Stat-340/341/342 – Final Exam

1 Part 1 - Multiple Choice

Enter your answers to the multiple choice questions on the provided bubble sheets. Each of the multiple choice question is worth 1 mark – there is no correction for guessing. Be sure your student name and number are completed on the bubble sheets.

Students in Stat-341 or Stat-342, should only answer questions labelled **SAS** or *R* as appropriate.

1. (*R*) Which is NOT a valid *R* expression?

   (a) TRUE + FALSE

   (b) 3/TRUE

   (c) c("Fred",13)

   (d) 3+("Fred"=="Fred")

   (e) 3+"10"

2. (*R*) The data values on blood pressure readings are stored in the file *bp.csv*.

```
Name  ,     Sex  ,  Year   ,  BP
C     ,     f    ,  2009   ,  120
C     ,     0    ,  2010   ,  130
D     ,     1    ,  NA     ,  140
M     ,     0    ,  2011   ,  140
M     ,     m    ,  2012   ,  150
```

   This data was read using

```
my.data <- read.csv("bp.csv", header=TRUE, strip.white=TRUE, as.is=TRUE)
```

   Which of the following is correct?

   (a)
```
> my.data[,3]
 2009 2010   NA 2011 2012
```

   (b)
```
> my.data[1,]
 C   0 2010 130
```

   (c)
```
> my.data[my.data$Sex=='f',]
  C   0 2010 130
```

   (d)
```
> mean(my.data[my.data$Name=="C","BP"])
  140
```

   (e)
```
> my.data[is.na(my.data$Year),"BP"]
 160
```

3. (*R*) Which of the following is correct about a standard deviation.

   (a) It measures the variation of individual items in the data

   (b) It measures how variable the population mean is when repeated samples are taken.

   (c) It measures the variation in the sample mean when a new sample is taken from the population.

   (d) It measures the variation of the confidence interval when repeated samples are taken from the population.

   (e) It measures how much the standard error would change when a new sample is taken.

4. (*R*) Which of the following is a correct statement?

   (a) An *R* vector can contain both integer and logical values.

   (b) An *R* list can contain vectors, arrays, and function objects.

   (c) An *R* data frame can have different number of rows for each column of data.

   (d) An *R* function can return multiple objects without using a list.

   (e) An *R* matrix must always have the same number of rows and columns.

5. (*R*) Which of the following is correct given the following information about a data frame on the composition of cereals:

```
> str(cereal)
'data.frame': 74 obs. of  16 variables:
 $ calories: int  70 120 70 50 110 110 110 130 90 90 ...
 $ protein : int  4 3 4 4 2 2 2 3 2 3 ...
 $ fat     : int  1 5 1 0 2 2 0 2 1 0 ...
 $ shelf   : Factor w/ 3 levels "1","2","3": 3 3 3 3 3 1 2 3 1 3 ...
 $ hot     : num  0 0 0 0 0 0 0 0 0 0 ...
```

   (a) The *lm(fat ~ calories, data=cereal)* is used to test if the mean number of calories differs by the amount of fat in the sample.

   (b) The *glm(hot ~ calories, data=cereal)* function is used to test hypotheses if the mean number of hot cereals (1) or a cold cereals (0) varies by the the number of calories/serving in the sample.

   (c) The *lm(shelf ~ calories, data=cereal)* function is used to test if the mean number of calories/serving varies over the different shelves in the population.

   (d) The *t.test(calories ~ shelf,data=cereal)* is used to test if the mean number of calories/serving varies over the different shelves in the population.

   (e) The *lm(fat ~ protein, data=cereal)* function is used to test hypotheses if the mean number of grams of fat varies by the amount of protein in a serving in the population.

6. (R) Here is some output from the *t.test()* function on the analysis of final grades in a course by the sex of the student.

```
        Welch Two Sample t-test

data:  grade by sex
t = 1.1489, df = 37.421, p-value = 0.2579
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.753133  9.970635
sample estimates:
mean in group f mean in group m
       79.79441        76.18566
```

   Which of the following is correct?

