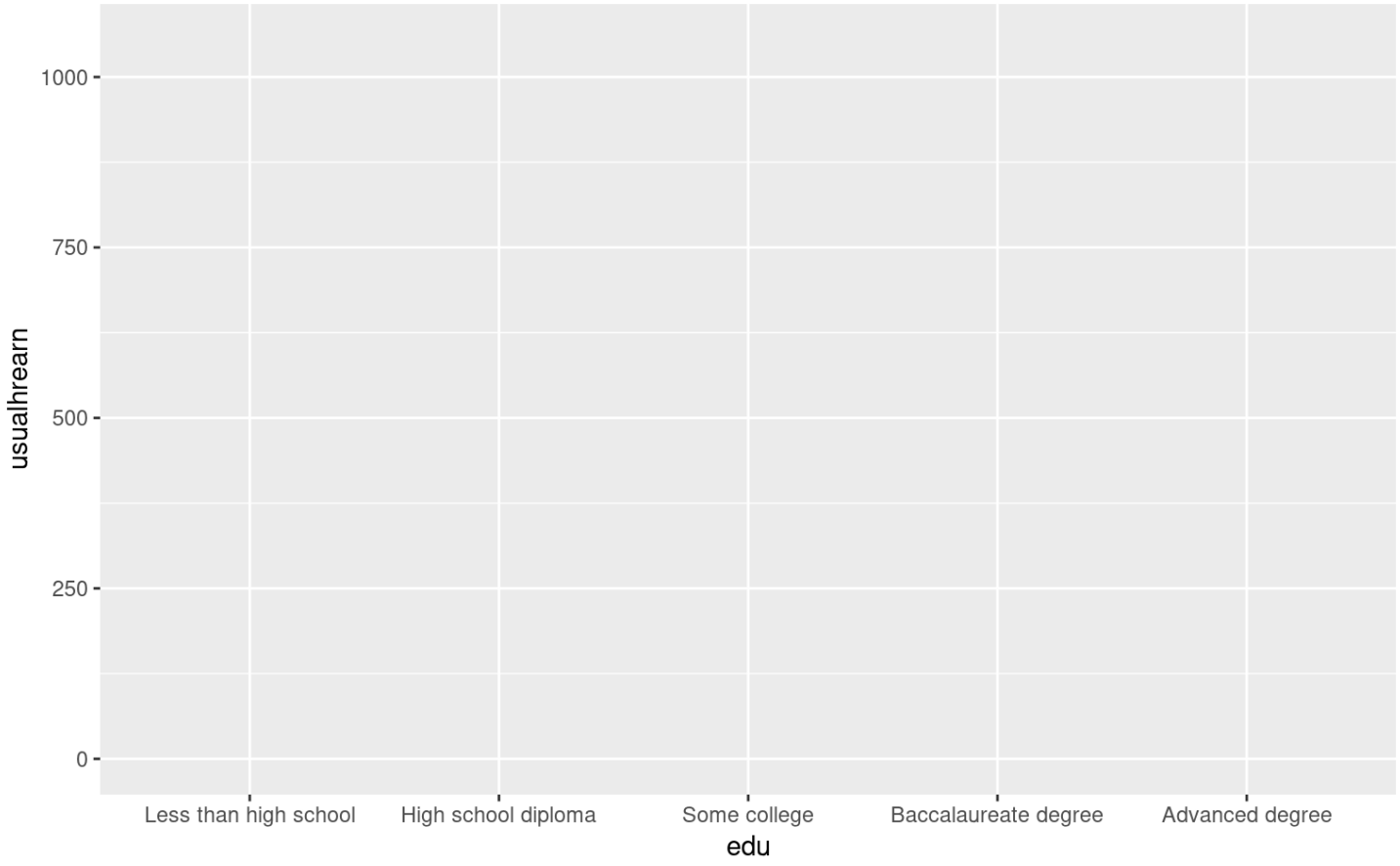
Let us use our same data set and build another common plot using the grammar of graphics in `ggplot`. In this example, we use the usual hourly earnings variable (`usualhrearn`) and education level (`edu`) to compare the *mean* usual hourly earnings

across different levels of education. Usual hourly earnings is a numerical variable expressed in dollars. Education level is an *ordinal* variable, which is a categorical variable, expressed descriptively as a string, but is encoded with an meaningful order. That is, the variable is coded so that the
value `"Baccalaureate degree" > "Some college"` and `"Some college" > "High school diploma"`, etc.

