Dot Plots

James M. Murray, Ph.D.
University of Wisconsin - La Crosse

February 27, 2017

Note on required packages: The following code requires the packages `ggplot2` and `scales` and code for a new function called `order.factor.by()`. GGplot is a package that allows you to build highly customized graphs in a way that is consistent with the grammar of graphics. We use the Scales library with GGplot to customize the scales of our axes. The function `order.factor.by()` is one of my own functions that makes it convenient to order levels of categorical variables according to a measure of another variable, such as a mean. If you have not already done so, download, install, and load the libraries with the following code:

```
install.packages("ggplot2") # This only needs to be executed once for your machine
install.packages("scales") # This only needs to be executed once for your machine
library("ggplot2") # This needs to be executed every time you load R
library("scales") # This needs to be executed every time you load R
source("http://murraylax.org/code/R/factorby.R")
```

---

1. Introduction

In this tutorial we create dot plots to illustrate differences in distributions of a numerical variable across different levels of a categorical variable. In a previous tutorial, we used bar plots to illustrate differences in mean usual hourly earnings (the numerical variable) across different industries (the categorical variable). We included error bars on the plot to illustrate the degree of uncertainty regarding these estimates.

The dot plots we create in this tutorial will communicate much of this same information and more. Again, we will illustrate means and confidence intervals across different levels of a categorical variable. Dot plots can go farther, providing visual evidence of the sample size and the shape of the distribution of data.

Data set: The following examples use 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 such 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("http://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  16 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 ...
```

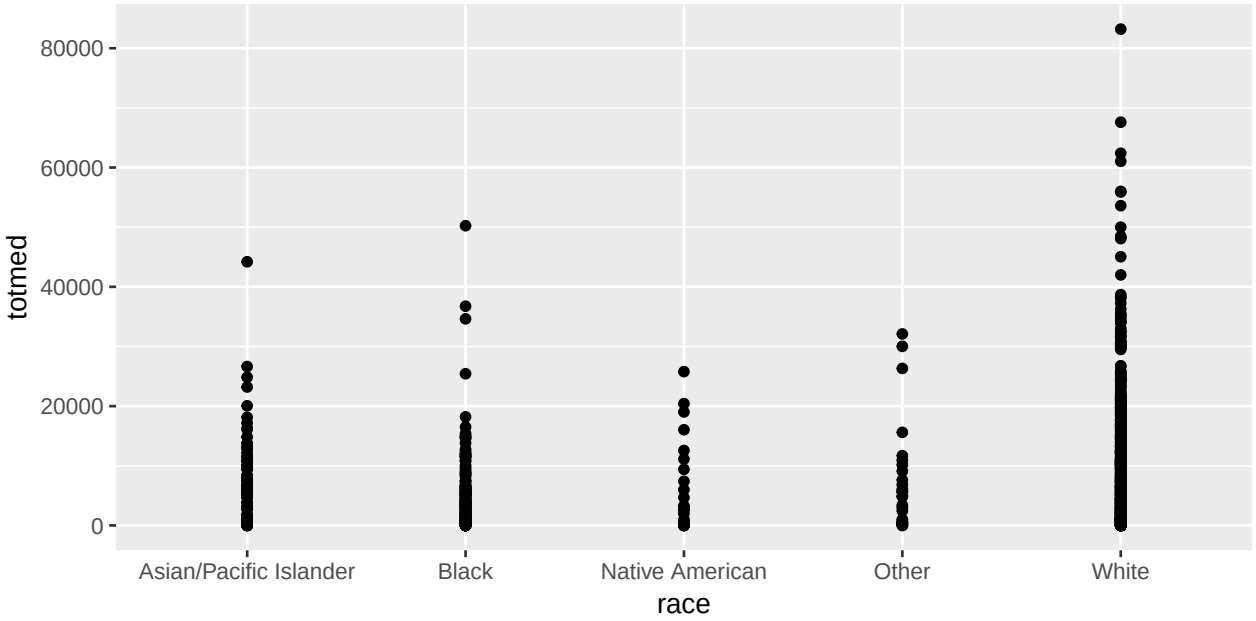
```
##  $ race       : Factor w/ 5 levels "Asian/Pacific Islander",..: 5 5 2 5 5 5 5 5 5 5 ...
##  $ empstat    : Factor w/ 4 levels "Armed Forces",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ inlf       : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ unempl     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ industry   : Factor w/ 7 levels "Agriculture, Forestry, and Fishing",..: 7 6 5 3 1 5 5 6 5 7 ...
##  $ 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 NA ...
##  $ vetran     : 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 ...
```

2. Basic Dot Plot

Let us create our first dot plot illustrating differences in total annual medical spending (`totmed`) by race (`race`). Total annual medical spending is a numerical variable measured in dollars and race is a factor (categorical) variable with descriptive strings for values.

Dot plots are similar to scatter plots in that every observation is plotted with a point. The difference is that with a dot plot, one of the variables is a categorical variable. In the code below, we create our first dot plot and save it to an object that we call `dot.p`.

