Data Analysis Coursebook

Marcell Granat & Zoltan Madari

Contents

Welcome

This is the online version of *Mastering Shiny*, a book **currently under early development** and intended for a late 2020 release by O'Reilly Media.

Shiny is a framework for creating web applications using R code. It is designed primarily with data scientists in mind, and to that end, you can create pretty complicated Shiny apps with no knowledge of HTML, CSS, or JavaScript. On the other hand, Shiny doesn't limit you to creating trivial or prefabricated apps: its user interface components can be easily customized or extended, and its server uses reactive programming to let you create any type of back end logic you want. Shiny is designed to feel almost magically easy when you're getting started, and yet the deeper you get into how it works, the more you realize it's built out of general building blocks that have strong software engineering principles behind them.

Today, Shiny is used in almost as many niches and industries as R itself is. It's used in academia as a teaching tool for statistical concepts, a way to get undergrads excited about learning to write code, a splashy medium for showing off novel statistical methods or models. It's used by big pharma companies to speed collaboration between scientists and analysts during drug development. It's used by Silicon Valley tech companies to set up realtime metrics dashboards that incorporate advanced analytics.

This book is designed to take you from knowing nothing about Shiny to being an expert developer who can write large complex apps that are still maintainable and performant. You'll gain a deep understanding of the reactive programming model that underlies Shiny, as well as building a tool box of useful techniques to solve common app challenges.

License

Syllabus

Table 1: Outline of Topics

Week	Lecture Topic	Seminar Topic	Homework/Exam
1		Introduction to R	
2		Functional programming, RMarkdown	
3		Tibble, Dplyr	
4		Ggplot2	
5		Statistical analysis, purrr	
6		Statistical analysis, broom	
7			

Week 1

Introduction

The following chapters give you a grab bag of useful techniques. I think everyone should start with Chapter **??**, because it gives you important tools for developing and debugging apps, and getting help when you're

stuck.

After that, there's no prescribed order and relatively few connections between the chapters: I'd suggest quickly skimming to get the lay of the land (and so you might remember these tools if related problems crop up in the future), and otherwise only deeply reading the bits that you currently need. Here's a quick run down of the main topics:

Let's begin by working on your workflow for developing apps.

1 Lecture 1

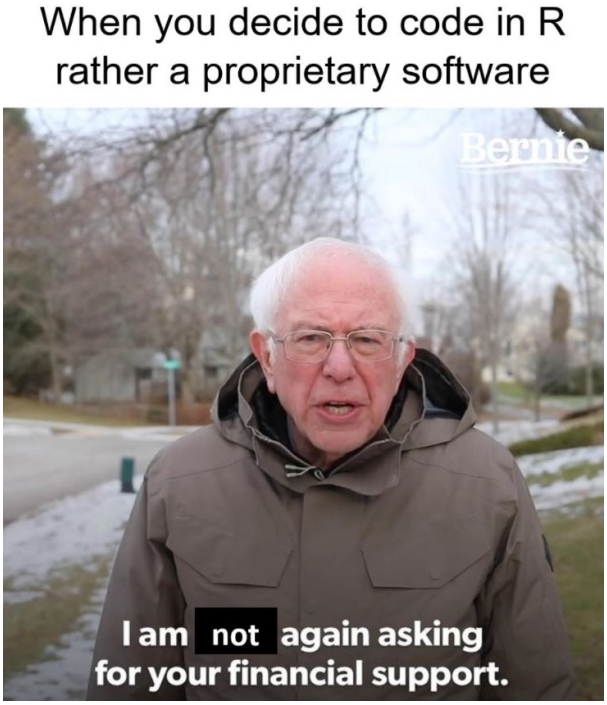
You can often make your app more usable by giving the user more insight into what is happening. This might take the form of better messages when inputs don't make sense, or progress bars for operations that take a long time. Some feedback occurs naturally through outputs, which you already know how to use, but you'll often need something else. The goal of this chapter is to show you some of your other options.

We'll start with techniques for **validation**, informing the user when an input (or combination of inputs) is in an invalid state. We'll then continue on to **notification**, sending general messages to the user, and **progress bars**, which give details for time consuming operations made up of many small steps. We'll finish up by discussing dangerous actions, and how you give your users peace of mind with **confirmation** dialogs or the ability to **undo** an action.

2 Introduction to R

2.1 Why R?

In this chapter we will discuss the basics of R programming. R is a free software, used by millions in the field of statistics, data science, economics and many others.