4.1 Data and Aesthetics Layers

We start with a call to `ggplot()` that specifies the data and aesthetics layer. We use the same data frame as before and map the x-axis to `edu` and the y-axis to `usualhrearn`.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn))
```



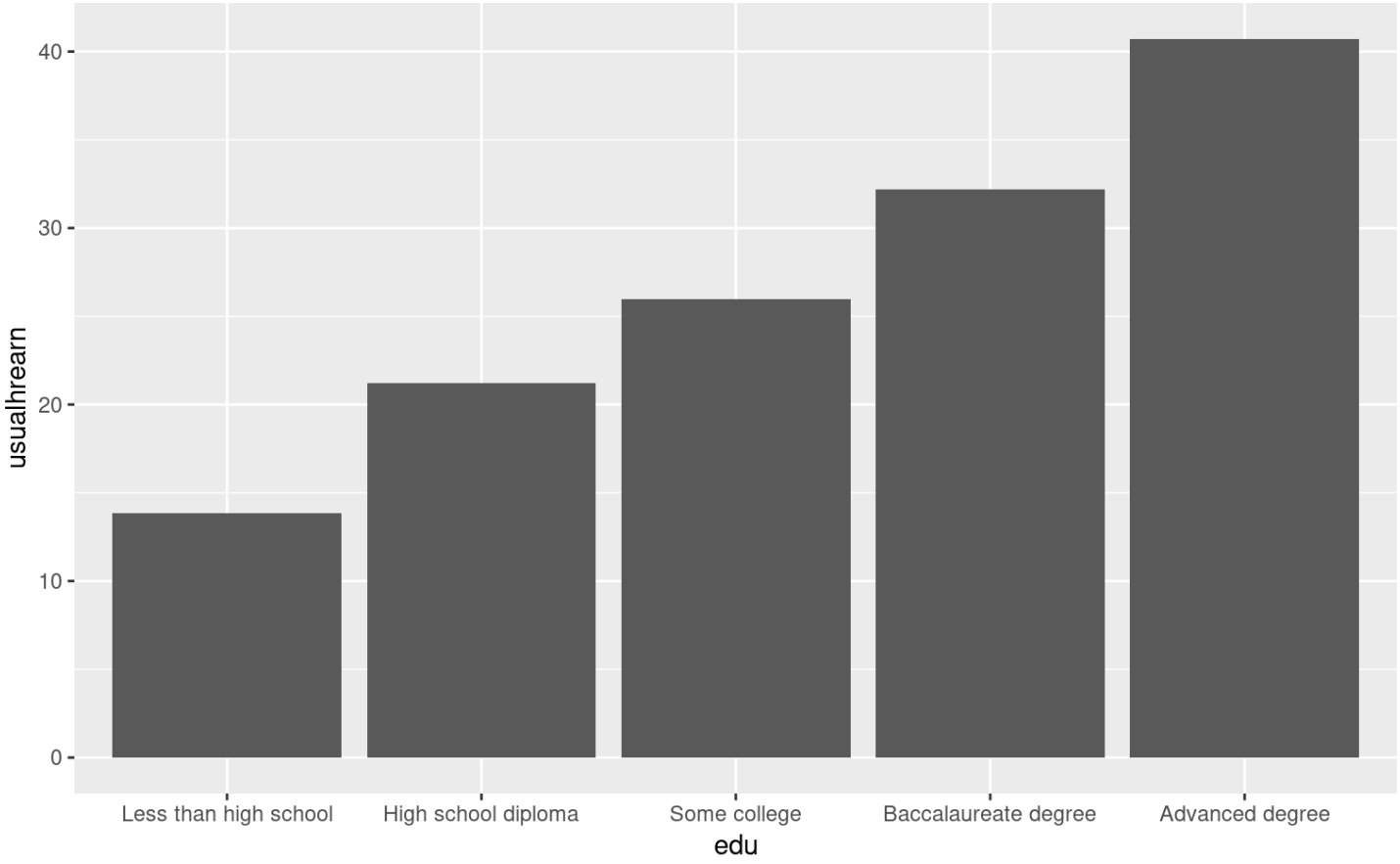
4.2 Statistics and Geometry Layers

In the case of a bar plot, we do not wish to graph the raw data, but rather a *statistic* that summarizes the data. Our statistic is the *mean*. We wish to compute the mean usual hourly earnings for each education level, and plot rectangular bars with a height equal to the

mean. To do this, we will not specify a geometry layer, but rather a **statistics layer** which will both compute the statistic we are interested in and specify the *bar* geometry.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar")
```

```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



We get a warning that there are many missing observations not included because there are a number of people in our sample that are not employed so they did not have usual weekly hours and/or wage or salary income. We still have over 1000 data points, so this is not problematic.