```
dot.p <- ggplot(data=df, mapping=aes(x=race,y=totmed)) + geom_point()
dot.p
```



In the code above, we create the data and aethetics layers in the call to `ggplot()`. We set the data equal to our data frame, `df`, and create a mapping (i.e. the aesthetics layer) that puts `race` on the x-axis and `totmed` on the y-axis.

This plot is quite unsatifactory. We have a significant overplotting problem. Overplotting is when a number of visuals in the plot line up on top of one another making it difficult to see it all. Because there are only five categories for race, we have many hundreds of dots lining on top of one another for every category of race.

2.1 Using Jittering to Handle Overplotting

A very common way to handle overplotting with categorical variables is to use **jittering**. Jittering means adding small random noise to the data to get them to spread out so that we can better see it all. Imagine the graph above is sitting on top of a washing machine and the dots are not glued down. When we turn on the washing machine, the surface will vibrate and it will shake the dots around, spreading out from their straight line which will let us see more of the dots.

We can add jittering to the graph above by altering the position of the geometry in the call to `geom_point()`. First we create a position object we call `posn.j` that creates a jittered position.

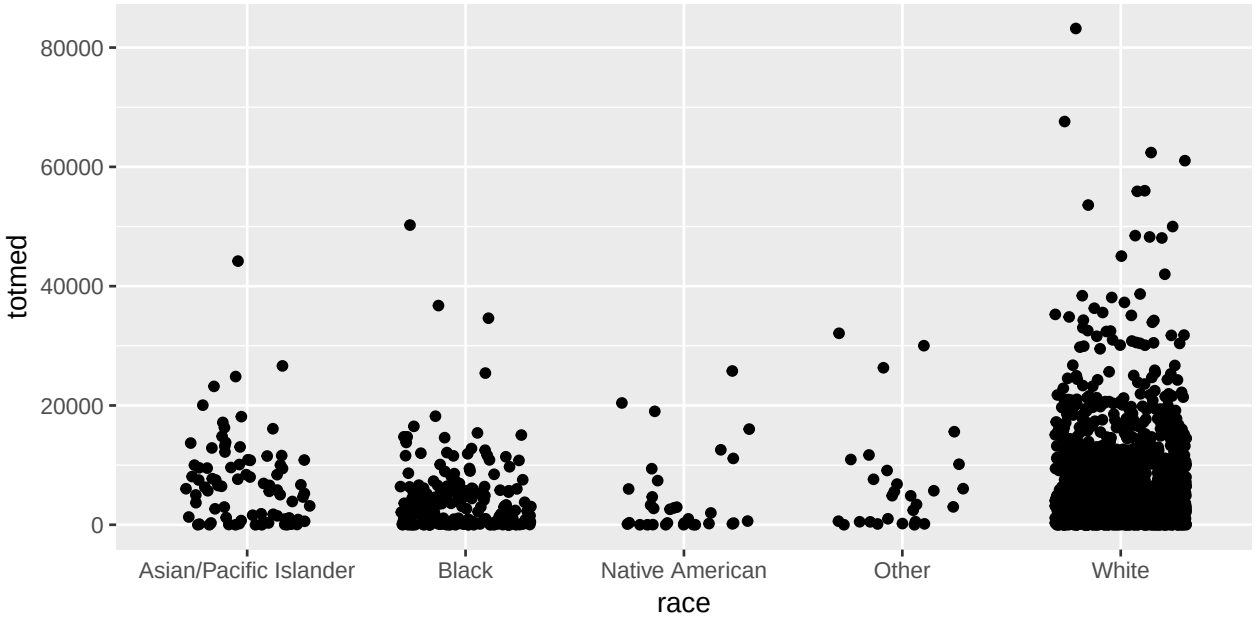
```
posn.j <- position_jitter(width=0.3,height=0)
```

The function `position_jitter()` creates the jittered position. We set the `width=0.3` which means the points will be evenly spread out to the left and to the right by at most 0.3 units. The distance between two levels (categories) of the race variable is 1 unit on the x-axis, so we don't want to move to the left or right more than 0.5, otherwise points from one race will overlap with points from another race.

We set height to 0 because we don't need to move the points up and down and the vertical distance is meaningful as it reveals the total medical expenditures for an individual. As long as we move the points from side to side and can still see what race each point aligns with, the jittering will be effective and not result in lost information.

We create our dot plot again, now setting the position in our call to `geom_point()`.

```
dot.p <- ggplot(data=df, mapping=aes(x=race,y=totmed)) + geom_point(position=posn.j)
dot.p
```



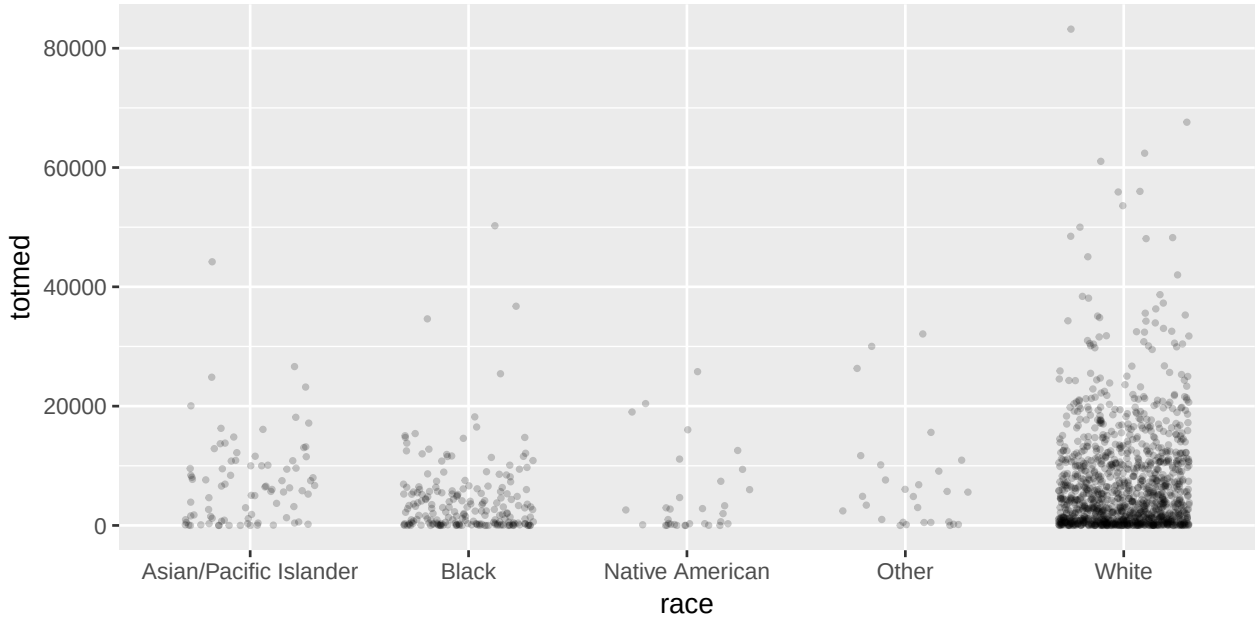
This is better. We get a better idea of how many observations there are in each racial group and get an understanding of the distribution of the data. We see there are more observations for the white racial group than any other, and that observations are clustered at the lower end of the distribution for total annual medical expenditures for every race.

2.2 More Methods to Handle Overplotting

We still have a significant overplotting problem, especially in the white racial category where we have hundreds of observations lining up on top of one another. We can employ two more strategies to remedy this problem: making our points smaller and making our points partially transparent.

In the code below, we create our dot plot again and add two more optional parameters to our call to `geom_point()`. The parameter `alpha=0.2` makes our points partially transparent so that we can see through them. A level of *alpha* equal to 1 is the default which makes our points completely opaque. A level of *alpha* equal to 0 makes the points completely invisible. We furthermore set the size equal to 0.7. With any graph, you probably need to experiment with different values for *alpha* and *size* to find values that produce a good graph.

```
dot.p <- ggplot(data=df, mapping=aes(x=race,y=totmed)) +
  geom_point(position=posn.j, alpha=0.2, size=0.7)
dot.p
```



The plot is again much improved, but we still have a lot of overlap of data near the medical expenditures equal to zero. Also notice more than the top half of the plot contains only a few points. More than half of the space on the plot is dedicated for total annual medical expenditures more than $40,000 per year, yet so few people in our sample of thousands are in this range.

If we do not wish to show the extreme outliers, and dedicate so much of our visualization to those outliers, we can change the scale of our vertical axes so that it's maximum height is a dollar value that includes all but the most extreme values.

To determine the maximum value for the vertical scale, let us compute the 95th percentile of medical expenditures. We can use the function `quantile()` to compute percentile we are interested in and set this equal to an object called `top95`.

