Applied Time Series Econometrics

Oxana Malakhovskaya, NRU HSE

September 10, 2019

Working with time series

The algorighm of working with times series models.

1. Importing data that we will do with `readr` or `readxl` packages.
2. Data visualisation that we will do with 'ggplot2' package (if necessary).
3. Data manipulation that we will do with `dplyr` package treating series as data frames.
4. Transformation data frames into special time series (ts) format if necessary (some packages work only with ts data).
5. Estimation and evaluation of models with special packages developed specially for a certain class of models.

Data visualisation: a first step

- A first option to make plots is to use the `ggplot()` function from `ggplot2` package
- An example dataset is `LifeCycleSavings` from an automatically installed and attached `datasets` package.

```
head(LifeCycleSavings,3)
```

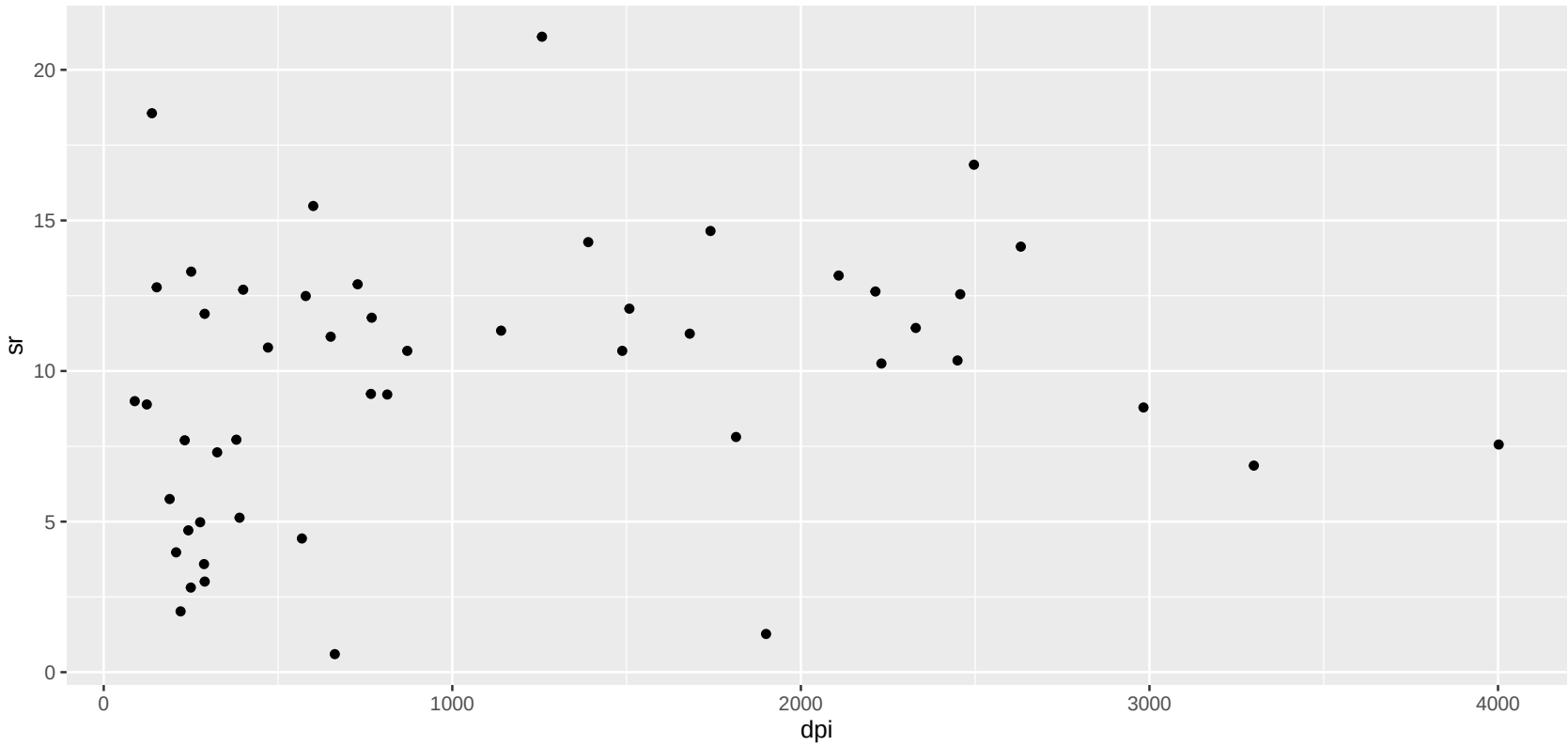
```
##                sr pop15 pop75     dpi ddpi
## Australia 11.43 29.35  2.87 2329.68 2.87
## Austria   12.07 23.32  4.41 1507.99 3.93
## Belgium   13.17 23.80  4.43 2108.47 3.82
```