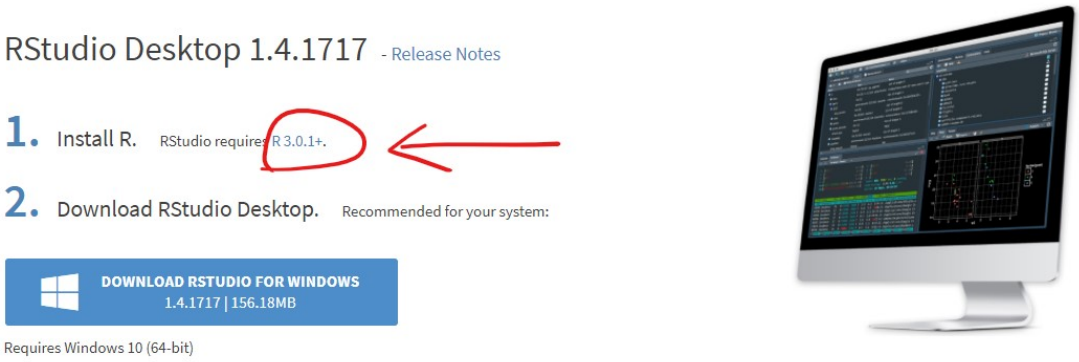


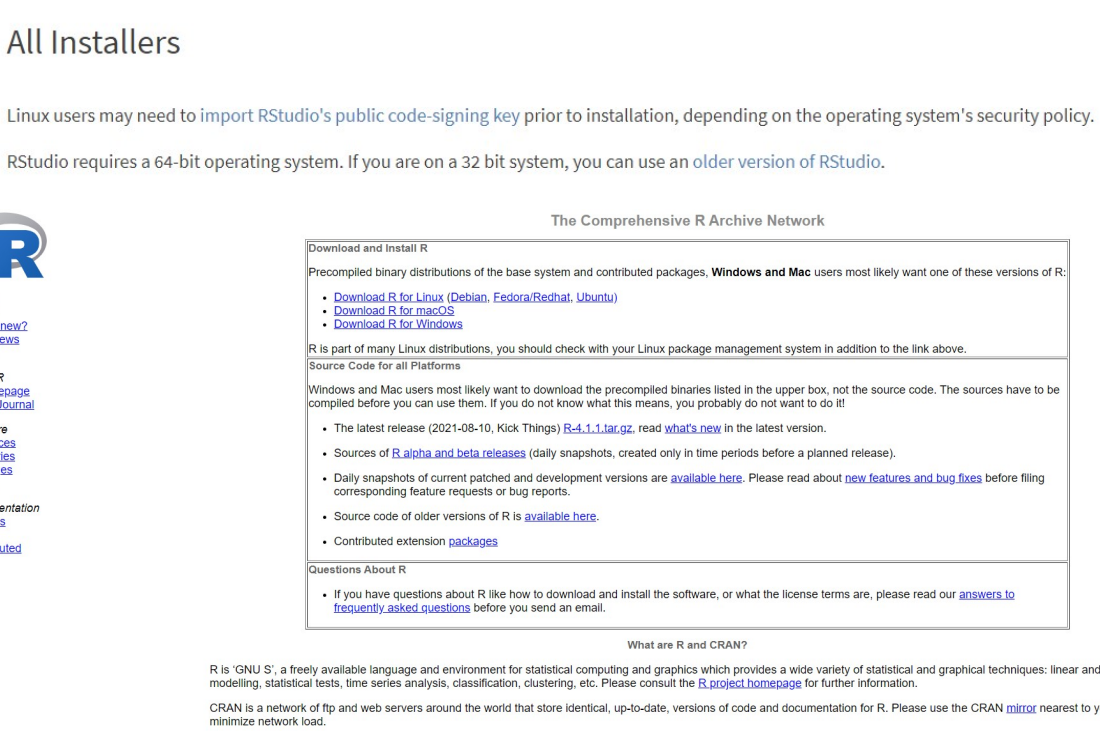
The R programming language is an important tool for data related tasks, but it is much more. Just like other programming languages, R has many additional packages, which can extend its basic functionality. R has a great (probably the best) graphical tools to create your charts, and with shiny, you can easily build your

minimalist web applications. We will learn about data manipulation, analysis and how to create awesome reports, like dashboards.

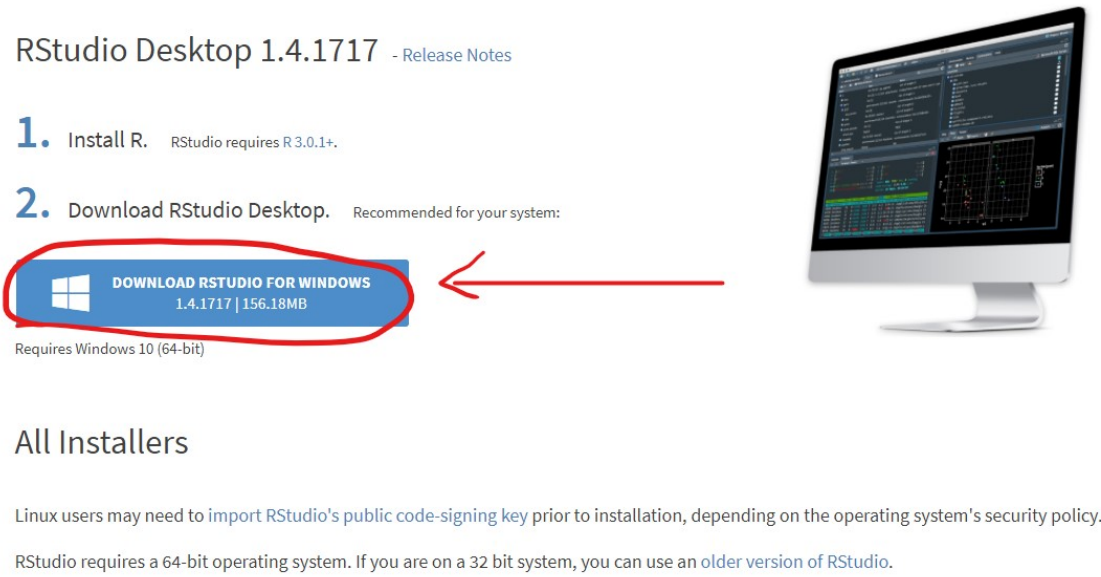
2.2 Setup

You can download R and RStudio from the official site of RStudio. Please install the appropriate version based on your OS, and do not forget that you also have to install R as well.





Run R's installer file after the downloading process is finished. Next, we will also need the RStudio.



If the installation process of R and RStudio is finished, then we can open RStudio and start to learn the software.

2.3 Bootstrap

To continue your app customisation journey, you'll need to learn a little more about the Bootstrap framework used by Shiny. Bootstrap is a collection of HTML conventions, CSS styles, and JS snippets bundled up into a convenient form. Bootstrap grew out of a framework originally developed for Twitter and over the last 10 years has grown to become one of the most popular CSS frameworks used on the web. Bootstrap is also popular in R — you've undoubtedly seen many documents produced by `rmarkdown::html_document()` and used many package websites made by pkgdown, both of which also use Bootstrap.

As a Shiny developer, you don't need to think too much about Bootstrap, because Shiny functions automatically generate bootstrap compatible HTML for you. But it's good to know that Bootstrap exists because then:

- You can use `bslib::bs_theme()` to customise the visual appearance of your code, Section 2.4.
- You can use the `class` argument to customise some layouts, inputs, and outputs using Bootstrap class names, as you saw in Section **??**.
- You can make your own functions to generate Bootstrap components that Shiny doesn't provide, as explained in "Utility classes".

It's also possible to use a completely different CSS framework. A number of existing R packages make this easy by wrapping popular alternatives to Bootstrap:

- shiny.semantic, by Appsilon, builds on top of formantic UI.
- shinyMobile, by RInterface, builds on top of framework 7, and is specifically designed for mobile apps.
- shinymaterial, by Eric Anderson, is built on top of Google's Material design framework.
- shinydashboard, also by RStudio, provides a layout system designed to create dashboards.

You can find a fuller, and actively maintained, list at https://github.com/nanxstats/awesome-shiny-extensions.

2.4 Themes

Bootstrap is so ubiquitous within the R community that it's easy to get style fatigue: after a while every Shiny app and Rmd start to look the same. The solution is theming with the bslib package. bslib is relatively new package that allows you to override many Bootstrap defaults in order to create an appearance that is uniquely yours. As I write this, bslib is mostly applicable only to Shiny, but work is afoot to bring its enhanced theming power to RMarkdown, pkgdown, and more.