```
top95 <- quantile(df$totmed, probs=0.95)
top95
```

```
##     95%
## 21867.5
```

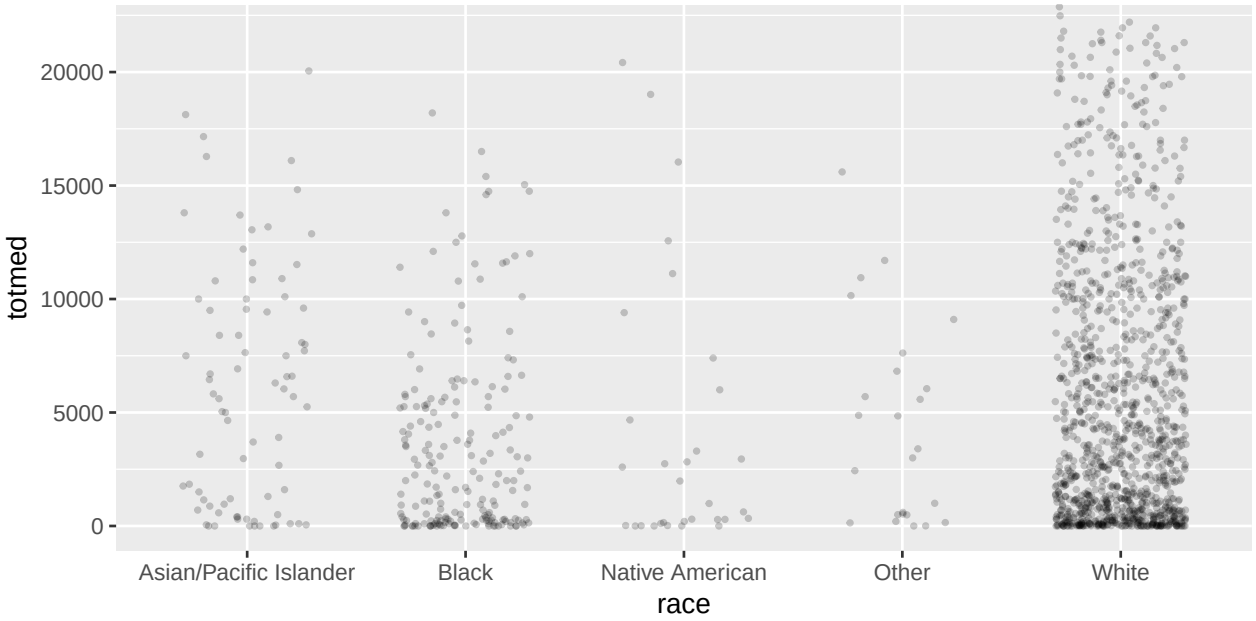
The result 21867.5 implies that 95% of the individuals in our sample pay $21,867.50 or less in annual medical expenses. The range of the vertical scale in our plot goes from $0 to more than $80,000, which implies almost

three-forths of the area if our plot is used for only 5% of our data. That 95% of the data is squished into the bottom quarter of our plot contributes to our overplotting problem.

Let us change the vertical scale to range from $0 to the value stored in `top95`, i.e. $21,867.50. While our visualization will no longer exclude the most extreme 5% of individuals, the visualization will more effectively show the distribution for the remaining 95% of the sample.

In the code below, we resuse our plot object we created above (`dot.p`) and customize the **coordinates layer** with a call to `coord_cartesian()`. In this call we set the optional parameter for `ylim` to the short list, `c(0,top95)`, which sets the minimum value to `0.0` and the maximum value to `top95`.

```
dot.p <- dot.p +
  coord_cartesian(ylim=c(0,top95))
dot.p
```



This plot is much improved. We still have a little overplotting in the white racial category near medical expenditures equal to zero, but we can see better how the distribution is clustered near zero and how it spreads out as medical expenditures increase.

2.3 Make it Pretty: Descriptive Title and Scales

Let us improve the visual appearance and the communication effectiveness by adding a descriptive plot title. Our plot title will be sufficiently descriptive that will make it unnecessary to also have the labels for the variables on horizontal and vertical axes, so we will also remove these.

We continue to add on to our existing `dot.p` ggplot object that we have created so far. In the code below we add the title and remove the axes labels with a call to the function `labs()` (short for 'labels'). In this function call, we set the title with the `title` parameter, and set the axes labels to blank strings by setting the `x` and `y` parameters.

```
dot.p <- dot.p + labs(title="Total Annual Medical Expenditures by Race", x="", y="")
```

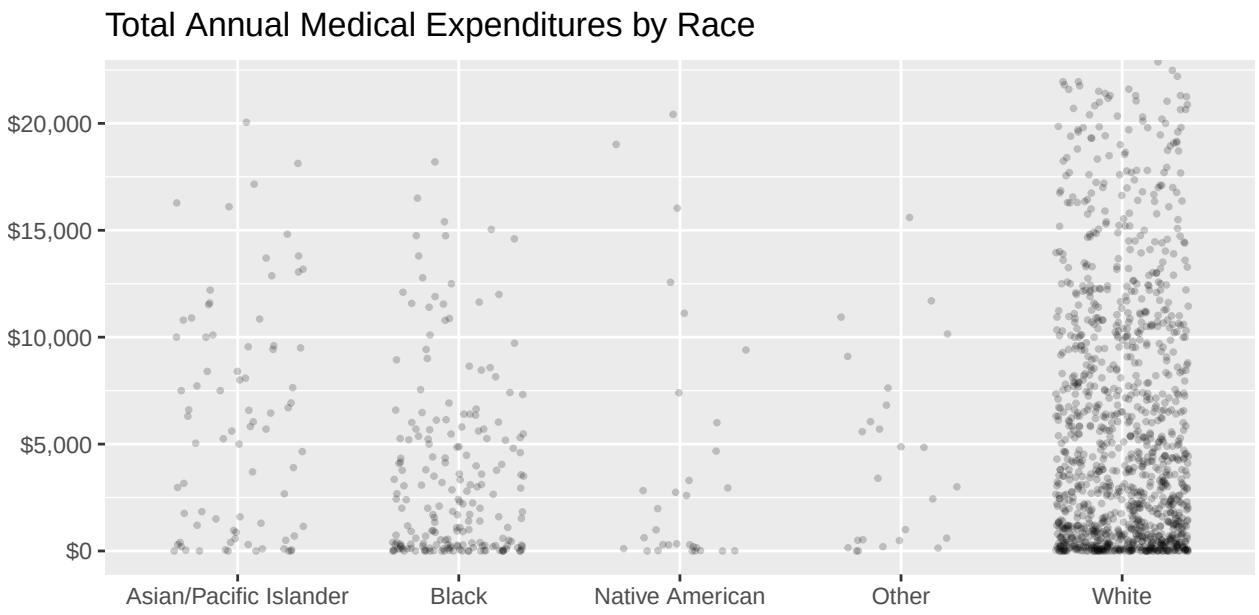
Let us also make clear that the vertical axes is expressed in dollars. The vertical scale tick labels shows some numbers between 0 and 20000, but it is not immediately clear that these are in dollars. We can customize the vertical scale with a call to `scale_y_continuous()` so that the numbers on the tick marks have a dollar sign on them, and commas placed at every third digit left of the decimal point. We set the `labels` parameter

equal to the *function* `dollar()` which accepts a number and returns a string with the number that includes the dollar sign and commas.