- Let's try to answer a question if countries with a greater per capita income save less or more than others.

Data visualisation: a scatterplot

Among the variables:
dpi - real per-capita disposable income,
sr - aggregate personal saving divided by disposable income

```
ggplot(data = LifeCycleSavings) +
  geom_point(mapping = aes(x = dpi, y = sr))
```



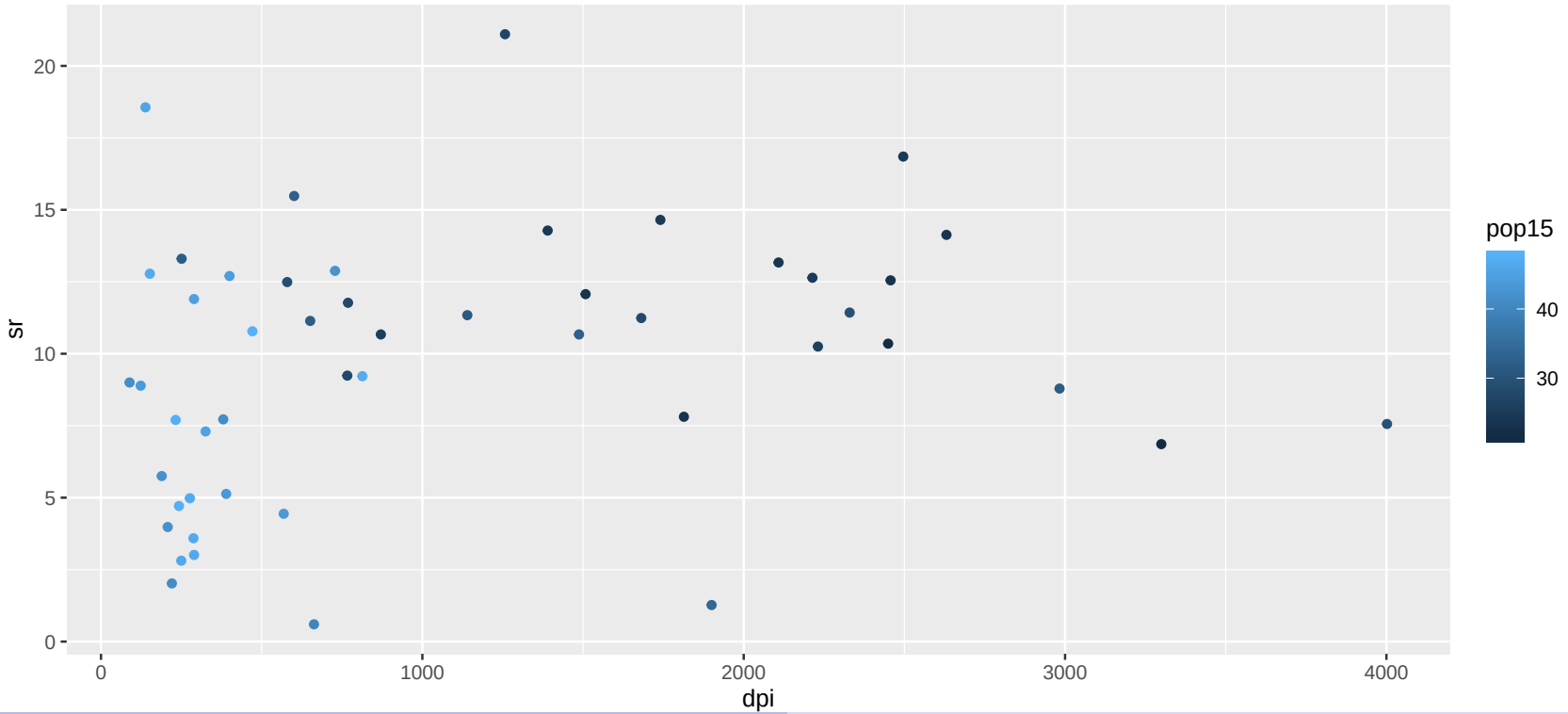
Data visualisation: functions and arguments

