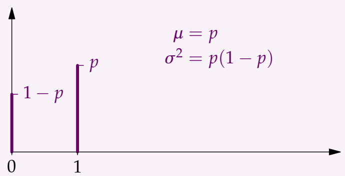
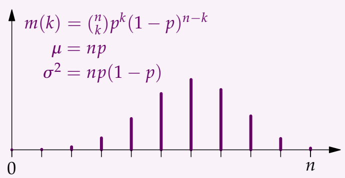
DATA 1010 · Brown University · Samuel S. Watson

Probability: Common Distributions

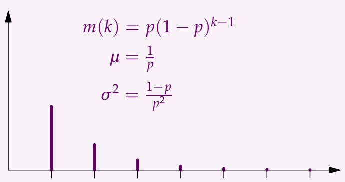
1 Bernoulli (Ber(p)): A weighted coin flip



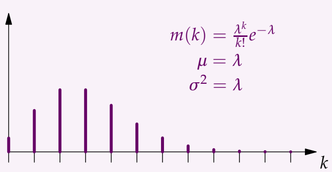
2 Binomial (Bin(n,p)): A sum of n independent Ber(p)'s



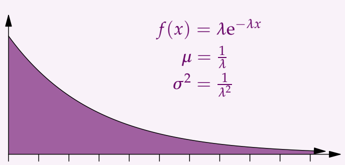
3 Geometric (Geom(p)): Time to first success (1) in a sequence of independent Ber(p)'s



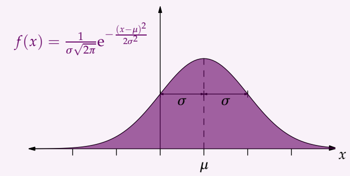
4 Poisson distribution (Poiss(λ)): Limit as $n \to \infty$ of Binomial($n, \frac{\lambda}{n}$)



5 Exponential distribution (Exp(λ)): Limit as $n \to \infty$ of distribution of $1/n$ times a Geometric(λ/n)



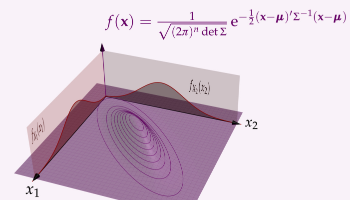
6 Normal distribution ($\mathcal{N}(\mu, \sigma^2)$): Limit as $n \to \infty$ of the distribution of $\frac{X_1 + X_2 + \cdots + X_n}{\sqrt{n}}$, for any independent sequence $X_1, \ldots, X_n$ of identically distributed random variables (i.i.d.) with $\mathbb{E}[X_1] = \mu$ and $\mathrm{Var}(X_1) = \sigma^2 < \infty$ (see Central Limit Theorem).



7 Multivariate normal distribution ($\mathcal{N}(\mathbf{0}, \Sigma)$): if $\boldsymbol{Z} = (Z_1, Z_2, \ldots, Z_n)$ is a vector of independent $\mathcal{N}(0,1)$'s, A is an $m \times n$ matrix of constants, and $\mu \in \mathbb{R}^m$, then the vector

$$\boldsymbol{X} = A\boldsymbol{Z} + \mu$$

is **multivariate normal**. The covariance matrix of $\mathbf{X}$ is $\Sigma = AA'$.



Machine Learning

1 The KDE cross-validation loss estimator is

$$J(f) = \int_{\mathbb{R}} \widehat{f}_\lambda^2 - \frac{2}{n} \sum_{i=1}^n \widehat{f}_\lambda^{(-i)}(X_i),$$

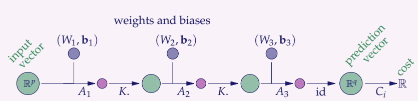
2 The logistic regression loss estimator is

$$L(r) = \sum_{i=1}^n \left[y_i \log \frac{1}{r(x_i)} + (1-y_i) \log \frac{1}{1-r(x_i)} \right],$$

3 The SVM loss estimator is

$$L(\boldsymbol{\beta}, \alpha) = \lambda |\boldsymbol{\beta}|^2 + \frac{1}{n} \sum_{i=1}^n \left[1 - y_i(\boldsymbol{\beta} \cdot \mathbf{x}_i - \alpha)\right]_+$$