```
dot.p <- dot.p + scale_y_continuous(labels=dollar)
```

Here is the plot as we have created it now.

```
dot.p
```



3 Illusrating Means and Confidence Intervals

The dot plots above give a visual of the distribution of annual medical expenses by race but they do not yet effectively communicate the average level of medical expenses. In this section, we add visuals for the sample mean for the total annual medical expenses for each race and a margin of error for this estimate.

3.1 Statistics and Geometry Layers

To do this, we add a **statistics layer** to our plot with a call to `stat_summary()`. This function uses the data in the data layer, computes some summary statistic(s), and then creates a **geometry** layer to visualize the summary statistic(s).

In the code below, we again use our `dot.p` ggplot object that we have been building. We add two statistics layers, each with a call to `stat_summary()`, each which creates another geometry layer. We save the result in a new ggplot object we call `dot.mcl.p` (I chose 'mcl' to be short hand for mean-confidence-limit).

```
dot.mcl.p <- dot.p +
  stat_summary(fun.y=mean, geom="point", size=1.5) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2)
```

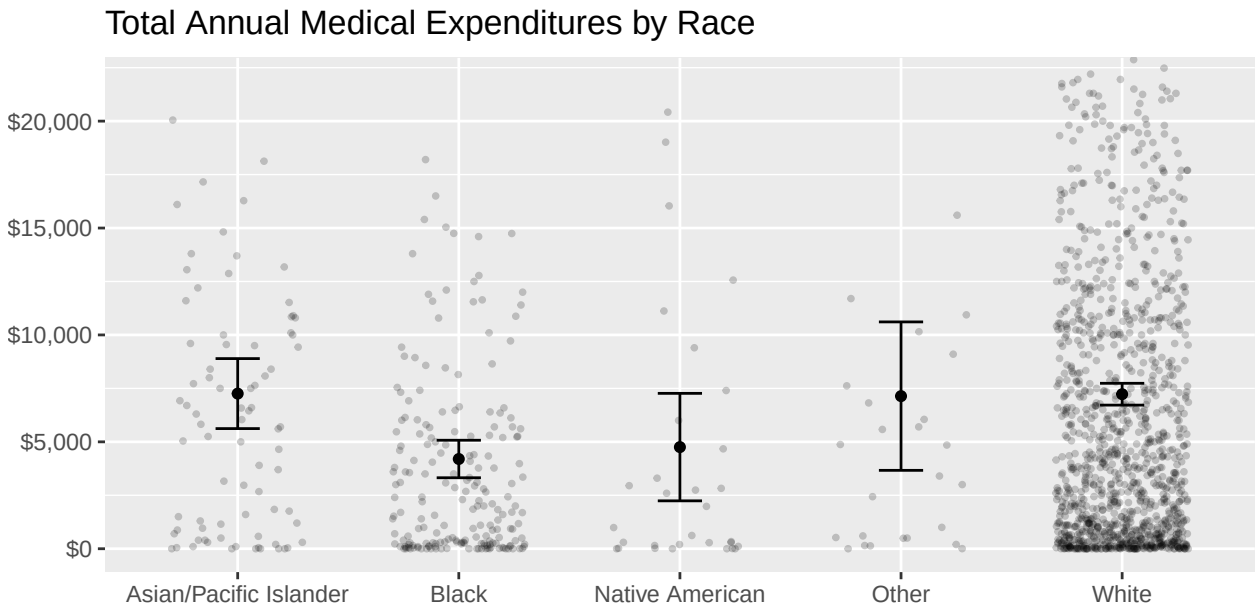
Let us first look at the first `stat_summary()` call above. This function call creates a statistics and geometry layer for the mean. We set the `fun.y` parameter equal to the *function* `mean()`, which tells `stat_summary()` to call the `mean()` function for each racial group, and use the result to put a geometry layer (a point) at that height. The `fun.y` parameter expects a function that returns a single value for the location on the y-axis, i.e. the height. The parameter `geom="point"` specifies that the geometry to place should be a point and

`size=1.5` indicates how big the point should be. We chose a point size larger than the other points on the plot so that the mean stands out.

The second `stat_summary()` call above creates statistics and geometry layers to create the error bars. We set the `fun.data` parameter equal to the *function* `mean_cl_normal()`. The parameter `fun.data` must be set equal to a function that will return three values, a y-value (like the only return value for `fun.y` above), a maximum y-value, and a minimum y-value. The function `mean_cl_normal()` computes a 95% confidence interval based on the normal distribution, returning a minimum value equal to the lower bound of the confidence interval, a maximum value equal to the upper bound, and the y-value equal to the mean. The `geom="errorbar"` specifies the geometry be an error bar, and the optional parameter `width=0.2` sets the width of the wiskers at the end of the bars.

Let us look at the new plot.

```
dot.mcl.p
```



3.2 Reordering Levels by Mean

It is often useful for visual communication if the levels for the categorical variable on the horizontal axis are in increasing order by the mean (or perhaps another summary statistic). This allows the reader to determine at a glance which racial categories are the highest and lowest in terms of medical expenses and where a particular race might fall relative to other races.

To order the levels by the mean, we need to create a new race variable that gives a different order for the levels. Currently race is a factor variable whose levels are presented in alphabetical order. We can verify this with a call to `levels()`.

