Graphs Part 1

Joshua F. Wiley

2019-06-10

Contents

Download the raw `R` markdown code here `https://jwiley.github.io/MonashHonoursStatistics/Graphs_Pt1.rmd`.

```
options(digits = 2)
```

```
## two new packages are: (1) cowplot and (2)
## ggthemes You can try installing by
## uncommenting the install.packages() code
## below

# install.packages('cowplot', type = 'binary')
# install.packages('ggthemes', type =
# 'binary')



## some people also report type='binary' does
## not work if that happens to you, try:
## install.packages('cowplot')
## install.packages('ggthemes')

## once installed, open the packages using the
## library() calls below

library(data.table)
```

```
library(ggplot2)
library(cowplot)
library(ggthemes)

## read in the dataset
d <- readRDS("aces_daily_sim_processed.RDS")

## small sample dataset
dt <- data.table(Age = c(20, 30, 40, 50, 60),
    Memory = c(7, 5, 6, 4.5, 4))

## convert a few variables in mtcars dataset
## into discrete factor variables
mtcars$cyl <- factor(mtcars$cyl)
mtcars$vs <- factor(mtcars$vs)
mtcars$am <- factor(mtcars$am)
```

1 *Grammar of Graphics*

`ggplot2` is based on the **g**rammar of **g**raphics, a framework for creating graphs.

The idea is that graphics or data visualization generally can be broken down into basic low level pieces and then combined, like language, into a final product.

Under this system, line plots and scatter plots are essentially the same. Both have data mapped to the x and y axes. The difference is the plotting symbol (**ge**ometries labelled `geoms` in `R`) in is a point or line. The data, axes, labels, titles, etc. may be identical in both cases.

`ggplot2` also uses aesthetics, which control how geometries are displayed. For example, the size, shape, colour, transparency level all are **aes**thetics.

2 *Univariate Graphs*

To begin with, we will make some simple graphs using `ggplot2`. The first step is specifying the dataset and mapping variables to axes. For basic, univariate plots such as a histograms, we only need to specify the dataset and what variable is mapped to the x axis. We can re-use this basic setup with different **geom**etries to make different graphs.

```
pb <- ggplot(data = mtcars, aes(x = mpg))
```

Using our basic mapping, we can "add" a histogram geometry to view a histogram[1].

[1] Histograms define equal width bins on the x axis and count how many observations fall within each bin. Bars display these where the width of the bar is the width of the bin and the height of the bar is the count (frequency) of observations falling within that range. Histograms show a univariate distribution.

```
pb + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```

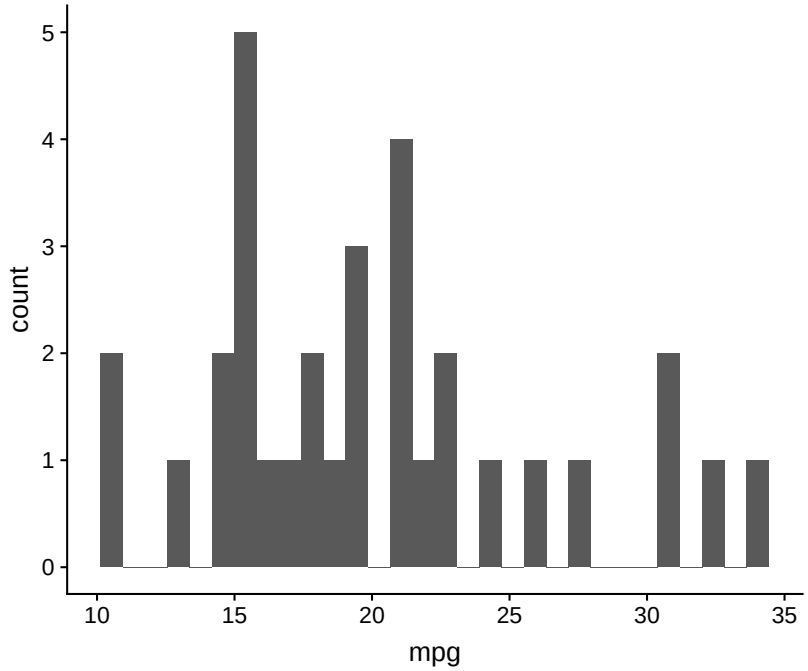


Figure 1: A histogram in ggplot2 for mpg

We also can make a density plot, which also attempts to show the distribution, but using a smooth density function rather than binning the data and plotting the frequencies. Like histograms, the height indicates the relative frequency of observations at a particular value. Density plots are designed so that they sum to one[2].

```
pb + geom_density()
```

Another type of plot are (stacked) dotplots. These are very effective at showing raw data for small datasets. Each dot represents one person. If two dots would overlap, they are stacked on top of each other. While these often are difficult to view with large datasets, for small datasets, they provide greater precision than histograms [3].

```
pb + geom_dotplot()
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```

Finally, we can make Q-Q plots, although these require sample values instead of values on just the x-axis. We use the `scale()` function

[2] Density plots attempt to provide an empirical approximation of the probability density function (PDF) for data. A probability density function always sums to one (i.e., if you integrated to get the area under the curve, it would always be one). The underlying assumption is that the observed data likely come form some relatively smooth distribution, so typically a smoothing kernel is used so that you see the approximate density of the data, rather than seeing exactly where each data point falls. Density plots show a univariate distribution.

[3] Dot plots show the raw data. The data values are on the x axis. If two data points would overlap, they are vertically displaced leading to another name: stacked dot plots. They are good for small datasets. The y axis is kind of like a discrete density. Dot plots show a univariate distribution.

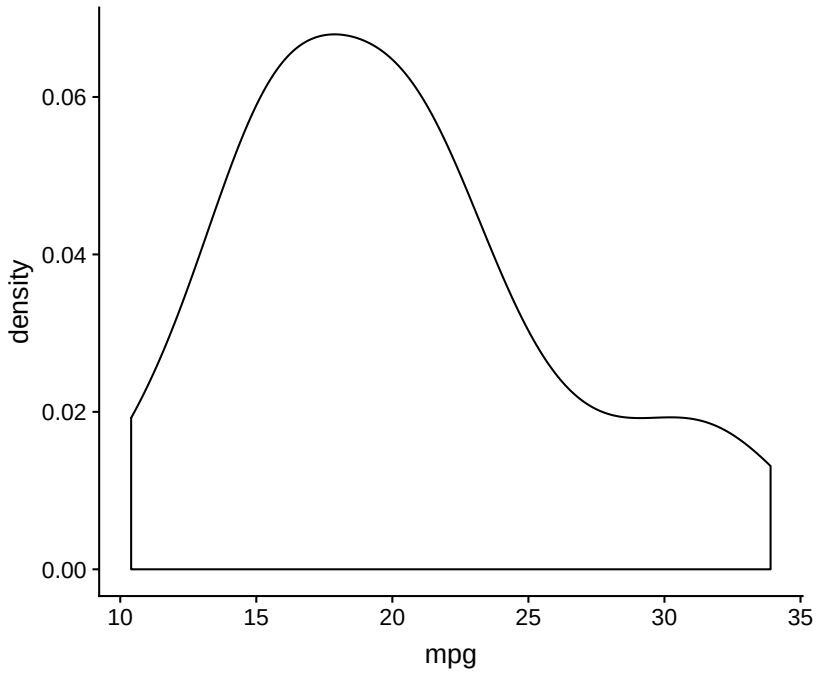


Figure 2: A density plot for mpg

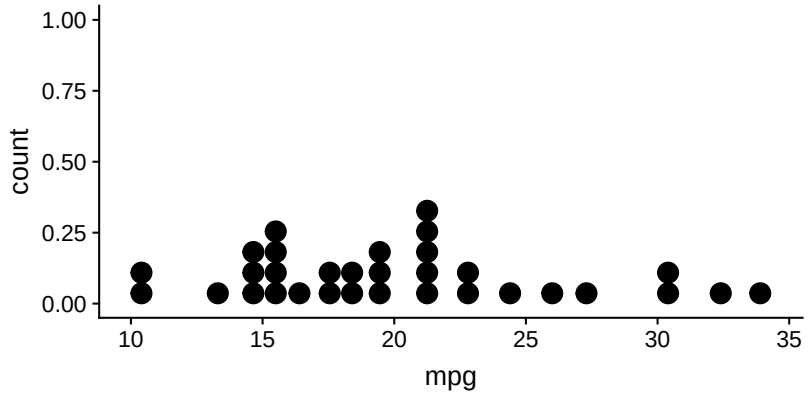
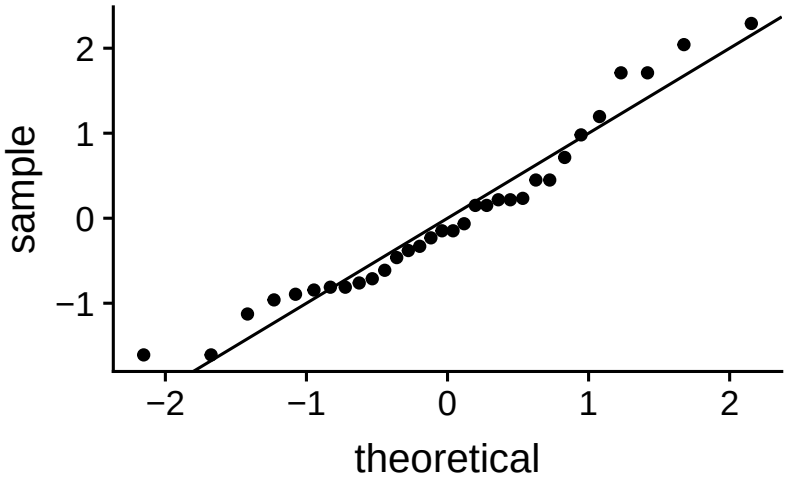


Figure 3: A dot plot for mpg

to z-score the data on the fly. Finally, we add a line with an intercept (`a`) and slope (`b`) using `geom_abline()` which is the line all the points would fall on if they were perfectly normally distributed.

```
ggplot(mtcars, aes(sample = scale(mpg))) + geom_qq() +
    geom_abline(intercept = 0, slope = 1)
```



2.1 *Mapping Additional Variables*

We can map additional variables to aesthetics such as the colour to include more information. To break the dataset down further, it is helpful to have a bit larger dataset than the 32 cars in `mtcars`. The `iris` dataset has measurements on 150 flowers from three species of iris. For density plots, separating by colour is easy, by adding the `Species` variable as an additional aesthetic[4].

[4] When there are multiple univariate distributions to view, density plots are probably one of the most efficient ways. Histograms are difficult to view because they either are stacked, which makes interpretation more difficult or dodged which is visually difficult to see, or overplotted, which can hide some of the data.

```
ggplot(iris, aes(Sepal.Length, colour = Species)) +
    geom_density()
```

For histograms, rather than control the colour of the lines, it is more helpful to control the fill colour. By default, overlapping bars are stacked on top of each other.

```
ggplot(iris, aes(Sepal.Length, fill = Species)) +
    geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick
## better value with `binwidth`.
```

Overlapping bars also can be dodged instead of stacked.

```
ggplot(iris, aes(Sepal.Length, fill = Species)) +
    geom_histogram(position = "dodge")
```