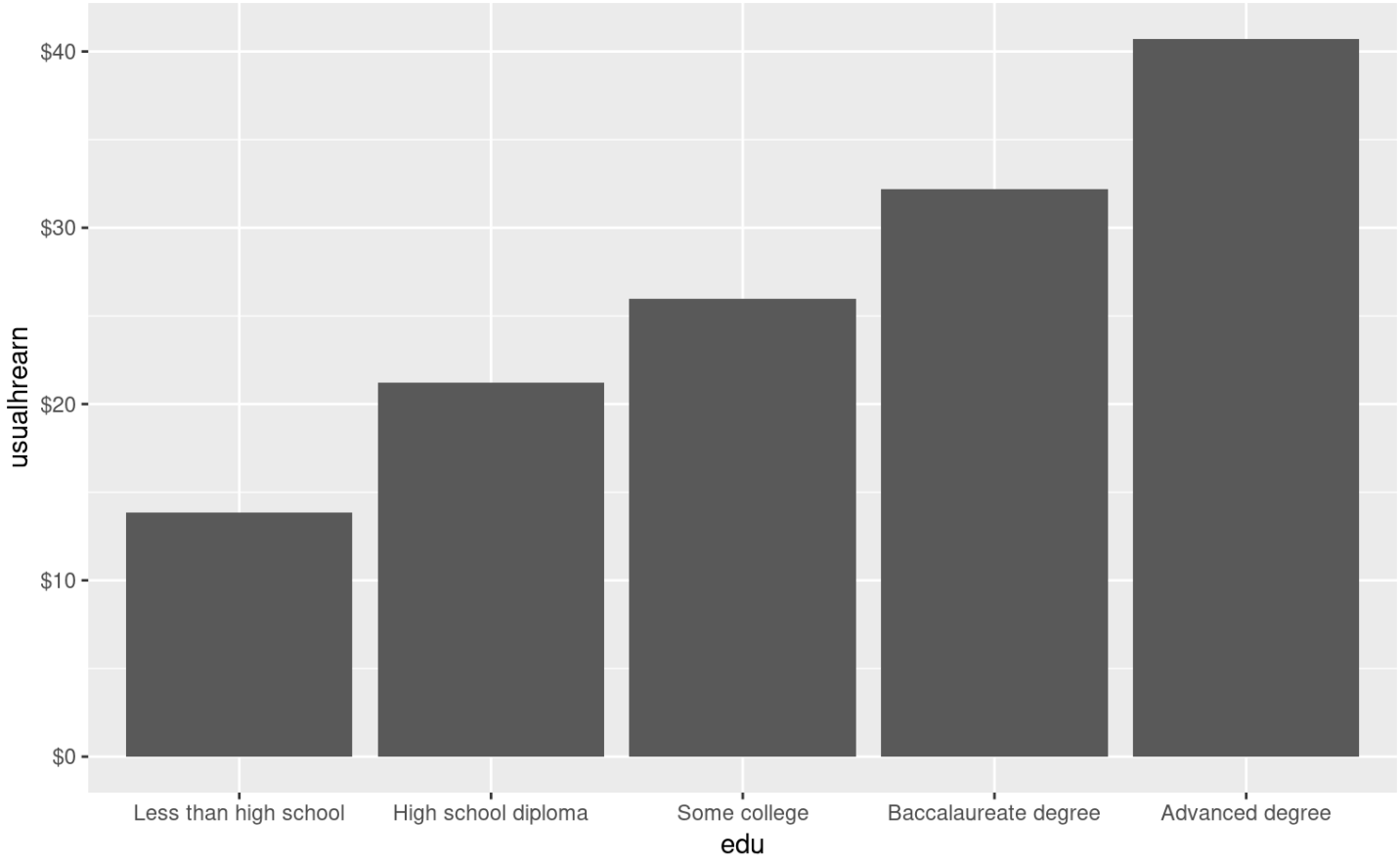
The call to `stat_summary()` specifies our statistics and geometry layer. The first parameter `fun.data=mean_sdl` tells `stat_summary()` what function to call to calculate the desired statistics. The function `mean_sdl()` comes from the `Hmisc` package. It takes a single variable as a parameter and computes the mean and the upper and lower limits of the 95% confidence interval, the latter which is not used for the bar geometry. The second parameter, `geom="bar"`, tells `stat_summary()` to use the bar geometry for the statistics it calculates.