- ggplot() only creates a coordinate system that we can add layers to, its argument is the dataset.
- geom_point function creates a new layer of points. The package `ggplot2` contains many different `geom` functions that are responsible for different types of layers.
- each `geom` function takes a `mapping` argument that determines which variables are used for the plot.
- the `aes()` argument makes the list of these variables and determines which of them are mapped to the horizontal and vertical axes.

Data visualisation: selecting subsets with color

- To understand if there is a clear correlation between the per capita income and saving ratio in countries with high youth ratio we can add a third variable by mapping it to the 'aes()' function.

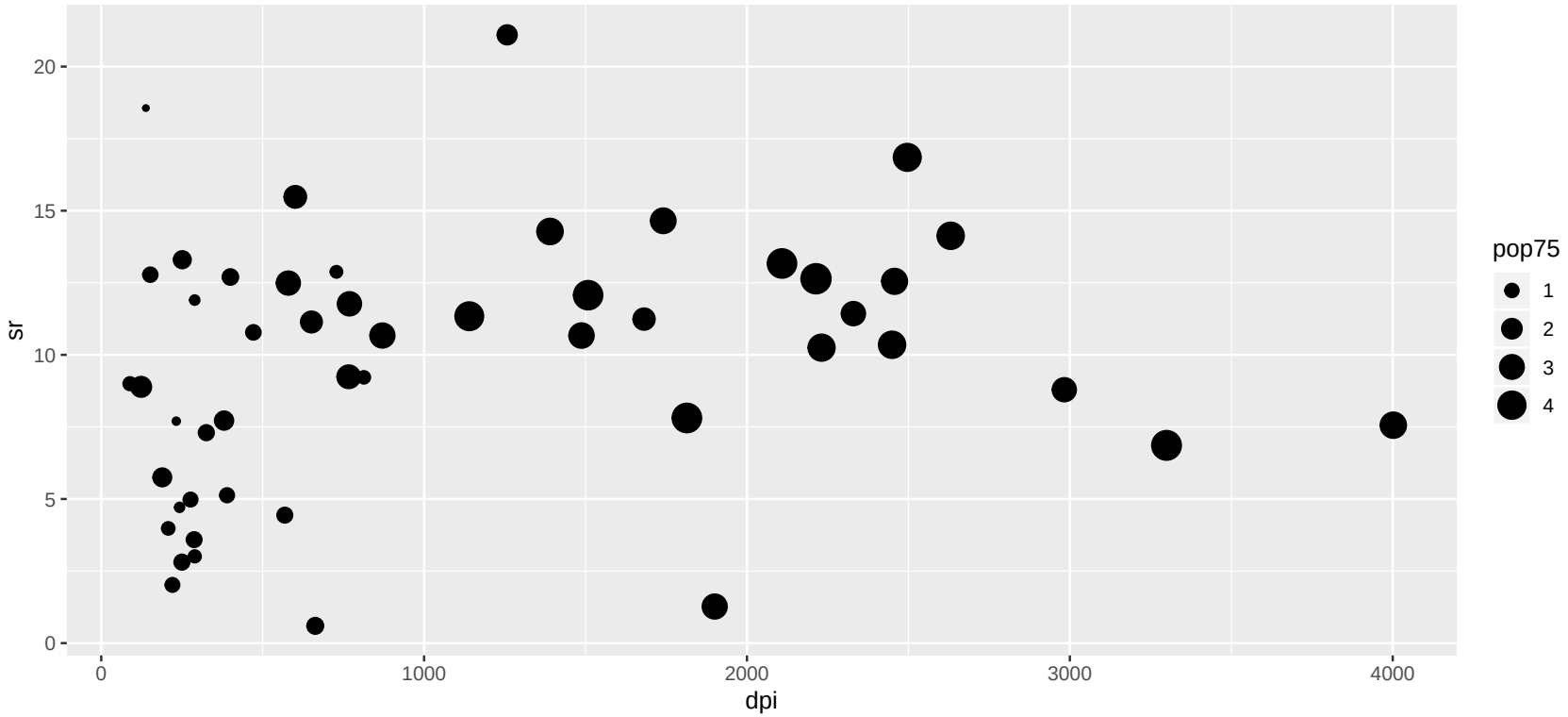
```
ggplot(data = LifeCycleSavings) +
geom_point(mapping = aes(x = dpi, y = sr, color = pop15))
```