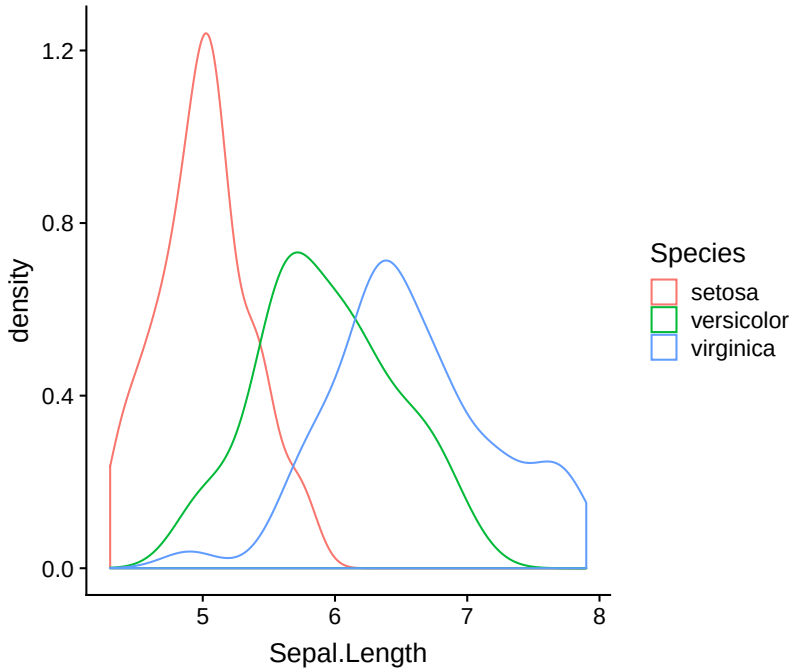


Figure 4: Density plot coloured by Species for Sepal Length

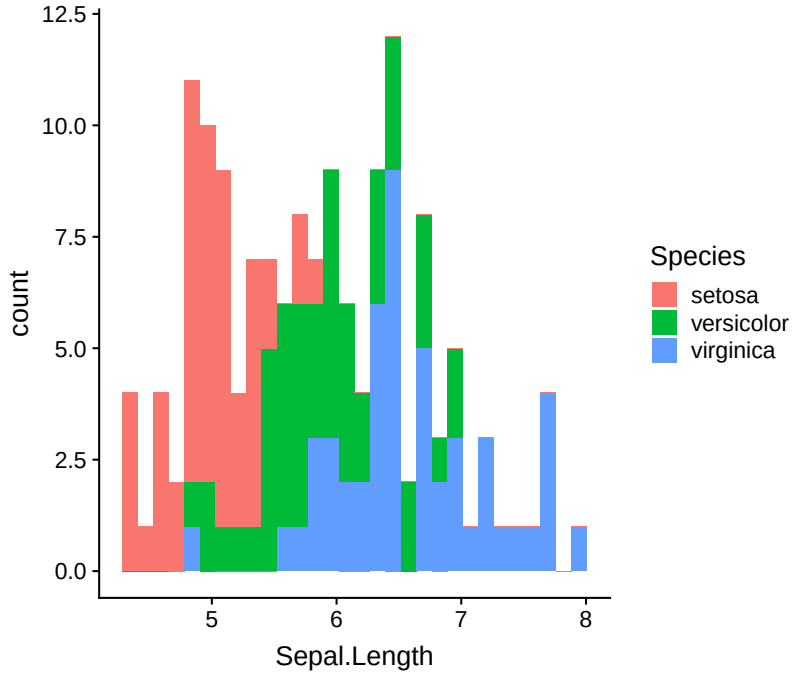


Figure 5: Histogram coloured by Species for Sepal Length

```
## 'stat_bin()' using 'bins = 30'. Pick
## better value with 'binwidth'.
```

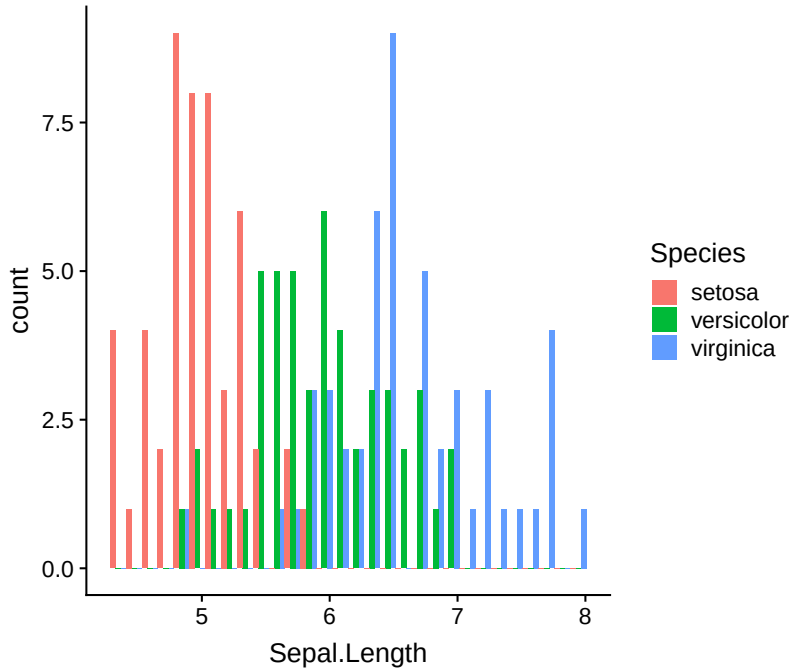


Figure 6: Dodged histogram coloured by Species for Sepal Length

Multiple geometric objects can be combined. The following figure combines dot and density plots into one[5].

```
ggplot(iris, aes(Sepal.Length, colour = Species)) +
    geom_dotplot() + geom_density()
```

```
## 'stat_bindot()' using 'bins = 30'. Pick better value with 'binwidth'.
```

[5] Note that when combining density and dot plots the scales do not always work out well. For example, if the range of the dotplot on the y axis is 0 to 1 and the range of the density plot y axis is 0 to .005, the two will not be easily plotted on the same figure. When they are more similar, however, they can be.

While the colour works well for the density plots, filling would work better for the dotplot. We can map separate aesthetics to each geometric object.

```
ggplot(iris, aes(Sepal.Length)) + geom_dotplot(aes(fill = Species)) +
    geom_density(aes(colour = Species))
```

```
## 'stat_bindot()' using 'bins = 30'. Pick better value with 'binwidth'.
```

2.2 *Try It - Univariate*

Using the `mtcars` dataset:

1. make a histogram for the variable: `qsec`.

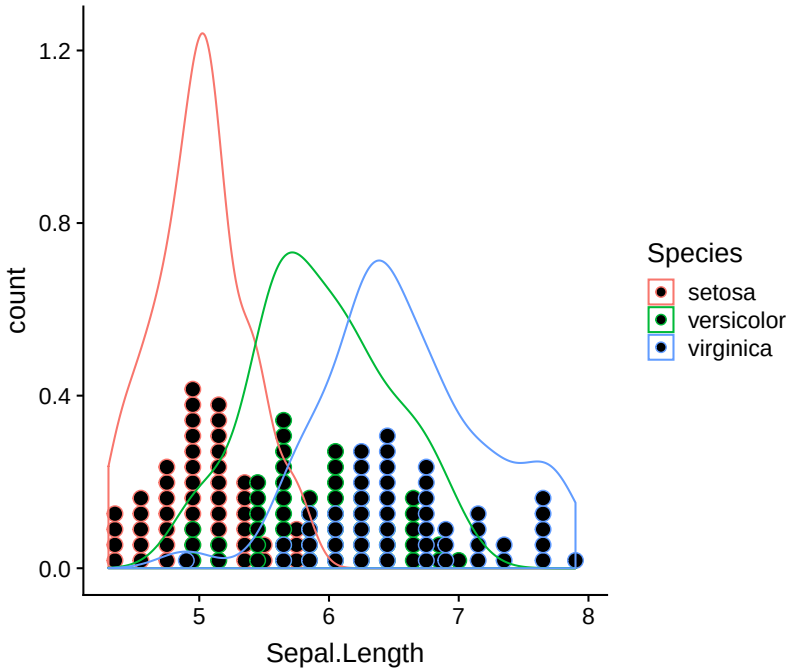


Figure 7: Density and dot plot plot coloured by Species for Sepal Length

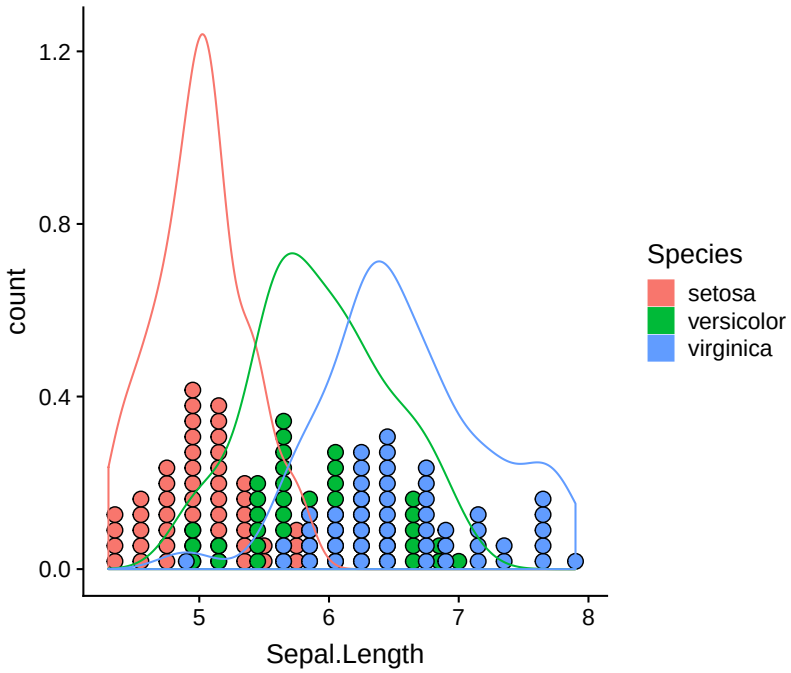


Figure 8: Density and dot plot plot coloured by Species for Sepal Length

2. make a QQ plot for qsec
3. Make a dotplot and density plot for: `disp` *separated* by `am`[6]:

[6] The dotplot and density plot do not show up well together with these particular data as the density plot has very small y axis values, relative to the dotplot.

```
## histogram code here (qsec)

## QQ plot code here (qsec)

## dotplot and density plot code here (disp by
## am)
```

3 *Bivariate Graphs*

We can make bivariate plots by mapping variables to both the x and y-axis. For a scatter plot, we use point geometric objects.

```
ggplot(mtcars, aes(x = mpg, y = hp)) + geom_point()
```

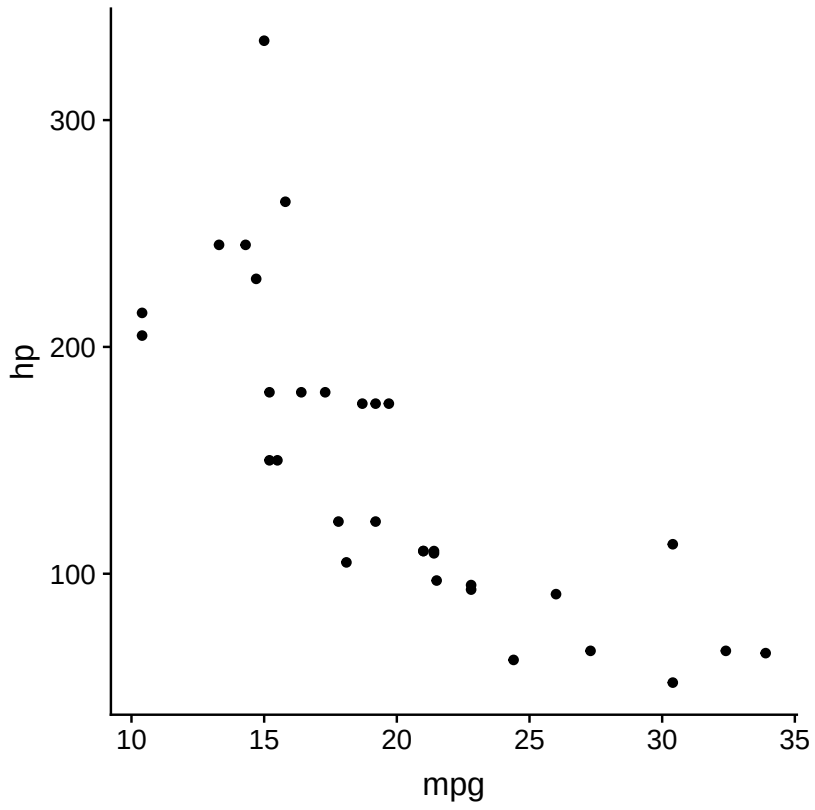


Figure 9: Scatter plot of mpg and hp

We also can use lines for bivariate data. For this example, we will use a small made up dataset. Compared to a scatter plot, we only change point to line geometric objects.

```
ggplot(dt, aes(Age, Memory)) + geom_line()
```