If you're producing apps for your company, I highly recommend investing a little time in theming — theming your app to match your corporate style guide is an easy way to make yourself look good.

2.4.1 Getting started

Create a theme with `bslib::bs_theme()` then apply it to an app with the `theme` argument of the page layout function:

```
fluidPage(
  theme = bslib::bs_theme(...)
)
```

If not specified, Shiny will use the classic Bootstrap v3 theme that it has used basically since it was created. By default, `bslib::bs_theme()`, will use Bootstrap v4. Using Bootstrap v4 instead of v3 will not cause problems if you only use built-in components. There is a possibility that it might cause problems if you've used custom HTML, so you can force it to stay with v3 with `version = 3`.

2.4.2 Shiny themes

The easiest way to change the overall look of your app is to pick a premade "bootswatch" theme using the `bootswatch` argument to `bslib::bs_theme()`. Figure **??** shows the results of the following code, switching `"darkly"` out for other themes.

Week 2

Introduction

The following chapters give you a grab bag of useful techniques. I think everyone should start with Chapter **??**, because it gives you important tools for developing and debugging apps, and getting help when you're stuck.

After that, there's no prescribed order and relatively few connections between the chapters: I'd suggest quickly skimming to get the lay of the land (and so you might remember these tools if related problems crop up in the future), and otherwise only deeply reading the bits that you currently need. Here's a quick run down of the main topics:

Let's begin by working on your workflow for developing apps.

3 Lecture 2

You can often make your app more usable by giving the user more insight into what is happening. This might take the form of better messages when inputs don't make sense, or progress bars for operations that take a long time. Some feedback occurs naturally through outputs, which you already know how to use, but you'll often need something else. The goal of this chapter is to show you some of your other options.

We'll start with techniques for **validation**, informing the user when an input (or combination of inputs) is in an invalid state. We'll then continue on to **notification**, sending general messages to the user, and

progress bars, which give details for time consuming operations made up of many small steps. We'll finish up by discussing dangerous actions, and how you give your users peace of mind with **confirmation** dialogs or the ability to **undo** an action.

4 Seminar 2

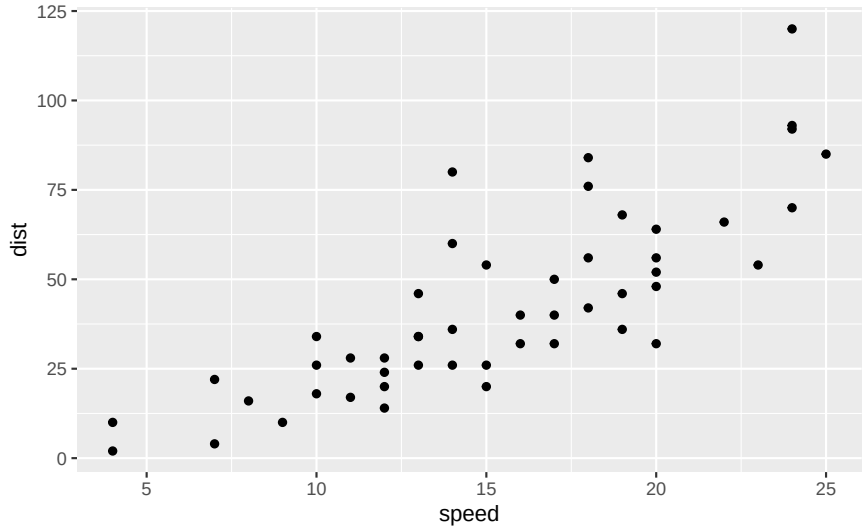
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

4.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to filter() which uses a data-variable (carat) and an env-variable (min):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



Week 3

Introduction

The following chapters give you a grab bag of useful techniques. I think everyone should start with Chapter **??**, because it gives you important tools for developing and debugging apps, and getting help when you're stuck.

After that, there's no prescribed order and relatively few connections between the chapters: I'd suggest quickly skimming to get the lay of the land (and so you might remember these tools if related problems crop up in the future), and otherwise only deeply reading the bits that you currently need. Here's a quick run down of the main topics:

Let's begin by working on your workflow for developing apps.

5 Lecture 3

You can often make your app more usable by giving the user more insight into what is happening. This might take the form of better messages when inputs don't make sense, or progress bars for operations that take a long time. Some feedback occurs naturally through outputs, which you already know how to use, but you'll often need something else. The goal of this chapter is to show you some of your other options.

We'll start with techniques for **validation**, informing the user when an input (or combination of inputs) is in an invalid state. We'll then continue on to **notification**, sending general messages to the user, and **progress bars**, which give details for time consuming operations made up of many small steps. We'll finish up by discussing dangerous actions, and how you give your users peace of mind with **confirmation** dialogs or the ability to **undo** an action.

6 Seminar 3

Transferring files to and from the user is a common feature of apps. You can use it to upload data for analysis, or download the results as a dataset or as a report. This chapter shows the UI and server components that you'll need to transfer files in and out of your app.

6.1 Upload

We'll start by discussing file uploads, showing you the basic UI and server components, and then showing how they fit together in a simple app.

6.1.1 UI

Week 4

Introduction

The following chapters give you a grab bag of useful techniques. I think everyone should start with Chapter **??**, because it gives you important tools for developing and debugging apps, and getting help when you're stuck.

After that, there's no prescribed order and relatively few connections between the chapters: I'd suggest quickly skimming to get the lay of the land (and so you might remember these tools if related problems crop up in the future), and otherwise only deeply reading the bits that you currently need. Here's a quick run down of the main topics:

Let's begin by working on your workflow for developing apps.

7 Lecture 4

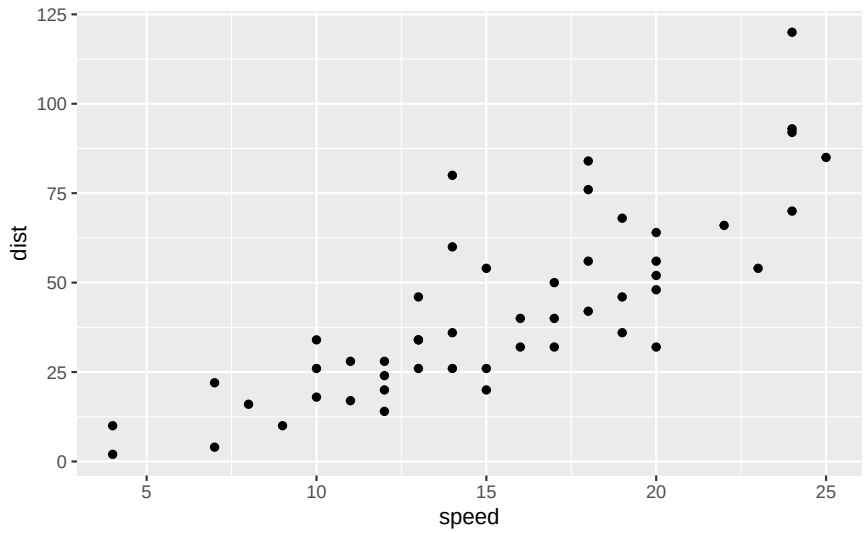
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