Data visualisation: selecting subsets with size

- Or may be a ratio of elderly people makes a differernce?

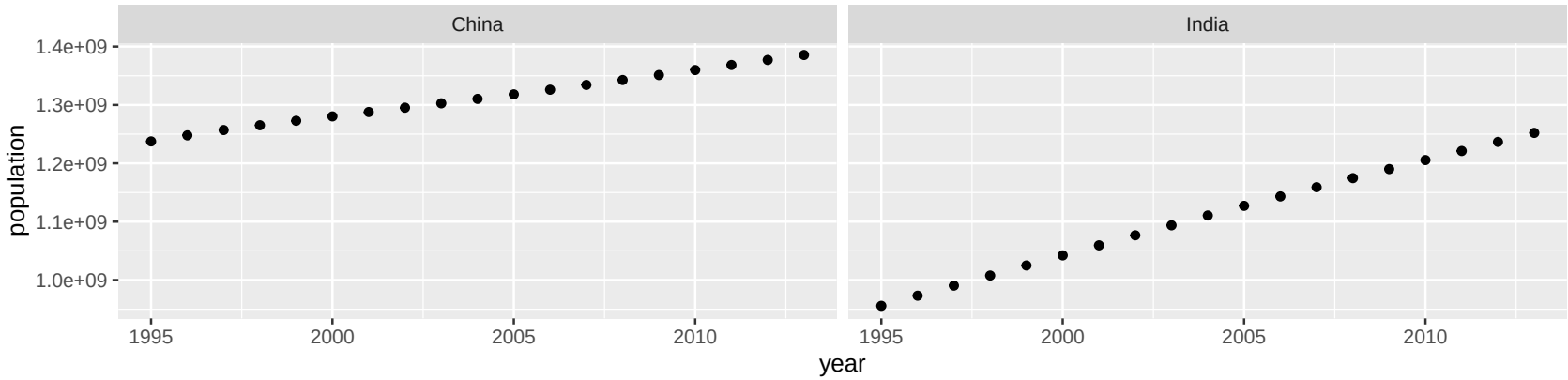
```
ggplot(data = LifeCycleSavings) +
  geom_point(mapping = aes(x = dpi, y = sr, size = pop75))
```



Data visualisation: facets

- Another way to add an additional variable to our plot is to split the dataset into different subsets with a `facet_wrap()` function. The argument of the function should be ~ sign followed by a discrete variable.