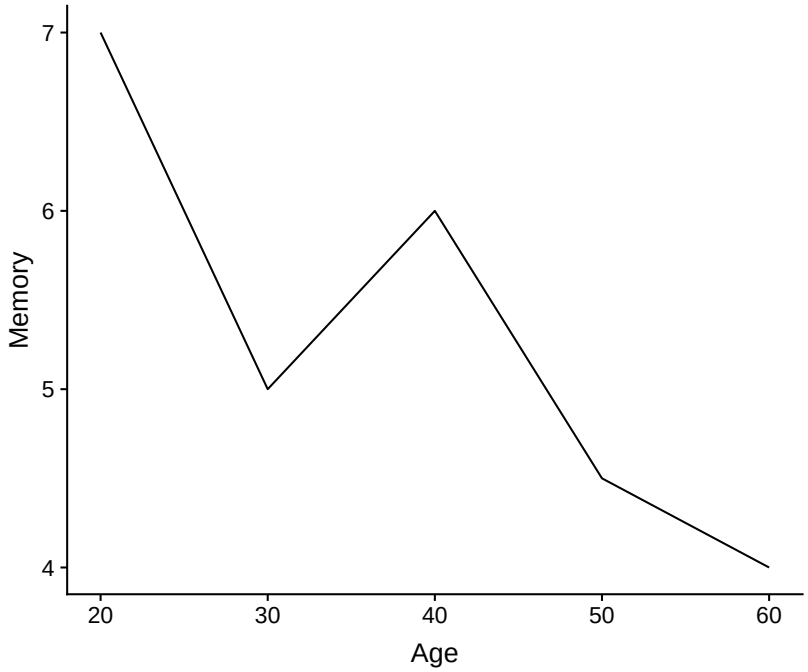


Figure 10: Line plot of age and memory

With relatively discrete x axis, we can use a barplot for bivariate data. By default, `geom_bar()` calculates the count of observations that occur at each x value, so if we want our values to be the actual bar height, we set `set = "identity"`.

```
ggplot(dt, aes(Age, Memory)) + geom_bar(stat = "identity")
```

The grammar of graphics is designed to be like sentences, where you can add or modify easily. For example, "There is a ball." or "There is a big, red, striped ball." are both valid sentences. So to with graphics, we often can chain pieces together to make it more nuanced. In `R` we just "add" more by separating each component with `+`. Note that most argument names (i.e., `data =`, `mapping =`) are not strictly required. `R` will match input to the correct argument by position, often. The two sets of code below yield the same plot. In the first, we explicitly label all arguments, in the second we rely on position matching. Positional matching does not always work, for example we still must specify `size =` because we don't always provide input for every argument, instead relying on defaults and only changing the specific arguments we want changed from defaults.

```
ggplot(data = dt, mapping = aes(x = Age, y = Memory)) +
```

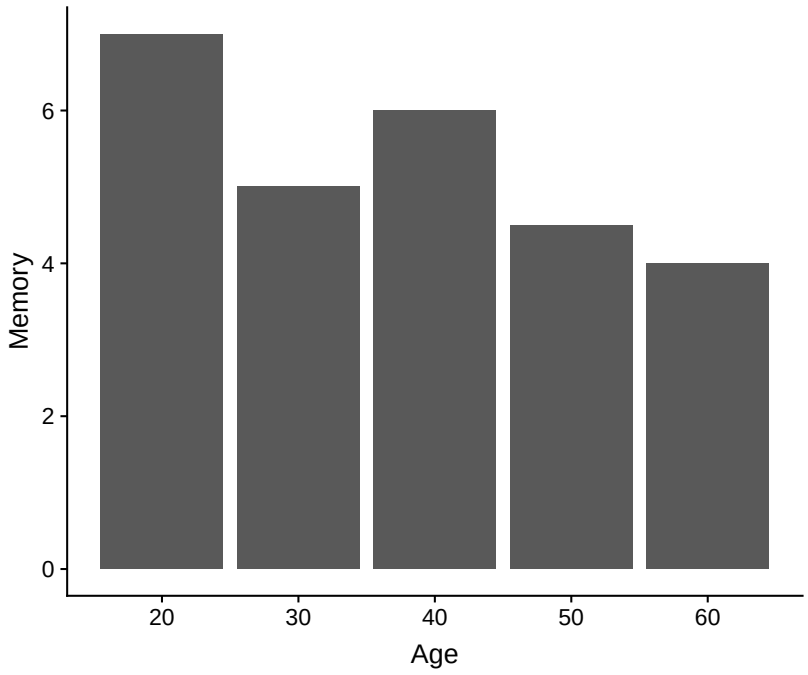


Figure 11: Bar plot of age and memory

```
    geom_bar(mapping = aes(fill = Age), stat = "identity") +
    geom_line(size = 2) + geom_point(size = 6)

ggplot(dt, aes(Age, Memory)) + geom_bar(aes(fill = Age),
    stat = "identity") + geom_line(size = 2) +
    geom_point(size = 6)
```

4 *Improving Data Visualization*

Before we continue examining graphs, it is helpful to think a bit more about what makes a good graph or good data visualization.

Edward Tufte and William Cleveland are two authors who have written extensively on data visualization and how to make good graphs. There work is well worth reading to improving understanding on how to efficiently convey data graphically.

- Tufte: `https://www.edwardtufte.com/tufte/`
- Cleveland: `http://www.stat.purdue.edu/~wsc/`

A few additional pages with good discussions on making good graphs: Graphic Design Stack Exchange 1 and Graphic Design Stack Exchange 2

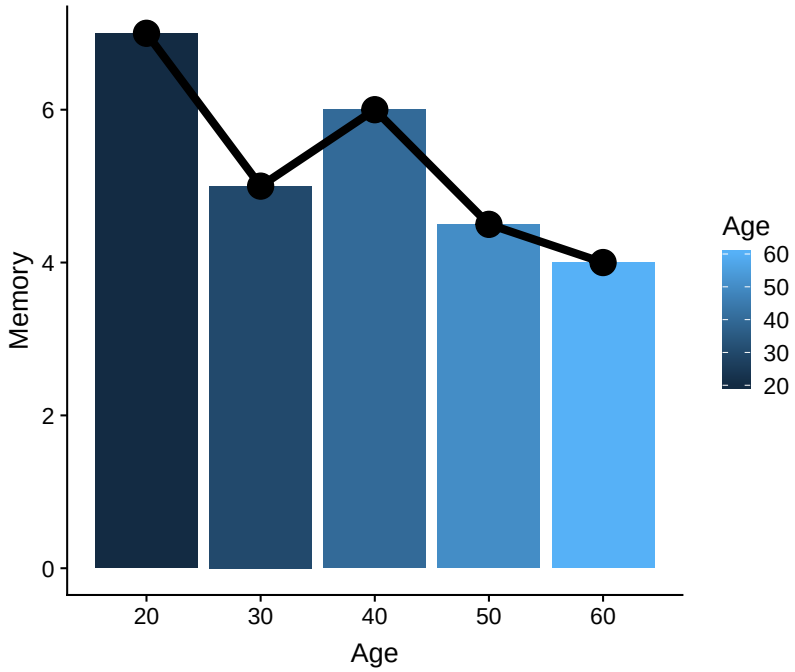


Figure 12: Bar plot of age and memory

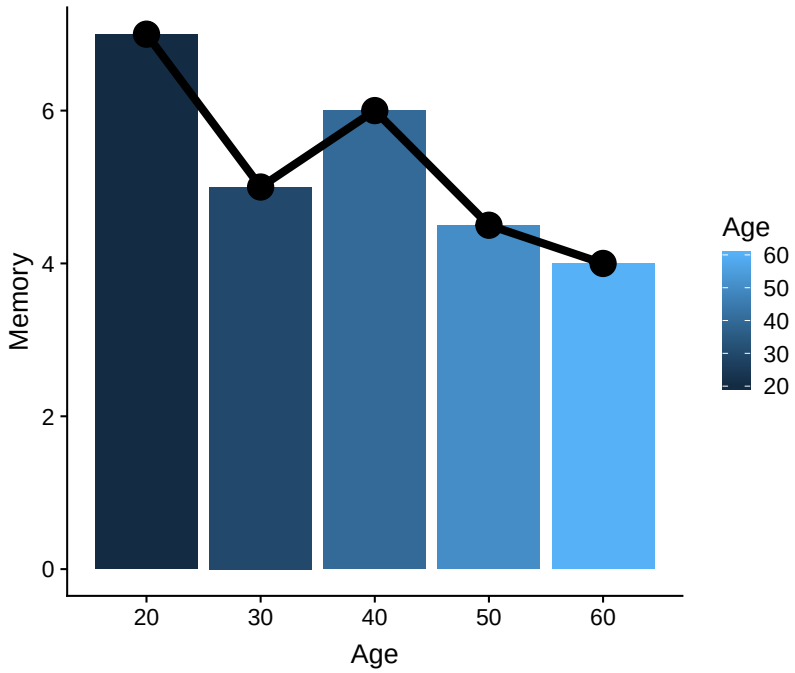


Figure 13: Bar plot of age and memory

One key principle that Tufte emphasises is the data to ink ratio. This ratio is how much data is conveyed versus ink used, and Tufte argues to try to maximize this (i.e., more data, less ink). To see this, consider the following graph, which is a fairly standard way stats programs (Excel, etc.) tend to make barplots.

```
ggplot(dt, aes(Age, Memory)) + geom_bar(stat = "identity") +
    theme_bw()
```

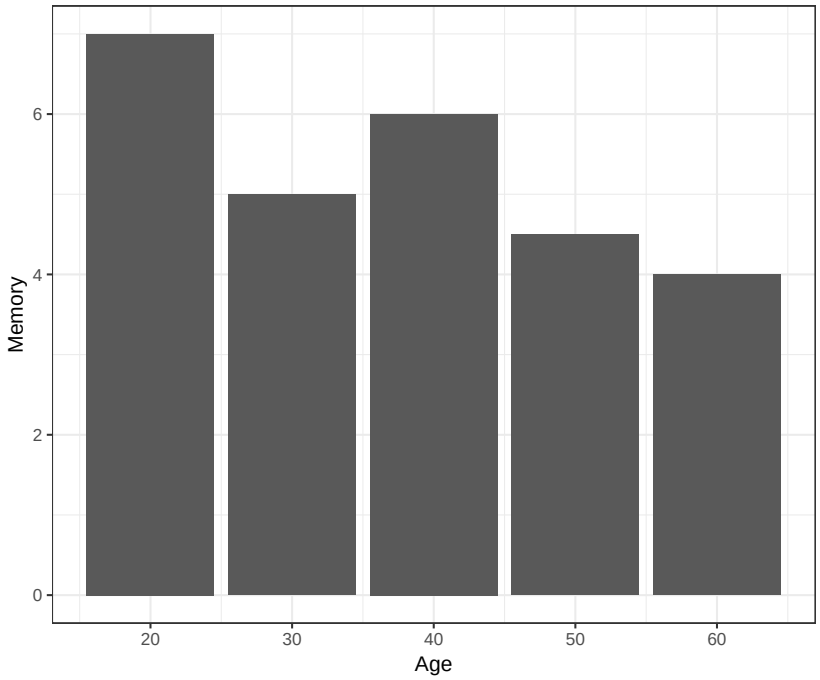


Figure 14: Bar plot of age and memory - Step 1

For starters, the borders tell us nothing. They edge the space but convey no information. This can be cleaned up using a different theme. Once we loaded the `cowplot` package, it automatically set its own theme as the default, so if we leave off the `theme_bw()` we get a cleaner graph.

```
ggplot(dt, aes(Age, Memory)) + geom_bar(stat = "identity")
```

However, there are still some borders, which we can strip away with no loss of data, but reducing the ink.

```
ggplot(dt, aes(Age, Memory)) + geom_bar(stat = "identity") +
    theme(axis.line = element_blank())
```

Next, think about what data are conveyed in this graph. The bars capture two pieces of information: (1) the age and (2) the memory