   (a) There is 26% chance that there is no difference in sex between the grades.

   (b) About 95% of individual grades lie between -3 and 10.

   (c) The p-value indicates no evidence that the population means are unequal.

   (d) Because the confidence interval includes the value of zero, all the males and females got the same grade.

   (e) The test statistic ($t = 1.1489$) is the estimated difference in means between the two sexes.

7. (*R*) The following section of code was run.

```
x <- c(1,3,5,7)
max(x,5)
```

   Which of the following is the correct output?

   (a) 5

   (b) 1 3 5 5

   (c) FALSE FALSE TRUE TRUE

   (d) 5 5 5 7

   (e) 7

8. (*R*) What is the function of the *as.is=TRUE* argument in the *read.csv()* function?

   (a) It ensures that numbers are read as numbers and dates as dates.

   (b) It ensures that missing values are added if the number of data values is too short on a data line.

   (c) It ensures that commas are used to delimit the input data.

   (d) It ensures that character strings are not converted to factors.

   (e) It ensures that extra blanks are retained in character strings.

9. (*R*) Consider the following data frame:

```
> my.data
  Name Sex Year  BP
1    C   f 2009 120
2    C   0 2010 130
3    D   1   NA 140
4    M   0 2011 140
5    M   m 2012 150
```

Which of the following is correct about the following section of code:

```
library(plyr)
ddply(my.data, "Sex", function(x){
  res <- mean(x$BP)
  names(res) <- "mean.BP"
  return(res)
})
```

(a) The created data frame has 4 rows.

(b) The final result only has a row for the male and female data.

(c) The *ddply()* function creates a list of mean blood pressure values.

(d) The *function(x)* argument of the *ddpy()* function indicates that an anova is fit between blood pressure and sex.

(e) The *library()* statement ensures that the *my.data* frame is available inside the call by *ddply()*.

10. (*R*) The following section of code was run on the cereal data frame used in class

```
ggplot(data=cereal, aes(x=fat, y=calories))+
    ggtitle("Calories vs. grams of fat")+
    xlab("Grams of fat")+ylab("Calories/serving")+
    geom_point(position=position_jitter())+
    geom_smooth(method="lm")
```

Which of the following is correct?

(a) A plot has calories on the X axis and grams of fat on the Y axis.

(b) The original data is plotted along with jittered values.

(c) The axis label for the Y axis is "Grams of fat".

(d) The plot is saved into a plot object for plotting or saving later in the program.

(e) A straight line is fit to the calories vs. fat and shown on the plot.

11. (*SAS*) How many observations and variables are contained in the following dataset?

```
data blah;
    infile datalines missover;
    length name $10 age height
    input name age height;
    if sex ="M" then delete;
    datalines;
Carl      56  185
Lois      56
Matthew   26 186
Marianne    23 .
David     22 187
;;;;
```

(a) 5 observations; 3 variables.

(b) 4 observations, 3 variables.

(c) 3 observation, 3 variables.

(d) 5 observations, 5 variables.

(e) 4 observations, 4 variables.

12. (*SAS*) The *dlm=','* option on the *INFILE* statement performs what function?

(a) Issues an error message and stops *SAS* if a data line has fewer values than variables on an INPUT statement.

(b) Indicates that data values are separated by at least one comma.

(c) Allows an input record to have more data values than variables on the INPUT statement.

(d) Because 256 characters is the default length for *SAS* input records, this option extends the maximum length of an input record.

(e) Inserts missing values into variables that try to read past the last data value on the input record.

13. (*SAS*) Which of the following is correct?

(a) *PROC GLM* is used to test hypotheses about population means.

(b) *PROC FREQ* is used to test hypotheses about sample proportions.

(c) *PROC REG* is used to test hypotheses about mean slopes.

(d) *PROC GENMOD* is used to test hypotheses about sample proportions.

(e) *PROC TTEST* is used to test hypotheses about sample means.

14. (*SAS*) Which of the following is INCORRECT about the bootstrap method to determine standard errors as seen in this class?