```
levels(df$race)
```

```
## [1] "Asian/Pacific Islander" "Black"
## [3] "Native American"        "Other"
## [5] "White"
```

To order the levels according to the mean total medical expenditures, we need to create a new race variable that includes the same data, but specifies an order for the levels based on calls to the `mean()` function. Unfortunately, the steps to do are somewhat complicated and beyond the scope of this tutorial. I provide a

convenient function that performs this operation, though, which you can download and call into memory with the following call to `source()`:

```
source("http://murraylax.org/code/R/orderfactor.R")
```

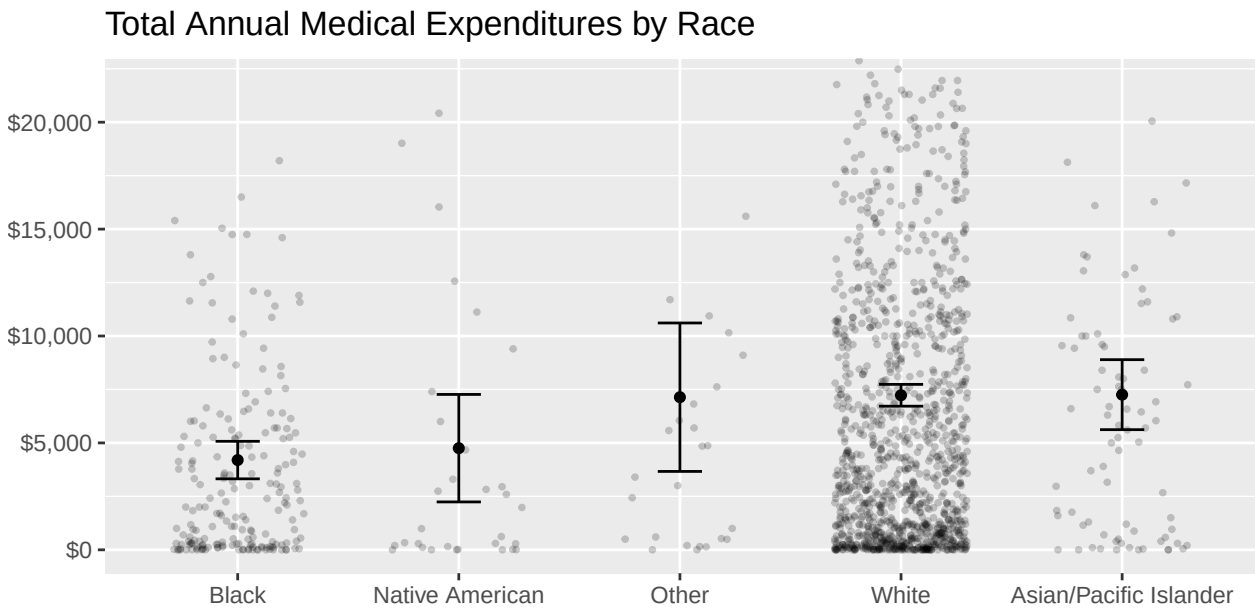
This file provides a function called `order.factor.by()` that takes four parameters: (1) `v.factor` is the factor variable to order levels, (2) `v.num` the numerical variable to use for computing the mean for each level, (3) `FUN` is the function we use for ordering - we pass along `mean`, but you can pass along another function like `median` or `sum`, and (4) `data` is the data frame to find the variables. The function returns a new factor variable whose levels are ordered appropriately, but is otherwise equal to the original factor.

We call `order.factor.by()` below and let the output override our current race variable in data frame `df`.

```
df$race <- order.factor.by(v.factor=race, v.num=totmed, FUN=mean, data=df)
```

We now need to recreate our plot from the beginning, as the data layer was the first layer we created. The code below recreates the plot using all of the steps we describe above and shows the resulting plot.

```
dot.mcl.p <- ggplot(data=df, mapping=aes(x=race, y=totmed)) +
  geom_point(position=posn.j, alpha=0.2, size=0.7) +
  coord_cartesian(ylim=c(0,top95)) +
  labs(title="Total Annual Medical Expenditures by Race", x="", y="") +
  scale_y_continuous(labels=dollar) +
  stat_summary(fun.y=mean, geom="point", size=1.5) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2)
dot.mcl.p
```



The levels of the `race` variable are now presented in increasing order according to the mean level of total medical expenditures for each race. Notice that the error bars have different lengths which communicate the level of *uncertainty* regarding each estimate of the mean. The error bar for whites is particularly small because there is such a large sample size in this category. The error bar for other races is particularly large because of the small sample size in this racial category.

3.3 Statistical Inference

Where error bars do not overlap, we have statistical evidence for differences in the means. We can see that the error bar for blacks does not overlap with the error bars for whites or Asians/Pacific Islanders. This indicates there is sufficient statistical evidence that the population mean total medical expenditure for black people is different that white people and Asian/Pacific Islanders.

A word of caution regarding making statistical inferences based on graphical visualizations: The confidence bounds in the plots are based on univariate (single-variable) confidence intervals, not on hypothesis tests or confidence intervals on the *difference* (two-variables) between means. When the categories are independent (different observations in different groups), non-overlapping confidence intervals does indicate statistically significant differences in means. When independent categories do have univariate confidence intervals overlap, but may be close together, there still may be enough statistical evidence for differences in the means. These questions can be answered by conducting the appropriate bivariate hypothesis tests.

4 Using Color to Highlight a Group

Generally speaking, you should not use different colors for different categories unless you are trying to communicate something meaningful. The levels for the race variable are already labeled on the horizontal axis, so there is no need to have different color dots or error bars for different races.

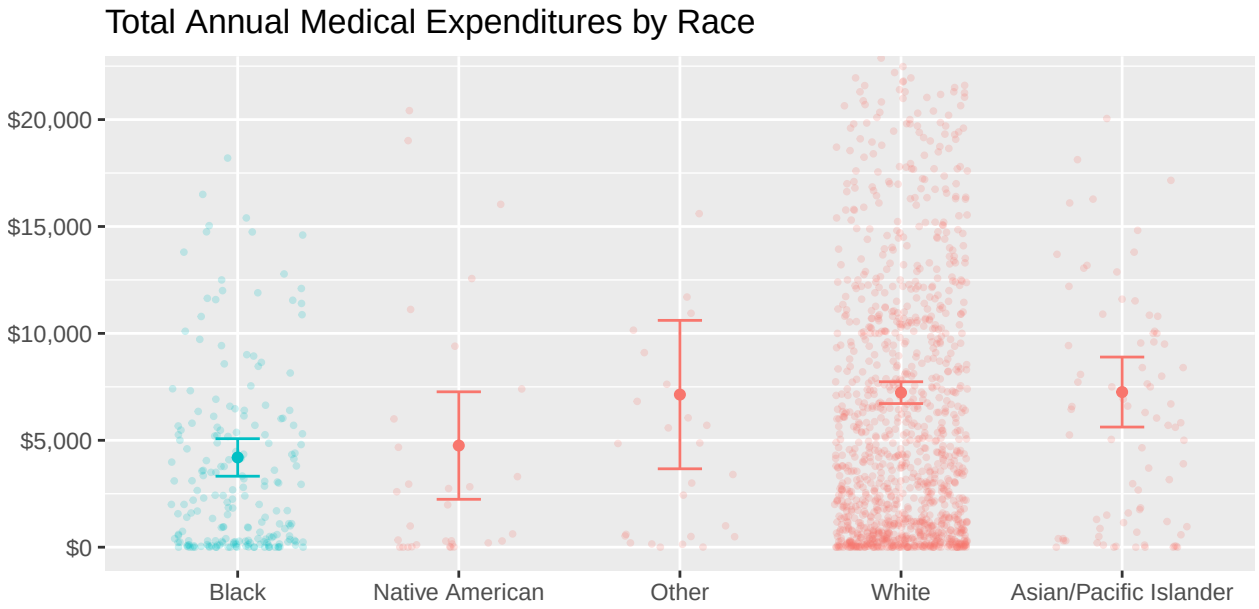
However, perhaps you wish to have one race stand out visually if the focus of your report or presentation is on one particular race and you wish to highlight statistics of that race relative to all others.

Let us suppose we wish to highlight the data for the black racial category with a different color than the other races. We first create a new boolean variable called `isblack` that is equal to `TRUE` if `race` is equal to "Black" and equal to `FALSE` otherwise. We also make sure that `isblack` is made part of the data frame object we are working with, `df`.