4 Our neural net diagram:



Learning Standards

- ☐ **1** [SETFUN] Correctly answer questions about basic set and function terminology
- ☐ **2** [JULIA] Write Julia code to solve simple algorithmic problems using conditionals, functions, arrays, dictionaries, and iteration.
- ☐ **3** [LINALG] Use vocabulary and results from linear algebra to solve problems involving linear independence, span, and rank.
- ☐ **4** [MATALG] Use matrix algebra (including matrix transposes) to solve problems involving projection and orthogonality
- ☐ **5** [EIGEN] Apply knowledge of determinants, eigendecomposition, and singular value decomposition to data problems and other applications
- ☐ **6** [OPT] Explain the Lagrange multipliers theorem and gradient descent and discuss issues surrounding applied optimization
- ☐ **7** [MATDIFF] Differentiate matrix expressions with respect to vectors and use this technique to solve optimization problems.
- ☐ **8** [MACHARITH] Reason about 64-bit and 32-bit floating point arithmetic
- ☐ **9** [NUMERROR] Discuss the categories of numerical error and identify points of concern in application
- ☐ **10** [PRNG] Discuss basic considerations surrounding the generation of pseudorandom numbers, such as seed, period, and statistical tests
- ☐ **11** [COUNTING] Use the fundamental principle of counting and binomial coefficients to solve basic counting problems
- ☐ **12** [PROBSPACE] Explain the elements of a probability space and use probability spaces to model random experiments
- ☐ **13** [PMF] Reason about discrete random variable distributions and use properties of discrete distributions to solve problems
- ☐ **14** [PDF] Reason about continuous random variable distributions and use properties of continuous distributions to solve problems
- ☐ **15** [CONDPROB] Use the conditional probability formula to translate back and forth between branching tree diagrams and their corresponding probability spaces
- ☐ **16** [BAYES] Use Bayes' theorem and other properties of conditional probability to solve conditional probability problems
- ☐ **17** [IND] Explain independence of random variables, construct a probability space with independent random variables, and use independence to solve probability problems
- ☐ **18** [EXP] Use the definition of a random variable, the distribution of the random variable, or linearity of expectation to find the expectation of a random variable
- ☐ **19** [COV] Calculate variances and covariances, recognize high or low variance and positive or negative covariance from graphical representations of distributions, and use properties of variance and covariance to solve problems about random variable distributions
- ☐ **20** [CONDEXP] Calculate conditional expectations and apply them to expectation problems
- ☐ **21** [COMDISTD] Discuss definitions and properties of common discrete distributions (Bernoulli, binomial, geometric) and recognize circumstances under which those distributions can be expected to fit the data well
- ☐ **22** [COMDISTC] Discuss definitions and properties of common continuous distributions (Poisson, exponential, multivariate normal)
- ☐ **23** [RVINEQ] Explain inequalities involving random variable expectations (such a Chebyshev's inequality) and use them to solve problems
- ☐ **24** [CLT] State and apply the central limit theorem, and recognize when the conclusion of the central limit theorem should not be expected to hold
- ☐ **25** [KDE] Apply kernel density estimators to data problems, and explain ways of dealing with the bias-variance tradeoff in density estimation
- ☐ **26** [LR] Explain the techniques of basic linear and polynomial regression, and discuss the advantages and disadvantages relative to nonparametric methods
- ☐ **27** [QDA] Discuss the assumptions of, the estimation methods for, and facts about quadratic and linear discriminant analysis
- ☐ **28** [STATLEARN] Explain the main points of statistical learning theory (regression vs classification, loss functional, target function, learner, training and test error, overfitting, inductive bias, bias-variance tradeoff)
- ☐ **29** [NPL] Apply classification vocabulary (confusion matrix, detection rate, false alarm rate, precision, receiver operating characteristic) and the Neyman-Pearson lemma to reason about classification problems
- ☐ **30** [SVM] Describe the mathematics and intuition behind support vector machines (both hard- and soft-margin)
- ☐ **31** [LOGIST] Describe, apply, and analyze logistic regression models
- ☐ **32** [NN] Describe, apply, and analyze multi-layer perceptrons for regression and classification
- ☐ **33** [DR] Describe and interpret dimension reduction methods, including principal component analysis (concept and technical details) and t-SNE (concept only)
- ☐ **34** [R] Perform basic programming tasks in R (defining variables, generating and indexing matrices, writing functions, and reading/writing to disk)
- ☐ **35** [GGPLOT] Use `ggplot` to create data visualizations (data, aesthetics, geometries, statistics, scales, faceting)
- ☐ **36** [DPLYR] Apply the six fundamental verbs in Hadley Wickham's grammar of data manipulation (`filter`, `arrange`, `select`, `mutate`, `group_by`, `summarise`) to transform data

	julia	python	R

System

julia
```julia
pwd() # print working directory
cd("/Users/sswatson") # change directory
readdir() # files and folders in current directory
```

python
```python
import os
os.getcwd()
os.chdir("/Users/sswatson")
os.listdir()
```

