R Tutorial at the WZB

7 - Reshaping data and plotting with ggplot2

Markus Konrad

December 06, 2018

Today's schedule

1. Review of last week's tasks
2. Reshaping data with `gather()` and `spread()`
3. Plotting with ggplot2

WZB
Wissenschaftszentrum Berlin für Sozialforschung

Review of last week's tasks

Solution for tasks #6

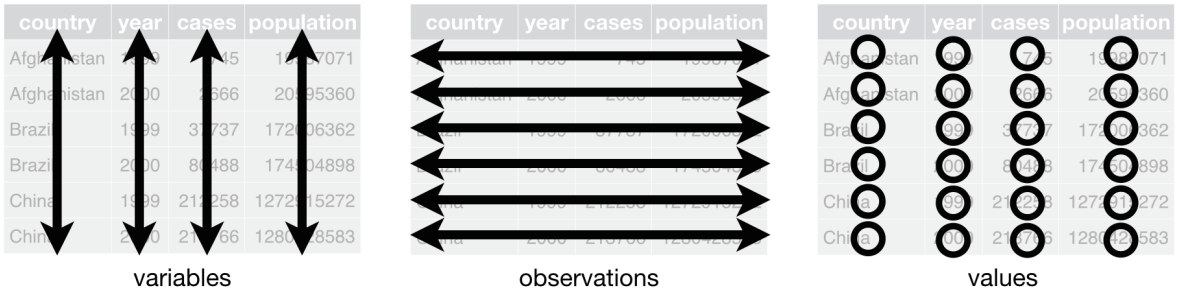
now online on
https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/

Reshaping data with `gather()` and `spread()`

Tidy data

Hadley Wickham introduced the concept of **tidy data** as a way how data should be organized so it is comfortable to work with (H. Wickham 2014: Tidy Data). He defined three rules that make a data set tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.



source: Grolemund, Wickham 2017: R for Data Science

Why tidy data?

- a tidy data set allows for easier variable selection, grouping, summarization and visualization
- tools and packages in the tidyverse like dplyr or ggplot2 require data to be organized in that way
- the problem: most data you'll get won't be "tidy" from the beginning on → you'll need to reshape it

Reshaping untidy data

`OBrienKaiser` from package carData

- data from imaginary study
- three treatments: A, B, control
- three measurement types: pretest, posttest, follow-up session
- each measured at five occasions (suffixes `.1` to `.5`)

```
 treatment gender pre.1 pre.2 pre.3 pre.4 pre.5 post.1 post.2 post.3 post.4
         A      M     7     8     7     9     9      9      9     10      8
         B      F     4     5     7     5     4      7      7      8      6
   control      M     1     2     4     2     1      3      2      5      3
       ...
```

Is this data tidy?

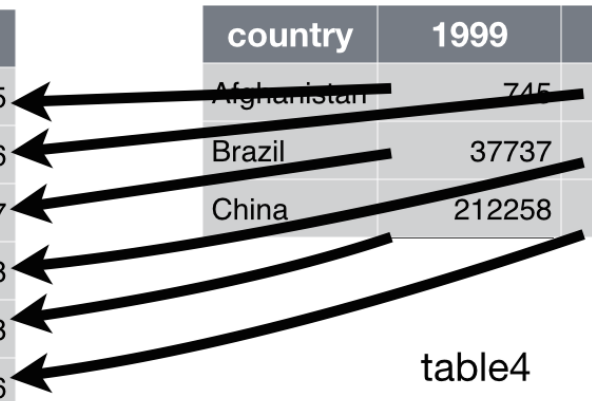
No! You have **several measurements (observations) per row**.
This is also called a wide table format.

→ hard to work with (e.g. compute mean of all three measurement types separately)

Reshaping untidy data

We can use `gather()` from package tidyr to put the measurements in separate rows.

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766



country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

source: Grolemund, Wickham 2017: R for Data Science

Reshaping untidy data

`gather()` takes the following arguments (among others):

1. The data to work with (omit this if you use the pipe operator %>%).
2. The "untidy" columns to gather.
3. `key`: The name of the new column containing the measurement types.
4. `value`: The name of the new column containing the measurement values.

```
(tidy_obk <- gather(OBrienKaiser, pre.1:fup.5,
                    key = 'meas_type_occasion', value = 'value'))
```

```
##     treatment gender meas_type_occasion value
## 1     control      M              pre.1     1
## 2     control      M              pre.1     4
## 3     control      M              pre.1     5
## 4     control      F              pre.1     5
## 5     control      F              pre.1     3
## 6           A      M              pre.1     7
## 7           A      M              pre.1     5
## 8           A      F              pre.1     2
## 9           A      F              pre.1     3
## 10          B      M              pre.1     4
## 11          B      M              pre.1     3
## 12          B      M              pre.1     6
## 13          B      F              pre.1     5
## 14          B      F              pre.1     2
## 15          B      F              pre.1     2
##  [ reached getOption("max.print") -- omitted 225 rows ]
```

Reshaping untidy data

Our data is already better to work with, but `meas_type_occasion` still contains two values like "pre.1" (pretest 1) or "fup.4" (follow-up test 4). This violates rule #3: "Each value must have its own cell.").

We can use `separate()` (package tidyr) to split columns that contain several values. It takes the following arguments (among others):

1. The data to work with (omit this if you use the pipe operator %>%).
2. The column to split.
3. `into`: The names of the new columns.
4. `sep`: A rule for how to split the values. The default is to split on anything that is not a number or character (e.g. slash, period, hyphen, …).

```
tidy_obk <- separate(tidy_obk, meas_type_occasion,
                     into = c('meas_type', 'meas_occasion'))
```

Reshaping untidy data

Some additional variable conversion and we're done:

```
tidy_obk <- mutate(tidy_obk,
                   meas_type = factor(meas_type, levels = c('pre', 'post', '
                   meas_occasion = as.integer(meas_occasion))
```

```
##     treatment gender meas_type meas_occasion value
## 1     control      M       pre             1     1
## 2     control      M       pre             1     4
## 3     control      M       pre             1     5
## 4     control      F       pre             1     5
## 5     control      F       pre             1     3
## 6           A      M       pre             1     7
## 7           A      M       pre             1     5
## 8           A      F       pre             1     2
## 9           A      F       pre             1     3
## 10          B      M       pre             1     4
## 11          B      M       pre             1     3
## 12          B      M       pre             1     6
##  [ reached getOption("max.print") -- omitted 228 rows ]
```

→ This is tidy data. It's also called the long table format.

Reshaping untidy data

With this data we can actually work!

```
tidy_obk %>%
  group_by(treatment, meas_type) %>%
  summarise(mean_measurement = mean(value))
```

```
## # A tibble: 9 x 3
## # Groups:   treatment [?]
##   treatment meas_type mean_measurement
##   <fct>     <fct>                <dbl>
## 1 control   pre                   4.2
## 2 control   post                  4
## 3 control   fup                   4.4
## 4 A         pre                   5
## 5 A         post                  6.5
## 6 A         fup                   7.25
## 7 B         pre                   4.14
## 8 B         post                  6.57
## 9 B         fup                   7.29
```

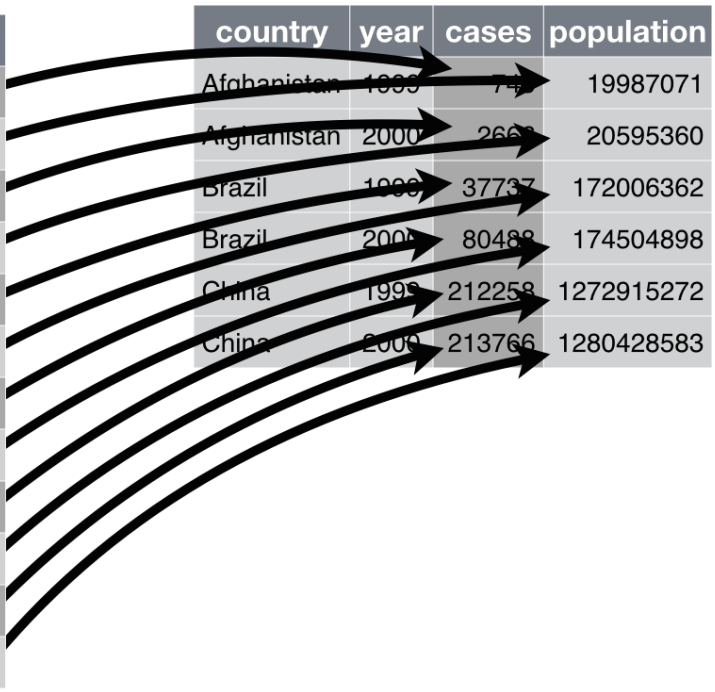
It's better spending some time on data cleanup than struggling with messy data sets!

Combining rows

`spread()` is the opposite of `gather()`: It combines observations from multiple rows into a single row with more columns. Hence it converts data from the long table format to the wide table format.

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2



source: Grolemund, Wickham 2017: R for Data Science

Combining rows

`economics_long` from package ggplot2 contains data in long table format:

```
date       variable    value
1967-07-01 pce         507.
1967-08-01 pce         510.
1967-09-01 pce         516.
...
1967-07-01 pop      198712
1967-08-01 pop      198911
1967-09-01 pop      199113
...
1967-07-01 psavert      12.5
1967-08-01 psavert      12.5
1967-09-01 psavert      11.7
...
```

For each date, there are five variable types (pce, pop, psavert, unemploy, uempmed) and their respective values.

Combining rows

`spread()` takes the following arguments:

1. The data to work with (omit this if you use the pipe operator %>%).
2. `key`: The column containing the variable names.
3. `value`: The column containing the respective variable values.

```
spread(economics_long, variable, value)
```

```
## # A tibble: 574 x 6
##    date         pce    pop psavert uempmed unemploy
##    <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
##  1 1967-07-01  507. 198712    12.5     4.5     2944
##  2 1967-08-01  510. 198911    12.5     4.7     2945
##  3 1967-09-01  516. 199113    11.7     4.6     2958
##  4 1967-10-01  513. 199311    12.5     4.9     3143
##  5 1967-11-01  518. 199498    12.5     4.7     3066
##  6 1967-12-01  526. 199657    12.1     4.8     3018
##  7 1968-01-01  532. 199808    11.7     5.1     2878
##  8 1968-02-01  534. 199920    12.2     4.5     3001
##  9 1968-03-01  545. 200056    11.6     4.1     2877
## 10 1968-04-01  545. 200208    12.2     4.6     2709
## # ... with 564 more rows
```

Plotting with ggplot2

Data visualization

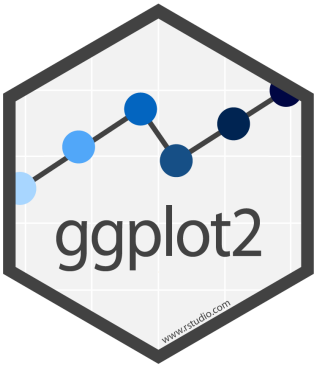
Grouping, aggregating and summarizing results are important for data analysis. They let you reduce complex data sets to simpler summarizations, compare groups and help to see patterns in your data.

Visualization of your data can have the same benefits and has the power to communicate a message much faster than a table of numbers. However, it requires a **sense for which type of graph or visual representation is appropriate for your data**.

A useful website for inspiration, code examples and caveats is https://www.data-to-viz.com.

What is ggplot2?

- R offers several plotting systems (e.g. base R, lattice or ggplot2)
- we'll use ggplot2 because:
  - it's versatile and elegant
  - it employs a well founded theory for declaratively creating graphics called "Grammar of Graphics" (Wilkinson 2005)
  - it's part of the tidyverse → plays together well with the tools we already learned



source:

tidyverse.org

General concepts behind ggplot2

There are three basic steps for constructing plots with ggplot2:

1. Supply a data set you want to plot to `ggplot()`.
2. Define an **aesthetic mapping** with `aes()`.
   This describes how variables of your data are mapped to visual properties, e.g. variable "age" is plotted on the x-axis and "income" on the y-axis.
3. Add layers of **geoms** (geometrical objects) that describe which graphical primitives to use (e.g. points in a scatter plot or bars in a bar plot).