7.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to filter() which uses a data-variable (carat) and an env-variable (min):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



8 Ggplot

8.1 Motiváció a ggplot package használatához

Bár maga a base R [értve itt ilyenkor mindig a plot paranccsal készült ábrákat] segítségével is lehet ábrákat készíteni, ez az egyszerűségét leszámítva nagyon ritkán ajánlott. Helyette érdemes a ggplot2-t alkalmazni, amely bár komplexebb, mint a base R, a személyre szabhatóságának szinte nincs felső határa.

Ám amelett, hogy szebb ábráink lesznek van egy sokkal fontosabb ok, amiért a ggplotot használjuk. A csomag a nevét Leland Wilkinson **Grammar of Graphics** c. munkája után kapta (innen a *gg*), amelyben megfogalmazásra került lényegében az ideális ábrakészítő algoritmus struktúrája. Mivel a ggplot2 ezt az elvet követi, így meglátjátok majd, hogy bár sok dolgot kell fejben tartani, minden logikus benne. Miután sikerül elsajátítani ezt az anyagot (és némi utána olvasás után) bármilyen szükséges ábrát el fogtok tudni készíteni.

8.2 Bevezetés

A ggplot2 telepíthető önállóan is, illetve ez a csomag része egy "csomaggyűjteménynek", amit tidyverse-nek hívnak. Jelenleg mi csak a ggplot2-vel fogunk foglalkozni, így annak telepítése és betöltése is bőven elegendő lesz. Így annak függvényébne, hogy le van-e már töltve a package telepítsük [install.packages("ggplot2")], vagy ne, majd töltsük be [library(ggplot2)]. A ggplot2 helyett lehet egyből a tidyverse-t is betölteni, azonban jelenleg a többi package redundáns nekünk.

```
library(ggplot2)
```

A ggplot2-vel kapunk néhány kiegészítő adattáblát, az egyszerűség kedvéért használjuk most ezek közül a diamonds-t. Pillanatra tekintsük meg az adattábla felépítését:

```
head(diamonds)
```

```
#> Error in tab_header(., "A diamonds adattábla"): could not find function "tab_header"
```

Lássuk az első ábránkat ggplottal!

```
ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point()
```

Amit látunk, hogy az ábrázolás a ggplot paranccsal kezdődik. Itt definiálni kell, hogy melyik táblának adatait

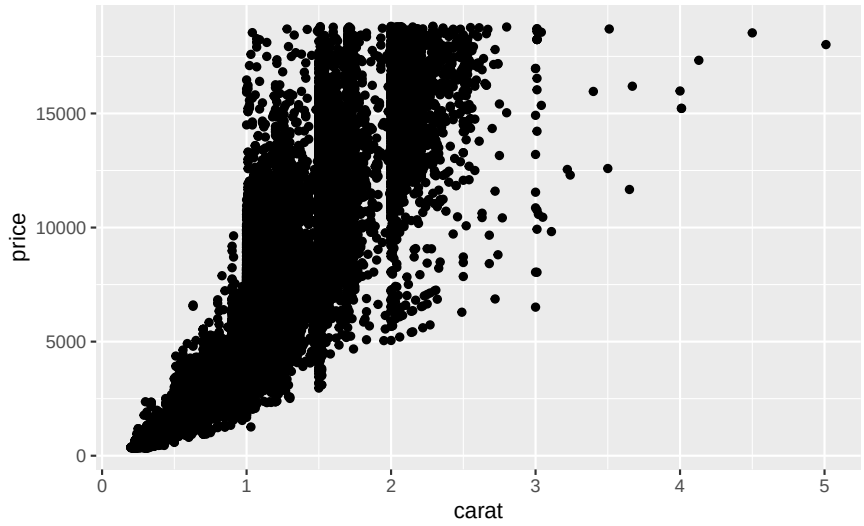


Figure 1: A legelső ábránk ggplot használatával

szeretnénk felhasználni (diamonds), majd pedig az ábrázoláshoz használt "keretet", ami az aes-es belül kap helyet. Itt általában (mint például most) az x és y tengelyen ábrázolt adatokat kell megadni. A ggplot-ot mindig egy geom parancs követi, amellyel az ábra típusát adjuk meg.

Ezen az ábrán azonban még nicsen semmilyen felirat. Minden felirat megadható a labs() függvényben. Példa:

```
ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
 geom_point() +
 labs(
   title = "Figyelemfelkeltő cím",
   subtitle = "Az alcím pontosítja a témát",
   x = "x-tengely változójának neve",
   y = "y-tengely változójának neve",
   caption = "Mindent megmagyarázó hosszú leírás."
 )
```

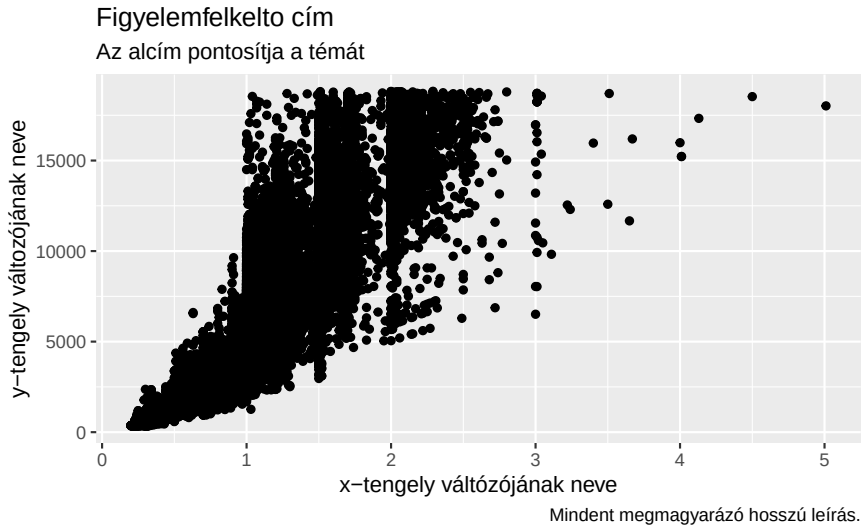


Figure 2: Feliratok elhelyezése az ábrán

8.3 Esquisse

Első látásra ez még lehet túl sok információ, illetve a következőkben a bonyolultabb ábrákra rátérve elveszítheti az ember a fonalat. Amíg bizonytalan a ggplot használata javaslom a esquisse packaget Ezt csupán telepíteni kell a install.packages("esquisse") paranccsal, majd az Addins menüből lehet előhívni. Ennek segítségével "manuálisan" adhatóak meg az ábra elemei. Lássuk például az előző ábrát!