(a) We compute the estimate for every bootstrap sample.

(b) Bootstrap samples are selected with replacement from the given sample with the same sample size.

(c) The average of the estimates over the bootstrap samples measures the standard error.

(d) About 1000 bootstrap samples should be chosen.

(e) The 95% confidence interval is found using the 2.5^{th} and 97.5^{th} percentile of the bootstrap estimates.

15. (*SAS*) Which informat is needed to read date values of the form "17/04/2014" (excluding the quotes)?

(a) input mydate:yymmdd10.;

(b) input mydate:ddmmyy10.;

(c) input mydate:dd/mm/yy10.;

(d) input mydate:dd/mm/yyyy10.;

(e) input mydate:ddmmyyyy10.;

16. (*SAS*) What is the function of the

```
ods pdf file='report.pdf';
   ...
odd pdf close;
```

(a) Create a PDF file with all output between the two ODS statements.

(b) Create a RTF file for inclusion of output in MSWord documents.

(c) Create a report in MSWord format for a client.

(d) Create an Excel spreadsheet with the data used in the analysis.

(e) Create a graphics file that contains the output from any SGplot command.

17. (*SAS*) Here are two data sets that are merged into a final dataset.

```
data ds1;
   input studentid name$ height;
   datalines;
1         12    Carla    150
2        123     Carl      .
3        456     Fred    190
4        789 Marianne    155
;;;;

data ds2;
   input studentid weight;
   datalines;
1         12      50
2        175      85
3        456      90
4        899      55
;;;;

data allds;
   merge ds1 ds2; by studentid;
run;
```

Which statement is FALSE?

(a) The first observation will have 12 Carla 150 50 as data values for the four variables.

(b) The second observation will have 123 Carl . 85 as the data values for the four variables.

(c) The third observation will have 175 . . 85 as the data values for the four variables.

(d) The fourth observation will have 456 Fred 190 90 as the data values for the four variables.

(e) The fifth observation will have 789 Marianne 155 . as the data values for the four variables.

18. (*SAS*) Consider the following code fragment to find the average grade from assignments for each student.

```
data assign;
   input studentid assign mark;
   datalines;
123  1  .
123  2  12
123  3  18
789  2  19
789  3  17
;;;;

proc means data=assign noprint;
  by studentid;
  var mark;
  output out=assign_avg mean=mean_assign;
run;

proc print data=assign_avg;
run;
```

Which statement is correct?

(a) The mean assignment mark for student 123 is 10.

(b) The mean assignment mark for student 123 is missing.

(c) The mean assignment mark for student 123 is 15.

(d) The mean assignment mark for student 789 is 12.

(e) The mean assignment mark for student 789 is missing.

19. (*SAS*) Consider the following code fragment to find the average grade from assignments for each student.

```
data assign;
   input studentid assign1 assign2 assign3;
   avg = (assign1 + assign2 + assign3)/3;
   datalines;
123  18  12 .
456  18  .   .
789  12 19  17
;;;;
```

Which statement is correct?

(a) The mean assignment mark for student 123 is 15.

(b) The mean assignment mark for student 123 is 10.

(c) The mean assignment mark for student 789 is 17.

(d) The mean assignment mark for student 456 is missing.

(e) The mean assignment mark for all students is computed to be 16.

20. (*SAS*) Which statement is correct about *Proc SGplot*?

(a) The *scatter* statement plots the points and then fits a line to the data points.

(b) The *series* statement plots the points and then fits a linear regression to the data points.

(c) The *highlow* statement joins points in a regression line.

(d) The *density* statement creates a histogram of the data value.

(e) The *band* statement draws a shaded band between an upper and lower bound.

2 Part II -Long Answer

Name

Student Number:

Put your name and student number on the upper right of each of the following pages as well in case the pages get separated.

Answer the following four questions in the space provided.

Stat-340 students should answer ALL four questions.
Stat-341 students should only answer the first 2 question (on using *R*).
Stat-342 students should only answer the last 2 question (on using *SAS*).