```
df$isblack <- df$race == "Black"
```

Next, we map our `isblack` variable to color in the *aesthetics* layer. Since we created the aesthetics layer with the original call to `ggplot()`, we need to start the code for our plot from the beginning.

```
dot.p <- ggplot(data=df, mapping=aes(x=race, y=totmed, col=isblack)) +
  geom_point(position=posn.j, alpha=0.2, size=0.7) +
  coord_cartesian(ylim=c(0,top95)) +
  labs(title="Total Annual Medical Expenditures by Race", x="", y="") +
  scale_y_continuous(labels=dollar) +
  stat_summary(fun.y=mean, geom="point", size=1.5) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  theme(legend.position="none")
dot.p
```



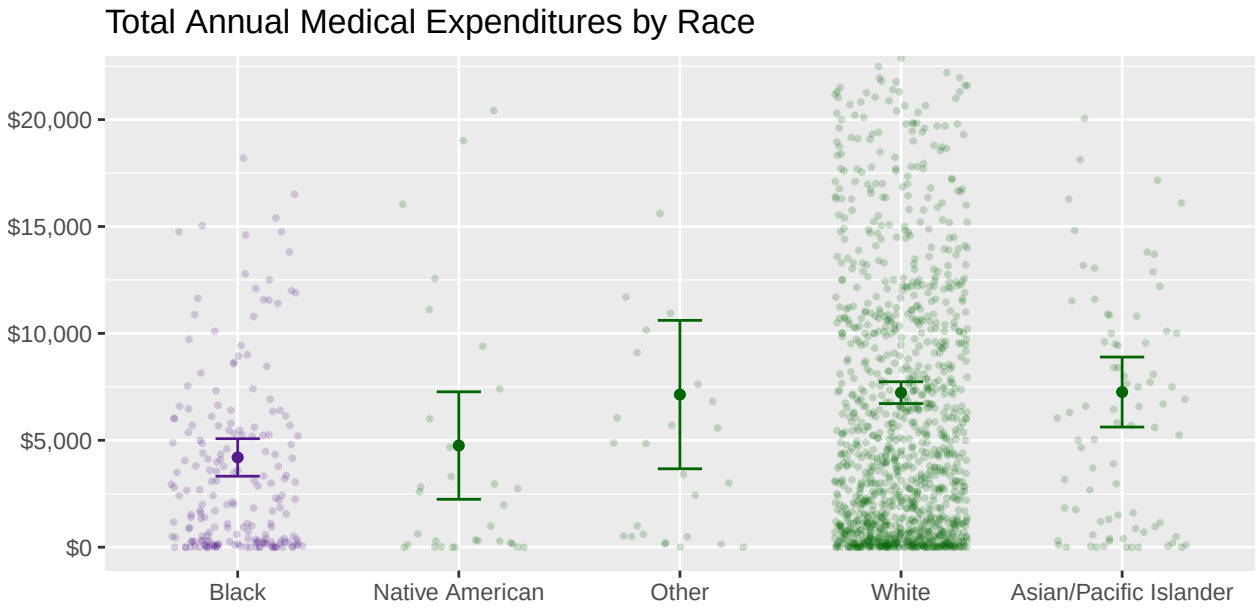
Most of the above should look familiar. There are two parts to this code that we have not already discussed. The first is in the call to `ggplot()` where we set the aesthetics layer in a nested call to `aes()`. Here we map our boolean variable identifying people who are black with the color aesthetic with `col=isblack`.

The second new aspect of this code is the setting of the legend position in the theme layer with the call `theme(legend.position="none")`. By default, when we map a variable to the color aesthetic, ggplot will create a legend. This is unnecessary, as it is clear without a legend that the black racial category is a different color than the others. Setting the `legend.position` equal to `"none"` makes it so there is no legend.

Changing the Colors

If you do not like the colors teal and pink, you can create your own color scale. Since we only have two possible outcomes for `isblack` (TRUE or FALSE) we need a color scale with only two colors. In the code below, we create a color scale called `mycols` with the colors purple and dark blue, then call `scale_color_manual()` to set the values for the color scale.

```
mycols <- c("darkgreen", "purple4")
dot.p <- dot.p + scale_color_manual(values=mycols)
dot.p
```



5 Multiple Categorical Variables

At this point we have created dot plots that communicated total annual medical expenditures across levels of one categorical variable, race. In this section, we expand out dot plots to include information on how our numerical outcome variable, medical expenditures, depends on multiple categorical variables. In the examples that follow, we look at categorical variables `race` and `sex`.

Since we have another variable, `sex`, that we need to map to an aesthetic, we need to start our plot over from the beginning. In the code below, we set up the data and aesthetic layers to serve as the foundation for our new plots. Again we map `race` to the x-axis and `totmed` to the y-axis. We also map `sex` to the `col` (color) aesthetic. We save the result in an object we call `dot.two.base.p`

```
dot.two.base.p <- ggplot(data=df, mapping=aes(x=race,y=totmed, col=sex))
```

As we have done earlier in this tutorial, let us also (1) add a title to the base plot, (2) set the scale on the vertical axis to be expressed in dollars, (3) zoom in on vertical scale so that we have a range of $0 to $21,867.50 (the 95th percentile for medical expenditures, still saved in object `top95`) and (4) set the custom colors for the color scale. This time the two colors will be mapped to values `Male` and `Female` for `sex`.