8.4 Szín

Az esquisse automatikusan hozzáadott 3 elemet, amit akkor itt tisztázni szükséges. A geom függvényen belül a size paraméter a pontok méretért felel, míg a color a pontok színéért. Színként megadható bármilyen hexadecimális érték, illetve az R számos színt ismer (red, green, blue, black, stb.).

```
#> Error in include_graphics("images/color_examples.png"): could not find function "include_graphics"
```

8.5 Téma

A 3. dolog amit esquisse csatolt a kódunk mellé az egy theme parancs. Ez a parancstípus az ábra hátterének, jelmagyarázatának és feliratainak esztétikáját adja meg. Lássunk ezekből is néhányat!

```
ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point() +
  theme_bw()

ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point() +
  theme_classic()

ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point() +
  theme_grey()

ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point() +
  theme_minimal()
```

```
#> Error: Can't add `(p + theme_classic() + ggtitle("theme_classic"))` to a ggplot object.
```

Persze van még rengeteg, válogass nyugodtan: https://www.datanovia.com/en/blog/ggplot-themes-gallery/

Ami még fontos, hogy a theme() függvénnyel módosíthatóak a témánk egyes elemei (háttérszín, betűszín stb.). Például legyen tényleg figyelemfelkeltő a fentebb bemutatott cím!

```
ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point() +
  labs(
    title = "Figyelemfelkeltő cím",
    subtitle = "Az alcím pontosítja a témát",
    x = "x-tengely változójának neve",
    y = "y-tengely változójának neve",
    caption = "Mindent megmagyarázó hosszú leírás."
  ) +
  theme(
    plot.title = element_text(color = "red", size = 30)
  )
```

8.6 Geom

A plot szempontjából lényeges fontosabb azonban a geom függvény. Maga a függvény adja meg az ábra típusát, azonban a benne foglalt paraméterek sem elhanyagolhatóak. Nézzük példát a típusokra (csak párat

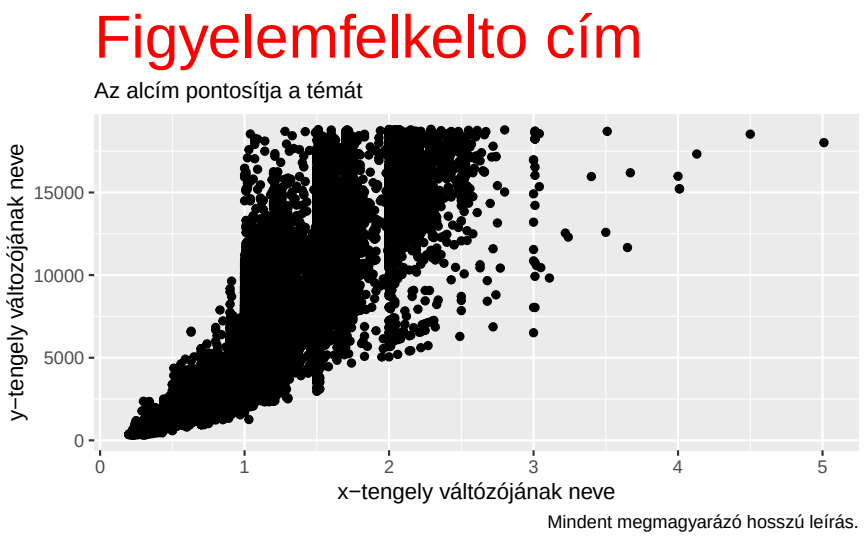


Figure 3: Téma egy elmének megváltoztatására példa

van módom említeni a végtelenül sokból)!

```
ggplot(data = diamonds, mapping = aes(x = carat)) +
  geom_histogram()

ggplot(data = diamonds, mapping = aes(x = carat)) +
  geom_histogram() +
  geom_vline(xintercept = mean(diamonds$carat), color = 'red4')

ggplot(data = diamonds, mapping = aes(x = cut)) +
  geom_bar()

ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
  geom_point()
```

```
#> Error: Can't add `ggplot(data = diamonds, mapping = aes(x = carat))` to a ggplot object.
```

A 2. kódból látszik, hogy több geom is tartozhat egyszerre egy ábrához, így lehet például egy hisztogramon feltüntetni az átlagot egy függőleges vonallal (geom_vline). Ennek párja is rendkívül hasznos tud lennni a geom_hline, amivel lehet akár 0 mentén húzni egy vízszintes vonalat. A cut egy kategorikus változó az adattáblánkban, amelynek az egyes kategóriákhoz tartozó darabszámait egy geom_bar-al gyorsan kirajzolhatjuk. Most nézzük a paramétereket! Eddig láttunk példát a size-ra és a color-ra, ami majdnem minden geom esetén csinál valamit, de egyes geomoknak vannak speciális paramétereik is, mint pélául az előbbi példában az xintercept. Lássunk néhány további fontos paraméterre példákat:

```
#> Error in cols_align(., align = "center"): could not find function "cols_align"
```

```
ggplot(data = diamonds, mapping = aes(x = carat)) +
  geom_histogram(color = 'black', fill = "red")

ggplot(data = diamonds, mapping = aes(x = carat)) +
  geom_vline(xintercept = mean(diamonds$carat), lty = 2)

ggplot(data = diamonds, mapping = aes(x = cut)) +
  geom_bar(alpha = .3)

ggplot(data = diamonds, mapping = aes(x = carat, y = price)) +
```

```
  geom_point(shape = 21, color = 'black', fill = 'green')
```

```
#> Error: Can't add `ggplot(data = diamonds, mapping = aes(x = carat))` to a ggplot object.
```

Az utolsó ábra itt még igényel némi magyarázatot. A pont diagram alapbeállítása szerint csak color paraméter adható meg. Azonban ha a shape-t átállítjuk, akkor van lehetőségünk olyan pontdiagramokat rajzolni, amelyeknek belsejéhez tartozó színt adja meg a fill, a szegélyét a color.