Additionally, you can further change the appearance of your plot by:

- altering the **scales** (e.g. use a logarithmic scale, modify display of factors, etc.)
- defining **facets** → create small multiples, each showing a different subset of the data
- changing the **coordinate system** (e.g. to display maps or radial plots)
- changing the overall appearance of the plot by adjusting its **theme** (e.g. change background color, rotate axis labels, etc.)

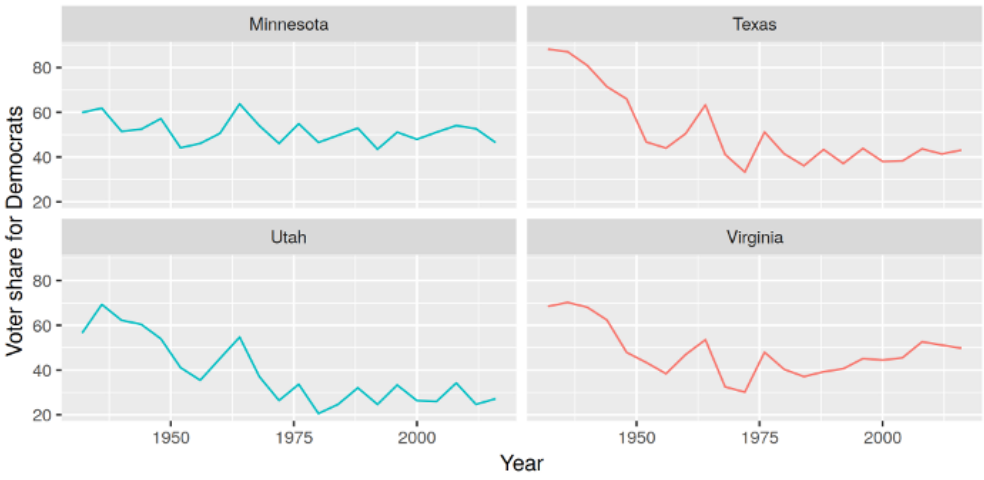
You combine all these steps with a **+**.

General concepts behind ggplot2

We'll make a plot for historical election results from four states (data set `presidentialElections` from package pscl):

```
## # A tibble: 88 x 4
##    state        demVote  year south
##    <chr>          <dbl> <int> <lgl>
##  1 Louisiana       92.8  1932 TRUE
##  2 Maine           43.2  1932 FALSE
##  3 South Dakota    63.6  1932 FALSE
##  4 Tennessee       66.5  1932 TRUE
##  5 Louisiana       88.8  1936 TRUE
##  6 Maine           41.5  1936 FALSE
##  7 South Dakota    54.0  1936 FALSE
##  8 Tennessee       68.8  1936 TRUE
##  9 Louisiana       85.9  1940 TRUE
## 10 Maine           48.8  1940 FALSE
## # ... with 78 more rows
```

General concepts behind ggplot2

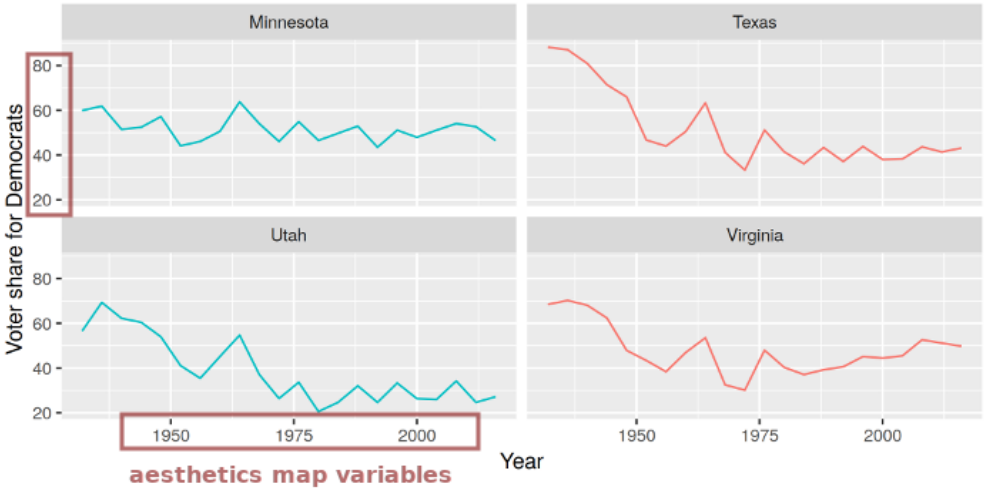


data to plot

```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

WZB Wissenschaftszentrum Berlin für Sozialforschung

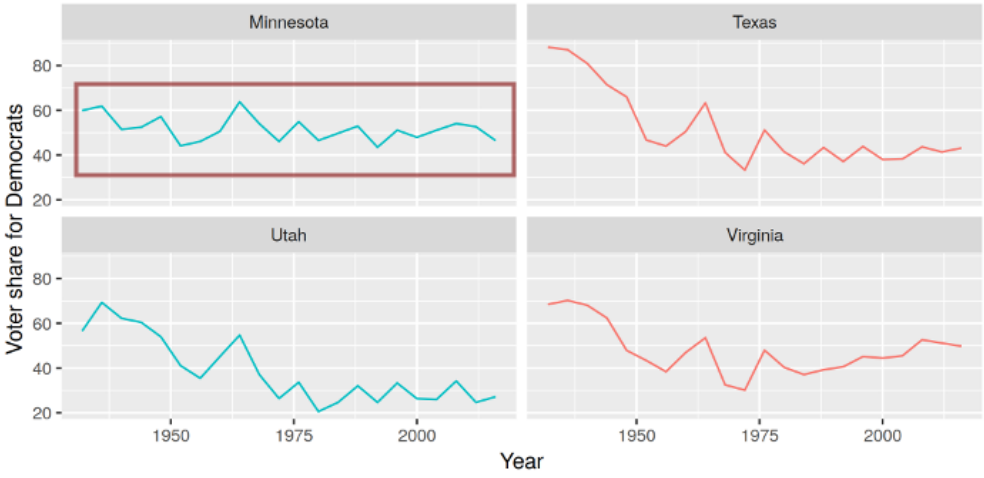
General concepts behind ggplot2




```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

WZB
Wissenschaftszentrum Berlin für Sozialforschung

General concepts behind ggplot2

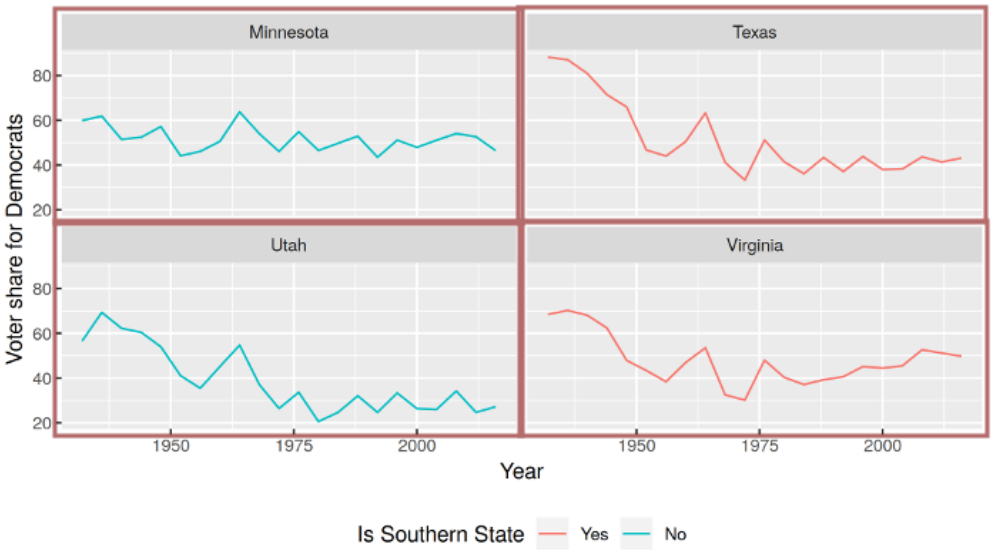



```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

geoms are the graphical primitives in the plot

WZB Wissenschaftszentrum Berlin für Sozialforschung

General concepts behind ggplot2

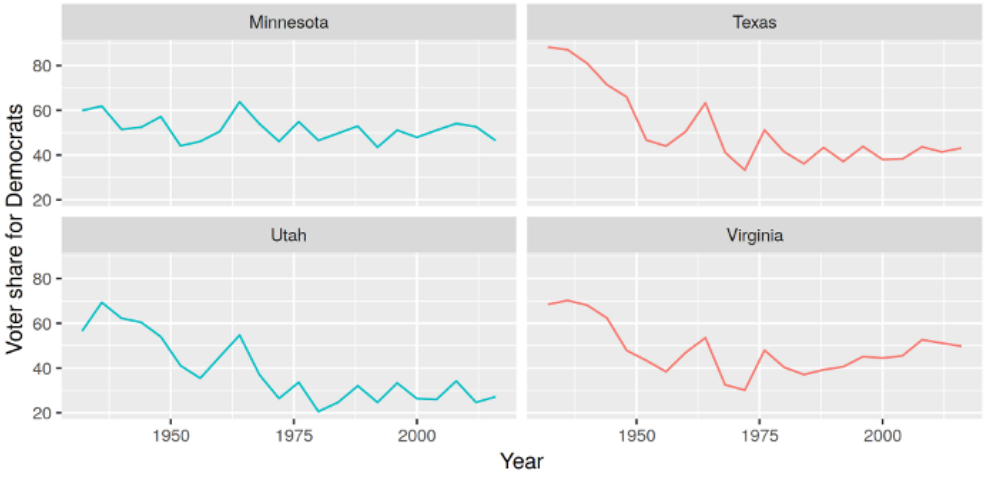



```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

facets allow to create "small multiples" for data that is subset by a variable

WZB Wissenschaftszentrum Berlin für Sozialforschung

General concepts behind ggplot2

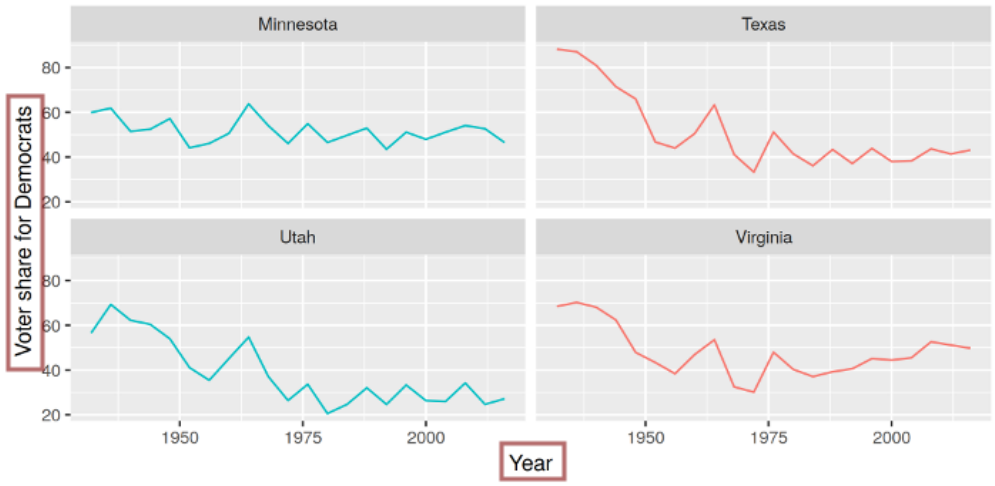



```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

each variable is mapped to a scale which can be adjusted

WZB Wissenschaftszentrum Berlin für Sozialforschung

General concepts behind ggplot2

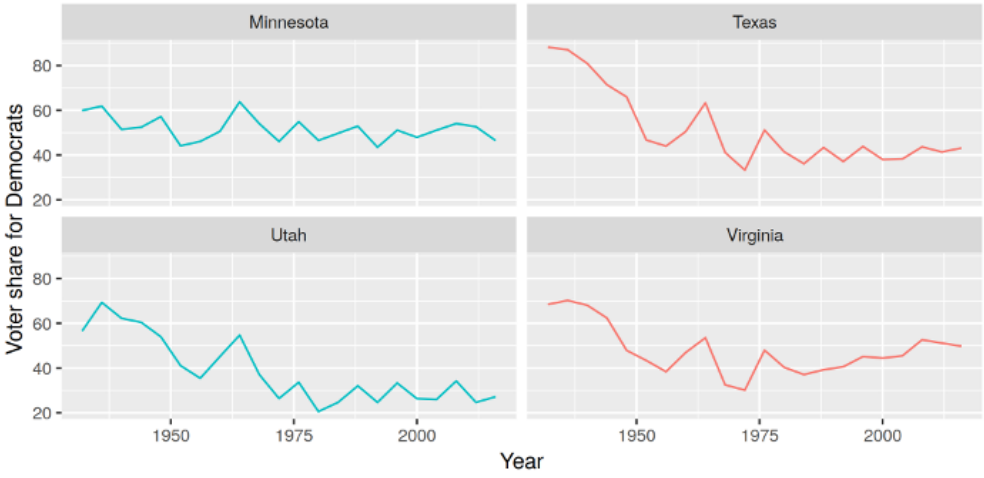