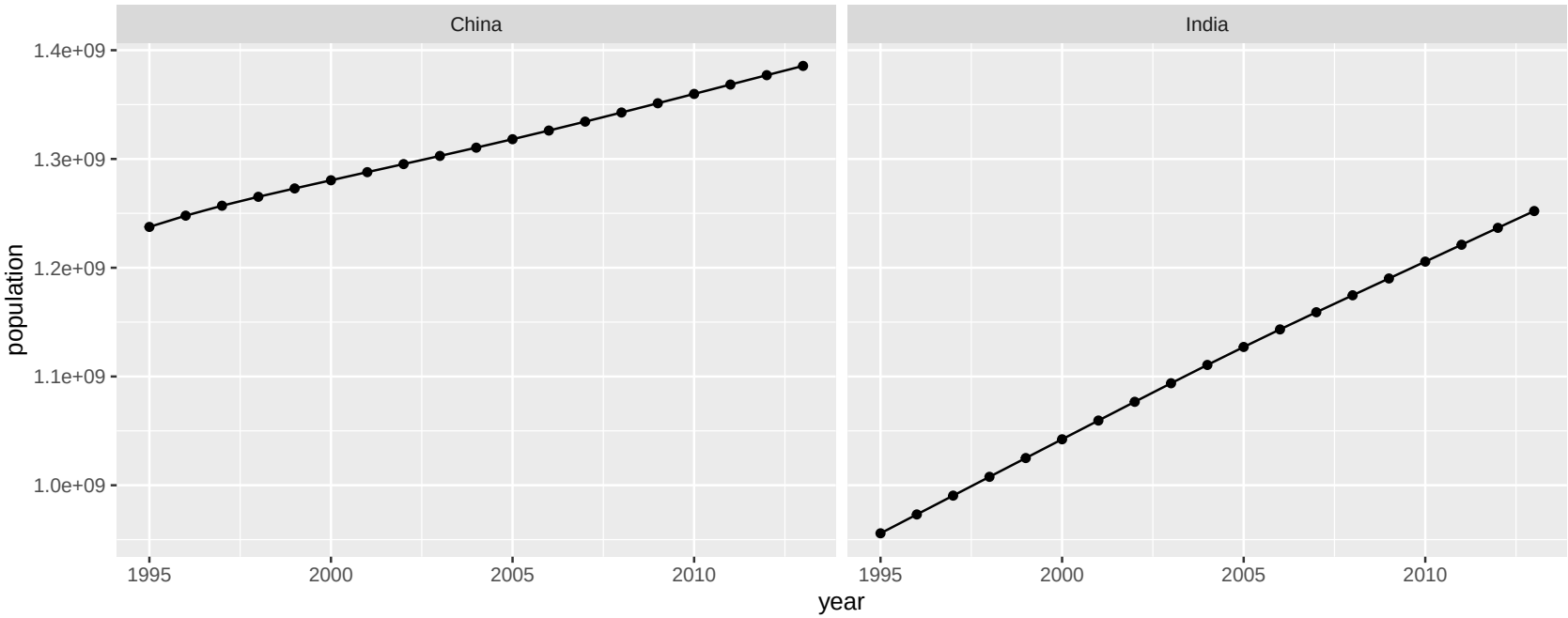
```
# Use dataset `population` from `tidyr` package and choose
# only data on China and India from the entire dataset
 pop <- population[c(784:802, 1681:1699),]
ggplot(data = pop) +
geom_point(mapping = aes(x = year, y = population)) +
facet_wrap(~ country)
```



Data visualisation: one more layer

- If we want to add a regular line to the same graph we just add one more layer with a new geom function:

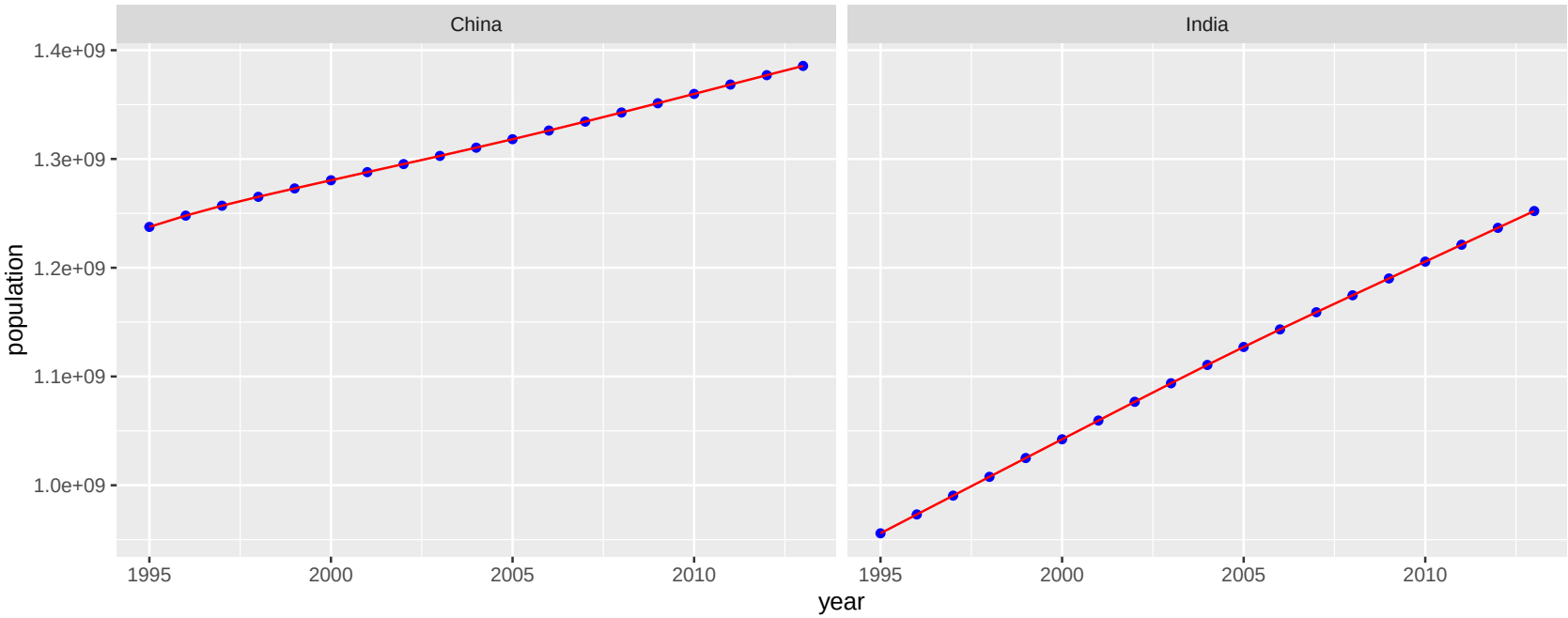
```
ggplot(data = pop) +
 geom_point(mapping = aes(x = year, y = population)) +
 facet_wrap(~ country)+geom_line(mapping = aes(x = +
 year, y = population))
```