R
```r
getwd()
setwd("/Users/sswatson/")
dir()
```

Packages

julia
```julia
# press ] at a Julia prompt for package mode
pkg> add Plots
julia> using Plots
```

python
```python
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
```

R
```r
install.packages('ggplot2')
library(ggplot2)
```

Arithmetic

julia
```julia
x = (1 + 2^3) % 4
x == 1 # returns true
```

python
```python
x = (1 + 2**3) % 4
x == 1
```

R
```r
x <- (1 + 2^3) %% 4
x == 1
```

Strings

julia
```julia
length("Hello World") # string length
"Hello" * "World" # concatenation
join(["Hello","World"],",") # joining
split("Hello, World",",") # splitting
'H' # single-quotes are for characters, not strings
```

python
```python
len('Hello world')
'Hello' + 'World'
','.join(['Hello','World'])
'Hello, World'.split(',')
"Hello, World" # alternate string syntax
```

R
```r
nchar('Hello World')
paste('Hello','World')
paste(c('Hello','World'),collapse='')
strsplit('Hello, World',',')
"Hello, World" # alternate string syntax
```

Booleans

julia
```julia
true && false == true  # and
false || true == true  # or
!true == false  # not
```

python
```python
True and False == False
False or True == True
not True == False
```

R
```r
TRUE && FALSE == FALSE
FALSE || TRUE == TRUE
!TRUE == FALSE
```

Loops

julia
```julia
for i = 1:10
    print(i)
end

while x > 0
    x -= 1
end
```

python
```python
for i in range(10):
    print(i)

while x > 0:
    x -= 1
```

R
```r
for (i in 1:10) {
    print(i)
}

while (x > 0) {
    x = x - 1
}
```

Conditionals

julia
```julia
if x > 0
    print("x is positive")
elseif x == 0
    print("x is zero")
else
    print("x is negative")
end

# ternary conditional
x > 0 ? 1 : -1
```

python
```python
if x > 0:
    print('x is positive')
elif x == 0:
    print('x is zero')
else:
    print('x is negative')

1 if x > 0 else -1
```

R
```r
if (x > 0) {
  print('x is positive')
}
else if (x == 0) {
  print('x is zero')
}
else {
  print('x is negative')
}

ifelse(x>0,1,-1)
```

Functions

julia
```julia
function f(x,y)
    x² = x * x # \^2[tab] gives the unicode superscript
    x² + sqrt(y*x²+1)
end # -or-
f(x) = x^2 + sqrt(y*x^2 + 1) # -or- (anonymous)
x -> x^2 + sqrt(y*x^2 + 1)
```

python
```python
def f(x,y):
    x2 = x * x
    return x2 + (y*x2+1)**(1/2)
# -or-
lambda x: x**2 + (y*x**2+1)**(1/2)
```

R
```r
f <- function(x,y) {
  x2 <- x * x
  x2 + sqrt(y*x2+1)
}
```

Splatting

julia
```julia
args = [1,2]
kwargs = (tol=0.1,maxiter=100) # a NamedTuple
f(args...;kwargs...) # equiv. to f(1,2;tol=0.1,maxiter=100)
```

python
```python
args = [1,2]
kwargs = {'tol':0.1,'maxiter':100} # a dictionary
f(*args,**kwargs) # equiv. to f(1,2,tol=0.1)
```

R
```r
library(plyr)
splat(f)(c(1,2)) # equiv. to f(1,2)
```

Lists

julia
```julia
myArray = [1,2,"a",[10,8,9]]
myArray[3] == "a"
myArray[4][2] == 8
myArray[end] == [10,8,9]
2 in myArray
```

python
```python
myList = [1,2,"a",[10,8,9]]
myList[2] == "a"
myList[3][2] == 9
myList[-1] == [10,8,9]
2 in myList
```

R
```r
myList <- list(1,2,"a",list(10,8,9))
myList[3] == "a"
myList[4][2] == 8
myList[length(myList)] # returns list(10,8,9)
2 %in% myList
```

A Julia-Python-R reference sheet – Samuel S. Watson

Mapping and filtering

julia

```julia
# Even perfect squares up to 10^2:
[x^2 for x=1:10 if x % 2 == 0]
# -or-
square(x) = x^2
square.(filter(iseven,1:10))
```

python

```python
[x**2 for x in range(1,11) if x % 2 == 0] # -or-
map(lambda x: x**2,filter(lambda x: x % 2 == 0,range(1,11)))
```

R

```r
A <- sapply(1:10,function(x) {x^2})
A[A %% 2 == 0]
```

Ranges

julia

```julia
range(0,stop=2π,step=0.1)
range(0,stop=2π,length=100)
0:5:20 == [0,5,10,15,20]
```

python

```python
np.arange(0,stop=2*np.pi,step=0.1)
np.linspace(0,stop=2*np.pi,num=100)
```