4.3 Labels and Titles

So that it is clear the vertical axis refers to usual hourly earnings in dollars, let us specify that the vertical scale labels be expressed in dollars.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar") +
  scale_y_continuous(labels=dollar)
```

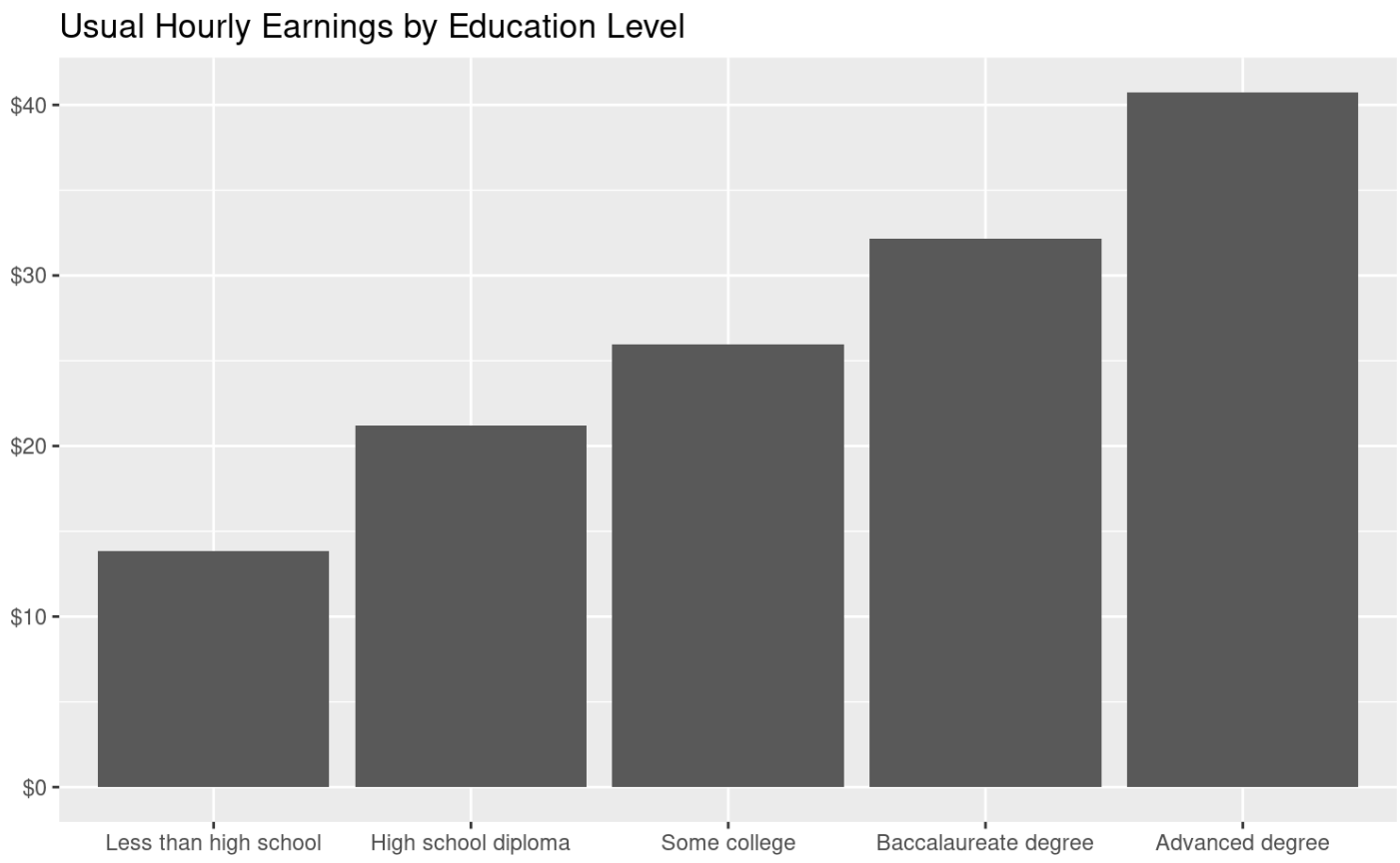
```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



Finally, we use the function `labs()` to add a descriptive title for plot and we remove the titles for the horizontal and vertical axes since those will be obvious from the title of the plot and the labels on the scales.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar") +
  scale_y_continuous(labels=dollar) +
  labs(title="Usual Hourly Earnings by Education Level", x="", y="")
```

```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



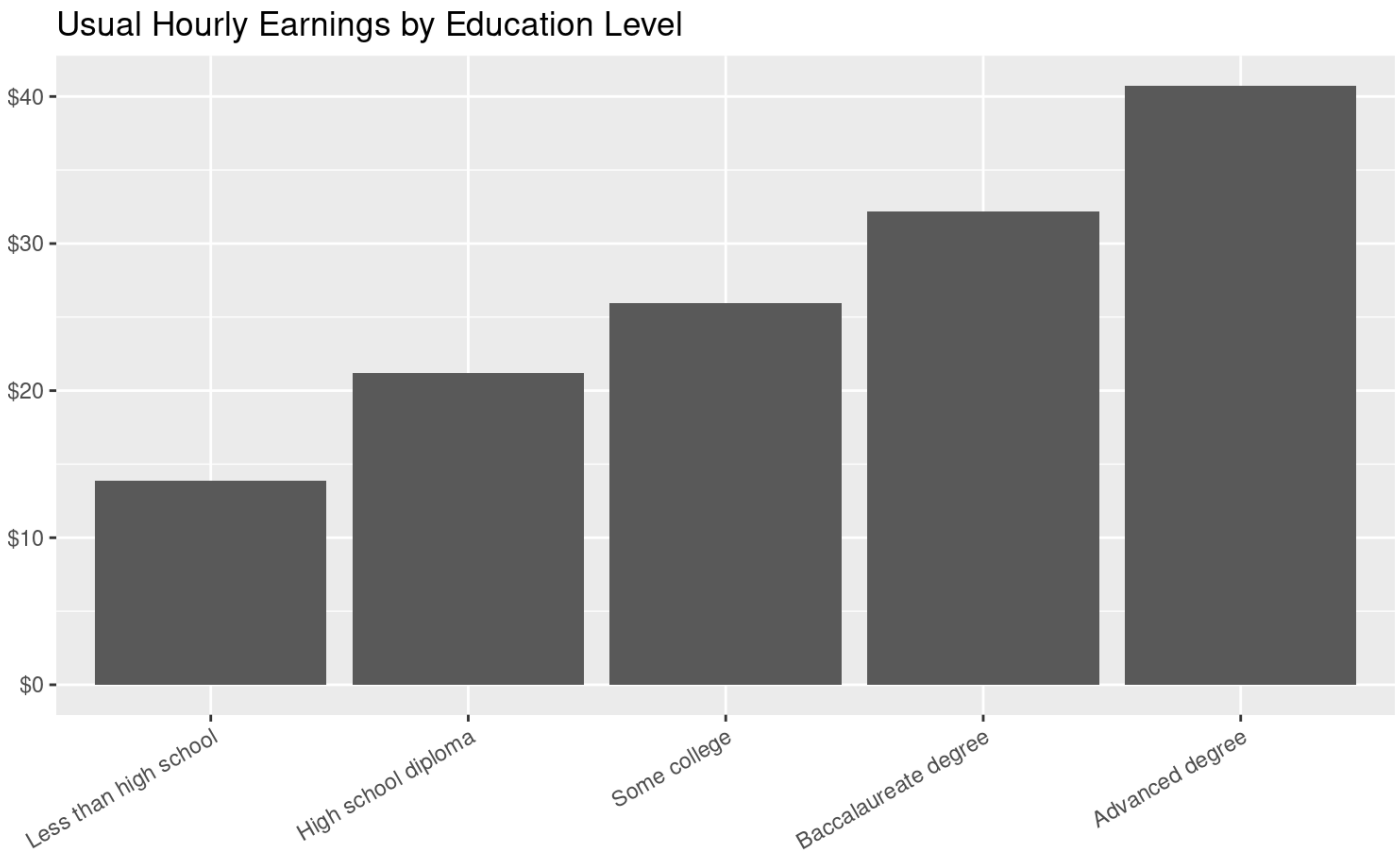
4.4 Fix Labels - Angled

The labels for the education levels are long and overlap with one another. To address this, we can specify a **theme layer**. The theme layer lets us calibrate all non-data ink on our graph. In this case, we will use the theme layer to display the x-axis labels at an angle so that the labels do not overlap and we can read them. The `theme()` function has dozens of possible parameters to calibrate everything from axes labels, titles, tick marks, legend, placement of all of these, background color and shading, and much, much more.

In the call below, we add a theme layer with a call to `theme()` to fix the x-axis labels.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar") +
  scale_y_continuous(labels=dollar) +