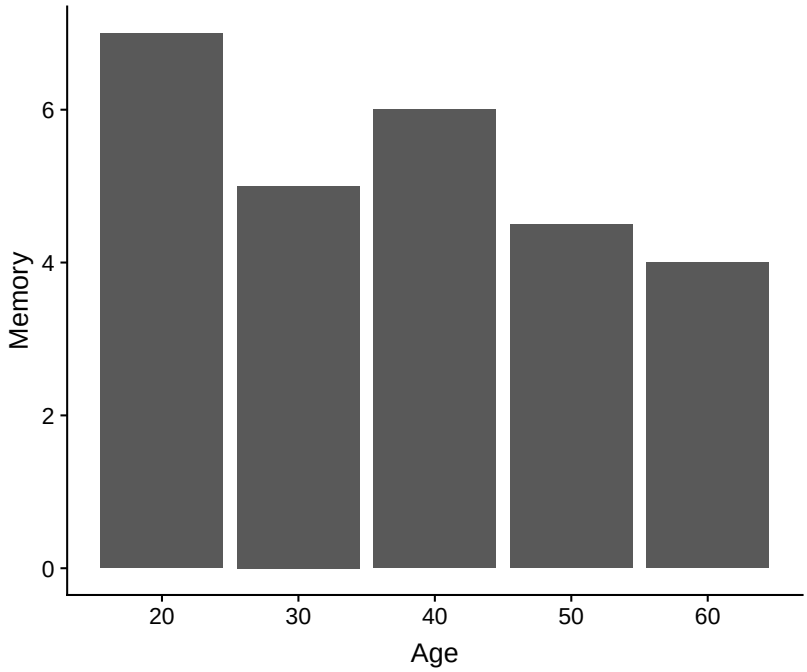


Figure 15: Bar plot of age and memory - Step 2

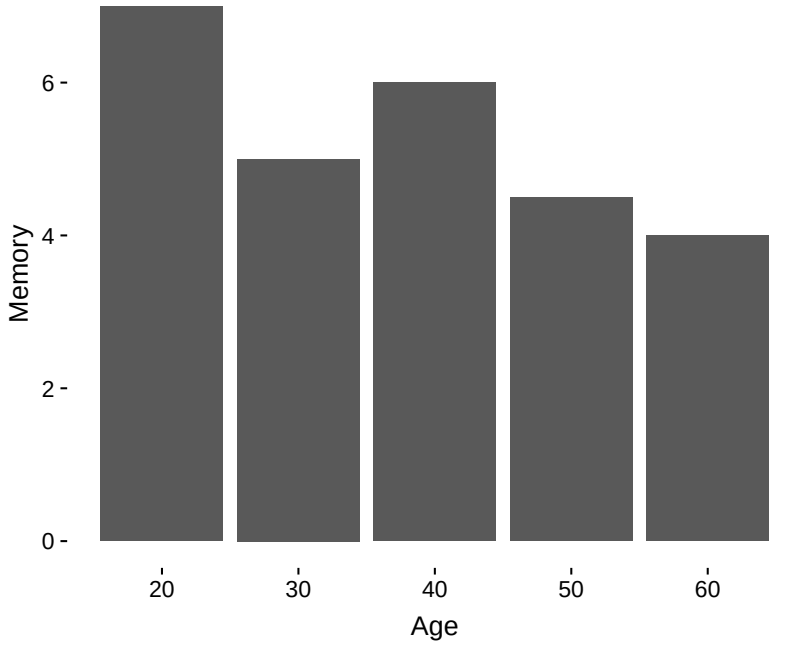


Figure 16: Bar plot of age and memory - Step 2

at that age. The only pieces of the bars we really need are the top. The rest of the bars take up a lot of ink, but convey no data. Points can do this more efficiently. The chart that follows has a much higher data-to-ink ratio as it is stripped back nearly to just the data.

```
ggplot(dt, aes(Age, Memory)) + geom_point(size = 4) +
    theme(axis.line = element_blank())
```

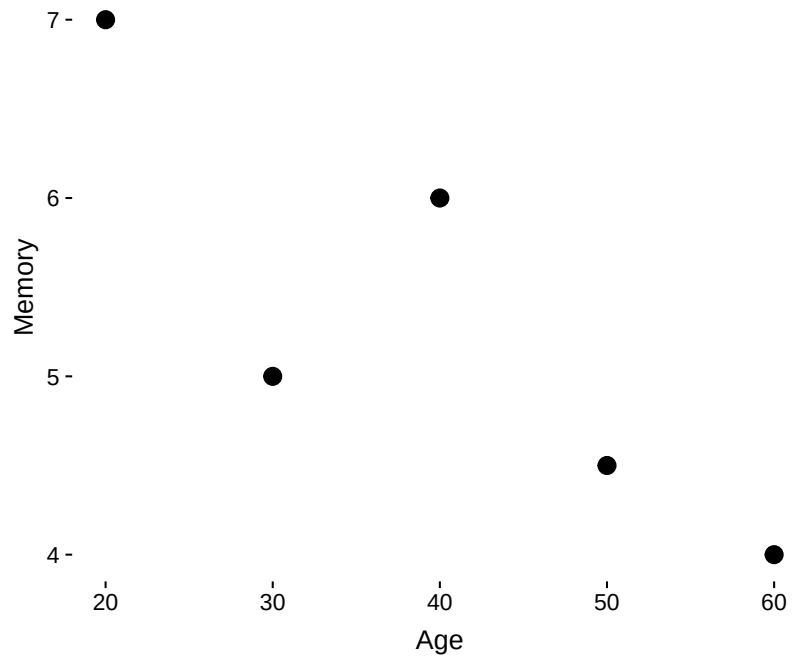


Figure 17: Dot plot of age and memory - Step 3

Depending on the number of data points, one may push a bit further. Many people in practice find they are unfamiliar with these sort of graphs and at first it can take a bit longer to read. We are trained and used to seeing plots with "chartjunk" and low data to ink ratios. However, a chart like this is a far more condensed display of data and removes distractions to really highlight the raw data or results.

```
ggplot(dt, aes(Age, Memory)) + geom_point(size = 4) +
    geom_text(aes(y = Memory + 0.5, label = Memory)) +
    theme(axis.line = element_blank(), axis.ticks.x = element_blank(),
        axis.ticks.y = element_blank(), axis.text.y = element_blank(),
        axis.title.y = element_blank()) + ggtitle("Working memory levels across the lifespan")
```

Another example is the popular scatter plot. By default scatter plots already have relatively high data to ink ratios.

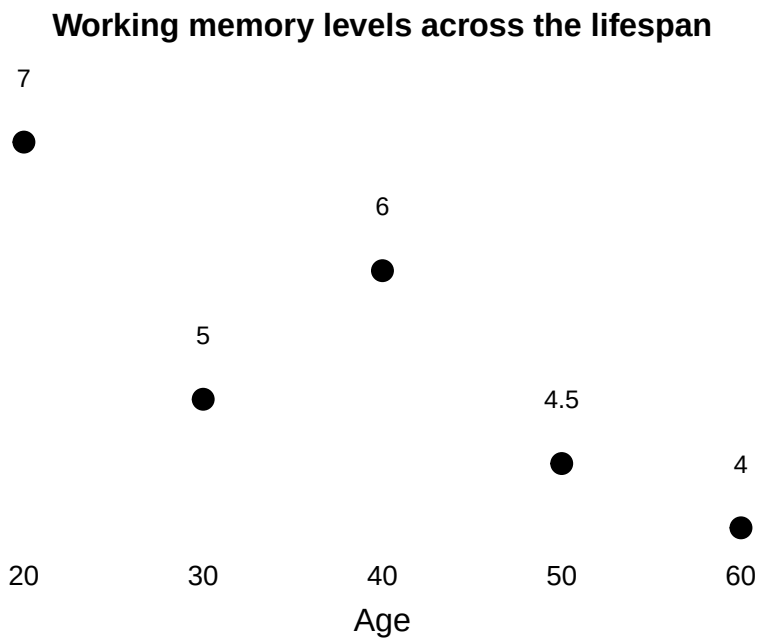


Figure 18: Dot plot of age and memory - Step 4

```
ggplot(mtcars, aes(mpg, hp)) + geom_point()
```

However, "normal" axes don't convey data, so they can be removed.

```
ggplot(mtcars, aes(mpg, hp)) + geom_point() +
    theme(axis.line = element_blank())
```

If we want something *like* axes but to be more useful, we can use the function `geom_rangeframe()` from the `ggthemes` package to put more informative data in. Range frames add "axes" but only that go the range of the observed data. Thus, these new axes show the minimum and maximum of each variable.

```
ggplot(mtcars, aes(mpg, hp)) + geom_point() +
    theme(axis.line = element_blank()) + geom_rangeframe()
```

Finally, we can make the axis labels more informative. Instead of presenting "pretty" numbers but that convey no data, we can pick axis labels and breaks at meaningful points of the data. One option is quantiles / percentiles: 0th, 25th, 50th (median), 75th and 100th percentiles are given by default from the `quantile()` function. Now almost every piece of ink in this figure conveys some useful information. We can visually see the range of the `mpg` and `hp` variables from the axes. We can see the median and interquartile range as well[7].

[7] These informative axes are created by asking `ggplot2` to create tick marks (breaks) on the x and y axis at the quintiles of the variables, which is the default output from the `quantile()` function. So now instead of the default tick marks / breaks on the axes, the breaks and numbers are: **minimum** (0th percentile), **25th** percentile (i.e., lower quartile), **50th** percentile (i.e., median), **75th** percentile (i.e., upper quartile), **maximum** (i.e., 100th percentile). This

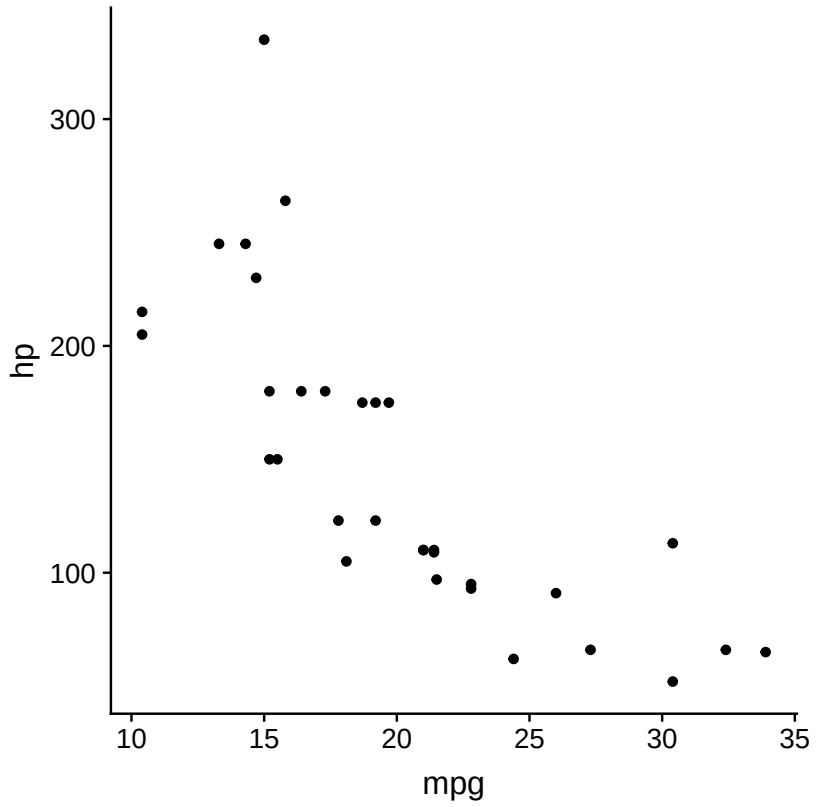
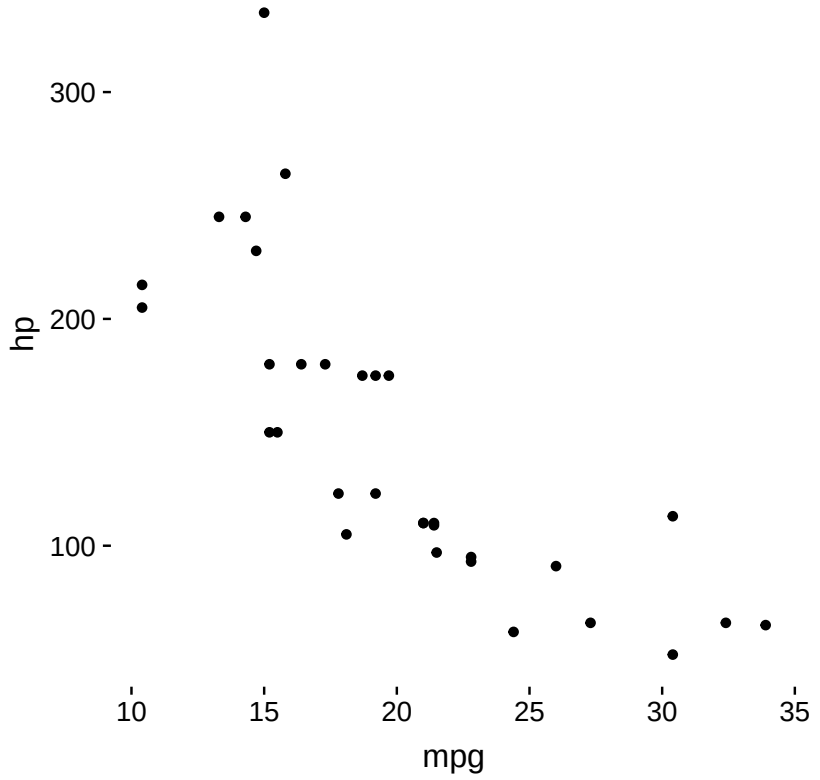