R

```r
seq(0,2*pi,by=0.1)
seq(0,2*pi,length=100)
0:5 == c(0,1,2,3,4,5)
```

Vectors and matrices

julia

```julia
A = [1 2; 3 4]
b = [1,2]
A'
size(A)
A \ b
b .> 0 # elementwise comparison
A.^2 # elementwise product
A * A # matrix product
findall(x -> x>0, b) # indices of true values
fill(2,(10,10)) # 10 x 10 matrix of 2's
I # multiplicative identity
hcat(A,b') # stack side by side
vcat(A,b) # stack vertically
```

python

```python
A = np.array([[1,2],[3,4]])
b = np.array([1,2])
np.transpose(A) # or A.T
A.shape
np.linalg.solve(A,b)
b > 0 # elementwise comparison
b**2 # elementwise function application
A @ A # matrix product
np.where(b > 0)
np.full((10,10),2)
np.eye(4) # 4 x 4 identity matrix
np.hstack((A,b[:,np.newaxis]))
np.vstack((A,b))
```

R

```r
A <- matrix(c(1,3,2,4),nrow=2) # column-wise!
b <- c(1,2)
t(A)
dim(A)
solve(A,b)
b > 0 # elementwise comparison
A^2 # elementwise product
A %*% A # matrix product
which(b > 0)
matrix(rep(2,100),nrow=10)
diag(4)
cbind(A,b)
rbind(A,b)
```

Slicing

julia

```julia
A = rand(10,10)
A[1:5,1:2:end] # first five rows, odd-indexed columns
```

python

```python
A = np.random.rand(10,10)
A[:5,1::2]
```

R

```r
A <- matrix(runif(100),nrow=10)
A[1:5,seq(1,10,by=2)]
```

Random numbers

julia

```julia
using Random; Random.seed!(1234)
rand(10,10) # matrix with Unif[0,1]'s
randn(10) # vector with N(0,1)'s
rand(10:99) # random two-digit number
```

python

```python
np.random.seed(1234)
np.random.rand(10,10)
np.random.randn(10)
np.random.randint(10,100)
```

R

```r
set.seed(1234)
matrix(runif(100),nrow=10)
rnorm(10)
sample(10:99,1)
```

Data frames

julia

```julia
using DataFrames, FileIO
myDataFrame = DataFrame(load("data.csv"))
save("mydata.csv",myDataFrame)

using DataFramesMeta, Feather
Feather.read("flights.feather") # see R column to write this file
@linq flights |>
    where(:month .== 1, :day .< 5) |>
    orderby(:day,:distance) |>
    select(:month, :day, :distance, :air_time) |>
    transform(speed = :distance ./ :air_time * 60) |>
    by(:day, avgspeed = mean(skipmissing(:speed)))
```

python

```python
import pandas as pd
myDataFrame = pd.read_csv("data.csv")
myDataFrame.to_csv("mydata.csv")

import feather
flights = feather.read_dataframe("flights.feather")
flights.query('month == 1 & day < 5') \
       .sort_values(['day','distance']) \
       [['month','day','distance','air_time']] \
       .assign(speed = flights.distance/flights.air_time * 60) \
       .groupby("day") \
       .agg({'day':'mean'})
```

R

```r
myDataFrame = read.csv("data.csv")
write.csv(myDataFrame,"mydata.csv")

library(dplyr); library(nycflights13)
flights %>%
    filter(month == 1, day < 5) %>% # filter rows
    arrange(day, distance) %>% # sort by day and distance
    select(month, day, distance, air_time) %>% # select columns
    mutate(speed = distance / air_time * 60) %>% # add a column
    group_by(day) %>% # group by day
    summarise(avgspeed = mean(speed,na.rm=TRUE)) # collapse columns
library(feather) # write flight data to disk for Python & Julia
write_feather(flights,"flights.feather")
```

Plotting

julia

```julia
using StatPlots
# select the rows with an air_time value and plot a histogram
(@linq flights |> where((!ismissing).(:air_time)))
    |> @df histogram(:air_time)
# scatter plot (using the first 10,000 records)
flights[1:10^4,:] |> @df scatter(:air_time,:distance,group=:carrier)
```

python

```python
import seaborn as sns
sns.pairplot(flights,x_vars='air_time',y_vars='distance',hue='carrier',
             plot_kws={'alpha': 0.2}) # scatter plot
sns.distplot(flights['air_time'].dropna()) # histogram
```

R

```r
library(ggplot2)
# aesthetic mapping: connects data to visual elements (x, y, size, color)
# geom: geometric object used to represent data (point, line, bar)
# geom functions return layers that you add to a ggplot
ggplot(data = flights) +
    geom_point(mapping=aes(x=air_time,y=distance,color=carrier),alpha=0.2)
```

Optimization

julia

```julia
using Optim
rosenbrock(x) =  (1.0 - x[1])^2 + 100.0 * (x[2] - x[1]^2)^2
result = optimize(rosenbrock, zeros(2), BFGS())
```

python

```python
from scipy.optimize import minimize
def rosenbrock(x):
    return (1-x[0])**2 +100*(x[1]-x[0]**2)**2
minimize(rosenbrock,[0,0],method='BFGS')
```