Be sure that your answers are legible.

The marks given to these four questions are 3, 7, 7 and 3 respectively.

This page intentionally left blank.

1. When does Carl Schwarz retire from SFU? - using *R*

As many of you know, I cycle to and from SFU in the spring/ summer/ fall on nice days. I keep track of the number of round trips that I do in a calendar year. My retirement decision rule is

> If the number of round trips in a calendar year is less than or equal to my age, it is time to retire.

Write a *R* function that takes the number of trips in a calendar year, the current date in a a string of the form yyyy-mm-dd and the year of my birth and returns either the string "Keep on working" or "Enjoy your retirement".

For example, your function should return the following results:

```
> CarlRetirement(63, "2017-12-31", 1956)
[1] "Keep on working"
> CarlRetirement(63, "2025-12-31", 1956)
[1] "Enjoy your retirement"
```

2. Group processing in *R*

Write a series of *R* statements to do the following:

- Read in the accidents data set from the *accidents.csv* file. You may assume that all of the variable names are in the first row. In particular, the data frame contains the variables
  - *AccSev* that takes the values (3 = fatality, 2 = serious injury, 1 = non-serious injury)
  - *AccDate* with the date of the accident as a character string in yyyy-mm-dd format.
- Converts the character dates to internal *R* format.
- Extracts the month from the dates
- Computes the proportion of fatalities in each month along with a large sample confidence interval. You can either compute these directly using *R* expressions or use a *R* modeling functions. If you are computing these directly, recall that the standard error of a sample proportion is found as $se_{\widehat{p}} = \sqrt{\frac{\widehat{p}(1-\widehat{p})}{n}}$. The one-step conversation from logits to probability is $prob = 1/(1 + exp(-logit))$. If you are using *lsmeans* you can assume that the resulting output has variables named *lsmeans*, *LCL* and *UCL*.
- Create a data frame with the month, proportion of fatalities, and lower and upper 95% confidence intervals for the proportion of fatalities.

3. Group processing in *SAS*

Write a series of *SAS* statements to do the following:

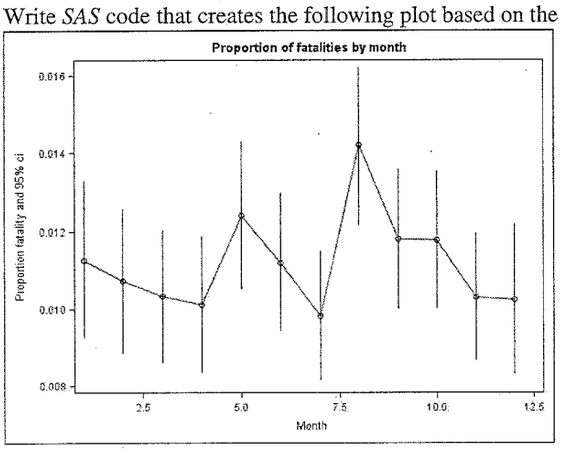
- Read in the accidents data set from the *accidents.csv* file. You may assume that the first two variables on each data line are:
  - *AccSev* that takes the values (3 = fatality, 2 = serious injury, 1 = non-serious injury)
  - *AccDate* with the date of the accident as a character string in yyyy-mm-dd format.

  You don't need any other variables from the file so you can stop reading after 2 variables.
- Converts the character dates to internal *SAS* format.
- Extracts the month from the dates
- Computes the proportion of fatalities in each month along with a large sample confidence interval. You can either compute these directly using *SAS* expressions or use a *SAS* modeling procedures. If you are computing these directly, recall that the standard error of a sample proportion is found as $se_{\widehat{p}} = \sqrt{\frac{\widehat{p}(1-\widehat{p})}{n}}$. If you are using a modeling procedure, you can assume that the ODS table name is *LSMEANS* and that the resulting data set contains the variables *estimate*, *lower* and *upper*. The one-step conversation from logits to probability is $prob = 1/(1 + exp(-logit))$.
- Create a data set with the month, proportion of fatalities, and lower and upper 95% confidence intervals for the proportion of fatalities.