Figure 19: Scatter plot - Step 1

Figure 20: Scatter plot - Step 2



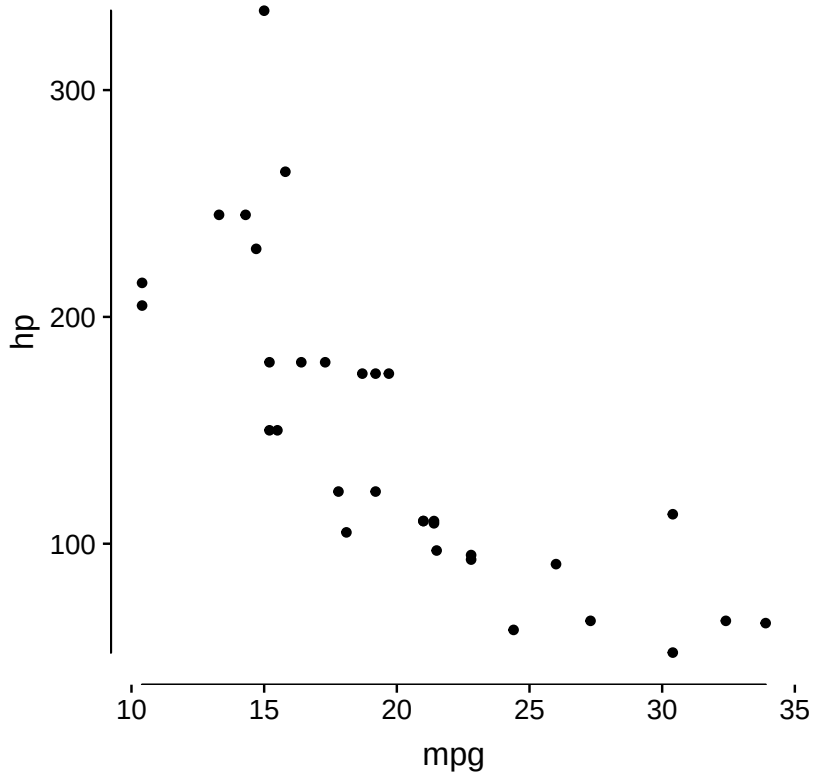
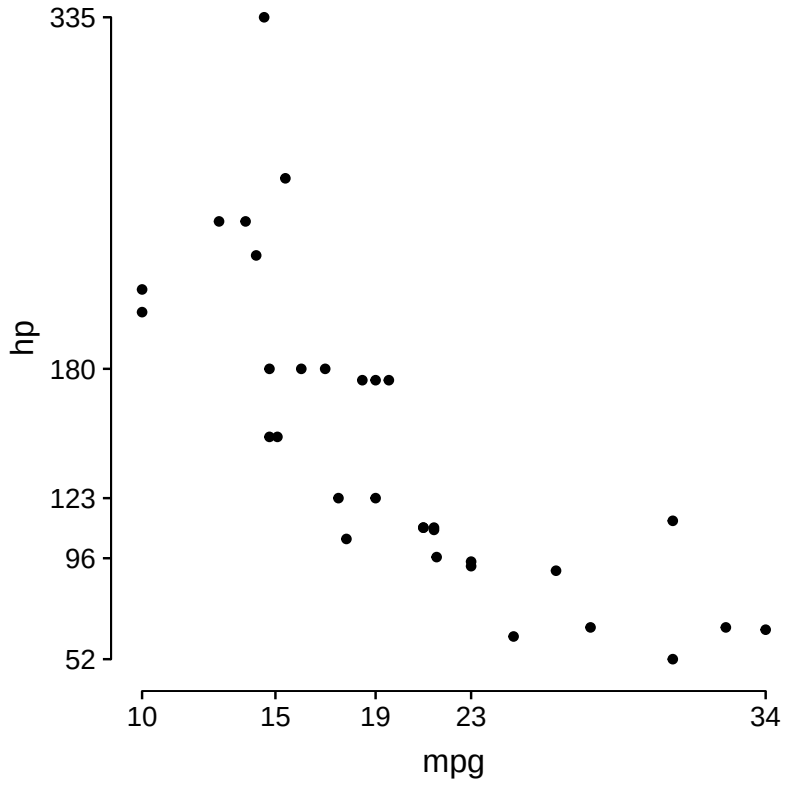


Figure 21: Scatter plot - Step 3

```
ggplot(mtcars, aes(mpg, hp)) + geom_point() +
    scale_x_continuous(breaks = as.numeric(quantile(mtcars$mpg))) +
    scale_y_continuous(breaks = as.numeric(quantile(mtcars$hp))) +
    theme(axis.line = element_blank()) + geom_rangeframe()
```

Figure 22: Scatter plot - Step 4



In the rest of the graphs we examine, we will try to implement this data to ink ratio principle. It does require some additional R code versus simply plotting with the defaults and often at first, you may need to spend a bit more time explaining your graph to people in text or in the figure legend. However, ultimately, such plots convey more data. For example, it makes it clear that cars in the top 25% of mpg (i.e., 23 to 34) all fall below the median hp (i.e., below 123). Conversely, cares in the bottom 25% of mpg(i.e., 10 to 15) all fall above the median hp (i.e., 123) with nearly all in the top 25% of hp.

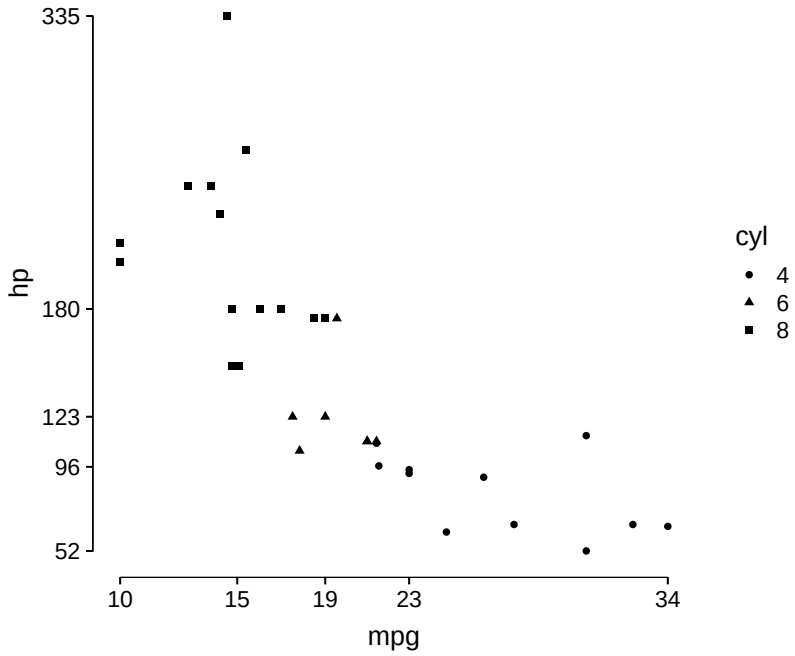
5 *Advanced Bivariate Graphs*

As with univariate graphs, we can map additional variables to additional aesthetics. This allows us to integrate more into standard

bivariate plots. The following graph shows a simple example of this.

```
ggplot(mtcars, aes(mpg, hp, shape = cyl)) + geom_point() +
    scale_x_continuous(breaks = as.numeric(quantile(mtcars$mpg))) +
    scale_y_continuous(breaks = as.numeric(quantile(mtcars$hp))) +
    theme(axis.line = element_blank()) + geom_rangeframe()
```

Figure 23: Scatter plot with shapes

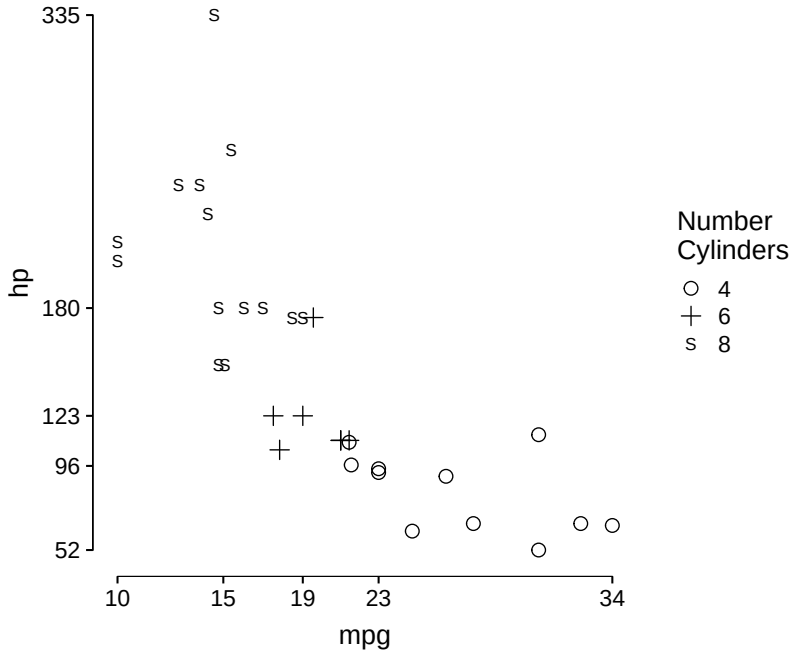


However, although the shapes are distinguishable, they are visually difficult. Cleveland did some studies around shape perception, particularly when points may be partially overlapping and on this basis suggested other shaped. We can manually specify the number for which shape we want applied to which value of `cyl` using the `scale_shape_manual()` function[8]. We also use the `name =` argument so that instead of getting the legend labelled `cyl` it is labelled `Number Cylinders`. In `R` the `\n` means add a line break so in the legend there will be a line break between "number" and "cylinders".