R

```r
rosenbrock <- function(x) {
    (1-x[1])^2 +100*(x[2]-x[1]^2)^2
}
optim(c(0,0), rosenbrock, method = "BFGS")
```

Root finding

julia

```julia
using Roots
f(x) = exp(x) - x^4
find_zero(f,3)
```

python

```python
import numpy as np
from scipy.optimize import root
def f(x):
    return np.exp(x[0]) - x[0]**4
root(f, [0])
```

R

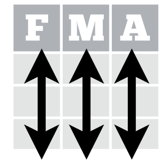
```r
f <- function(x) {
    exp(x) - x^4
}
uniroot(f,c(0,3))
```

Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

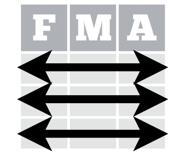
Tidy Data - A foundation for wrangling in R

In a tidy data set:



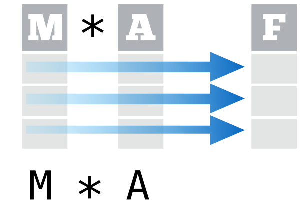
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

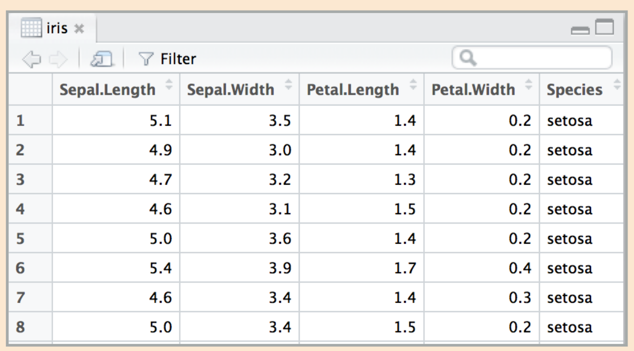
```
Source: local data frame [150 x 5]

  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3
4          4.6         3.1          1.5
5          5.0         3.6          1.4
..         ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

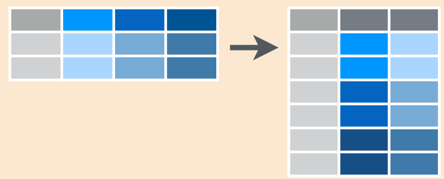
`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

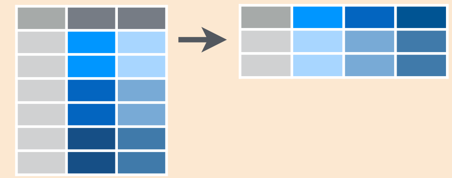
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Reshaping Data - Change the layout of a data set



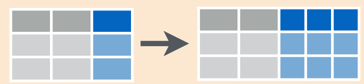
tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



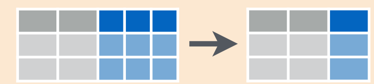
tidyr::**spread(pollution, size, amount)**

Spread rows into columns.



tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.



tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

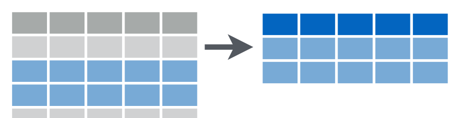
dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)



dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

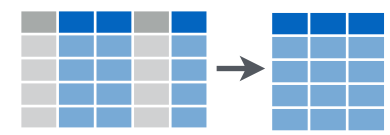
dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)



dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, **contains(".")**)

Select columns whose name contains a character string.

select(iris, **ends_with("Length")**)

Select columns whose name ends with a character string.

select(iris, **everything()**)

Select every column.

select(iris, **matches(".t.")**)

Select columns whose name matches a regular expression.

select(iris, **num_range("x", 1:5)**)

Select columns named x1, x2, x3, x4, x5.

select(iris, **one_of(c("Species", "Genus"))**)

Select columns whose names are in a group of names.

select(iris, **starts_with("Sepal")**)

Select columns whose name starts with a character string.

select(iris, **Sepal.Length:Petal.Width**)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, **-Species**)

Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets
Learn more with **browseVignettes(package = c("dplyr", "tidyr"))** • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Summarise Data

dplyr::summarise(iris, avg = mean(Sepal.Length))
Summarise data into single row of values.

dplyr::summarise_each(iris, funs(mean))
Apply summary function to each column.

dplyr::count(iris, Species, wt = Sepal.Length)
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::first
First value of a vector.

dplyr::last
Last value of a vector.

dplyr::nth
Nth value of a vector.

dplyr::n
# of values in a vector.

dplyr::n_distinct
# of distinct values in a vector.

IQR
IQR of a vector.

min
Minimum value in a vector.

max
Maximum value in a vector.

mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

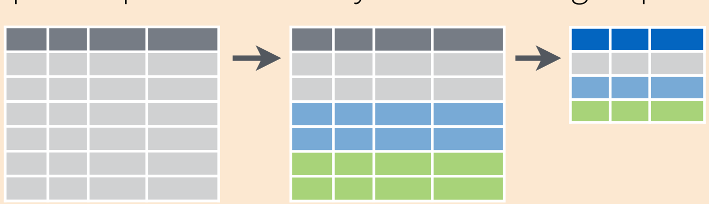
sd
Standard deviation of a vector.

Group Data

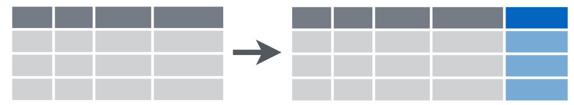
dplyr::group_by(iris, Species)
Group data into rows with the same value of Species.

dplyr::ungroup(iris)
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)
Compute separate summary row for each group.



Make New Variables



dplyr::mutate(iris, sepal = Sepal.Length + Sepal. Width)
Compute and append one or more new columns.

dplyr::mutate_each(iris, funs(min_rank))
Apply window function to each column.

dplyr::transmute(iris, sepal = Sepal.Length + Sepal. Width)
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::lead
Copy with values shifted by 1.

dplyr::lag
Copy with values lagged by 1.

dplyr::dense_rank
Ranks with no gaps.

dplyr::min_rank
Ranks. Ties get min rank.

dplyr::percent_rank