```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

labels and annotations can be added to the overall plot

WZB Wissenschaftszentrum Berlin für Sozialforschung

General concepts behind ggplot2




```
ggplot(sampled_pres, aes(x = year, y = demVote, color = south)) +
  geom_line() +
  facet_wrap(~ state) +
  scale_color_discrete(guide = guide_legend(title = "Is Southern State"),
                       limits = c(TRUE, FALSE),
                       labels = c('Yes', 'No')) +
  xlab('Year') +
  ylab('Voter share for Democrats') +
  theme(legend.position = 'bottom')
```

theme options control the overall appearance

WZB Wissenschaftszentrum Berlin für Sozialforschung

Example data

Example data: `politicalInformation` from pscl package

> Interviewers administering the 2000 American National Election Studies assigned an ordinal rating to each respondent's "general level of information" about politics and public affairs.

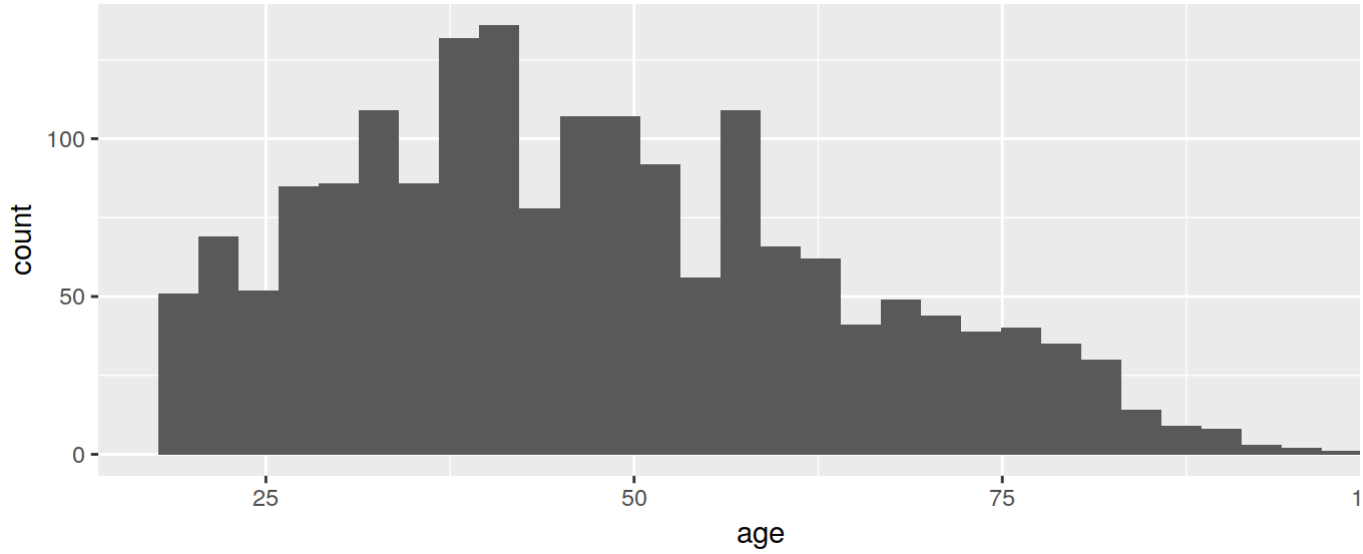
```
##                y collegeDegree female age homeOwn govt length  id
## 1    Fairly High           Yes     No  49     Yes   No  58.40   1
## 2        Average            No    Yes  35     Yes   No  46.15   2
## 3      Very High            No    Yes  57     Yes   No  89.52   3
## 4        Average            No     No  63     Yes   No  92.63   4
## 5    Fairly High           Yes    Yes  40     Yes   No  58.85   4
## 6        Average            No     No  77     Yes   No  53.82   4
## 7        Average            No     No  43     Yes   No  58.47   5
##  [ reached getOption("max.print") -- omitted 1800 rows ]
```

Distribution of a single variable

At first, you should make sure that you either loaded the tidyverse package or ggplot2 package. We define:

1. The data set `politicalInformation`.
2. The aeshetic mapping for a single variable `age`.
3. The graphical representation as histogram with `geom_histogram()`.

```
ggplot(politicalInformation, aes(age)) + geom_histogram()
```



WZB Wissenschaftszentrum Berlin für Sozialforschung

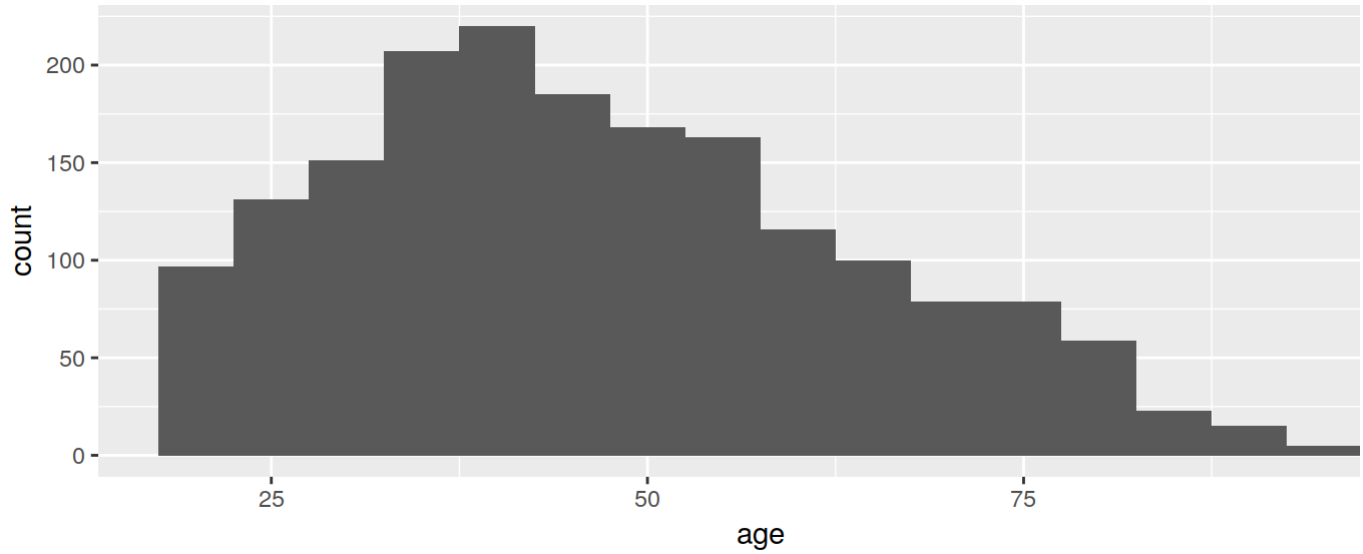
Distribution of a single variable

I hid two messsages produced by ggplot:

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
Removed 9 rows containing non-finite values (stat_bin).
```

- the first tells us that a default value for the number of bins was used → we set `binwidth = 5` below to make a histogram of age with five-year steps
- the second complains about 9 NA values in the `age` variable → we can ignore this

```
ggplot(politicalInformation, aes(age)) + geom_histogram(binwidth = 5)
```



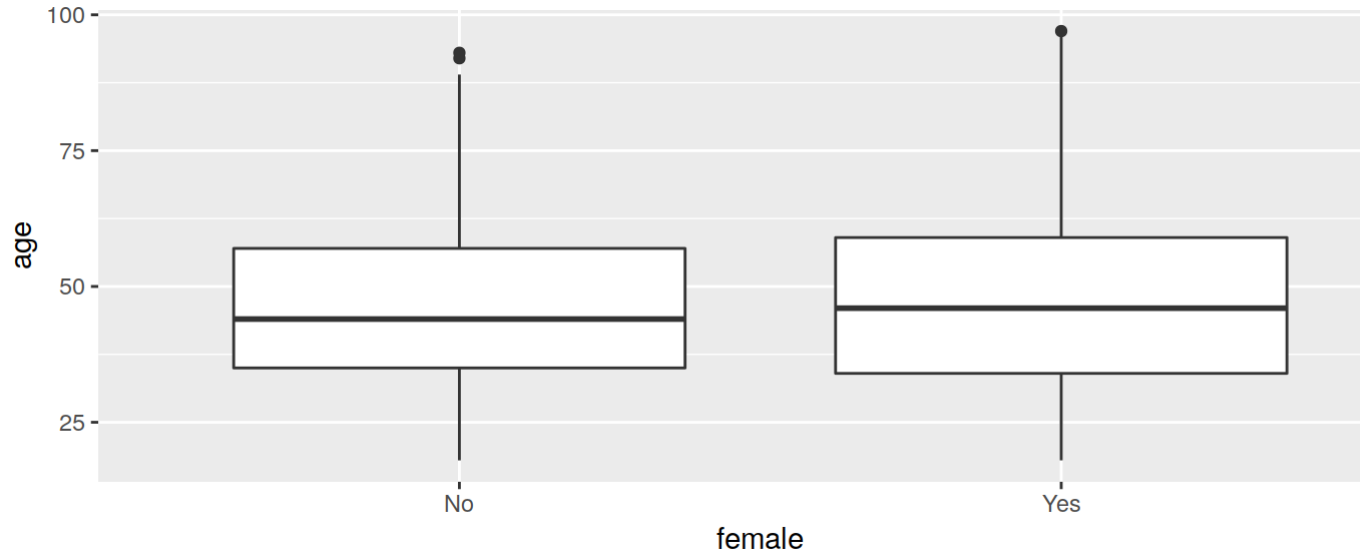
WZB Wissenschaftszentrum Berlin für Sozialforschung

Distribution of a single variable

What if we wanted to know if the distribution of `age` in our sample is different for men and women?

We can produce boxplots for both subsets, putting `female` on the x-axis (see how the aesthetics mapping was changed!):

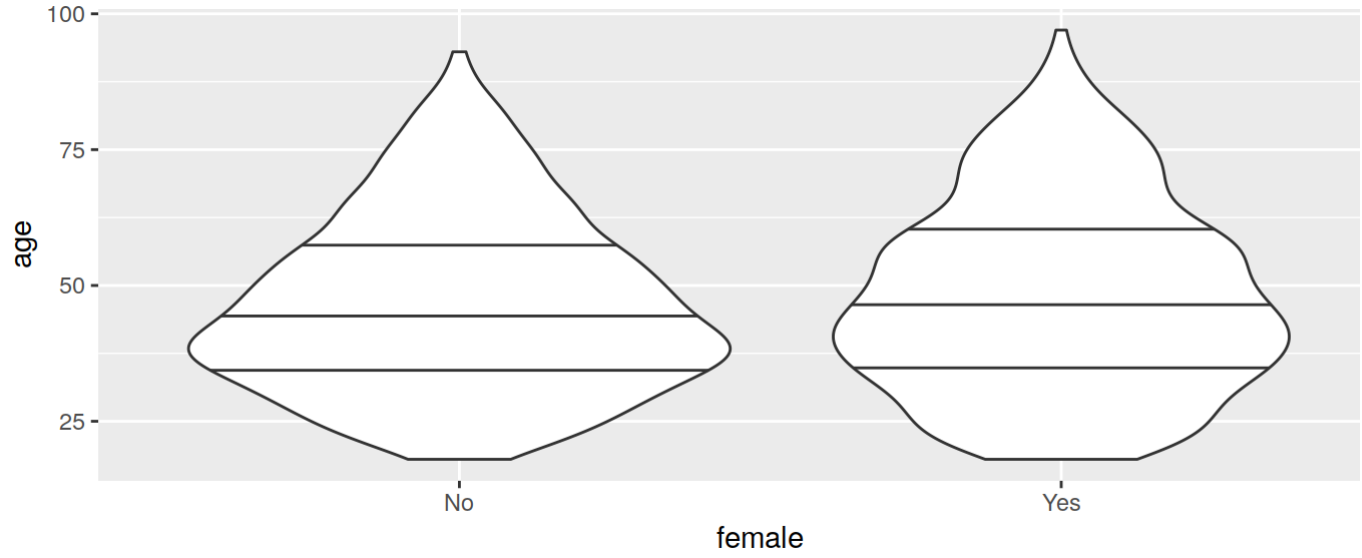
```
ggplot(politicalInformation, aes(x = female, y = age)) + geom_boxplot()
```



Distribution of a single variable

A violin plot offers better insight about the actual shape of the distribution:

```
ggplot(politicalInformation, aes(x = female, y = age)) +
  geom_violin(draw_quantiles = c(0.25, 0.5, 0.75))  # horizontal lines for q
```



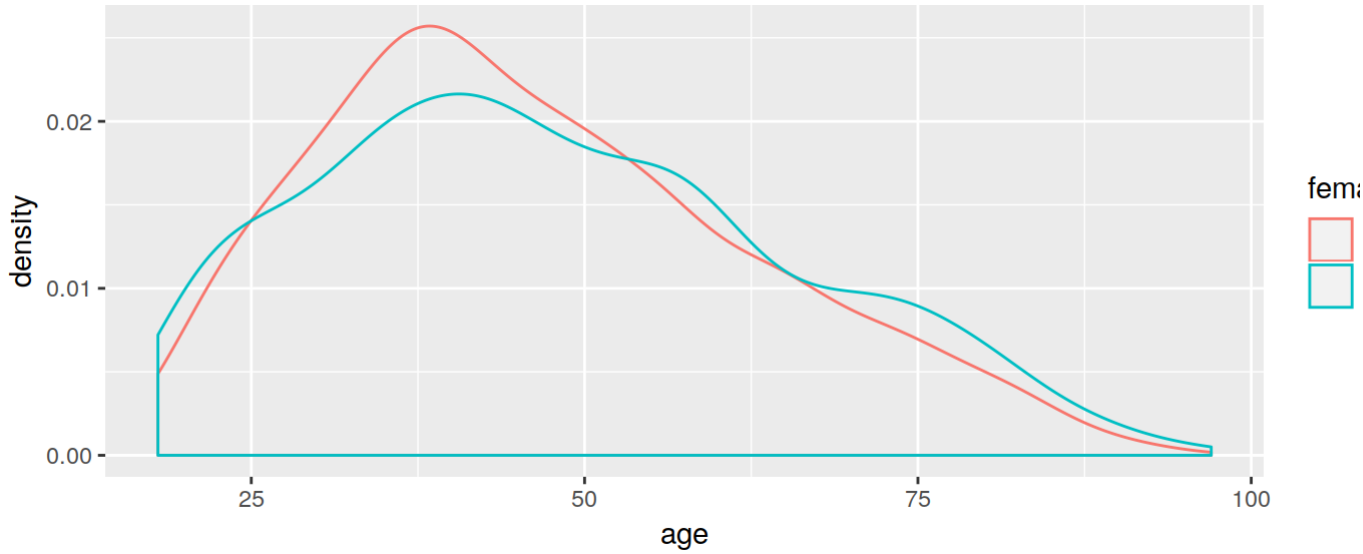
WZB Wissenschaftszentrum Berlin für Sozialforschung

33/60

Distribution of a single variable

A density plot also reveals that there are more younger men in the sample:

```
ggplot(politicalInformation, aes(age, color = female)) + geom_density()
```

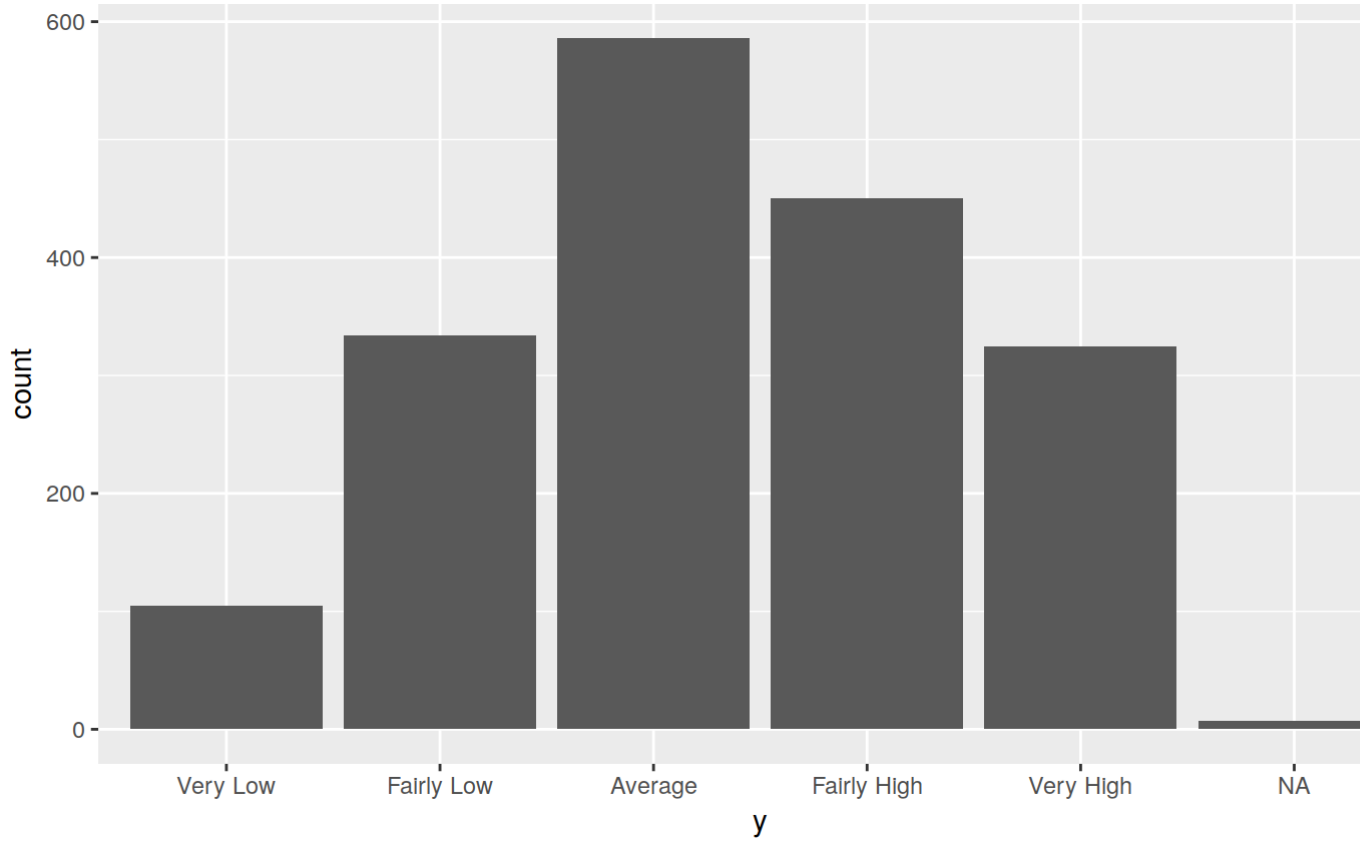


WZB
Wissenschaftszentrum Berlin für Sozialforschung

Counts of a single categorical variable

Histograms can be used to bin continuous variables. To show the distribution of a categorical variable we can use a simple bar chart:

```
ggplot(politicalInformation, aes(y)) + geom_bar()
```



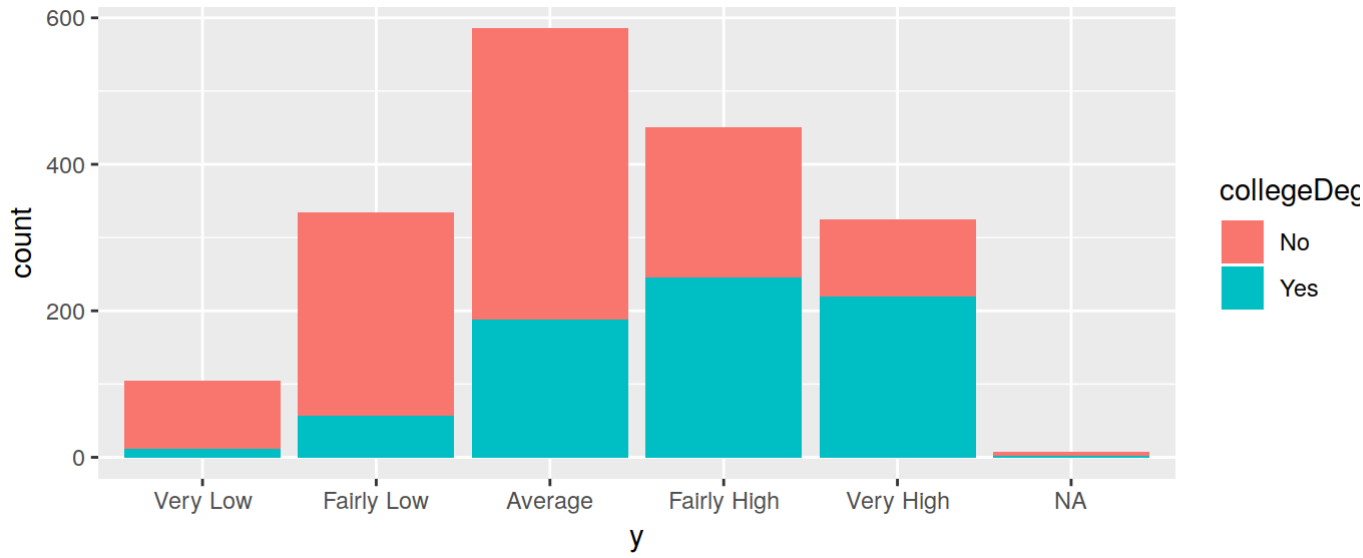
WZB Wissenschaftszentrum Berlin für Sozialforschung

Counts of a single categorical variable

A stacked and grouped bar charts allows to compare groups. Here, we compare groups based on the `collegeDegree` variable by mapping the "fill (color)" aesthetic to it.

By default, the bars get stacked:

```
ggplot(politicalInformation, aes(y, fill = collegeDegree)) +
  geom_bar()
```

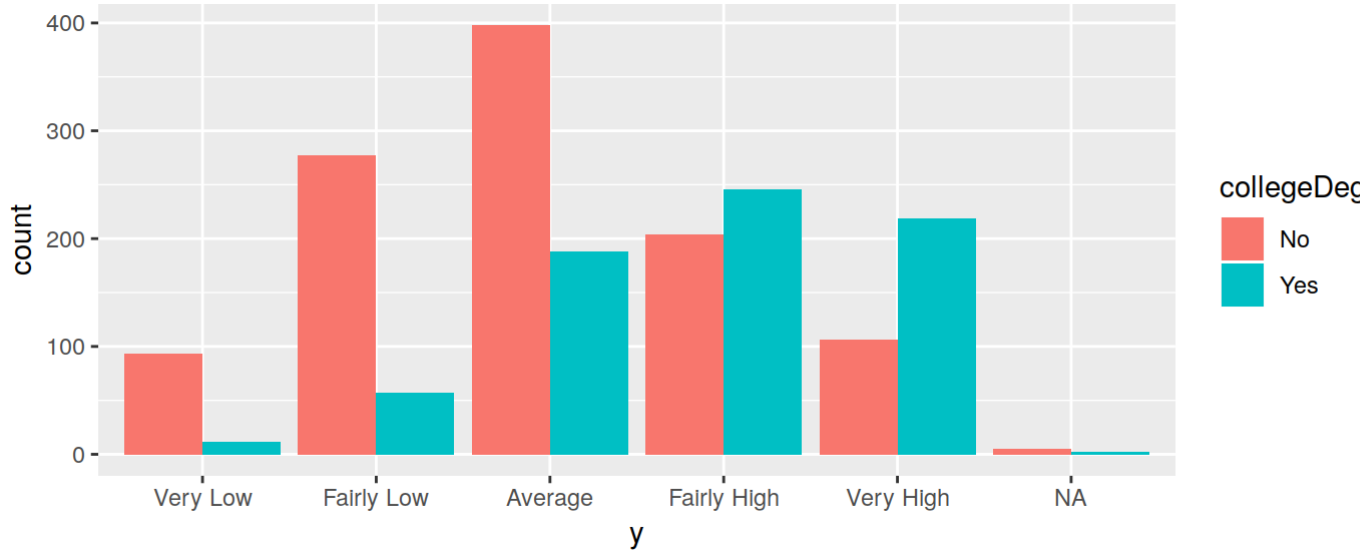


Counts of a single categorical variable

A stacked and grouped bar charts allows to compare groups. Here, we compare groups based on the `collegeDegree` variable by mapping the "fill (color)" aesthetic to it.

Setting the bars' `position` to `position_dodge()` creates a grouped bar chart:

```
ggplot(politicalInformation, aes(y, fill = collegeDegree)) +
  geom_bar(position = position_dodge())
```



Scatterplots

A scatterplot displays the relationship between two numeric variables. We'll use the `airquality` and have a look at the relationship between ozone and temperature.

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

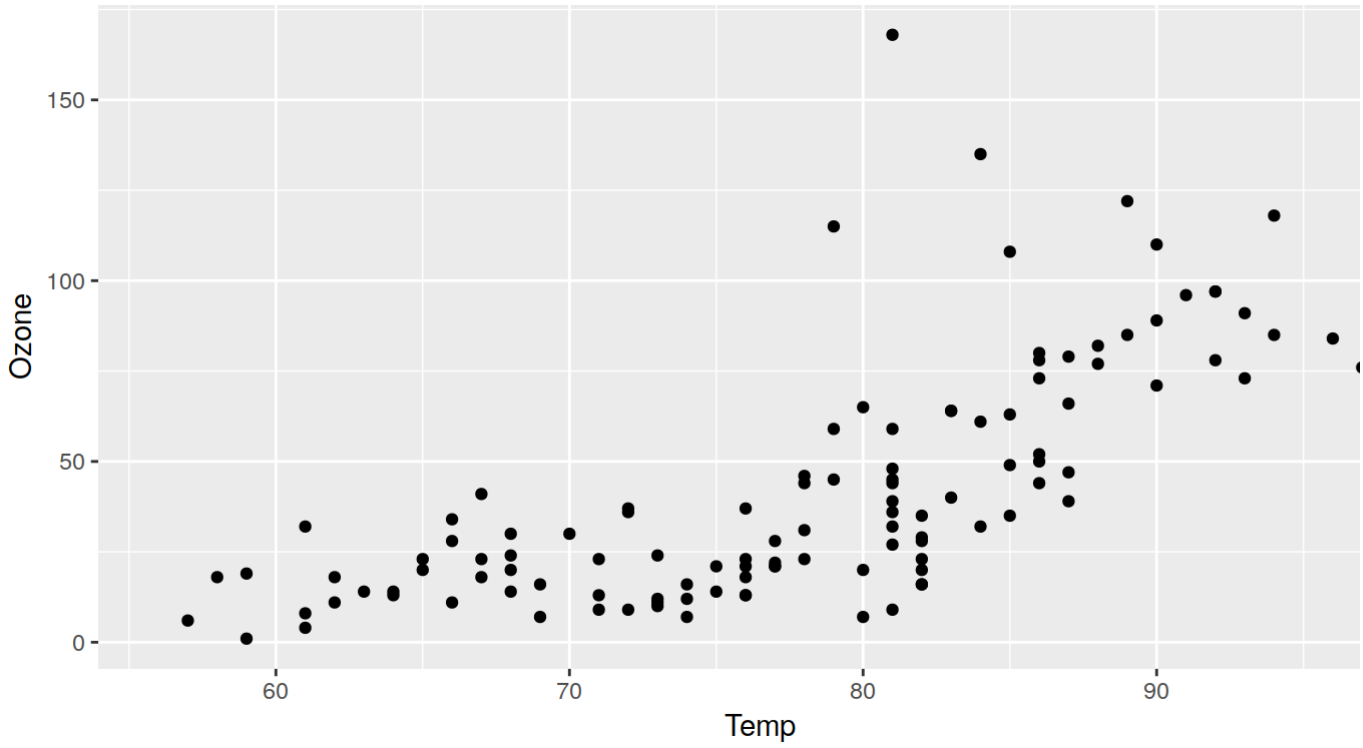
WZB Wissenschaftszentrum Berlin für Sozialforschung

Scatterplots

A scatterplot displays the relationship between **two numeric variables**. We'll use the `airquality` and have a look at the relationship between ozone and temperature.

We map `Ozone` to the y-axis and `Temp` to the x-axis and use `geom_point()`:

```
ggplot(airquality, aes(x = Temp, y = Ozone)) + geom_point()
```



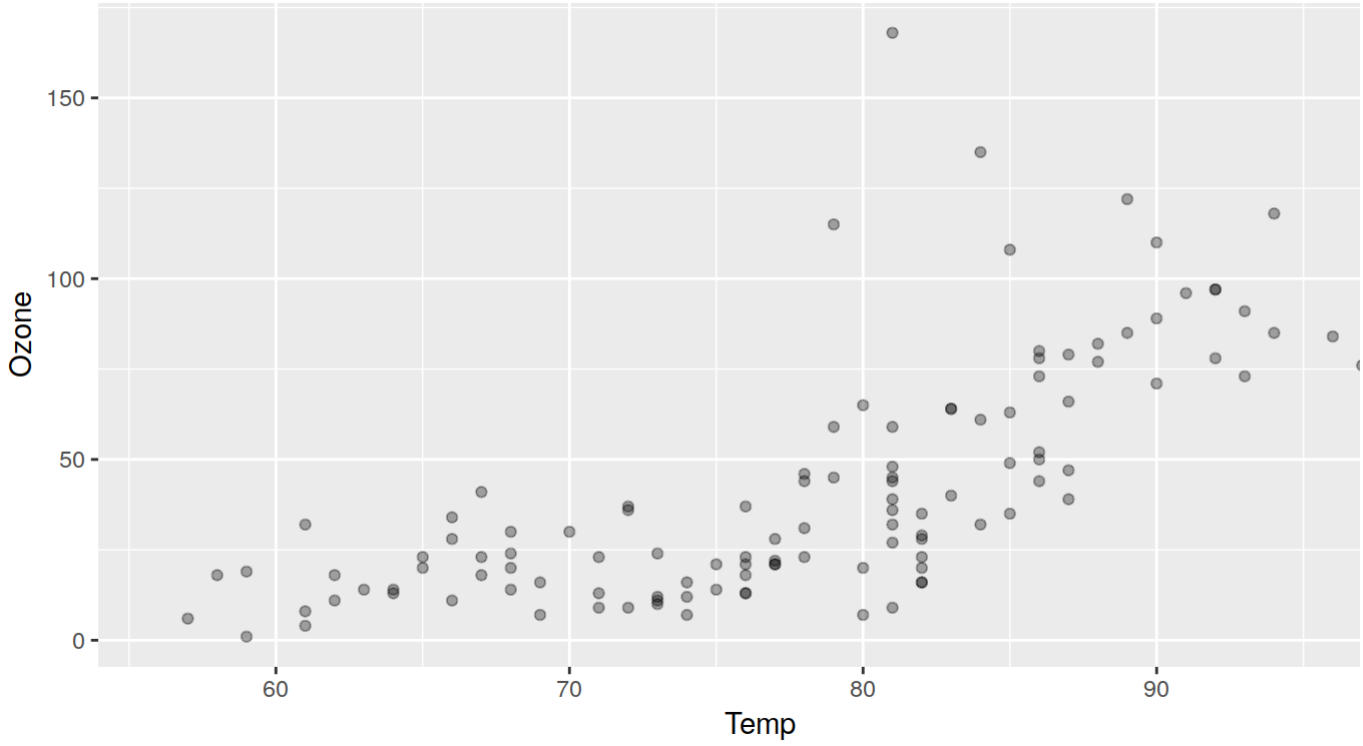
WZB Wissenschaftszentrum Berlin für Sozialforschung

Scatterplots

Overplotting can easily occur, especially with large data sets.

- happens when multiple data points are drawn on the same spot
- fix it with setting a semi-transparent fill color or apply jittering

```
ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(alpha = 0.33)  # alpha of 0 is invisible, 1 is opaque
```



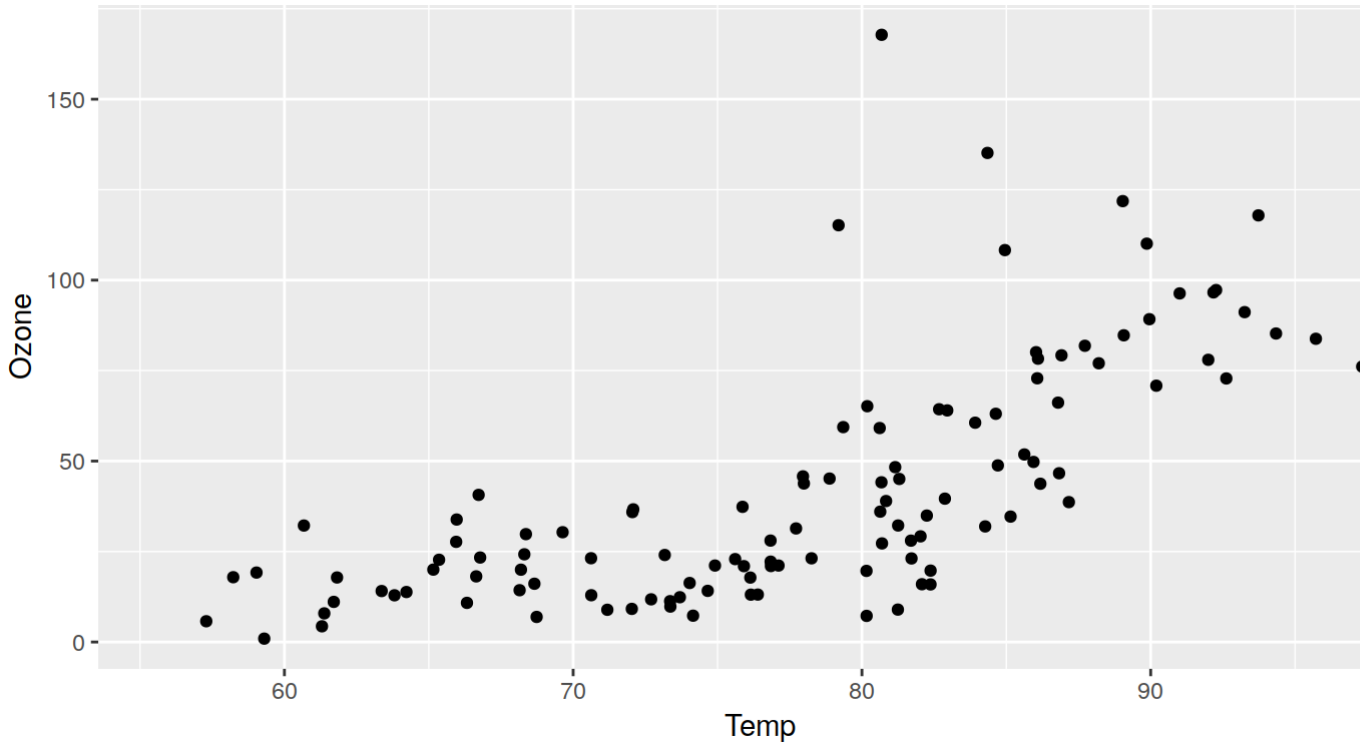
WZB
Wissenschaftszentrum Berlin für Sozialforschung

Scatterplots

Overplotting can easily occur, especially with large data sets.

- happens when multiple data points are drawn on the same spot
- fix it with setting a semi-transparent fill color or apply jittering