```
dot.two.base.p <- dot.two.base.p +
  labs(title="Total Annual Medical Expenditures by Race and Sex", x="", y="") +
  scale_y_continuous(labels=dollar) +
  coord_cartesian(ylim=c(0,top95)) +
  scale_color_manual(values=mycols)
```

5.1 Multiple Bars

We still do not have a plot to visualize as we have not yet set up the geometry or statistics layers. We again use `geom_point()` to set the geometry for the data points, but we have to set up the position variable first. Again we use jittering so that all the points do not line up on top of one another. It is also useful to separate the position for males versus females (the levels of our factor variable `sex`), so that they do not line up on top of one another. To create a position that includes both jittering and separation (called "dodging") we

call the function, `position_jitterdodge()`. The code below creates the appropriate position and saves in in an object called `posn.jd`.

```
posn.jd <- position_jitterdodge(jitter.height=0, jitter.width=0.5, dodge.width=0.6)
```

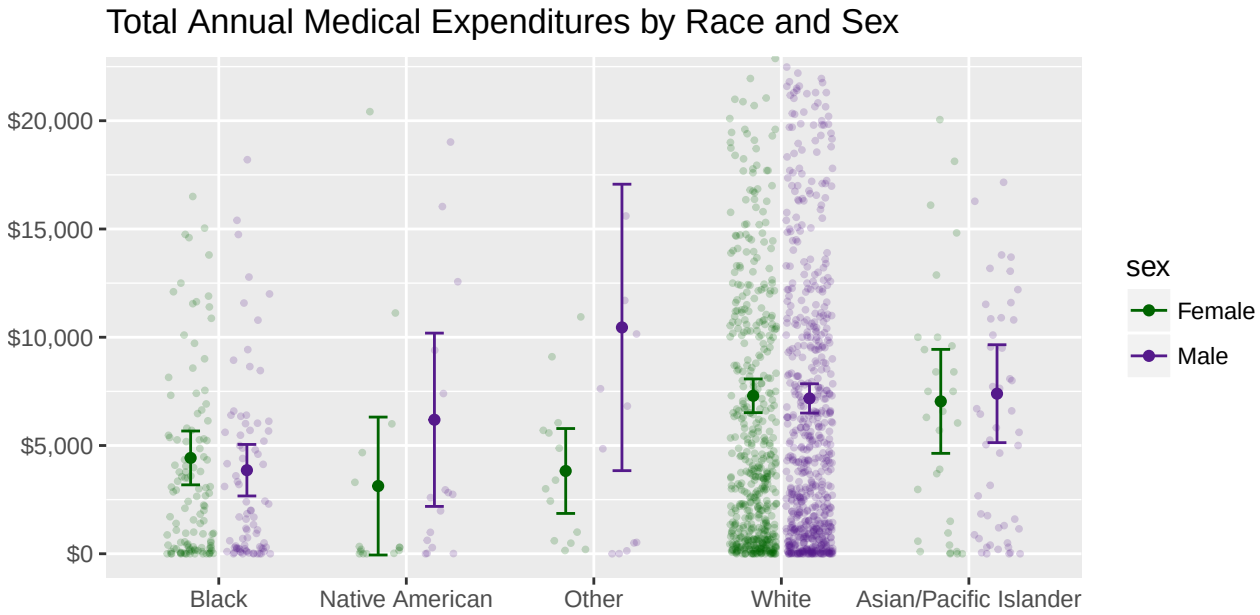
It is important to have the `dodge.width` parameter the same size or larger than the `jitter.width` parameter if you do not want any of the points between the categories for `sex` to overlap. You might need to experiment with different jitter and dodge widths to get a plot that looks good.

Let us also create a position object for the error bars. We make this a dodge position without jitter, call it `posn.d`, and set it equal to the same dodge width.

```
posn.d <- position_dodge(width=0.6)
```

Now let us finish our plot with calls to create our geometry and statistics layers. We begin with the base plot saved in `dot.two.base.p` and save our result in an object we call `dot.two.p`.