  labs(title="Usual Hourly Earnings by Education Level", x="", y="") +
  theme(axis.text.x=element_text(angle=30, vjust=1, hjust=1))
```

```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



We set the parameter for `axis.text.x` and call a function `element_text()` that allows us to specify an angle to write the text, vertical justification, and horizontal justification. We set the text at a 30 degree angle, and set both vertical and horizontal justification equal to `1` so that text labels are right-aligned and top-aligned, which puts the end of the labels close to the horizontal axis.

4.5 Fix Labels - Text Wrap

Alternatively, we can wrap our x-axis labels onto new lines so that they do not overwrite each other. To do this, we will change the text for the levels of the ordered factor variable, `edu`. Let us remind ourselves what are these levels:

```
levels(df$edu)
```

```
## [1] "Less than high school" "High school diploma"   "Some college"
## [4] "Baccalaureate degree"  "Advanced degree"
```

The code below calls the function `str_wrap()` to re-write the levels for `edu`:

```
levels(df$edu) <- str_wrap( levels(df$edu), width=15 )
```

The function `str_wrap()` takes as parameters a string or vector of strings and a desired width in characters. We pass the vector of strings that `levels(df$edu)` returns, we specify a maximum width of 15 characters, and `str_wrap()` outputs a new vector of strings that includes new lines as necessary to assure that no string goes over 15 characters in a single line. We save that output to `levels(df$edu)`, which overrides the existing levels.

Now let us view our levels.