8.7 Aes

Korábban csak átsiklottunk az aes() függvényen, ami az ábra "keretét" adta. Most nézzük meg kicsit részletesebben! A mapping paraméter, aminek megadásához használjuk fel az aes()-t az előzőekben a ggplot() függvényen belül szerepelt. Ennek azonban nem kell feltétlenül így lennie. A következő 3 kód, mind ugyanazt hajtja végre (erre még visszatérünk):

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point()

ggplot(diamonds) +
  aes(x = carat, y = price) +
  geom_point()

ggplot(diamonds) +
  geom_point(aes(x = carat, y = price))
```

Az aes-ben azonban nem csak x- és y-tengely változóit definiálhatjuk. Minden ami geom-beni paraméter, az megadásra kerülhet az aes()-ben is, azonban ilyenkor az adott változó értéke szerint kerül megadásra (pl.: egy harmadik változó szerint van definiálva a méret, vagy a szín). Ami paraméter az aes()-ben kerül megadásra, az automatikusan a jelmagyarázatra is felkerül. Lássunk két példát!

```
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point()

ggplot(diamonds, aes(x = carat, y = price, size = cut)) +
  geom_point(alpha = .3)
```

```
#> Error in (ggplot(diamonds, aes(x = carat, y = price, color = cut)) + geom_point())/(ggplot(diamonds, : non-numeric argument to binary operator
```

Ha egy változó szerint adjuk meg a színeket, akkor érdemes a színezésnek módját is megadni. Erre leginkább a viridis függvények a javasoltak (számos alkalommla láthatod tudományos publikációkban + *végződésre figyeljünk: viridis_d, mint diszkért/viridis_c, mint folytonos*), de ha például fontos, hogy fekete-fehér legyen, arra is van mód, illetve "kézileg" is meg lehet adni színeket.

```
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point()

ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point() +
  scale_color_viridis_d(option = "magma")

ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point() +
  scale_color_manual(
    values = c("red", "green", "blue", "yellow", "orange")
    )
```

```
#> Error in (ggplot(diamonds, aes(x = carat, y = price, color = cut)) + geom_point())/(ggplot(diamonds, : non-numeric argument to binary operator
```

És most vissza oda, hogy miért is hasznos, hogy több aes() definiálható egyetlen ábrán. Elképzelhető, hogy két külön geom-ot használunk együtt. Legyen mondjuk egy pontdiagram és egy azokra illesztett "trendvonal".

Utóbbi a geom_smooth paranccsal érhető el, s amennyiben lineáris függvény formára vágyunk (yes), úgy adjuk meg neki a method = "lm" paramétert.

```
ggplot(diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point() +
  geom_smooth(method = "lm")
#> `geom_smooth()` using formula 'y ~ x'
```

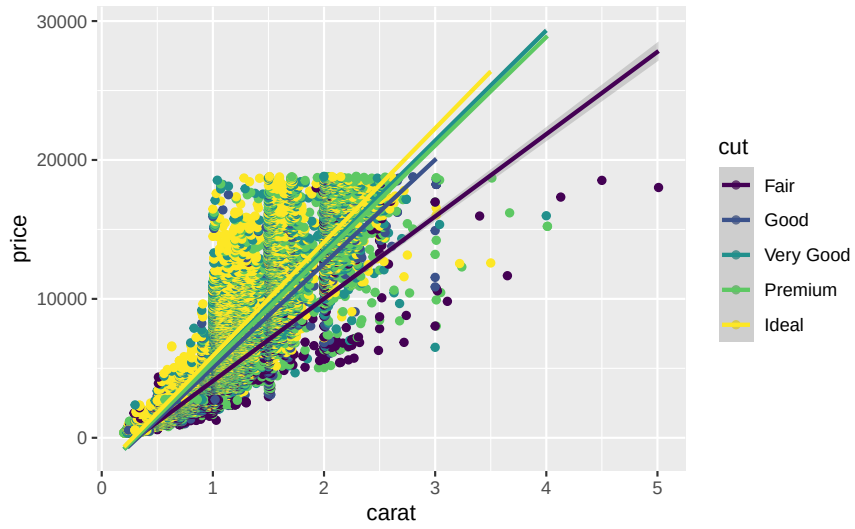


Figure 4: Lineáris trend illesztése pontdiagramra

Mint látjuk jelen esetben a geom_smooth ugyanazt az aes()-t alkalmazza, mint a geom_point. Ez azonban problémás, ha nem akarunk a kategorikus ismérv mentén 5 külön trendvonalat illeszteni. Ilyenkor jelent megoldás egy új aes() megadása egyetlen ábrán belül.

```
ggplot(diamonds) +
  geom_point(aes(x = carat, y = price, color = cut)) +
  geom_smooth(aes(x = carat, y = price), method = "lm")
#> `geom_smooth()` using formula 'y ~ x'
```

8.8 Facet

A facet az utolsó parancs amit érinteni kívánunk. Ennek segítségével lehetőség van egy ábra felosztására valamely változó mentén. Például a cut változó szerint csoportosítva megtekintve a pontdiagramok:

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point() +
  facet_wrap(~ cut)
```

Ha pedig soronként és oszloponként is akarunk csoportosítani:

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point() +
  facet_grid(color ~ cut)
```

8.9 Kitekintés

A ggplot2 logikusan felépített működése annyira a felhasználók kedvencévé tette, hogy számos kiegészítő package épült rá: ggpubs, GGally, gganimate, stb. Ezek jelenleg nem részei a tananyagnak, azonban érdemes

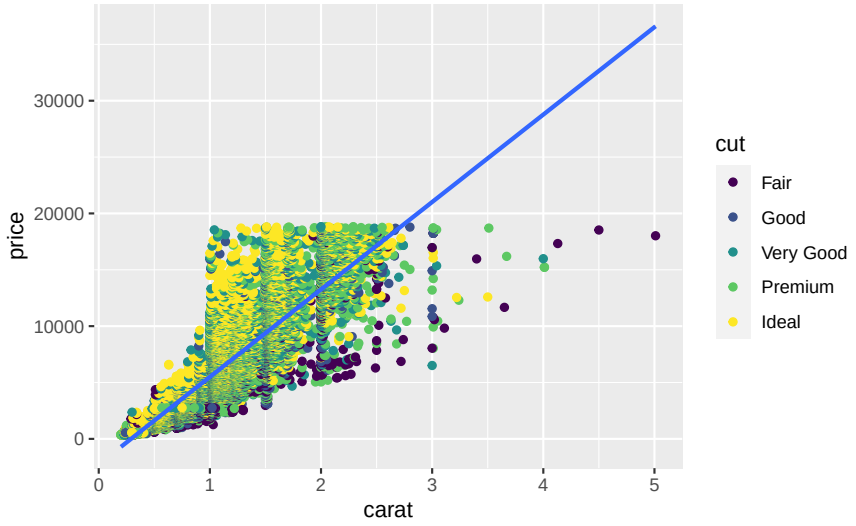


Figure 5: Lineáris trend illesztése pontdiagramra két külön esztétika függvénnyel

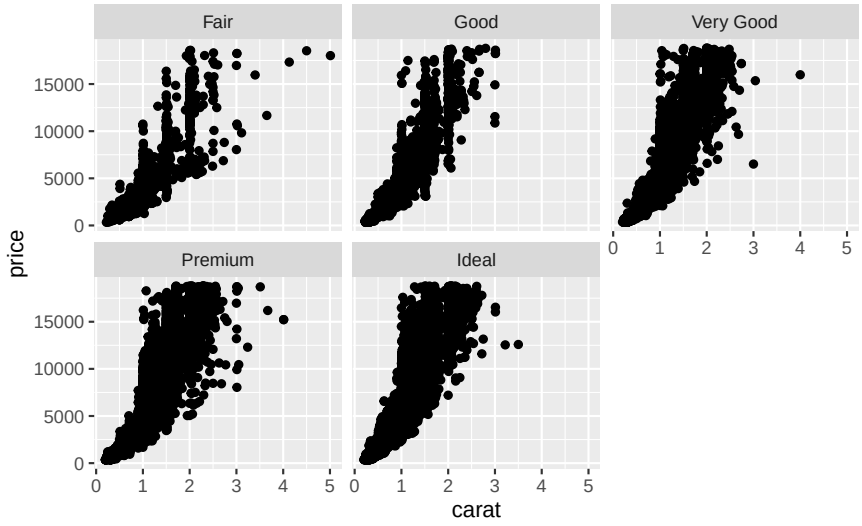


Figure 6: Facet alkalmazása egy változó szerint

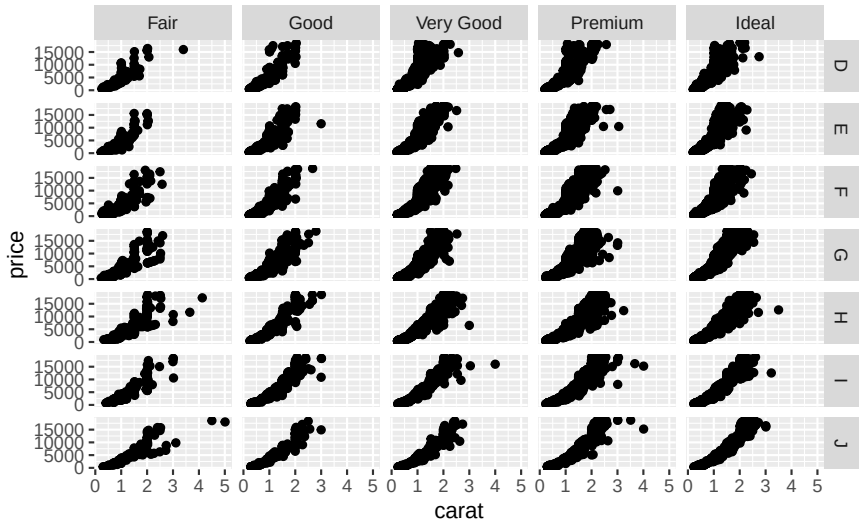


Figure 7: Facet alkalmazása két változó szerint

rájuk nézni, ha szakdolgozat vagy TDK prezentálása során szeretnéd lenyűgözni a közönséged. Kedvcsináló:

Week 5

Introduction

The following chapters give you a grab bag of useful techniques. I think everyone should start with Chapter **??**, because it gives you important tools for developing and debugging apps, and getting help when you're stuck.

After that, there's no prescribed order and relatively few connections between the chapters: I'd suggest quickly skimming to get the lay of the land (and so you might remember these tools if related problems crop up in the future), and otherwise only deeply reading the bits that you currently need. Here's a quick run down of the main topics:

Let's begin by working on your workflow for developing apps.

9 Lecture 5

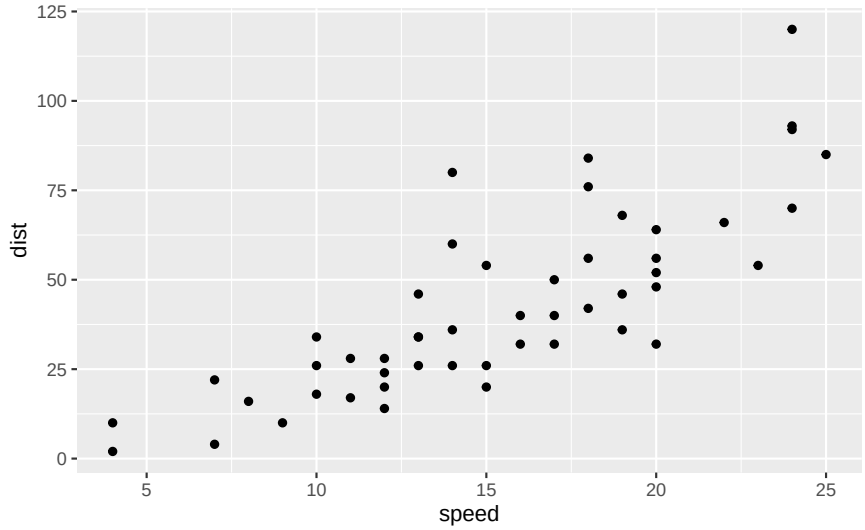
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

9.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to filter() which uses a data-variable (carat) and an env-variable (min):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



10 Seminar 5

Week 6

Introduction

The following chapters give you a grab bag of useful techniques. I think everyone should start with Chapter **??**, because it gives you important tools for developing and debugging apps, and getting help when you're stuck.