[8] There are many other shapes available. To see a list, go to: `http://sape.inf.usi.ch/quick-reference/ggplot2/shape`.

```
ggplot(mtcars, aes(mpg, hp, shape = cyl)) + geom_point(size = 3) +
    scale_shape_manual(name = "Number\nCylinders",
        values = c('4' = 1, '6' = 3, '8' = 83)) +
    scale_x_continuous(breaks = as.numeric(quantile(mtcars$mpg))) +
    scale_y_continuous(breaks = as.numeric(quantile(mtcars$hp))) +
    theme(axis.line = element_blank()) + geom_rangeframe()
```

Figure 24: Scatter plot with shapes



Scatter plots are a helpful way to represent the raw data and association between two variables. In some cases, however, it can be harder to see the results. The following graph is a scatter plot of stress and positive affect in the daily diary data we have used. First we remove any missing data on those variables, then we plot.

```
d2 <- d[!is.na(STRESS) & !is.na(PosAff)]

ggplot(d2, aes(STRESS, PosAff)) + geom_point(alpha = 0.2) +
    scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = as.numeric(quantile(d2$PosAff))) +
    theme(axis.line = element_blank()) + geom_rangeframe()
```

There are so many assessments and each overlap with each other so much that it is very difficult to see any pattern or trend. `STRESS` is skewed so both the minimum value and the 25th percentile are 0, which is why we only see four labels on the x axis instead of five. We can try to better see the data by shrinking the raw points, making them transparent and adding some noise (a.k.a., jittering) on stress. This works relatively well for these data and helps us see the pattern more clearly.

```
ggplot(d2, aes(jitter(STRESS), PosAff)) + geom_point(size = 0.6,
    alpha = 0.2) + scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
```

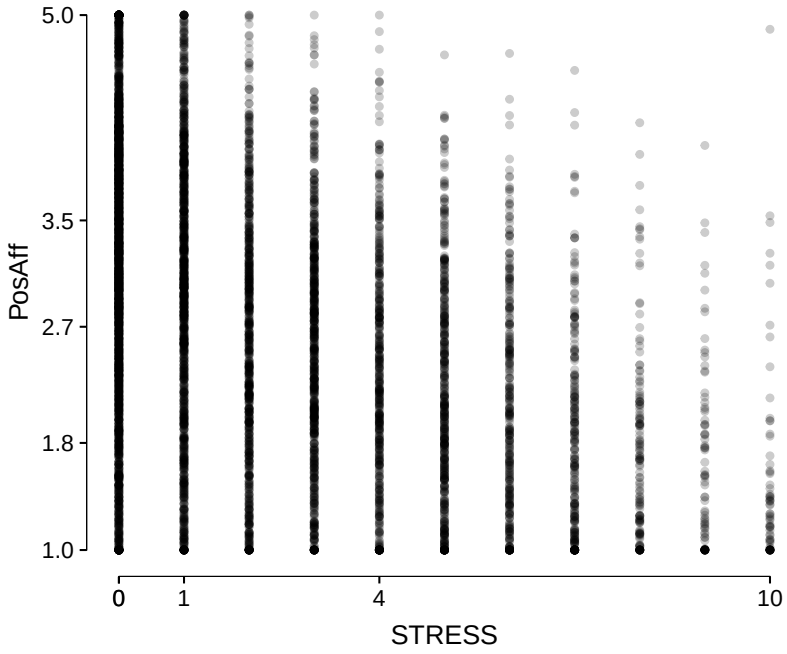


Figure 25: Scatter plot with a lot of data

```
    scale_y_continuous(breaks = as.numeric(quantile(d2$PosAff))) +
    theme(axis.line = element_blank()) + geom_rangeframe()
```

The **breaks** for positive affect are useful but may be printed in too much precision. We can use the `round()` function to reduce this precision and clean up the numbers on the y axis (e.g., if we wanted to round to 1 decimal point).

```
ggplot(d2, aes(jitter(STRESS), PosAff)) + geom_point(size = 0.6,
    alpha = 0.2) + scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        digits = 1)) + theme(axis.line = element_blank()) +
    geom_rangeframe()
```

To more clearly see the trend in scatter plot data, we can add smooth lines. R can add these easily using some flexible regression-based models but that allow flexibility in the association. With repeated measures data, the standard errors will be wrong, so we turn them off using the argument `se = FALSE`.

```
ggplot(d2, aes(jitter(STRESS), PosAff)) + geom_point(size = 0.6,
    alpha = 0.2) + scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        digits = 2)) + theme(axis.line = element_blank()) +
    geom_rangeframe() + stat_smooth(se = FALSE)
```

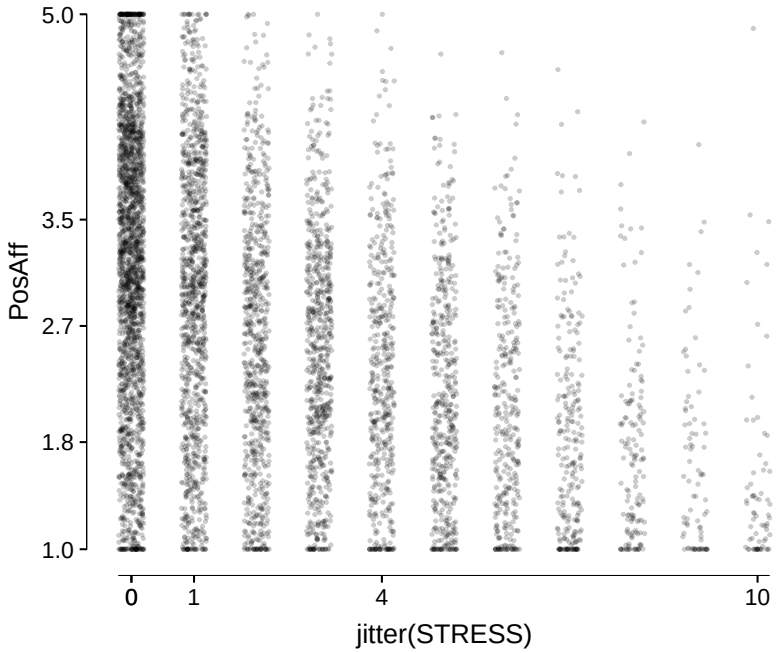


Figure 26: Scatter plot with a lot of data reducing overplotting

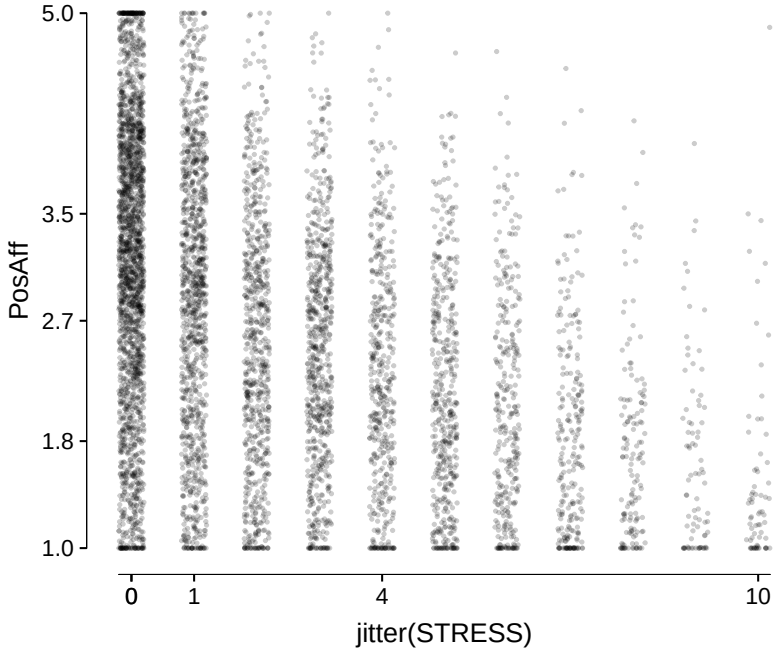


Figure 27: Scatter plot with a lot of data reducing overplotting plus rounding the axis breaks for positive affect

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

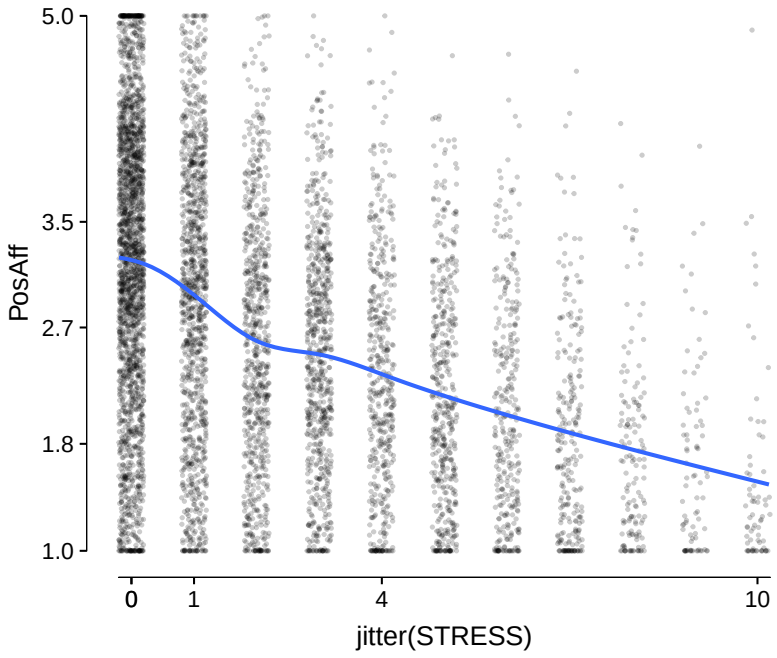


Figure 28: Scatter plot with a lot of data reducing overplotting with rounding for positive affect and a smooth trend line added

Another option for large amounts of data where plotting the raw points may be too much is to use summaries. For example, to examine univariate distributions, we used histograms and density plots. Similar options exist for bivariate data. Before we look at some of these, we start with a 3D graph of the density of data by stress and positive affect.

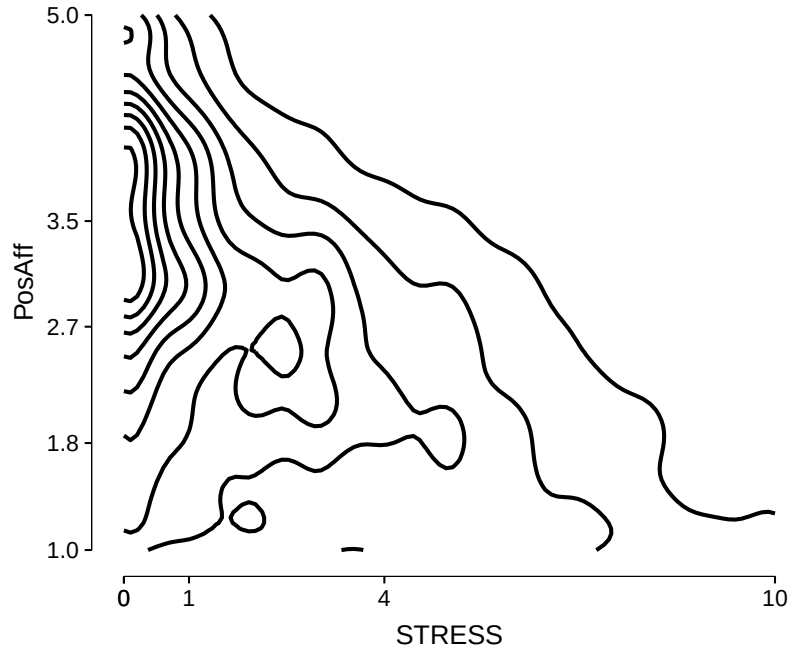
You must enable Javascript to view this page properly.

Once you are familiar with this 3D representation, now we can look at the 2D versions. One common approach is a contour plot. Each line in a contour plot represents the same value on the third dimension, here density. They are equally spaced, so if contours are near together, it means that the level of the third variable changes rapidly with a change in x or y. If they are far apart, it means the value of the third variable changes slowly with x or y[9]. For reference, a scatter plot of the data are also shown here.

```
ggplot(d2, aes(STRESS, PosAff)) + geom_density2d(size = 1,
    colour = "black") + scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        2)) + theme(axis.line = element_blank()) +
    geom_rangeframe()
```

[9] To learn how to read contour plots, first watch this video about reading contour lines on maps: `https://youtu.be/CoVcRxza8nI`. The same principles apply to reading contour plots of density, except that instead of different lines for different altitudes, they are different lines for different densities, which represents how many observations fall in any particular region. Note that if data are bivariate normal, the bivariate distribution should have a single mode (peak) that is at the mean (roughly centered) at the mean of each variable of the bivariate distribution. The contours would look like circles or ellipsoids expanding out from the single peak. Multiple points of tightly spaced contour lines may suggest more than one peak (i.e., multimodal) on one

Figure 29: 2D contour plot of densities




```
ggplot(d2, aes(STRESS, PosAff)) + geom_point() +
    scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        2)) + theme(axis.line = element_blank()) +
    geom_rangeframe() + ggtitle("Scatter plot for comparison")
```

Following is an example of a contour plot for bivariate normal data, where the two variables are positive correlated with each other.

```
set.seed(1234)
normaldat <- data.frame(x = rnorm(50000))
normaldat$y <- 2 * normaldat$x + rnorm(50000)

ggplot(normaldat, aes(x, y)) + geom_density2d(size = 1,
    colour = "black") + theme(axis.line = element_blank()) +
    geom_rangeframe()


ggplot(normaldat, aes(x, y)) + geom_point() +
    theme(axis.line = element_blank()) + geom_rangeframe() +
    ggtitle("Scatter plot for comparison")
```

We can use the `fill` argument to add colour to help. Higher numbers indicate higher density (i.e., more observations). This is still a