```
levels(df$edu)
```

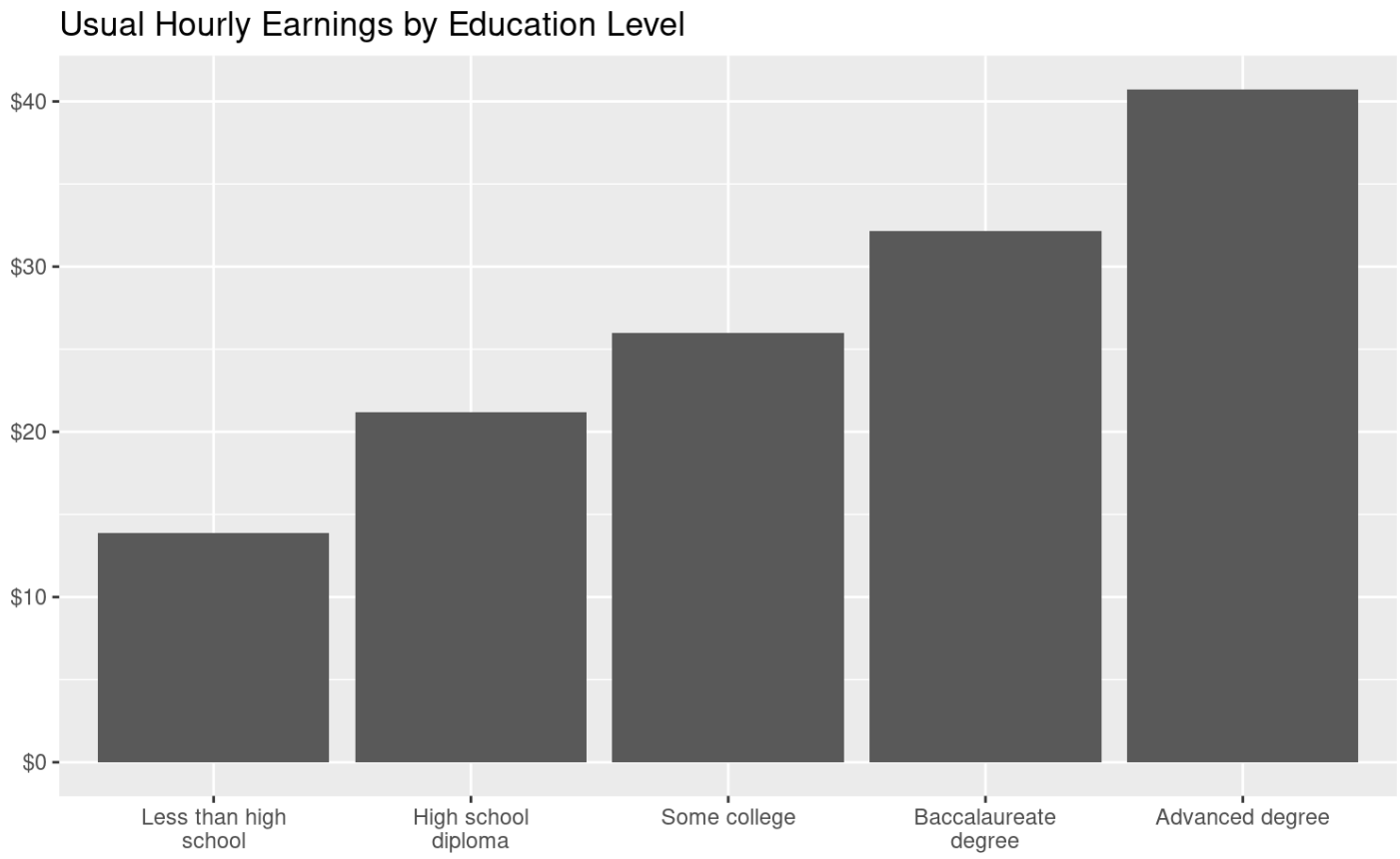
```
## [1] "Less than high\nschool" "High school\ndiploma"
## [3] "Some college"           "Baccalaureate\ndegree"
## [5] "Advanced degree"
```

The newline characters, `\n`, indicate where a new line will appear.

With the new levels created, let's re-run the code above that creates the plot.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar") +
  scale_y_continuous(labels=dollar) +
  labs(title="Usual Hourly Earnings by Education Level", x="", y="")
```

```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



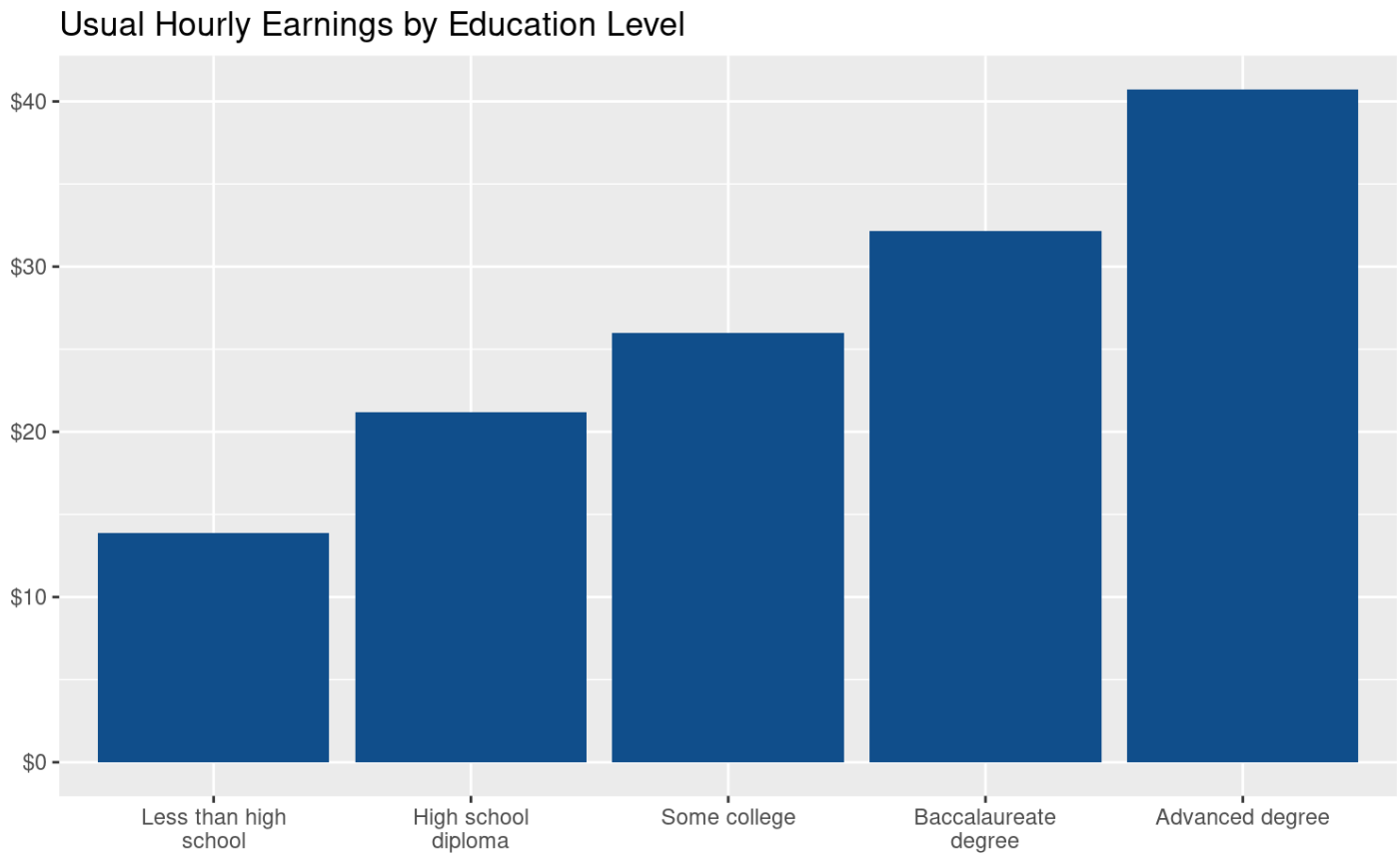
4.6 Make it Look Pretty with Color and Themes

Let's add some color to our plot. There are many ways of adding color to a plot. Sometimes you may want to map color to a variable, for example, so that different levels of education are represented by different colors. In this case, we will use just one color, but make it something prettier than gray.

We again recreate the plot from scratch, copying and pasting most of the above code. We add an optional parameters to the `stat_summary()`, `fill="dodgerblue4"`.
The `fill` parameter sets the color of the bars.
See http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf for a long-list of recognized colors.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar", fill="dodgerblue4") +
  scale_y_continuous(labels=dollar) +
  labs(title="Usual Hourly Earnings by Education Level", x="", y="")
```

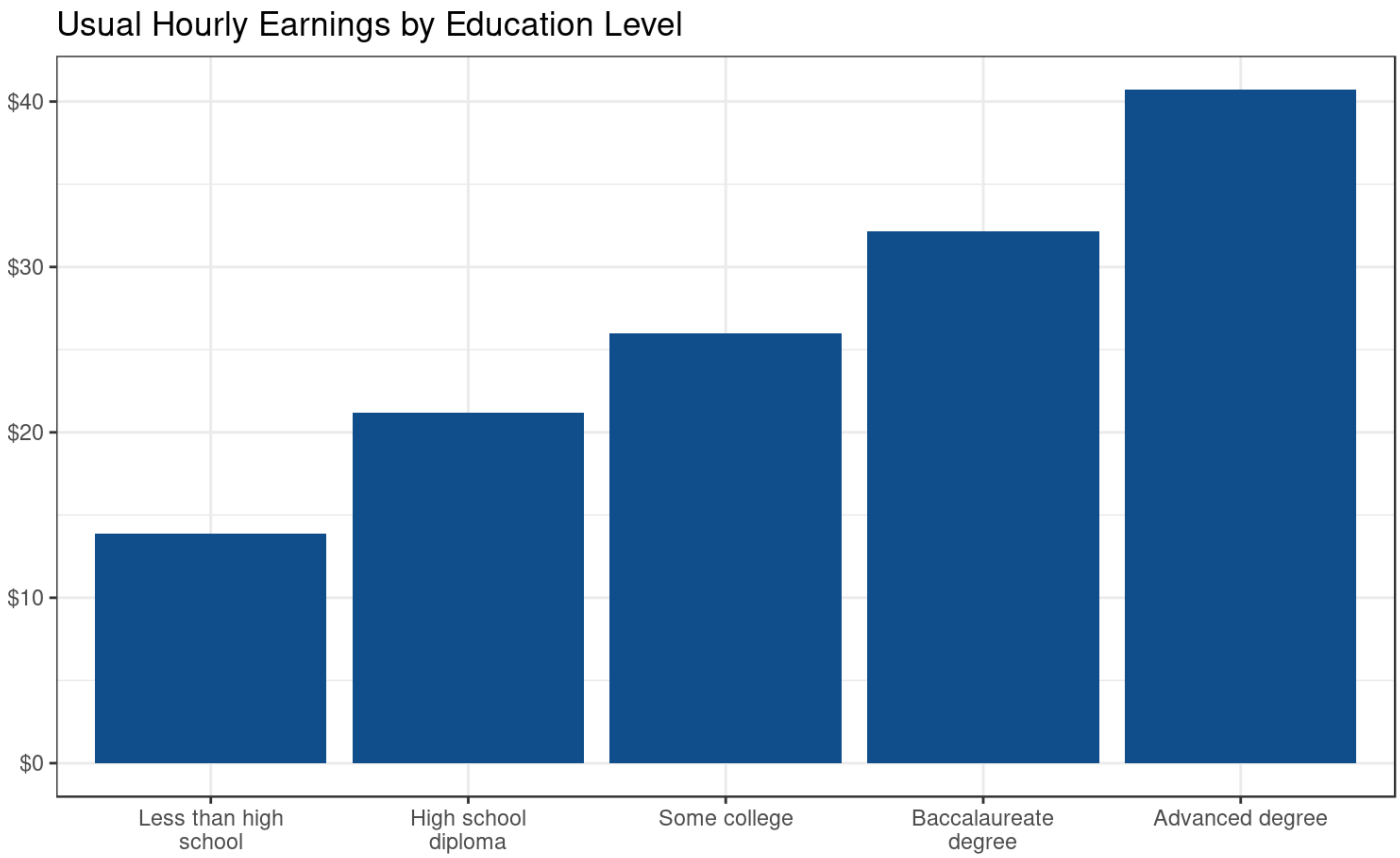
```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



Let's finally remove the gray background, in favor of a more clean background. We can set aspects of the *theme layer*. The theme layer is used to customize all the non-data ink on the graph. You can use the `theme()` function to set one or dozens of options, or use `theme_` functions that set many of these for you to create a visually attractive theme. We will use the `theme_bw()` function.