Data visualisation: one more layer(2)

- To avoid repetition of `mapping` arguments in two geom functions, we can include mapping argument as an argument of ggplot.

```
ggplot(data = pop, mapping = aes(x = year,
y = population)) + geom_point(color = 'blue') +
facet_wrap(~ country)+geom_line(color = 'red')
```



Data visualisation: a cheetsheet

- A comprehensive overview of `ggplot()` function usage can be found in the `ggplot2` cheetsheet at: http://rstudio.com/cheatsheets

Data visualisation: a cheetsheet(2)

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

data + geom (x = F, y = A) = coordinate system → plot

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

data + geom (x = F, y = A, color = F, size = A) = coordinate system → plot

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                   required
<GEOM_FUNCTION> (mapping = aes( <MAPPINGS> ),
 stat = <STAT> , position = <POSITION> ) +                 Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round', linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

visualizing error

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map) **+ expand_limits**(x = map$long, y = map$lat), map_id, alpha, color, fill, linetype, size

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.5.0 • tibble 1.2.0 • Updated: 2017-01

Working with time series

The algorighm of working with times series models.

1. Importing data that we will do with `readr` or `readxl` packages.
2. Data visualisation that we will do with `ggplot2` package (if necessary).
3. Data manipulation that we will do with ‘dplyr‘ package treating series as data frames.
4. Transformation data frames into special time series (ts) format if necessary (some packages work only with ts data).
5. Estimation and evaluation of models with special packages developed specially for a certain class of models.

Data transformation: `dplyr` package

- Usually we do not have data exactly in the same form that we need.
- We have to transform the datset to make it more convenient to use.
- The main functions of `dplyr` package:
  - `filter()` selects observations by their values
  - `select()` selects variables by their names
  - `arrange()` sorts the rows in a particular order
  - `mutate()` creates new variables using functions of existing variables
  - `summarise()` collapses many values down to a single summary
- All the functions can be preceded by a `group_by()` function that makes any data tranformation function to work on a group-by-group basis.