Ranks rescaled to [0, 1].

dplyr::row_number
Ranks. Ties got to first value.

dplyr::ntile
Bin vector into n buckets.

dplyr::between
Are values between a and b?

dplyr::cume_dist
Cumulative distribution.

dplyr::cumall
Cumulative `all`

dplyr::cumany
Cumulative `any`

dplyr::cummean
Cumulative `mean`

cumsum
Cumulative `sum`

cummax
Cumulative `max`

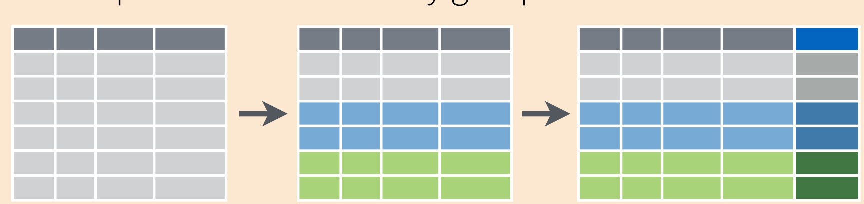
cummin
Cumulative `min`

cumprod
Cumulative `prod`

pmax
Element-wise `max`

pmin
Element-wise `min`

iris %>% group_by(Species) %>% mutate(…)
Compute new variables by group.



Combine Data Sets

a

x1	x2
A	1
B	2
C	3

\+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::left_join(a, b, by = "x1")
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::right_join(a, b, by = "x1")
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::full_join(a, b, by = "x1")
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::semi_join(a, b, by = "x1")
All rows in a that have a match in b.

x1	x2
C	3

dplyr::anti_join(a, b, by = "x1")
All rows in a that do not have a match in b.

y

x1	x2
A	1
B	2
C	3

\+

z

x1	x2
B	2
C	3
D	4

=

Set Operations

x1	x2
B	2
C	3

dplyr::intersect(y, z)
Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::union(y, z)
Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::setdiff(y, z)
Rows that appear in y but not z.

Binding

x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::bind_rows(y, z)
Append z to y as new rows.

x1	x2	x1	x2
A	1	B	2
B	2	C	3
C	3	D	4

dplyr::bind_cols(y, z)
Append z to y as new columns.
Caution: matches rows by position.

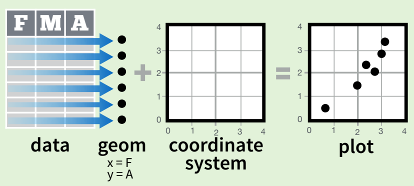
devtools::install_github("rstudio/EDAWR") for data sets
Learn more with **browseVignettes(package = c("dplyr", "tidyr"))** • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Data Visualization
with ggplot2
Cheat Sheet

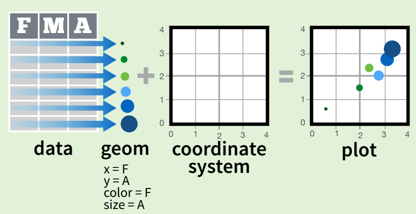
RStudio

Basics

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



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings · data · geom

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

ggplot(data = mpg, **aes**(x = cty, y = hwy))
Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()
Returns the last plot

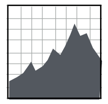
ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

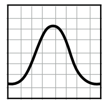
Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

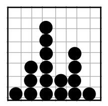
One Variable

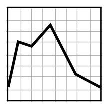
Continuous

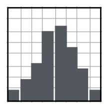
a <- ggplot(mpg, aes(hwy))


a + **geom_area(stat = "bin")**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")


a + **geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

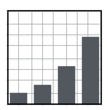

a + **geom_dotplot()**
x, y, alpha, color, fill


a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))


a + **geom_histogram(**binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

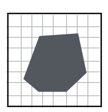
Discrete

b <- ggplot(mpg, aes(fl))

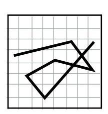

b + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

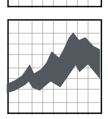
Graphical Primitives

c <- ggplot(map, aes(long, lat))

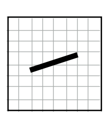

c + **geom_polygon(**aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

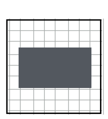
d <- ggplot(economics, aes(date, unemploy))


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


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

e <- ggplot(seals, aes(x = long, y = lat))

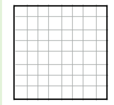

e + **geom_segment(**aes(
xend = long + delta_long,
yend = lat + delta_lat)**)**
x, xend, y, yend, alpha, color, linetype, size

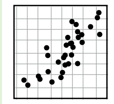

e + **geom_rect(**aes(xmin = long, ymin = lat,
xmax= long + delta_long,
ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

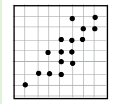
Two Variables

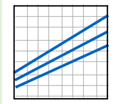
Continuous X, Continuous Y

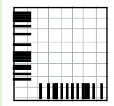
f <- ggplot(mpg, aes(cty, hwy))

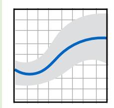

f + **geom_blank()**

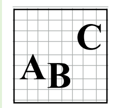