```
dot.two.p <- dot.two.base.p +
  geom_point(position=posn.jd, alpha=0.2, size=0.7) +
  stat_summary(fun.y=mean, geom="point", size=1.5, position=posn.d) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2, position=posn.d)
dot.two.p
```



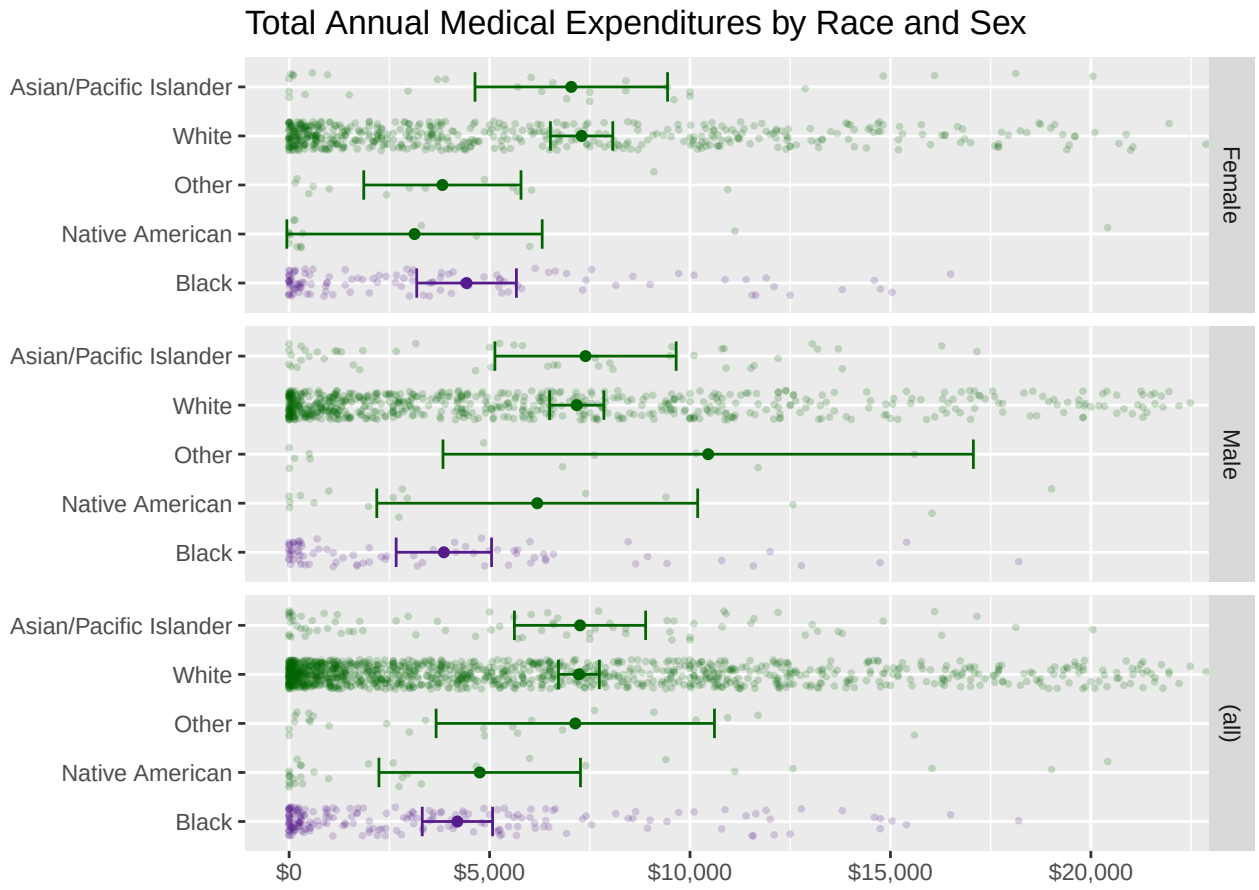
The above code should look familiar. We create the points with a call to `geom_point()` setting the position variable equal to our new dodge-jitter-type position object. We also give the points some transparency and set the size.

We create the large points for the mean with the first call to `stat_summary()`. Here we set the function for the height on the y-axis equal to the function `mean()`, set the size to be roughly twice as large as the other points on the graph, and set the position equal to the dodge position, `posn.d`.

We create the error bars with the second call to `stat_summary()`. Here we set the function for the minimum and maximum limits equal to the function `mean_cl_normal()`. We set the geometry to be error bars, set the width of the line of bars equal to 0.2, and align the error bars associated with each sex with the points with the dodge position variable `posn.d`.

5.2 Multiple Facets

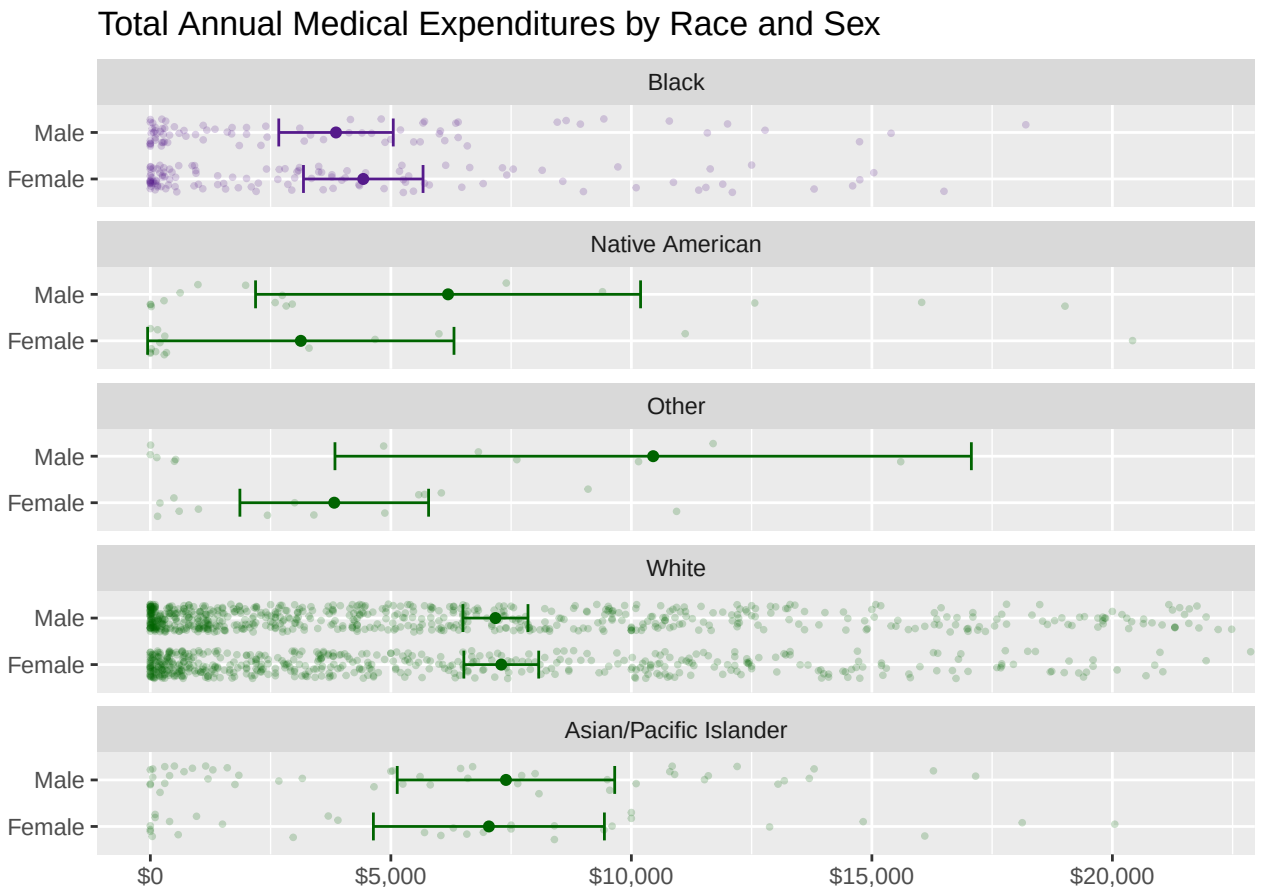
```
dot.grid.p <- ggplot(data=df, mapping=aes(x=race,y=totmed,col=isblack)) +
  labs(title="Total Annual Medical Expenditures by Race and Sex", x="", y="") +
  scale_y_continuous(labels=dollar) +
  scale_color_manual(values=mycols) +
  geom_point(position=posn.j, alpha=0.2, size=0.7) +
  stat_summary(fun.y=mean, geom="point", size=1.5) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.6) +
  coord_flip(ylim=c(0,top95)) +
  facet_grid(sex~., margins=TRUE) +
  theme(legend.position="none")
dot.grid.p
```



Or this...

```
dot.grid.p <- ggplot(data=df, mapping=aes(x=sex,y=totmed,col=isblack)) +
  labs(title="Total Annual Medical Expenditures by Race and Sex", x="", y="") +
  scale_y_continuous(labels=dollar) +
  scale_color_manual(values=mycols) +
  geom_point(position=posn.j, alpha=0.2, size=0.7) +
  stat_summary(fun.y=mean, geom="point", size=1.5) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.6) +
  coord_flip(ylim=c(0,top95)) +
  facet_wrap(~race,ncol=1) +
  theme(legend.position="none")
```

```
dot.grid.p
```


Total Annual Medical Expenditures by Race and Sex
Black
Male
Female
Native American
Male
Female
Other
Male
Female
White
Male
Female
Asian/Pacific Islander
Male
Female
$0
$5,000
$10,000
$15,000
$20,000