4. More accident fatality rates by month

The results of fatalities by month that you created in the previous question are stored in a *report* dataset that contains a separate record for each month with following information: month (*month*); proportion of accidents with fatality (*pfatal*); and lower and upper confidence bound for the proportion of fatalities (*pfatal_lcl* and *pfatal_ucl* respectively).

Write *SAS* code that creates the following plot based on the *report* data set.



R Reference Card 2.0

Public domain, v2.0 2012-12-24.
V 2 by Matt Baggott, matt@baggott.net
V 1 by Tom Short, t.short@ieee.org
Material from *R for Beginners* by permission of Emmanuel Paradis.

Getting help and info

help(topic) documentation on topic
?topic same as above; special chars need quotes: for example **?'&&'**
help.search("topic") search the help system; same as **??topic**
apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help
summary(x) generic function to give a "summary" of x, often a statistical one
str(x) display the internal structure of an R object
ls() show objects in the search path; specify pat="pat" to search on a pattern
ls.str() str for each variable in the search path
dir() show files in the current directory
methods(x) shows S3 methods of x
methods(class=class(x)) lists all the methods to handle objects of class x
findFn() searches a database of help packages for functions and returns a data.frame (*sos*)

Other R References

CRAN task views are summaries of R resources for task domains at: cran.r-project.org/web/views Can be accessed via *ctv* package
R FAQ: cran.r-project.org/doc/FAQ/R-FAQ.html
R Functions for Regression Analysis, by Vito Ricci: cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf
R Functions for Time Series Analysis, by Vito Ricci: cran.r-project.org/doc/contrib/Ricci-refcard-ts.pdf
R Reference Card for Data Mining, by Yanchang Zhao: www.rdatamining.com/docs/R-refcard-data-mining.pdf
R Reference Card, by Jonathan Baron: cran.r-project.org/doc/contrib/refcard.pdf

Operators

<-	Left assignment, binary
->	Right assignment, binary
=	Left assignment, but not recommended
<<-	Left assignment in outer lexical scope; not for beginners
$	List subset, binary
-	Minus, can be unary or binary
+	Plus, can be unary or binary
~	Tilde, used for model formulae
:	Sequence, binary (in model formulae: interaction)
::	Refer to function in a package, i.e, pkg::function; usually not needed
*	Multiplication, binary
/	Division, binary
^	Exponentiation, binary
%x%	Special binary operators, x can be replaced by any valid name
%%	Modulus, binary
%/%	Integer divide, binary
%*%	Matrix product, binary
%o%	Outer product, binary
%x%	Kronecker product, binary
%in%	Matching operator, binary (in model formulae: nesting)
! x	logical negation, NOT x
x & y	elementwise logical AND
x && y	vector logical AND
x | y	elementwise logical OR
x || y	vector logical OR
xor(x, y)	elementwise exclusive OR
<	Less than, binary
>	Greater than, binary
==	Equal to, binary
>=	Greater than or equal to, binary
<=	Less than or equal to, binary

Packages

install.packages("pkgs", lib) download and install pkgs from repository (lib) or other external source
update.packages checks for new versions and offers to install
library(pkg) loads pkg, if pkg is omitted it lists packages
detach("package:pkg") removes pkg from memory

Indexing vectors

x[n]	nth element
x[-n]	all but the nth element
x[1:n]	first n elements
x[-(1:n)]	elements from n+1 to end
x[c(1,4,2)]	specific elements
x["name"]	element named "name"
x[x > 3]	all elements greater than 3
x[x > 3 & x < 5]	all elements between 3 and 5
x[x %in% c("a","if")]	elements in the given set

Indexing lists

x[n]	list with elements n
x[[n]]	nth element of the list
x[["name"]]	element named "name"
x$name	as above (w. partial matching)

Indexing matrices

x[i,j]	element at row i, column j
x[i,]	row i
x[,j]	column j
x[,c(1,3)]	columns 1 and 3
x["name",]	row named "name"

