Data Analysis using R

Visualization

Sven Werenbeck-Ueding

21.09.2023

Source: Wickham and Grolemund (2016)

Prerequisites

```
# Load required packages
# install.packages("wooldridge")
library(tidyverse)
library(RColorBrewer)

# Load the CPR data set from the wooldridge package
df_cpr <- wooldridge::wage1 %>%
  mutate(
    sex = ifelse(female == 1, "Female", "Male"),
    region = case_when(
      west == 1 ~ "West",
      south == 1 ~ "South",
      northcen == 1 ~ "North Central",
      T ~ "East"
    )
  )
```

Graphics in base R

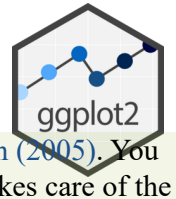
- Functions for graphics in base R are fast, but limited
- Useful for simple tasks such as quickly plotting histograms
- Graphics content can only be written on top of each other
- Elements of a plot cannot be modified or deleted

```
hist(df_cpr$wage)
```

Graphics in base R

- `base` R is rather limited when it comes to creating beautiful, customized plots
- Layered approach: Adds plot components on top of each other
  - Opens up for more flexibility
  - Layers can overwrite each other to allow for post-hoc editing of components

ggplot2



> `ggplot2` is a system for declaratively creating graphics, based on "The Grammar of Graphics" by Wilkinson (2005). You provide the data, tell `ggplot2` how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.
>
> Wickham (2016)

The Layered Grammar of Graphics

Component	Description
Layer	Geometric elements (**geoms**, e. g. points and lines) and statistical transformations (**stats** that summarize the data)
Scale	Mapping of data to aesthetics (e. g. color, shape or size) of geoms and draws legends and axes
Coord	Mapping of data to the plane of the graphic and provides axes/gridlines (usually the Cartesian coordinate system)
Facet	Plots subsets of the data
Theme	Styling of the plot such as fonts and background

The Layered Grammar of Graphics



From top left to bottom: Geometric objects, scales and coordinate system (Wickham, 2010).

The Layered Grammar of Graphics

Source: Wickham (2010)

The Layered Grammar of Graphics

- Graphics template consists of seven parameters (in brackets)
- *Any* plot can be generated from this template
- `ggplot2` provides defaults for everything except for data, mappings and geoms
- Syntax is similar to using the pipe operator

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

Source: Wickham and Grolemund (2016)

Initialize a Plot

```
?ggplot
```

> `ggplot()` initializes a ggplot object. It can be used to declare the input data frame for a graphic and to specify the set of plot aesthetics intended to be common throughout all subsequent layers unless specifically overridden.

Initialize a Plot – Function Call and Arguments

```
ggplot(
  data = NULL,
  mapping = aes(),
  ...
)
```

Initialize a Plot – Function Call and Arguments

```
ggplot(
  data = NULL,
  mapping = aes(),
  ...
)
```

`data`

- Default data set to use for the plot
- If necessary will be converted to a `data.frame`
- If not specified, must be supplied in each layer added to the plot

Initialize a Plot – Function Call and Arguments

```
ggplot(
  data = NULL,
  mapping = aes(),
  ...
)
```

`mapping`

- Default list of aesthetic mappings to use for plotting
- If not specified must be supplied in each layer added to the plot

Initialize a Plot

Code

> Initialize a `ggplot` object using the CPR data set and map `educ` to the x-axis and `wage` to the y-axis.

```
ggplot(data = df_cpr,
       mapping = aes(x = exper,
                     y = wage))
```

Plot

Add Geometric Objects

Code

> Create a scatterplot of `wage` vs. `educ` by adding points as geoms.

```
ggplot(df_cpr,
       aes(x = exper,
           y = wage)) +
  geom_point()
```

Plot

Map Data to Geom Aesthetics

Code

> Map `sector` to the `color` aesthetics and change the transparency of points to handle overlapping of points.

```
ggplot(data = df_cpr,
       mapping = aes(x = exper,
                     y = wage,
                     color = sex)) +
  geom_point(alpha = .5)
```