```
ggplot(data=df, mapping=aes(x=edu, y=usualhrearn)) +
  stat_summary(fun.data=mean_sdl, geom="bar", fill="dodgerblue4") +
  scale_y_continuous(labels=dollar) +
  labs(title="Usual Hourly Earnings by Education Level", x="", y="") +
  theme_bw()
```

```
## Warning: Removed 271 rows containing non-finite values (stat_summary).
```



5 Conclusion

In this tutorial, we introduced the concept of the grammar of graphics and illustrated how to use the `ggplot` package to create graphics using a minimal number of grammar layers. There are still more layers in the grammar of graphics to learn and a whole world of data visualizations and graphical customization that can be built with ggplot. This tutorial sets the framework that is used for all graphics and illustrates the grammar of graphics with two of the most simple and common plot types.

6 Exercises

1. Use the data frame in this tutorial and create the scatter plots described in the following problems.
    1. Create a scatter plot that illustrates the relationship between a respondent's age (`age`) and total medical expenses (`totmed`). What problems do you notice about the scatter plot?
    2. Recreate the plot and do the following to remedy over-plotting problems:
        - Decrease the point size (parameter is `size`) to 1.0
        - Set transparency (parameter is `alpha`) to 0.4

- Zoom in: Set the limits for ages to between 18 and 70 years and the limits for total annual medical expenses to between $0 and $30,00. Create a coordinates layer and use parameters `xlim=c(18,70)` and `ylim=c(0,30000)`.