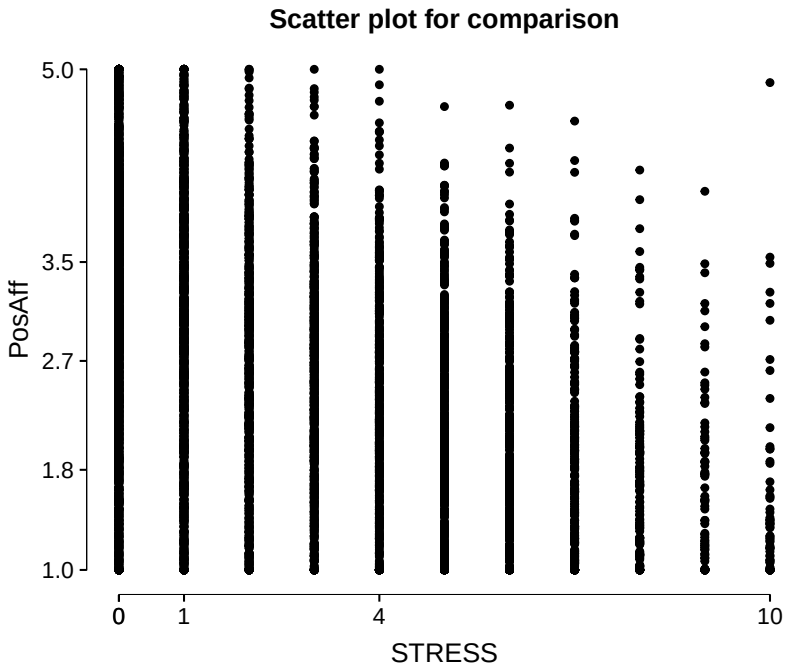


Figure 30: 2D contour plot of densities

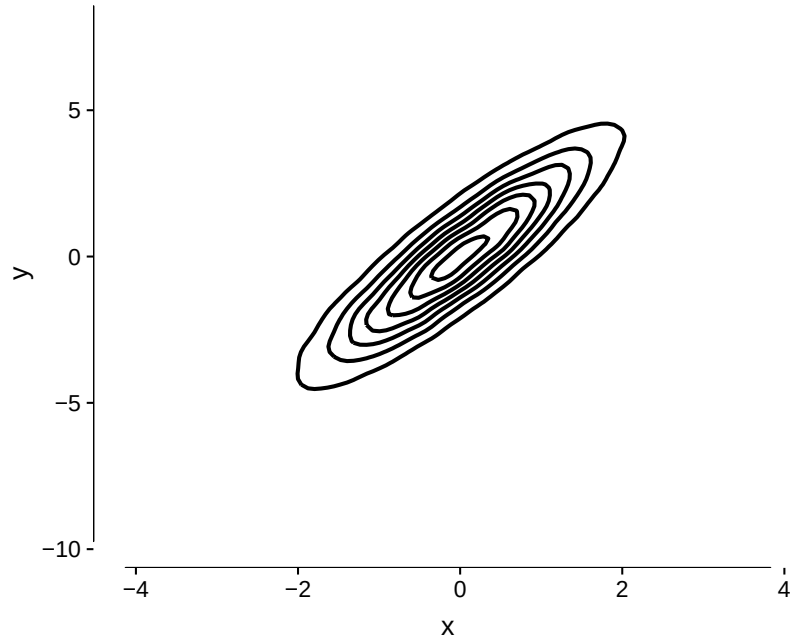


Figure 31: 2D contour plot of densities for a bivariate normal distribution and a comparison scatter plot

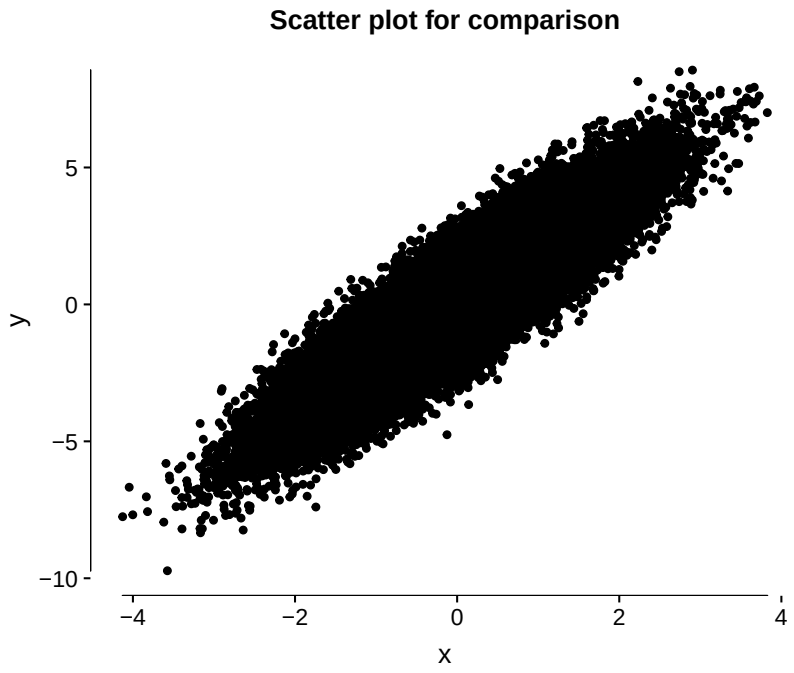


Figure 32: 2D contour plot of densities for a bivariate normal distribution and a comparison scatter plot

contour plot, its just filled in the space in between the contour lines with colour.

```
ggplot(d2, aes(STRESS, PosAff)) + stat_density_2d(aes(fill = ..level..),
    geom = "polygon", colour = "black") + scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        2)) + theme(axis.line = element_blank()) +
    geom_rangeframe()
```

One last option we will look at for large amounts of two-dimensional data is kind of like a histogram. The data are binned, and counted. Instead of binning on one dimension, data are binned on two dimensions. The colour of each hexagon indicates how many people fall into the area covered by the hexagon, just as the height of a bar in a histogram indicates how many observations fall between the width of the bar[10].

```
ggplot(d2, aes(STRESS, PosAff)) + geom_hex(bins = 20) +
    scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        2)) + theme(axis.line = element_blank()) +
    geom_rangeframe()
```

A challenge with these plots can be figuring out the ideal number of bins to use, especially withvery skewed or "clumpy" data.

[10] On the right hand side of the plot, the legend shows a continuous colour gradient, where brighter blues indicate more observations. In this instance, the brightest blues represent a count of over 150 observations. The darker blues represent something under 50 observations. It will be difficult visually to distinguish 50 observations in a bin from say 20 or 10 because all of them are a relatively dark blue. It is a fairly continuous colour gradient from dark to bright blue, but visually we can only discern movement of a certain magnitude. When you have some high frequency bins requiring that bright blues be used for say 150+, then it gives less "space" on the colour gradient for the low frequencies (5, 10, 20) which means all of those (visually) appear very similar as a dark blue, and we cannot well distinguish some bins with relatively lower frequencies apart as

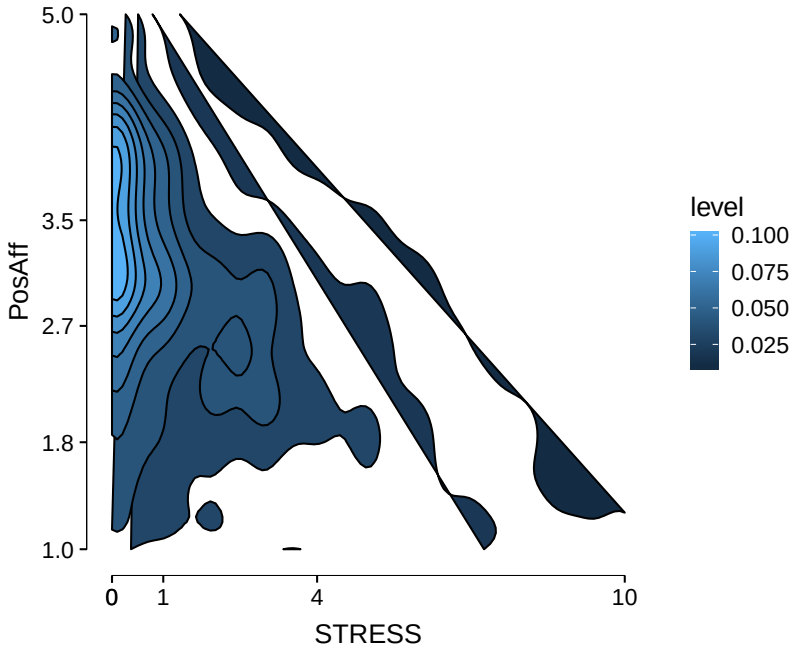


Figure 33: 2D filled contour plot of densities

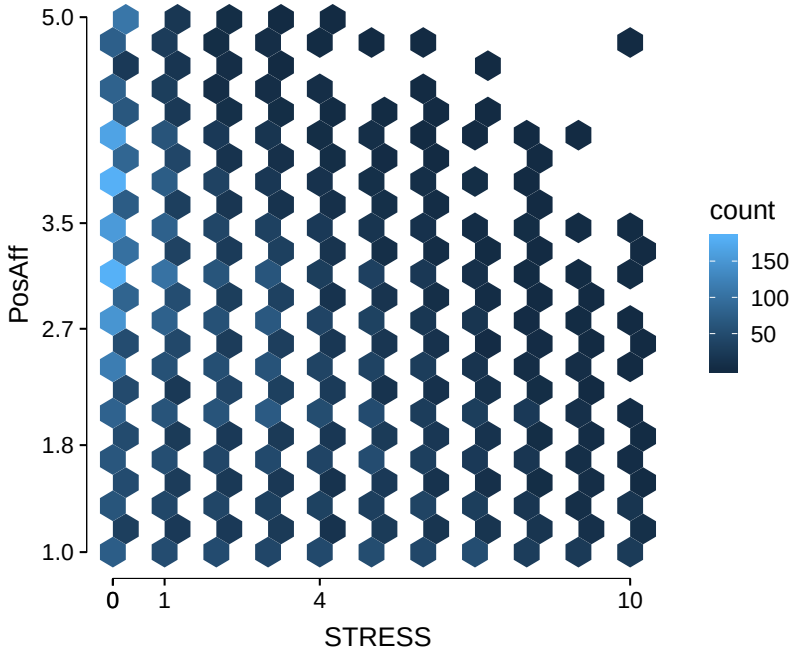


Figure 34: 2D histogram by hexagonal binning

Too many people in one bin means that the colour scale must go up so high that you do not get much granularity to compare different frequencies on the lower end (e.g., below with `bins = 10` the colour scale is in 100s of observations). The `bins = 10` option roughly means use about 10 bins in each variable, although its used as part of an algorithm so you don't get exactly 10 bins in the rows or columns.

```
ggplot(d2, aes(STRESS, PosAff)) + geom_hex(bins = 10) +
    scale_x_continuous(breaks = as.numeric(quantile(d2$STRESS))) +
    scale_y_continuous(breaks = round(as.numeric(quantile(d2$PosAff)),
        2)) + theme(axis.line = element_blank()) +
    geom_rangeframe()
```

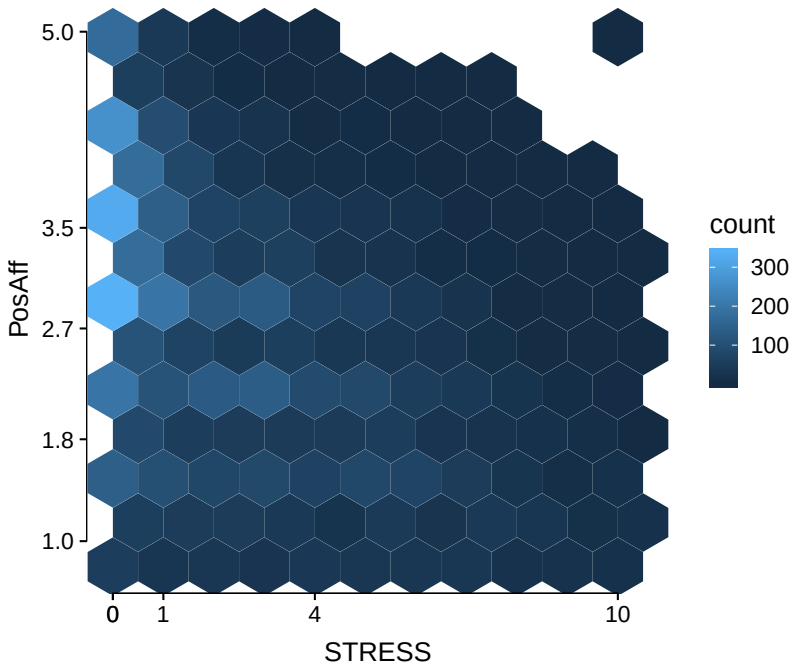


Figure 35: 2D histogram by hexagonal binning

5.1 *Try It - Bivariate*

Using the daily diary dataset we have worked with all semester, `d` dataset:

1. Make a scatter plot (points) for `SUPPORT` and `COPEPrb`. Use the good visualization principles we have learned. Make your own decisions about jittering, size, and alpha transparency to make the scatter plot most useful to read.