Plot

Add Titles and Labels

Code

> Add a title to the plot and modify axis labels and legend title.

```
ggplot(df_cpr,
       aes(x = exper,
           y = wage,
           color = sex)) +
  geom_point(alpha = .5) +
  labs(title = "Hourly Wage ~ Years of Experience",
       x = "Years of Experience",
       y = "Wage [USD/hour]",
       color = "Sex")
```

Plot

Change the Styling

Code

> Change the theme to `theme_bw()` and adjust the position of the legend.

```
ggplot(df_cpr,
       aes(x = exper,
           y = wage,
           color = sex)) +
  geom_point(alpha = .5) +
  labs(title = "Hourly Wage ~ Years of Experience",
       x = "Years of Experience",
       y = "Wage [USD/hour]",
       color = "Sex") +
  theme_bw() +
  theme(legend.position = "top")
```

Plot

Create Facets of Plots

Code

> Create facets to display subplots for `region`.

```
ggplot(df_cpr,
       aes(x = exper,
           y = wage,
           color = sex)) +
  geom_point(alpha = .5) +
  labs(title = "Hourly Wage ~ Years of Experience",
       x = "Years of Experience",
       y = "Wage [USD/hour]",
       color = "Sex") +
  facet_wrap(~ region) +
  theme_bw() +
  theme(legend.position = "top")
```

Plot

i

For brevity, we will use the same `ggplot` object (short-hand `gg`) over the next slides. This way, we need not specify every time which data to use for plotting and only have to add geoms.

```
gg <- df_cpr %>%
  ggplot()
```

Univariate Graphs

Categorical Data

Count Bars

Bar Plot

```
bar_plot <- gg +
  geom_bar(aes(x = region)) +
  labs(x = "US Region of Residence",
       y = "Count")
```

Frequency Bars

Bar Plot with Frequencies

By mapping `..count../sum(..count..)` to the y-axis, the scale of the y-axis is changed to display frequencies:

```
freq_bar_plot <- gg +
```

Numerical Data

Histogram

Histogram

```
hist <- gg +
  geom_histogram(aes(x = wage)) +
  labs(x = "Wage [USD/hour]",
       y = "Count")
```

Density

Kernel Density Plot

```
density_plot <- gg +
  geom_density(aes(x = wage)) +
  labs(x = "Wage [USD/hour]",
```

Bivariate Graphs

Categorical vs. Categorical

Stacked Bars

Stacked Bar Plot

```
stack_bar_plot <- gg +
  geom_bar(aes(x = sex, fill = region)) +
  labs(x = "Sex",
       y = "Count",
       fill = "US Region of Residence")
```

Grouped Bars

Grouped Bar Plot

```
group_bar_plot <- gg +
  geom_bar(aes(x = sex, fill = region),
```

Numerical vs. Numerical

Scatter

Scatterplot

```
scatter_plot <- gg +
  geom_point(aes(x = exper, y = wage)) +
  labs(x = "Years of Work Experience",
       y = "Wage [USD/hour]")
```

Line

Line Plot

```
line_plot <- wooldridge::approval %>%
  mutate(period = year + 1/12*month) %>%
  ggplot() +
```

Categorical vs. Numerical

Grouped Density

Grouped Kernel Density Plot

```
grouped_density_plot <- gg +
  geom_density(aes(x = wage, fill = sex),
               alpha = .4) +
  labs(x = "Wage [USD/hour]", y = "Density")
```

Error Bars

Mean with Standard Error Bars

```
error_bars <- group_by(df_cpr, sex) %>%
  summarize(mean_wage = mean(wage),
            sd_wage = sd(wage)) %>%
```

Multivariate Graphs

Creating Multivariate Graphs

Including multiple variables into one graph can be achieved by combining the geometric objects seen on the previous slides, through mapping of different aesthetics and facetting.

Mapping

Variables can be mapped to various aesthetics of geometric objects, such as

- `shape` (only categorical)
- `size` (only numerical)
- `color` (both)
- ...