3. Recreate the plot and set the scale labels and titles so that the plot more easily communicates to the reader what variables the graph shows and how these variables are measured. Change the labels on the vertical axis *scale* so that they are expressed in dollars. Change the text labels of the axes so that they are descriptive and create a title for the overall plot.
4. Recreate the plot from part (c) and add the geometry, `geom_smooth()`. This function creates a curve that describes the relationship between the two variables, which also includes a shaded region depicting the margin of error in this estimate for the relationship.

2. Use the data frame in this tutorial and create bar charts as described in the following problems.

    1. Create a bar chart illustrating average annual income by industry.
    2. Recreate the graph in (a) and fix the x-axis labels so that they do not overlap.
    3. Recreate the plot and set the scale labels and titles so that the plot more easily communicates to the reader what variables the graph shows and how these variables are measured. Change the labels on the vertical axis *scale* so that they are expressed in dollars. Create a descriptive title for the overall plot that makes clear what variables are being compared and remove the labels for the horizontal and vertical axes.
    4. Recreate the graph in (c) and add another statistics layer with the following function call, `stat_summary(fun.data=mean_cl_boot, geom="errorbar")`.
    The `mean_cl_boot` function computes 95% confidence bounds on the mean using the data to estimate the distribution of medical expenses rather than assuming a normal distribution. The `geom="errorbar"` specifies the geometry used to illustrate these confidence bounds. Does the graph look nice? What does it communicate to you in terms of the relationship between industry and annual income?