f + **geom_jitter()**
x, y, alpha, color, fill, shape, size


f + **geom_point()**
x, y, alpha, color, fill, shape, size


f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight


f + **geom_rug(**sides = "bl"**)**
alpha, color, linetype, size

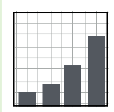

f + **geom_smooth(**model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

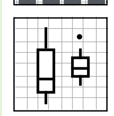


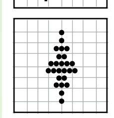
f + **geom_text(**aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

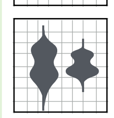
Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))


g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

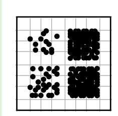

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight


g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill


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

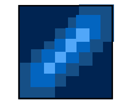
Discrete X, Discrete Y

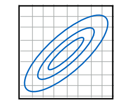
h <- ggplot(diamonds, aes(cut, color))

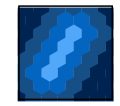

h + **geom_jitter()**
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

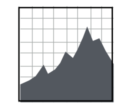

i + **geom_bin2d(**binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

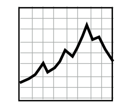

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

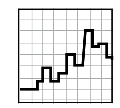

i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

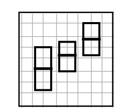

j + **geom_area()**
x, y, alpha, color, fill, linetype, size

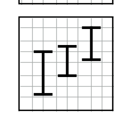

j + **geom_line()**
x, y, alpha, color, linetype, size

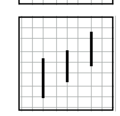

j + **geom_step(**direction = "hv"**)**
x, y, alpha, color, linetype, size

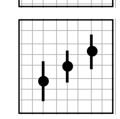
Visualizing error

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


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

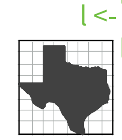

k + **geom_errorbar()**
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)


k + **geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size


k + **geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

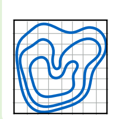
Maps

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

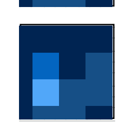

l + **geom_map(**aes(map_id = state), map = map**)** +
expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

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

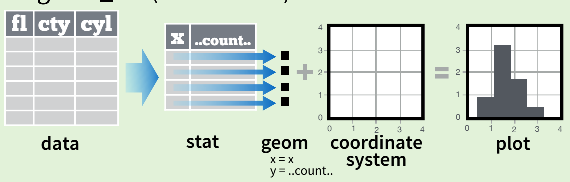

m + **geom_contour(**aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight


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


m + **geom_tile(**aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. **stat_bin(geom="bar")** does the same as **geom_bar(stat="bin")**

stat function | layer specific mappings | variable created by transformation

i + **stat_density2d**(aes(fill = ..level..), geom = "polygon", n = 100)

geom for layer | parameters for stat

1D distributions

a + **stat_bin**(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + **stat_bindot**(binwidth = 1, binaxis = "x")
x, y, | ..count.., ..ncount..
a + **stat_density**(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + **stat_bin2d**(bins = 30, drop = TRUE)
x, y, fill | ..count.., ..density..
f + **stat_binhex**(bins = 30)
x, y, fill | ..count.., ..density..
f + **stat_density2d**(contour = TRUE, n = 100)
x, y, color, size | ..level..

3 Variables

m + **stat_contour**(aes(z = z))
x, y, z, order | ..level..
m+ **stat_spoke**(aes(radius= z, angle = z))
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + **stat_summary_hex**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..
m + **stat_summary2d**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

Comparisons

g + **stat_boxplot**(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + **stat_ydensity**(adjust = 1, kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

f + **stat_ecdf**(n = 40)
x, y | ..x.., ..y..
f + **stat_quantile**(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
x, y | ..quantile.., ..x.., ..y..