Creating Multivariate Graphs

Including multiple variables into one graph can be achieved by combining the geometric objects seen on the previous slides, through mapping of different aesthetics and facetting.

Facets

Creating subplots for groups of variables, using `facet_wrap()` for one-dimensional and `facet_grid()` for two-dimensional sequences of panels.

Variables to create facets for can be passed to `facet_*()` using R formulas, e. g. `~ sector` (one-dimensional) or `female ~ sector` (two-dimensional). Formulas of type `~ sector + region` can be used to create groups for combinations of variables.

Correlation Matrix

Plot

Code

```
library(ggcorrplot)

corr_plot <- df_cpr %>%
  select(wage, educ, exper, tenure, numdep) %>% # Select numerical variables
  cor(use = "complete.obs") %>%                 # Corr. matrix for observations w/o NAs
  # Display correlation in a ggplot object
  ggcorrplot(hc.order = T,                      # Sort variables
             type = "lower",                    # Show only lower half of the matrix
             lab = T)                           # Show correlations in tile labels

corr_plot
```

Customizing Plots

Color Schemes

Choosing an appropriate color scheme for your graph is an important step in conveying the information!

Make sure that your plot is readable. Avoid light on light and dark on dark colors!

RColorBrewer

```
?RColorBrewer
```

> The `RColorBrewer` package provides color palettes from ColorBrewer 2.0 for creating thematic graphs. Run the `display.brewer.all()` function for a display of available color palettes.

Color Palettes for Numerical Data

Sequential

Sequential color palettes assign light colors to low values of the data and dark colors to high values, e. g.

Diverging

Diverging color palettes emphasize differences between extreme values by assigning contrasting dark colors to low and high values while assigning light colors to mid-range values, e. g.

Color Palettes for Categorical Data

Qualitative

Qualitative color palettes are used, when there is no difference in magnitude between values, i. e. for categorical data. The primary concern is to visualize differences between classes, e. g.

Change Discrete Color Palettes

Code

- Discrete color palettes for `ggplot` aesthetics, such as `color` or `fill`, can be changed to `RColorBrewer` palettes using the `scale_*_brewer()` functions, where the name of the palette can be passed as a string to the argument `palette`
- `RColorBrewer` supports only discrete color palettes

```
ggplot(
  data = df_cpr,
  mapping = aes(
    x = exper,
    y = wage,
    color = region
  )
) +
  geom_point() +
  scale_color_brewer(palette = "Set1")
```

Plot

Change Continuous Color Palettes

Code

For continuous color palettes, it is best to use `ggplot2`'s built-in functions:

- `scale_*_gradient()`: Creates a two color gradient based on strings passed to arguments `low` and `high`
- `scale_*_gradient2()`: Creates a diverging color gradient based on strings passed to arguments `low`, `mid` and `high`. The `midpoint` defaults to 0 and, in most cases, has to be set manually to ensure a meaningful color gradient.

```
ggplot(
  data = df_cpr,
  mapping = aes(
    x = exper,
    y = wage,
    color = tenure
  )
) +
  geom_point() +
  scale_color_gradient(low = "yellow",
                       high = "red")
```

Plot

Themes

```
?theme
```

Themes are used to modify non-data components of plots, such as titles, labels, ticks, fonts for text elements, background etc. `ggplot2` provides out-of-the-box themes for quick and easy adjustments of the overall look:

`theme_gray()`


`theme_bw()`


`theme_light()`


`theme_dark()`


`theme_minimal()`


...