2. Extend your final plot from number 1 to include an overall trend line.
3. Create a plot of the distribution/density of `SUPPORT` and `COPEPrb`. This can be a contour plot, hexgonal bin, or if you are very keen, a 3D plot.

```
## scatter plot code here


## scatter plot + trend line code here


## 2d distribution plot code here
```

6 *Longitudinal Plots*

For longitudinal data or time series data, line plots often make sense. We have seen simple line plots before, but when there are multiple people, using an overall line plot is not effective.

The following code subset the data to just 4 randomly sampled people with non missing support values and creates a line plot[11].

[11] The `set.seed()` function is used to set the random seed. This controls how `R` generates random numbers. It allows multiple computers to generate the same set of random numbers. So that over and over, this example will randomly select the **same** random 4 IDs to plot. If you want to plot different participants, you can change the random seed to some other number (e.g., 52349) and then you will get a different 4 random participants' selected and plotted.

```
set.seed(123)
d3 <- d[!is.na(SUPPORT) & UserID %in% sample(unique(d$UserID),
    4)]

ggplot(d3, aes(SurveyDay, SUPPORT)) + geom_line()
```

The vertical jumps in the lines are because there are multiple assessments on the same day. To address this, we need to separate the lines by person. We can do this by using another aesthetic, such as colour, and/or by specifying a grouping variable. We do both below.

```
ggplot(d3, aes(SurveyDay, SUPPORT, colour = UserID,
    group = UserID)) + geom_line()
```

Just as we looked at plotting some smooth lines for scatter plots, we can do that for each person. All we need to do is add another line with the `stat_smooth()` function.

```
ggplot(d3, aes(SurveyDay, SUPPORT, colour = UserID,
    group = UserID)) + geom_line() + stat_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

If we think that the smooth lines are relatively good approximations, a cleaner image may be made by just showing these, which smooths out some of the "noise" in the raw data.

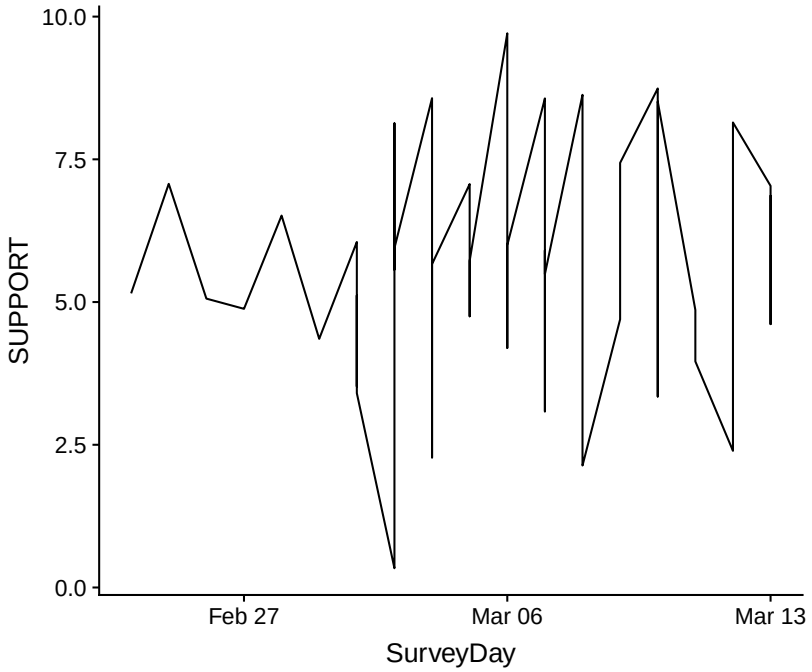


Figure 36: Longitudinal plots

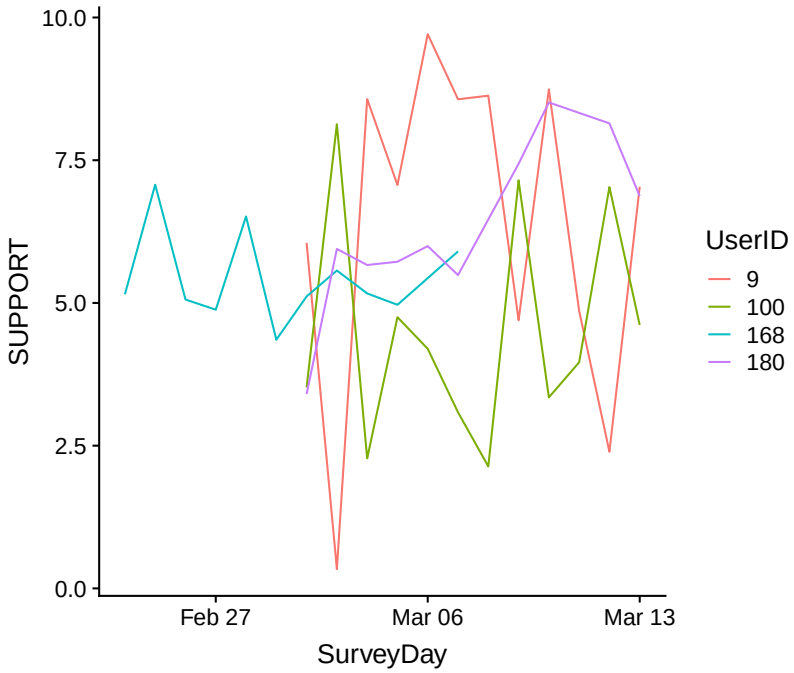


Figure 37: Longitudinal plots by person

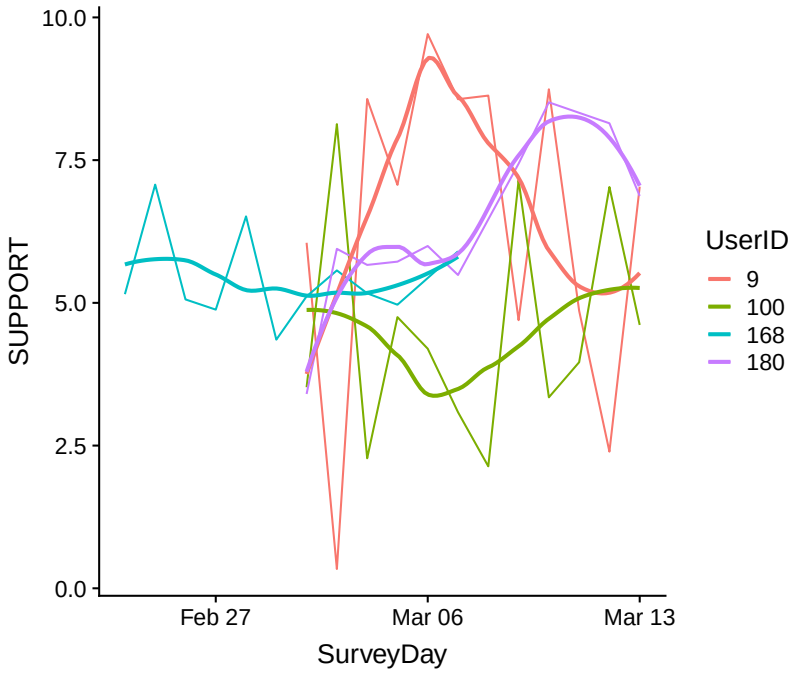


Figure 38: Longitudinal plots by person with smooths

```
ggplot(d3, aes(SurveyDay, SUPPORT, colour = UserID,
    group = UserID)) + stat_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

Finally, by default R uses flexible smooths. However, by specifying `method = "lm"` we can force a linear smooth function (linear regression), although this might be a (very) poor fit to some data, such as ID 9.

```
ggplot(d3, aes(SurveyDay, SUPPORT, colour = UserID,
    group = UserID)) + geom_line() + stat_smooth(method = "lm",
    se = FALSE)
```

6.1 Try It - Longitudinal

Using the daily diary dataset we have worked with all semester, `d` dataset:

1. Change the random seed and subset the dataset to just a few participants' IDs.
2. Plot problem focused coping over time for those few people.

```
## subset the data
```

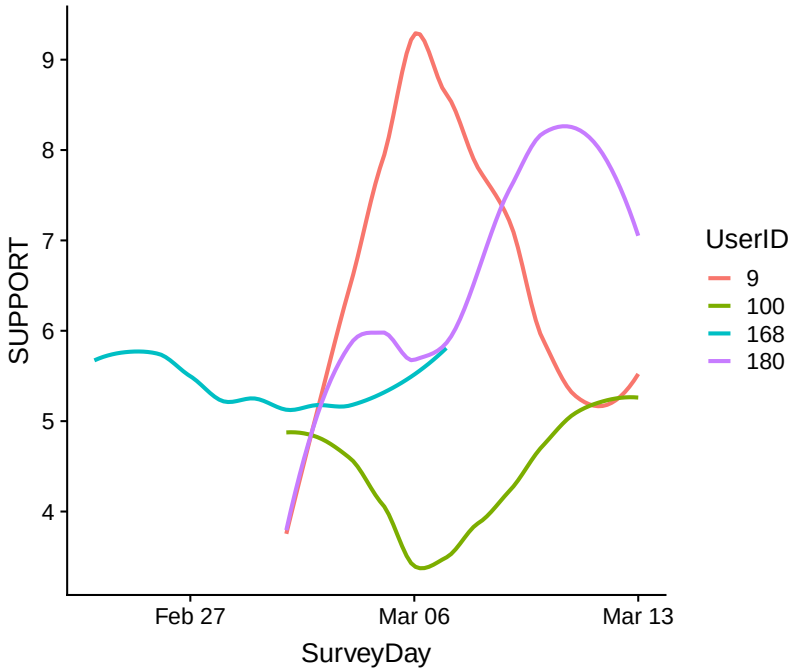


Figure 39: Longitudinal smooth plots by person

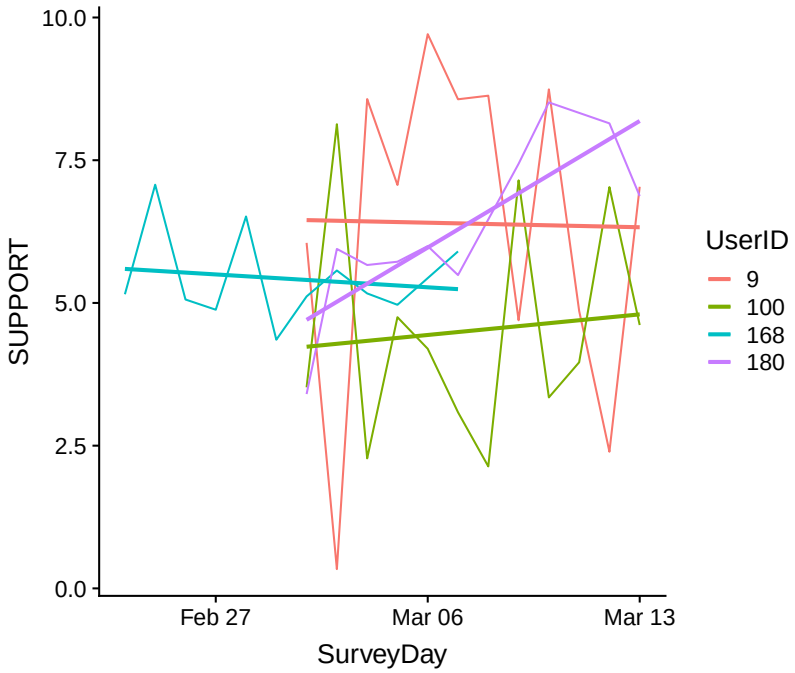


Figure 40: Longitudinal linear smooth plots by person

```
## longitudinal plot
```