Mammals sleep dataset

- As an example we take a dataset about mammals sleep

```
head(msleep)
```

```
## # A tibble: 6 x 11
##   name  genus vore  order conservation sleep_total slee
##   <chr> <chr> <chr> <chr> <chr>              <dbl>    <
## 1 Chee~ Acin~ carni Carn~ lc                  12.1
## 2 Owl ~ Aotus omni  Prim~ <NA>                17
## 3 Moun~ Aplo~ herbi Rode~ nt                  14.4
## 4 Grea~ Blar~ omni  Sori~ lc                  14.9
## 5 Cow   Bos   herbi Arti~ domesticated         4
## 6 Thre~ Brad~ herbi Pilo~ <NA>                14.4
## # ... with 3 more variables: awake <dbl>, brainwt <dbl>
```

filter() function

```r
# to select only herbivore mammals
herbivore_sleep <- filter(msleep, vore == "herbi")
```

```r
# to select only herbivore artiodactyl mammals
herbi_arti_sleep <- filter(msleep, vore == "herbi",
                           order == "Artiodactyla")
```

```r
# to select herbivore and carnivore mammals
herbi_carni <- filter(msleep, vore == "herbi" |
                      vore == "carni")
```

```r
# The same selection can be done in an alternative way
herbi_carni <- filter(msleep,
                      vore %in% c("herbi","carni"))
```

Missing values: NA

- NA means 'not available'
- As a default, `filter()` keeps only rows where the condition is TRUE and drop those where the condition is FALSE or the value is missing.

```
# Default mode
domest <- filter(msleep, conservation == "domesticated")
# We can keep missing values if necessary
domest_or_NA <- filter(msleep, conservation ==
                         "domesticated"|is.na(conservation))
```

- `is.na()` serves to determine if a value is missing

```
mv <- NA
is.na(mv)
```

```
## [1] TRUE
```

select() function

```
# to pick up variables we are interesed in
select(msleep, name, sleep_total, awake)
```

```
# to pick up all columns between two of them)
select(msleep, name, awake:bodywt)
```

```
# to drop  columns from the dataset (the original
# dataset does not change)
select(msleep, -(genus:awake))
```

```
# To select variables titles containing a string
select(msleep, name, contains("sleep"))
```

arrange() function

```
# sort mammals according their body weight
arrange(select(msleep,name, order, bodywt), bodywt)
```

```
# sort mammals according their order and
# their  body weight
arrange(select(msleep,name, order, bodywt),
        order, bodywt)
```

```
# an option 'desc()' allows to sort values in
# descending order
arrange(select(msleep,name, order, bodywt),
            desc(order), desc(bodywt))
```

mutate() function

As mutate() always adds a new variable in the end of the dataset, it is reasonable to start with limiting the dataset.

```
# To create a new variable as a function of some others.
msleep_new <- select(msleep, name, starts_with("sleep_"))
sleep_phases <- mutate(msleep_new,
                       nonrem_sleep = sleep_total - sleep_rem)
```

```
# To keep only the new variable(s) use transmute
sleep_phases <- transmute(msleep_new,
                          nonrem_sleep = sleep_total - sleep_rem)
head(sleep_phases)
```

summarise() function

summarise() collapses a data frame to a single row

```
# To find an average weight of mammels in the sample
summarise(msleep, mam_weight = mean(bodywt))
```