More Themes

The `ggthemes` package offers more out-of-the-box themes. Have a look at e. g. the Stata theme:

Custom Themes

Plot

- The appearance of presentations and seminar papers greatly improves by using a consistent design throughout your work
- When outputting several plots, managing the design of each plot individually becomes very cumbersome
- Instead, you may want to think about writing your own custom theme function that is applied to each plot
- Changing your theme function then automatically changes the appearance of all your plots

Code

```
grouped_density_plot <- df_cpr %>%
  ggplot() +
  geom_density(aes(x = wage, fill = sex),
               alpha = .4) +
  labs(title = "Wage Distribution Across Sex and Region",
       x = "Wage [USD/hour]",
       y = "Density",
       fill = "Sex") +
```

Custom Themes

Plot

- A good starting point for a custom theme is `ggplot`'s `theme_classic()`: A black-and-white theme with minimal styling
- Take some time to think about how the components of your plot should look like
    - Font family and size
    - Legend position and orientation
    - Background and border colors
    - ...
- Play around with the options until you decide on your design
- Run `?ggplot2::theme` for a full list of theme arguments

Code

```
theme_custom <- function() {
  theme_bw()
```

Custom Themes

Plot

- Once you decided on what your design should look like, start implementing the options in your function
- The theme arguments of `theme_classic()` can be overwritten using `%+replace%`
- We may want to consider making the following changes to our design:
    - Change the default text font to "Helvetica"
    - Increase the base font size to 14.
    - Left-align the title, set its font to bold and its size to 150% of the base font size
    - Increase the top and bottom margin around the title

Code

```
theme_custom <- function(base_family = "Helvetica",
                         base_size = 14,
                         padding = base_size*.75,
                         title_size = base_size*1.25) {
```

Custom Themes

Plot

- Adjust the grid: Darker and thicker dashed lines to increase the grid's visibility
- Change the legend position and orientation
- Set the facet background to dark blue, its text color to white and increase the padding around the text

Code

```
theme_custom <- function(base_family = "Helvetica",
                         base_size = 12,
                         padding = base_size*.75,
                         title_size = base_size*1.25) {
  theme_bw() %+replace%
    theme(
      # Add the code below to theme() in the theme_custom() function
      panel.grid = element_line(color = "#C8C8C8",
                                size = .5,
                                linetype = "dashed"),
```

Custom Themes

Plot

- Including control flows into your custom theme function is useful for easily toggling on or off specific components of your theme
- Perhaps you want to have a short-hand argument for removing the legend title
- In the theme shown on the right side, you can set `show_legend_title` to `FALSE` to remove the legend title

Code

```
theme_custom <- function(base_family = "Helvetica",
                         base_size = 12,
                         padding = base_size*.75,
                         title_size = base_size*1.25,
                         show_legend_title = TRUE) {
  out <- theme_bw() %+replace%
    theme() # Include the previous theme arguments here

  if(!show_legend_title) {
    out <- out %+replace%
```

Exporting Plots

ggsave()

```
?ggplot2::ggsave
```

- By default, `ggsave()` saves your last plot to the path given by `filename` and `path`
  - Specify which plot in the environment to save by setting `plot` accordingly
  - Instead of giving both, `filename` and `path`, the directory and file name can be specified via `filename` alone
- The `device` to use, can be set implicitly by `filename` or explicitly by setting `device` to e. g. "png" or "jpeg".
- Set `units` to "px" and change `width` and `height` according to the desired resolution

```
save_plot <- grouped_density_plot +
  theme_custom(base_size = 6)

save_path <- "output/plots/density_plot.png"

ggsave(filename = save_path,
       plot = save_plot,
       units = "px",
       # Full HD: 1920 x 1080 pixels
       width = 1920,
```

References

Wickham, H. (2010). "A Layered Grammar of Graphics". In: *Journal of Computational and Graphical Statistics* 19.1, pp. 3-28. DOI: 10.1198/jcgs.2009.07098. eprint: https://doi.org/10.1198/jcgs.2009.07098. URL: https://doi.org/10.1198/jcgs.2009.07098.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

Wickham, H. and G. Grolemund (2016). *R for data science. import, tidy, transform, visualize, and model data*. O'Reilly. URL: https://r4ds.had.co.nz/.

Wilkinson, L. (2005). *The Grammar of Graphics*. 2nd ed. Springer New York, NY. ISBN: 978-0-387-28695-2. DOI: https://doi.org/10.1007/0-387-28695-0.