After that, there's no prescribed order and relatively few connections between the chapters: I'd suggest quickly skimming to get the lay of the land (and so you might remember these tools if related problems crop up in the future), and otherwise only deeply reading the bits that you currently need. Here's a quick run down of the main topics:

Let's begin by working on your workflow for developing apps.

11 Lecture 6

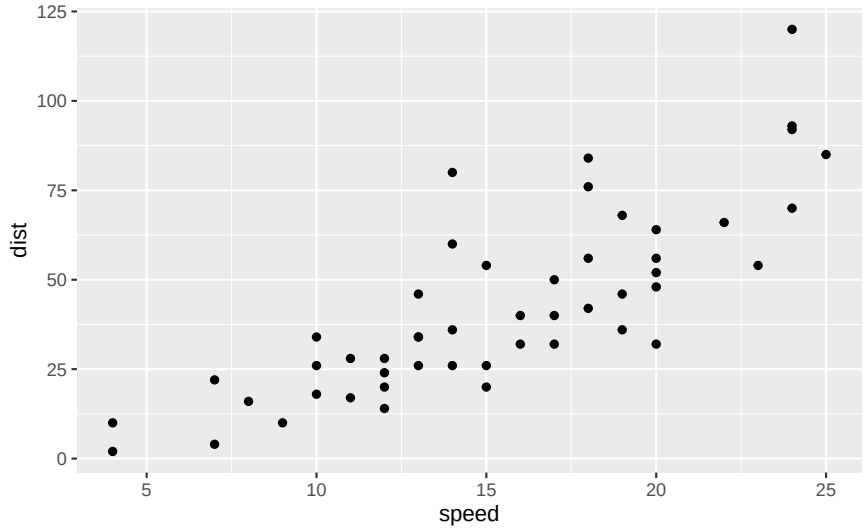
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

11.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to `filter()` which uses a data-variable (`carat`) and an env-variable (`min`):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



12 Seminar 6

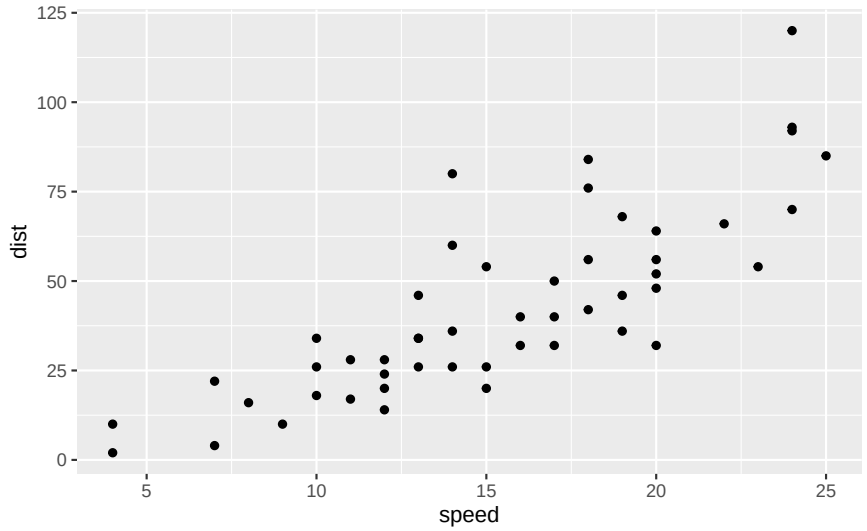
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

12.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to filter() which uses a data-variable (carat) and an env-variable (min):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



Week 7

13 Lecture 7

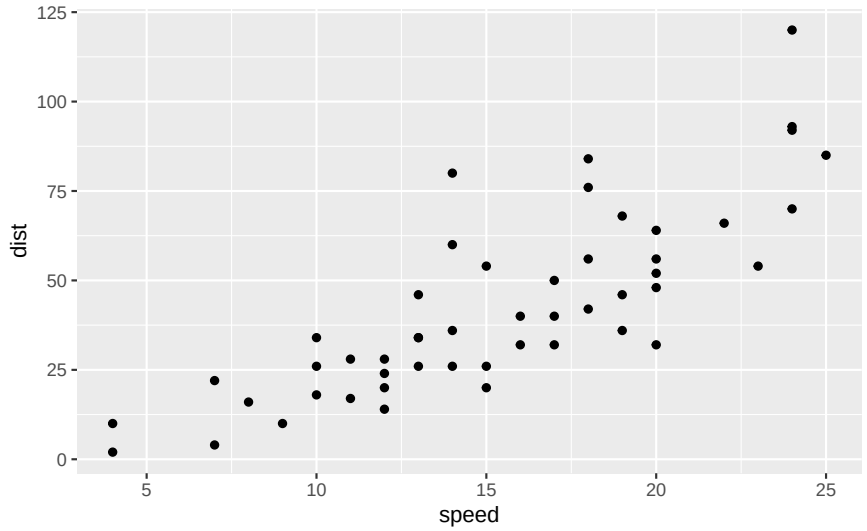
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

13.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to filter() which uses a data-variable (carat) and an env-variable (min):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```



14 Seminar 7

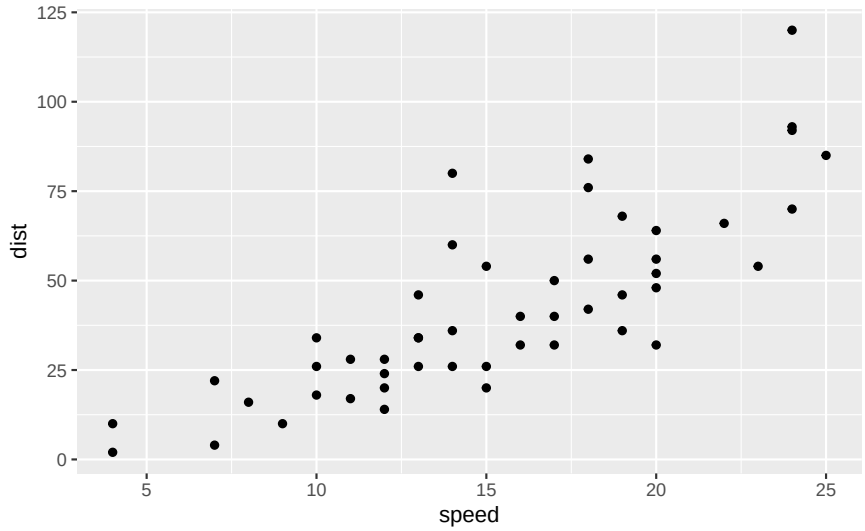
In this chapter, you'll learn how to wrap ggplot2 and dplyr function in a Shiny app. (If you don't use the tidyverse, you can skip this chapter .) The techniques for wrapping ggplot2 and dplyr functions in a functions and package, are a a little different and covered in other resources like *Using ggplot2 in packages* or *Programming with dplyr*.

14.1 Motivation

Imagine I want to create an app that allows you to filter a numeric variable to select rows that are greater than a threshold. You might write something like this:

Let's begin with this call to filter() which uses a data-variable (carat) and an env-variable (min):

```
ggplot(cars, aes(speed, dist)) +
  geom_point()
```


125
100
75
50
25
0
dist
5
10
15
20
25
speed