Indexing matrices data frames (same as matrices plus the following)

X[["name"]]	column named "name"
x$name	as above (w. partial matching)

Input and output (I/O)

R data object I/O

data(x) loads specified data set; if no arg is given it lists all available data sets
save(file,...) saves the specified objects (...) in XDR platform-independent binary format
save.image(file) saves all objects
load(file) load datasets written with save

Database I/O

Useful packages: *DBI* interface between R and relational DBMS; *RJDBC* access to databases through the JDBC interface; *RMySQL* interface to MySQL database; *RODBC* ODBC database access; *ROracle* Oracle database interface driver; *RpgSQL* interface to PostgreSQL database; *RSQLite* SQLite interface for R

Other file I/O

read.table(file), read.csv(file), read.delim("file"), read.fwf("file") read a file using defaults sensible for a table/csv/delimited/fixed-width file and create a data frame from it.

write.table(x,file), write.csv(x,file) saves x after converting to a data frame

txtStart and **txtStop**: saves a transcript of commands and/or output to a text file (*TeachingDemos*)

download.file(url) from internet

url.show(url) remote input

cat(..., file="", sep=" ") prints the arguments after coercing to character; sep is the character separator between arguments

print(x, ...) prints its arguments; generic, meaning it can have different methods for different objects

format(x,...) format an R object for pretty printing

sink(file) output to file, until sink()

Clipboard I/O

File connections of functions can also be used to read and write to the clipboard instead of a file.

Mac OS: **x <- read.delim(pipe("pbpaste"))**

Windows: **x <- read.delim("clipboard")**

See also **read.clipboard** (*psych*)

Data creation

c(...) generic function to combine arguments with the default forming a vector; with recursive=TRUE descends through lists combining all elements into one vector

from:to generates a sequence; ":" has operator priority; 1:4 + 1 is "2,3,4,5"

seq(from,to) generates a sequence by= specifies increment; length= specifies desired length

seq(along=x) generates 1, 2, ..., length(along); useful in for loops

rep(x,times) replicate x times; use each to repeat "each" element of x each times; rep(c(1,2,3),2) is 1 2 3 1 2 3; **rep(c(1,2,3),each=2)** is 1 1 2 2 3 3

data.frame(...) create a data frame of the named or unnamed arguments data.frame (v=1:4, ch= c("a","B","c","d"), n=10); shorter vectors are recycled to the length of the longest

list(...) create a list of the named or unnamed arguments; list(a=c(1,2),b="hi", c=3);

array(x,dim=) array with data x; specify dimensions like dim=c(3,4,2); elements of x recycle if x is not long enough

matrix(x,nrow,ncol) matrix; elements of x recycle

factor(x,levels) encodes a vector x as a factor

gl(n, k, length=n*k, labels=1:n) generate levels (factors) by specifying the pattern of their levels; k is the number of levels, and n is the number of replications

expand.grid() a data frame from all combinations of the supplied vectors or factors

Data conversion

as.array(x), as.character(x), as.data.frame(x), as.factor(x), as.logical(x), as.numeric(x), convert type; for a complete list, use **methods(as)**

Data information

is.na(x), is.null(x), is.nan(x); is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x); for a complete list, use **methods(is)**

x prints x

head(x), tail(x) returns first or last parts of an object

summary(x) generic function to give a summary

str(x) display internal structure of the data

length(x) number of elements in x

dim(x) Retrieve or set the dimension of an object; **dim(x) <- c(3,2)**

dimnames(x) Retrieve or set the dimension names of an object

nrow(x), ncol(x) number of rows/cols; **NROW(x), NCOL(x)** is the same but treats a vector as a one-row/col matrix

class(x) get or set the class of x; **class(x) <- "myclass";**

unclass(x) removes the class attribute of x

attr(x,which) get or set the attribute which of x

attributes(obj) get or set the list of attributes of obj

Data selection and manipulation

which.max(x), which.min(x) returns the index of the greatest/smallest element of x