f + **stat_smooth**(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + **stat_function**(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
x | ..y..
f + **stat_identity**()
ggplot() + **stat_qq**(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..x.., ..y..
f + **stat_sum**()
x, y, size | ..size..
f + **stat_summary**(fun.data = "mean_cl_boot")
f + **stat_unique**()

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.

n <- b + **geom_bar**(aes(fill = fl))
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + **scale_fill_manual**(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), **breaks** =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete

n <- b + **geom_bar**(
aes(fill = fl))
n + **scale_fill_brewer**(
palette = "Blues")
For palette choices:
library(RcolorBrewer)
display.brewer.all()
n + **scale_fill_grey**(
start = 0.2, end = 0.8,
na.value = "red")

Continuous

o <- a + **geom_dotplot**(
aes(fill = ..x..))
o + **scale_fill_gradient**(
low = "red",
high = "yellow")
o + **scale_fill_gradient2**(
low = "red", hight = "blue",
mid = "white", midpoint = 25)
o + **scale_fill_gradientn**(
colours = terrain.colors(6))
Also: rainbow(), heat.colors(),
topo.colors(), cm.colors(),
RColorBrewer::brewer.pal()

Shape scales

p <- f + **geom_point**(
aes(shape = fl))
p + **scale_shape**(
solid = FALSE)
p + **scale_shape_manual**(
values = c(3:7))
Shape values shown in chart on right

Manual shape values

0 □	6 ▽	12 ⊞	18 ◆	24 ▲
1 ○	7 ⊠	13 ⊠	19 ●	25 ▼
2 △	8 ✳	14 ⊠	20 ●	· ·
3 +	9 ◇	15 ■	21 ●	· ★
4 ×	10 ⊕	16 ●	22 ■	0 O
5 ◇	11 ✡	17 ▲	23 ◆	O O

Size scales

q <- f + **geom_point**(
aes(size = cyl))
q + **scale_size_area**(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + geom_bar()

r + **coord_cartesian**(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

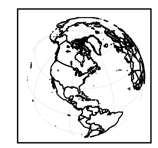
r + **coord_fixed**(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + **coord_flip()**
xlim, ylim
Flipped Cartesian coordinates

r + **coord_polar**(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + **coord_trans**(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

z + **coord_map**(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)



Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + **facet_grid(. ~ fl)**
facet into columns based on fl
t + **facet_grid(year ~ .)**
facet into rows based on year
t + **facet_grid(year ~ fl)**
facet into both rows and columns
t + **facet_wrap(~ fl)**
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(y ~ x, **scales = "free"**)
x and y axis limits adjust to individual facets
- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, **labeller = label_both**)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(. ~ fl, **labeller = label_bquote**(alpha ^ .(x)))

α^c	α^d	α^e	α^p	α^r

t + facet_grid(. ~ fl, **labeller = label_parsed**)

c	d	e	p	r

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(**position = "dodge"**)
Arrange elements side by side
s + geom_bar(**position = "fill"**)
Stack elements on top of one another, normalize height
s + geom_bar(**position = "stack"**)
Stack elements on top of one another
f + geom_point(**position = "jitter"**)
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + geom_bar(**position = position_dodge(width = 1)**)

Labels

t + **ggtitle**("New Plot Title")
Add a main title above the plot
t + **xlab**("New X label")
Change the label on the X axis
t + **ylab**("New Y label")
Change the label on the Y axis
t + **labs**(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

Legends

t + **theme**(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
t + **guides**(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + **scale_fill_discrete**(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.

Themes

r + **theme_bw()**
White background with grid lines
r + **theme_classic()**
White background no gridlines
r + **theme_grey()**
Grey background (default theme)
r + **theme_minimal()**
Minimal theme

ggthemes - Package with additional ggplot2 themes

Zooming

Without clipping (preferred)
t + **coord_cartesian**(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)
t + **xlim**(0, 100) + **ylim**(10, 20)
t + **scale_x_continuous**(limits = c(0, 100)) + **scale_y_continuous**(limits = c(0, 100))