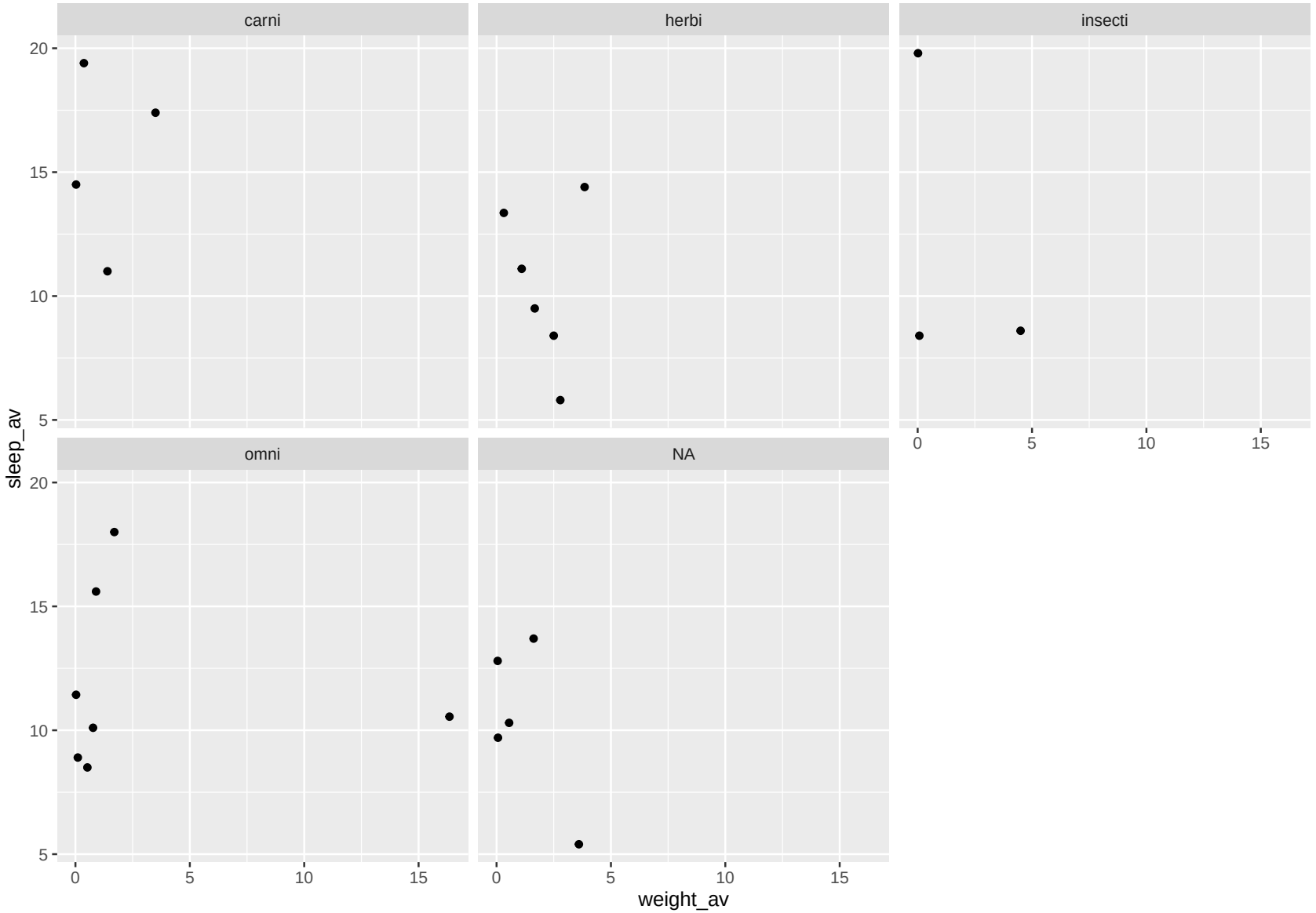
```
# Combining `summarise()` with group_by is much
# more useful  as in this case a summary for each
# group is returned
mamm_groups <- group_by(msleep, vore, order)
summarise(mamm_groups, mam_weight = mean(bodywt))
```

Step-by-step code

Imagine we want to check visually if there is a relationship between the body weight and the length of sleep in different groups of mammals.

```
mamm_groups <- group_by(msleep, vore, order)
mamm_sum <- summarize(mamm_groups,
                      weight_av = mean(bodywt),
                      sleep_av = mean(sleep_total))
mamm_sum2 <- filter(mamm_sum, weight_av < 50)
ggplot(data = mamm_sum2) +
  geom_point(mapping = aes(x = weight_av, y = sleep_av)) +
  facet_wrap(~vore)
```

Step-by-step code: result



Pipe (%>%) operator

The code is too heavy as we have to give a title to each intermediary data frame. We can do it shorter with the pipe operator.

```
mamm_sum <- msleep %>% group_by(vore, order) %>%
  summarise(weight_av = mean(bodywt),
sleep_av = mean(sleep_total)) %>% filter(weight_av < 50)
ggplot(data = mamm_sum) +
  geom_point(mapping = aes(x = weight_av, y = sleep_av)) +
  facet_wrap(~vore)
```

Working with time series

The algorighm of working with times series models.

1. Importing data that we will do with `readr` or `readxl` packages.
2. Data visualisation that we will do with `ggplot2` package (if necessary).
3. Data manipulation that we will do with `dplyr` package treating series as data frames.}
4. Transformation data frames into special time series (ts) format if necessary (some packages work only with ts data).
5. Estimation and evaluation of models with special packages developed specially for a certain class of models.