

Data Transformation with dplyr :: CHEAT SHEET



dplyr functions work with pipes and expect **tidy data**. In tidy data:



&



Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**



pipes

$x \%>\% f(y)$ becomes $f(x, y)$

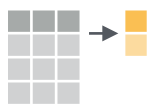
Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



summarise(.data, ...)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))



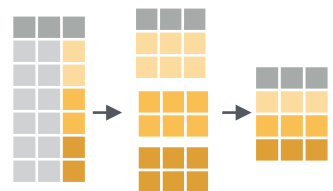
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also **tally()**.
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



*mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))*

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
g_iris <- group_by(iris, Species)

ungroup(x, ...)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



filter(.data, ...) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*



distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)



sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, ...) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%		xor()
>	>=	!is.na()	!	&	

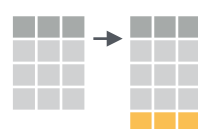
See **?base::logic** and **?Comparison** for help.

ARRANGE CASES



arrange(.data, ...) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)



select(.data, ...)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. *select(iris, starts_with("Sepal"))*

contains(match)	num_range(prefix, range)	⋮, e.g. <i>mpg:cyl</i>
ends_with(match)	one_of(...)	⋮, e.g. <i>-Species</i>
matches(match)	starts_with(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

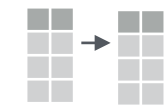
vectorized function



mutate(.data, ...)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)



transmute(.data, ...)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)



mutate_all(.tbl, .funs, ...) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))



mutate_at(.tbl, .cols, .funs, ...) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for **select()**.
mutate_at(iris, vars(-Species), funs(log(.)))

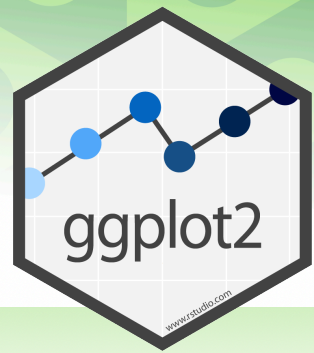


add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*



rename(.data, ...) Rename columns.
rename(iris, Length = Sepal.Length)

Data Visualization with ggplot2 : : CHEAT SHEET

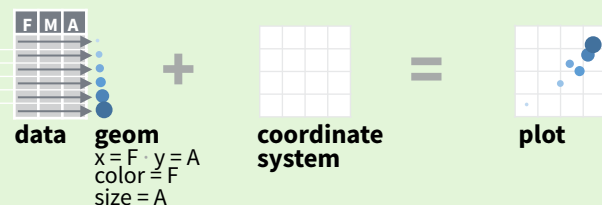


Basics

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.



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



Complete the template below to build a graph.

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

required

Not required, sensible defaults supplied

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

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 = 1) - 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 = 1:155, 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

- ```
f <- ggplot(mpg, aes(class, 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

- ```
sealsSz <- 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

#### maps

- ```
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