```
ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(position = position_jitter())
```

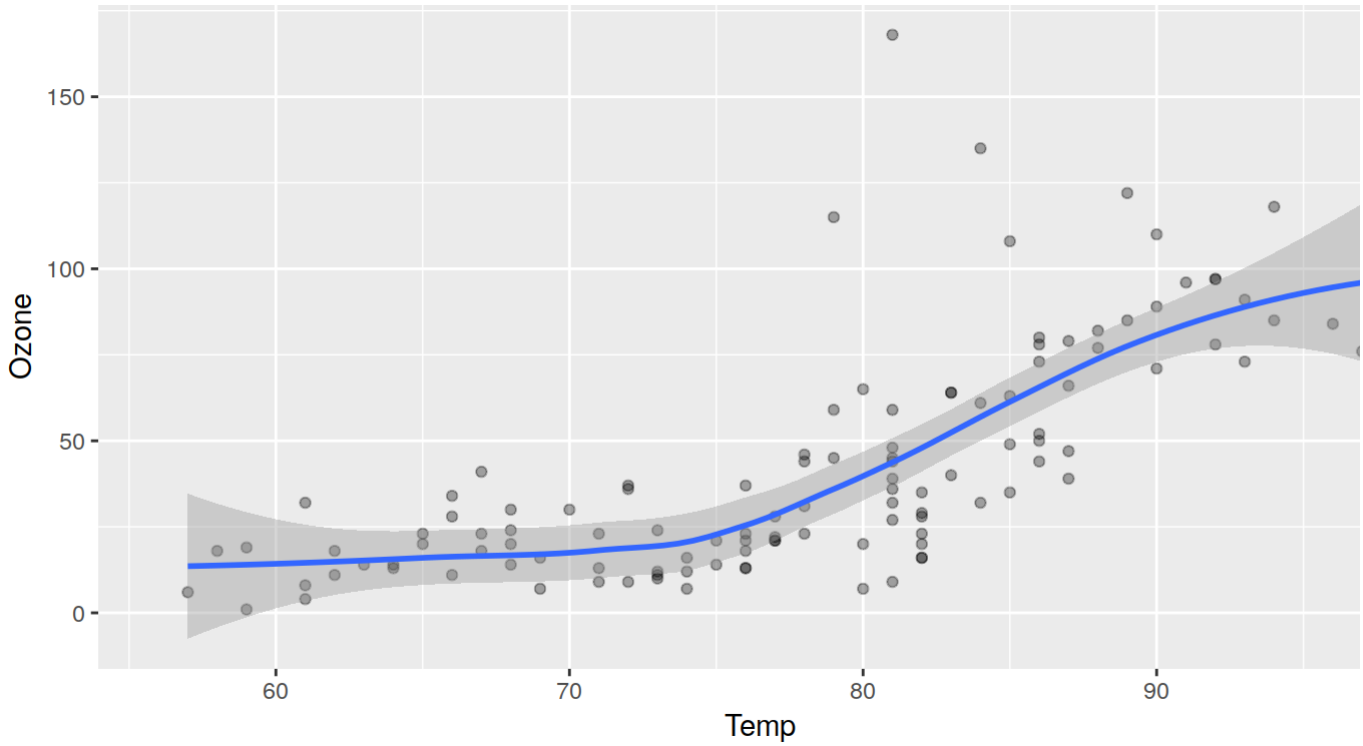


WZB Wissenschaftszentrum Berlin für Sozialforschung

Scatterplots

`geom_smooth()` aids the eye in seeing patterns by adding a (local polynomial) regression line and its confidence interval:

```
ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point(alpha = 0.33) +
  geom_smooth()
```



WZB
Wissenschaftszentrum Berlin für Sozialforschung

Line graphs

A line graph can be used whenever you have an **ordered numeric variable on the x-axis**, such as years.

We'll use a subset of the `presidentialElections` data set from the package pscl:

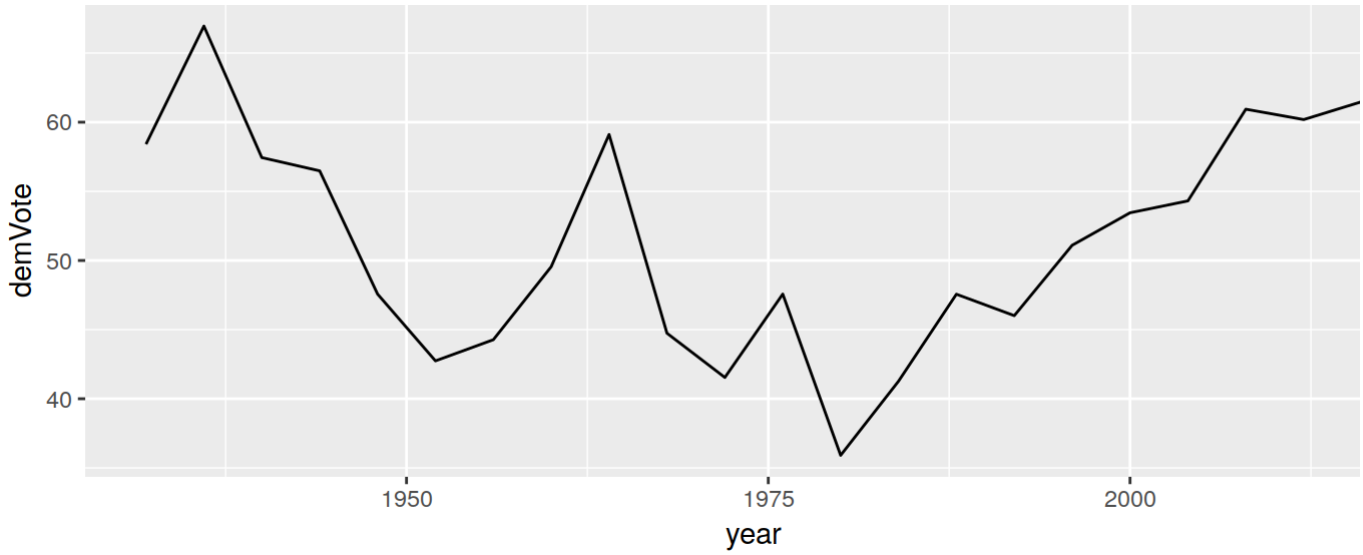
```
(cal_pres <- presidentialElections %>% filter(state == 'California'))
```

```
## # A tibble: 22 x 4
##    state      demVote  year south
##    <chr>        <dbl> <int> <lgl>
##  1 California    58.4  1932 FALSE
##  2 California    67.0  1936 FALSE
##  3 California    57.4  1940 FALSE
##  4 California    56.5  1944 FALSE
##  5 California    47.6  1948 FALSE
##  6 California    42.7  1952 FALSE
##  7 California    44.3  1956 FALSE
##  8 California    49.6  1960 FALSE
##  9 California    59.1  1964 FALSE
## 10 California    44.7  1968 FALSE
## # ... with 12 more rows
```

Line graphs

We map `year` on the x-axis, the vote share for Democrats `demVote` on the y-axis and add a `geom_line()` layer:

```
ggplot(cal_pres, aes(x = year, y = demVote)) + geom_line()
```

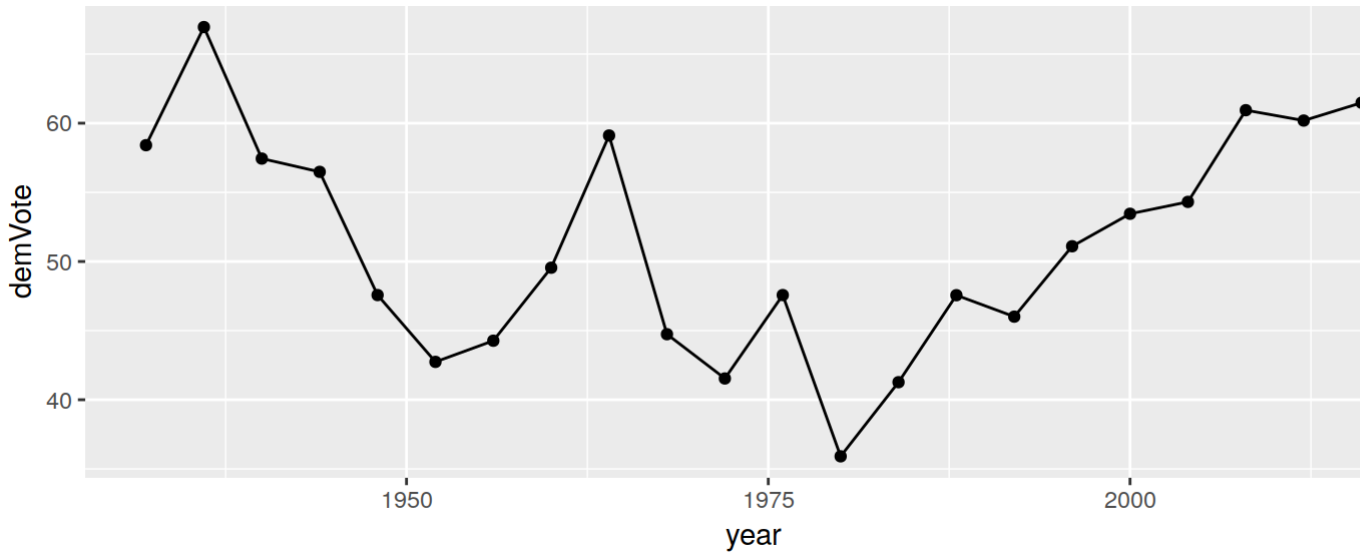


WZB Wissenschaftszentrum Berlin für Sozialforschung

Line graphs

We can additionally add points like in a scatterplot:

```
ggplot(cal_pres, aes(x = year, y = demVote)) +
  geom_line() +
  geom_point()
```



WZB
Wissenschaftszentrum Berlin für Sozialforschung

Line graphs

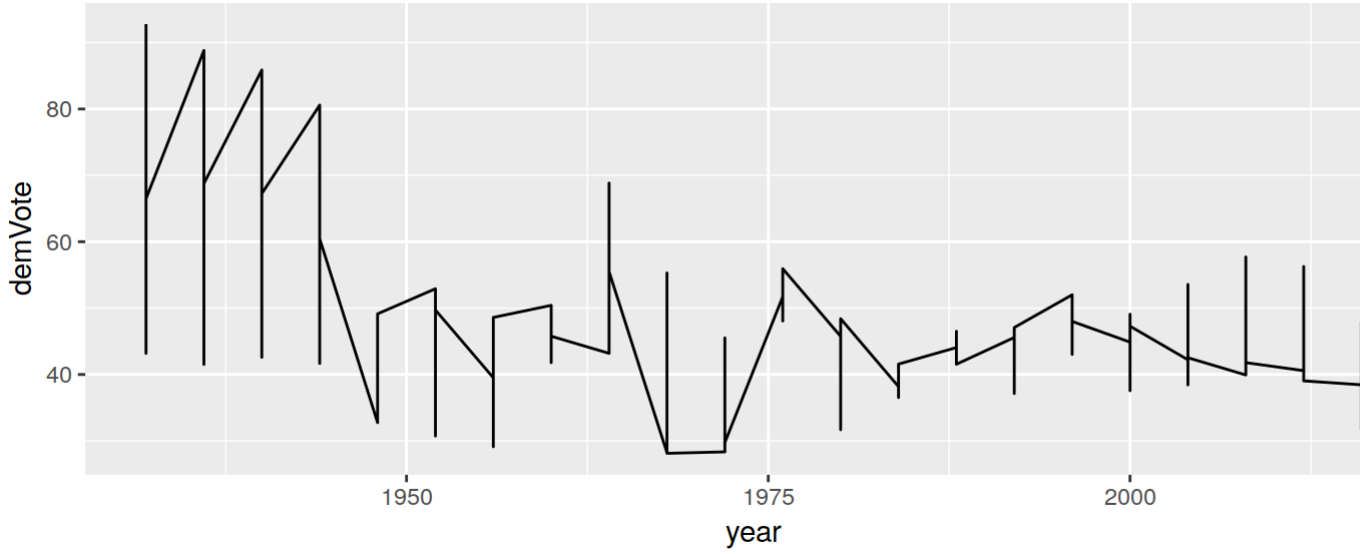
Again, we'll plot the historical election results from four states (data set `presidentialElections` from package pscl) stored in an object `sampled_pres`:

```
## # A tibble: 88 x 4
##    state        demVote  year south
##    <chr>          <dbl> <int> <lgl>
##  1 Louisiana       92.8  1932 TRUE
##  2 Maine           43.2  1932 FALSE
##  3 South Dakota    63.6  1932 FALSE
##  4 Tennessee       66.5  1932 TRUE
##  5 Louisiana       88.8  1936 TRUE
##  6 Maine           41.5  1936 FALSE
##  7 South Dakota    54.0  1936 FALSE
##  8 Tennessee       68.8  1936 TRUE
##  9 Louisiana       85.9  1940 TRUE
## 10 Maine           48.8  1940 FALSE
## # ... with 78 more rows
```

Line graphs

If we simply use the same command as before, we produce some interesting chaos:

```
ggplot(sampled_pres, aes(x = year, y = demVote)) + geom_line()
```

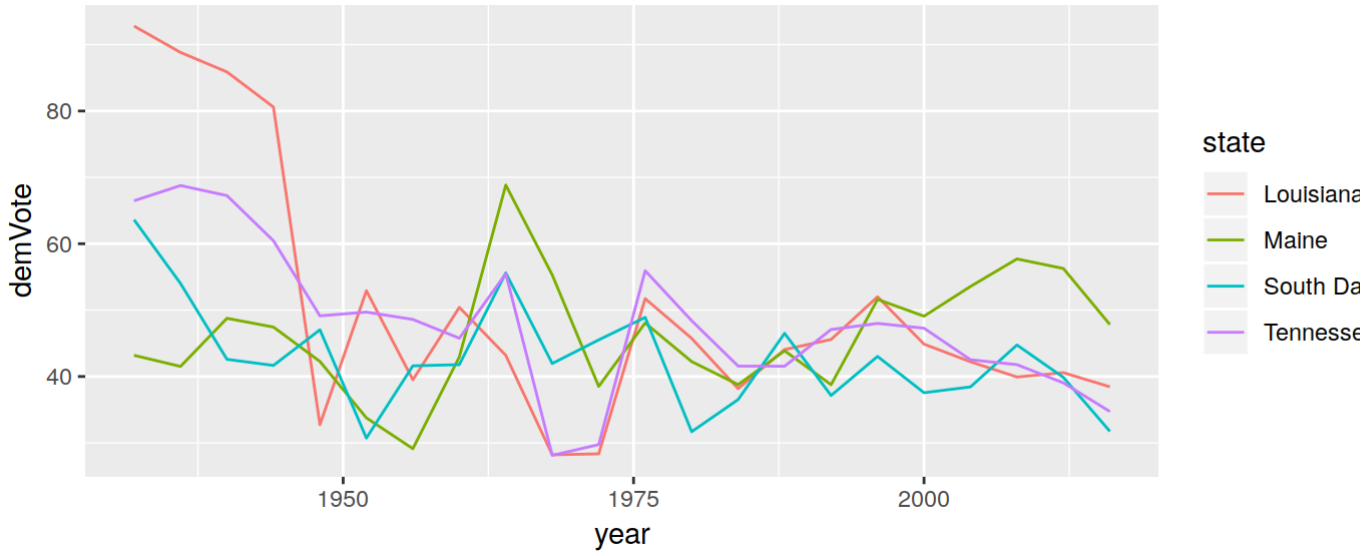


WZB
Wissenschaftszentrum Berlin für Sozialforschung

Line graphs

We have to add another aesthetic mapping (`color = state`) in order to tell apart the states' lines:

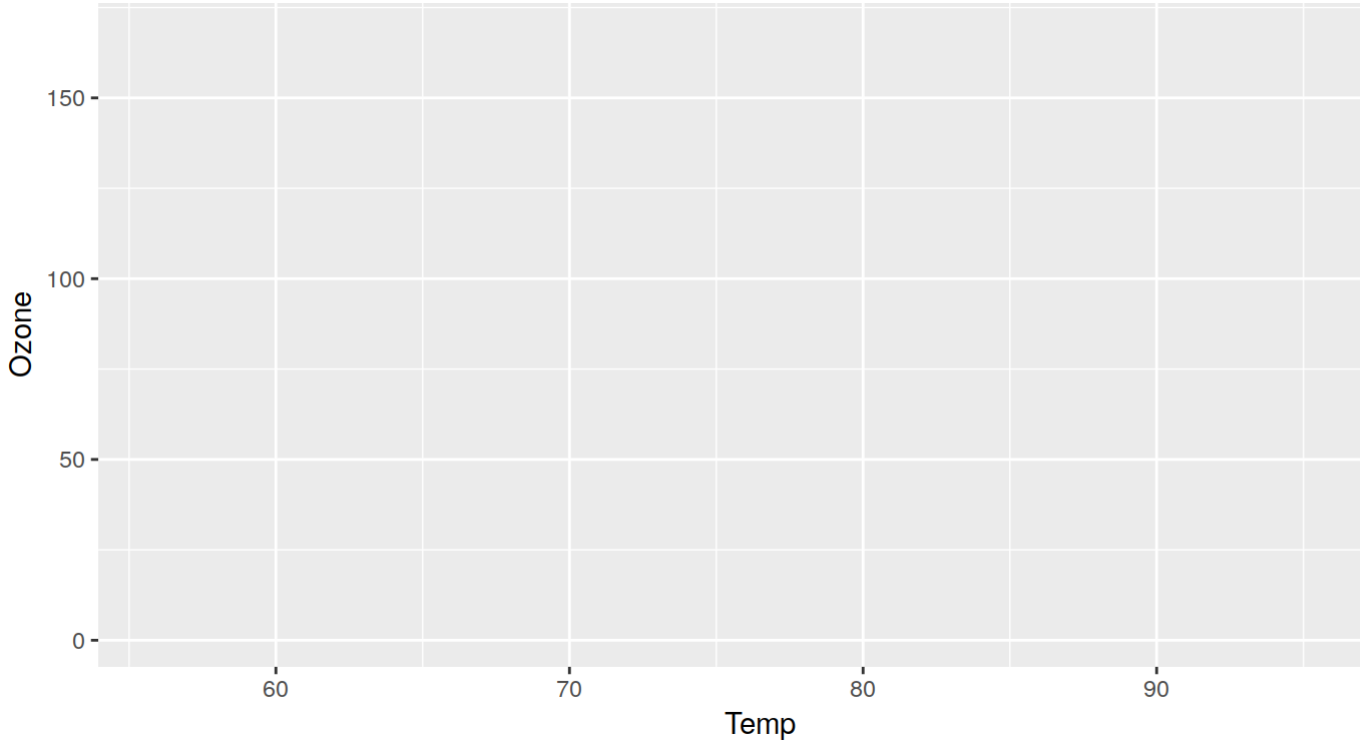
```
ggplot(sampled_pres, aes(x = year, y = demVote, color = state)) + geom_line(
```



Saving plots

A ggplot object can be assigned a name just as any other object in R:

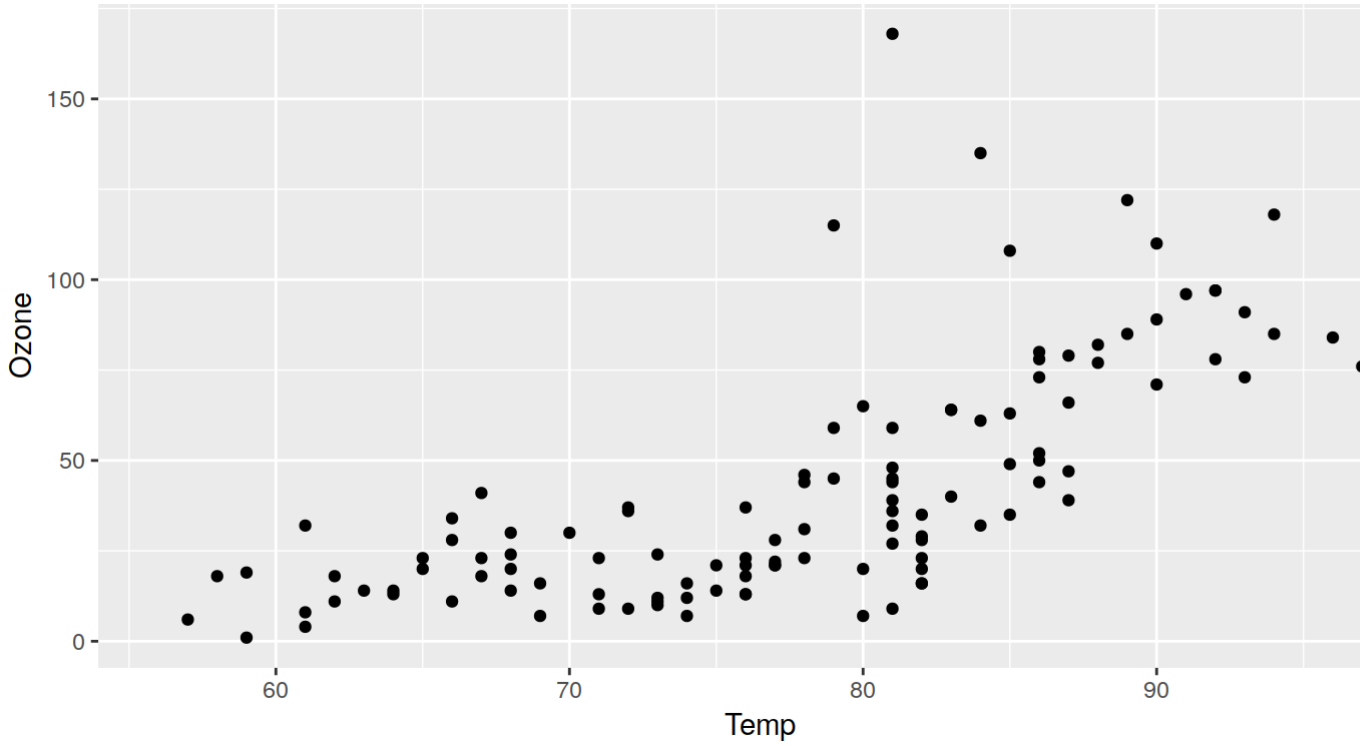
```
myplot <- ggplot(airquality, aes(x = Temp, y = Ozone))
myplot   # shows an "empty" plot
```



Saving plots

You can re-use the ggplot object and try out different layers:

```
myplot + geom_point()
```

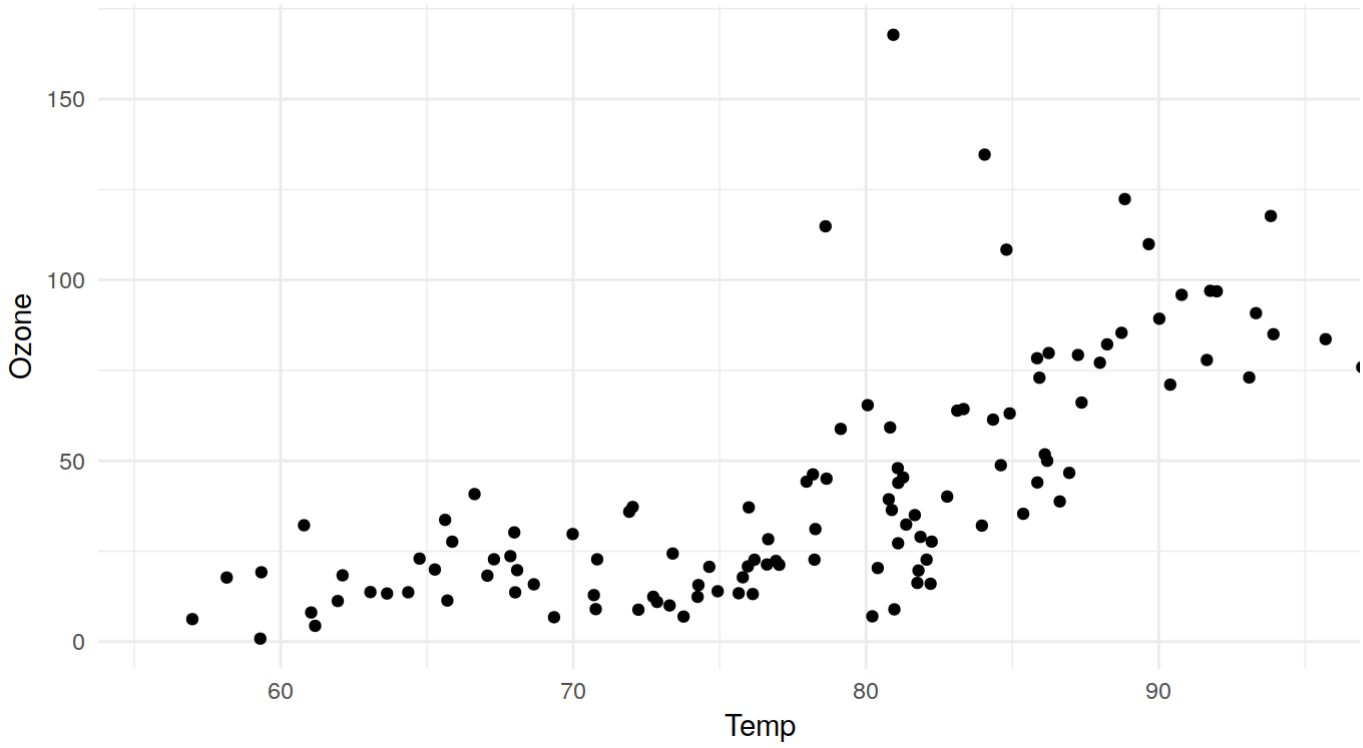


WZB
Wissenschaftszentrum Berlin für Sozialforschung

Saving plots

You can re-use the ggplot object and try out different layers or themes:

```
myplot + geom_point(position = position_jitter()) + theme_minimal()
```



WZB
Wissenschaftszentrum Berlin für Sozialforschung

Saving plots

You can eventually save the plot to disk with `ggsave()`:

```
final_plot <- myplot + geom_point(position = position_jitter())
ggsave('07plotting-resources/example_saved_figure.png',
       plot = final_plot,
       width = 4,
       height = 3)
```

There are several options to configure the output file (see `?ggsave`):

- plot dimensions (by default in inch)
- plot resolution
- format (PNG, PDF, etc.) – determined by file extension

Common mistakes

A very common mistake is to accidentally put + on a new line:

```
ggplot(airquality, aes(x = Temp, y = Ozone))
  + geom_point()
```

```
Error: Cannot use "+.gg()" with a single argument. Did
you accidentally put + on a new line?
```

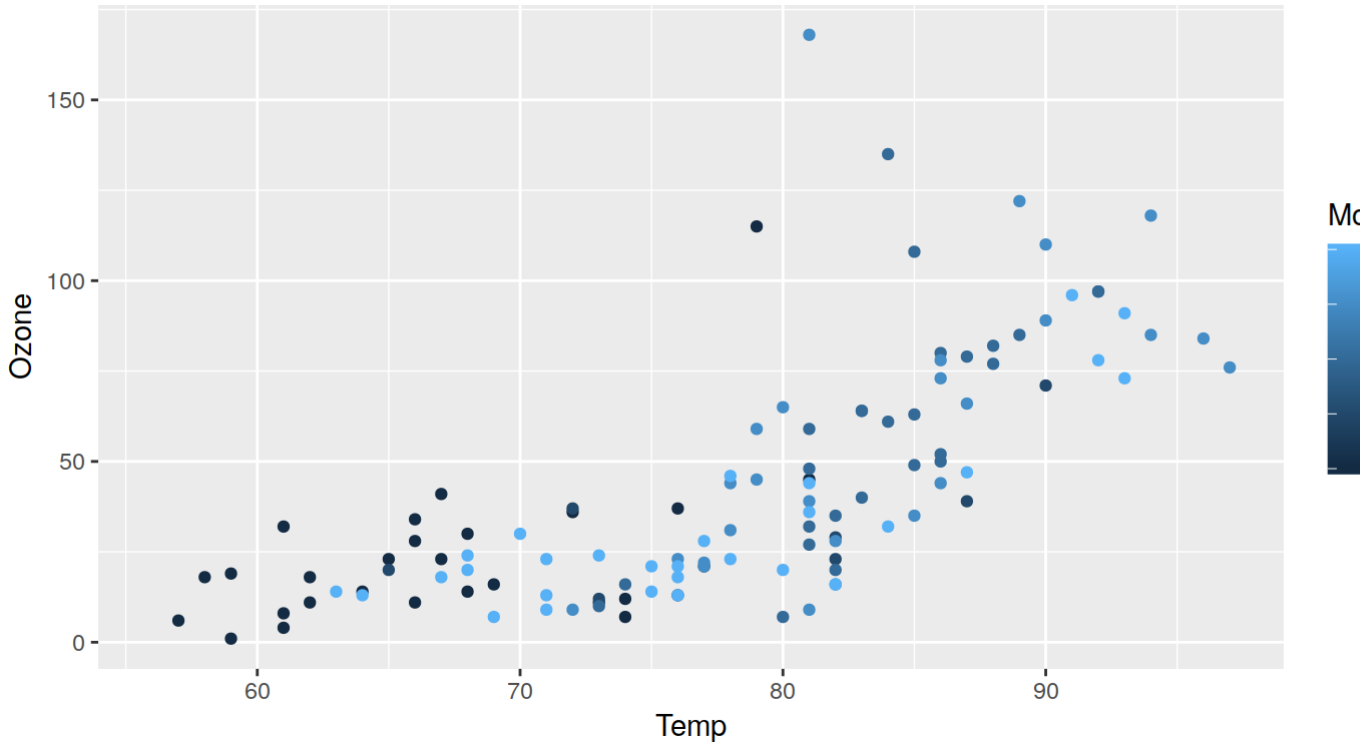
The + operator must appear before the line break (the same is true for other operators like %>% used in dplyr):

```
ggplot(airquality, aes(x = Temp, y = Ozone)) +
  geom_point()
```

Common mistakes

The type of your variables determines its scale for plotting. E.g. here you might want to use a discrete scale:

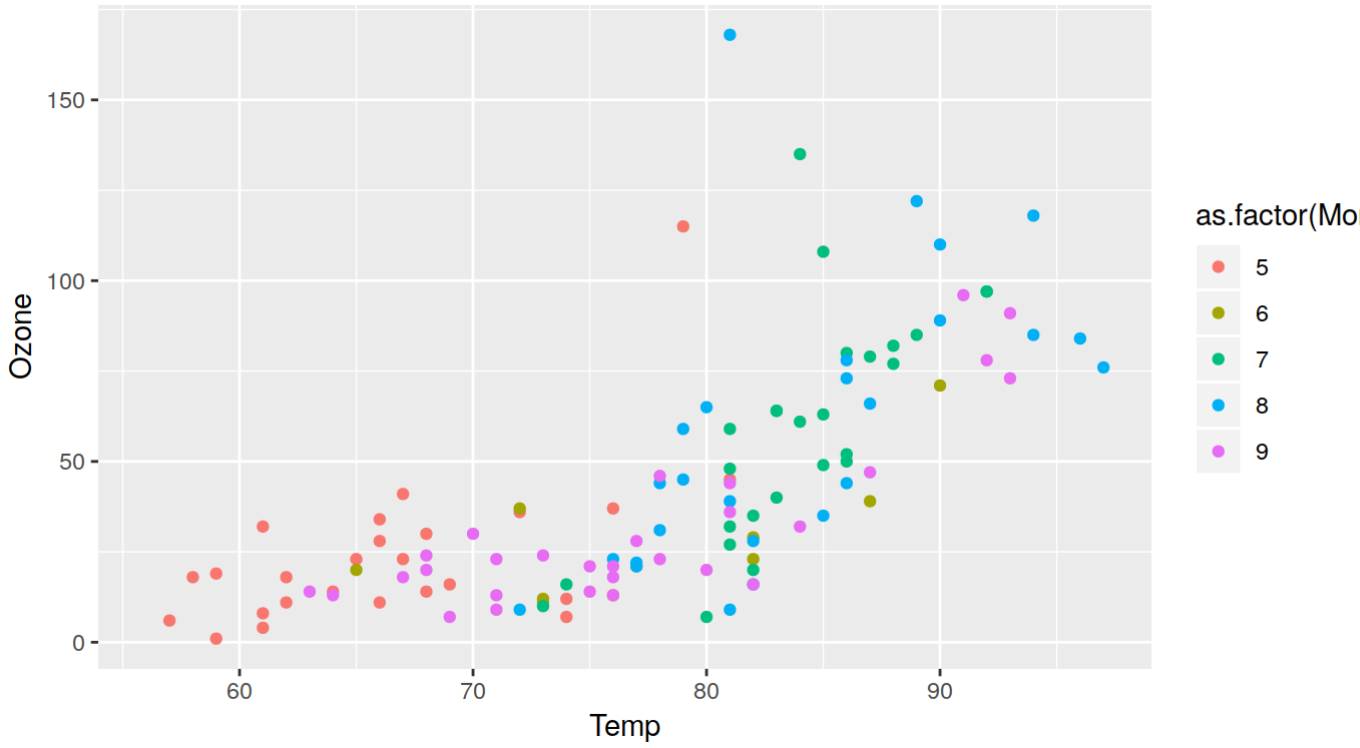
```
ggplot(airquality, aes(x = Temp, y = Ozone, color = Month)) +
  geom_point()
```



Common mistakes

Converting the numerical to a factor tells ggplot that a discrete scale is appropriate:

```
ggplot(airquality, aes(x = Temp, y = Ozone, color = as.factor(Month))) +
  geom_point()
```



What we didn't cover

This was only a short intro about a few types of plots that can be made with ggplot2. For more advanced plots you should learn about:

- other **geoms** that allow to create other types of plots
- **facets** to create "small multiples"
- **scales** to adjust colors, legends, scale intervals, etc.
- **themes** to adjust the overall appearance of a plot

ggplot extensions

There are several packages that extend ggplot's capabilities:

- ggmap: Spatial visualization (plot geographic data on maps)
- ggraph: Graphs and networks
- ggdag: Analyze and create elegant Directed Acyclic Graphs (DAGs)
- ggmosaic: Mosaic plots
- ggwordcloud: Word clouds
- gghighlight: Highlight lines and points
- cowplot: Combine multiple labelled plots
- plotly: Create interactive web graphics

Documentation and other resources

The documentation is excellent. If you don't know how a "geom" can be controlled, have a look at it's documentation, e.g.

```
[...]
Aesthetics:
     'geom_point' understands the following aesthetics (required
     aesthetics are in bold):
        • *'x'*
        • *'y'*
        • 'alpha'
        • 'colour'
        • 'fill'
        • 'group'
        • 'shape'
        • 'size'
        • 'stroke'
[...]
```

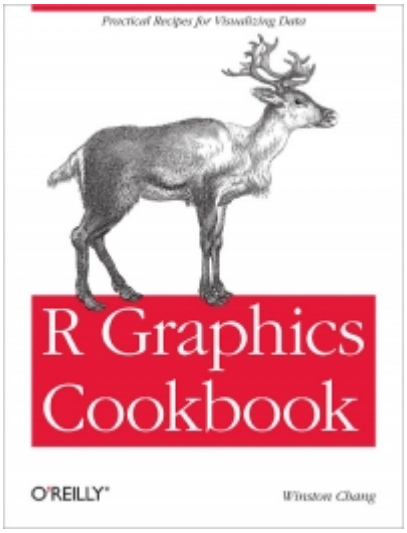
→ you must specify the aesthetics x and y

→ you can make several other properties (e.g. transparency or fill color) dependent on some variable

Documentation and other resources

For more, see:

- ggplot2 website incl. reference: https://ggplot2.tidyverse.org/
- W. Chang, R Graphics Cookbook
- The R Graph Gallery: https://www.r-graph-gallery.com/



Tasks

See dedicated tasks sheet on the tutorial website.