rev(x) reverses the elements of x

sort(x) sorts the elements of x in increasing order; to sort in decreasing order: **rev(sort(x))**

cut(x,breaks) divides x into intervals (factors); breaks is the number of cut intervals or a vector of cut points

match(x, y) returns a vector of the same length as x with the elements of x that are in y (NA otherwise)

which(x == a) returns a vector of the indices of x if the comparison operation is true (TRUE), in this example the values of i for which x[i] == a (the argument of this function must be a variable of mode logical)

choose(n, k) computes the combinations of k events among n repetitions = n!/[(n − k)!k!]

na.omit(x) suppresses the observations with missing data (NA)

na.fail(x) returns an error message if x contains at least one NA

complete.cases(x) returns only observations (rows) with no NA

unique(x) if x is a vector or a data frame, returns a similar object but with the duplicates suppressed

table(x) returns a table with the numbers of the different values of x (typically for integers or factors)

split(x, f) divides vector x into the groups based on f

subset(x, ...) returns a selection of x with respect to criteria (..., typically comparisons: x$V1 < 10); if x is a data frame, the option select gives variables to be kept (or dropped, using a minus)

sample(x, size) resample randomly and without replacement size elements in the vector x, for sample with replacement use: replace = TRUE

sweep(x, margin, stats) transforms an array by sweeping out a summary statistic

prop.table(x,margin) table entries as fraction of marginal table

xtabs(a b,data=x) a contingency table from cross-classifying factors

replace(x, list, values) replace elements of x listed in index with values

Data reshaping

merge(a,b) merge two data frames by common col or row names

stack(x, ...) transform data available as separate cols in a data frame or list into a single col

unstack(x, ...) inverse of stack()

rbind(...) , cbind(...) combines supplied matrices, data frames, etc. by rows or cols

melt(data, id.vars, measure.vars) changes an object into a suitable form for easy casting, (*reshape2* package)

cast(data, formula, fun) applies fun to melted data using formula (*reshape2* package)

recast(data, formula) melts and casts in a single step (*reshape2* package)

reshape(x, direction...) reshapes data frame between 'wide' (repeated measurements in separate cols) and 'long' (repeated measurements in separate rows) format based on direction

Applying functions repeatedly

(m=matrix, a=array, l=list; v=vector, d=dataframe)

apply(x,index,fun) input: m; output: a or l; applies function fun to rows/cols/cells (index) of x

lapply(x,fun) input l; output l; apply fun to each element of list x

sapply(x,fun) input l; output v; user friendly wrapper for **lapply()**; see also **replicate()**

tapply(x,index,fun) input l output l; applies fun to subsets of x, as grouped based on index

by(data,index,fun) input df; output is class "by", wrapper for tapply

aggregate(x,by,fun) input df; output df; applies fun to subsets of x, as grouped based on index. Can use formula notation.

ave(data, by, fun = mean) gets mean (or other fun) of subsets of x based on list(s) by

***plyr* package functions** have a consistent names: The first character is input data type, second is output. These may be d(ataframe), l(ist), a(rray), or _(discard). Functions have two or three main arguments, depending on input:

a*ply(.data, .margins, .fun, ...)
d*ply(.data, .variables, .fun, ...)
l*ply(.data, .fun, ...)

Three commonly used functions with ply functions are **summarise(), mutate(), and transform()**

Math

Many math functions have a logical parameter na.rm=FALSE to specify missing data removal.

sin,cos,tan,asin,acos,atan,atan2,log,log10,exp
min(x), max(x) min/max of elements of x
range(x) min and max elements of x
sum(x) sum of elements of x
diff(x) lagged and iterated differences of vector x
prod(x) product of the elements of x
round(x, n) rounds the elements of x to n decimals
log(x, base) computes the logarithm of x
scale(x) centers and reduces the data; can center only (scale=FALSE) or reduce only (center=FALSE)
pmin(x,y,...), pmax(x,y,...) parallel minimum/maximum, returns a vector in which ith element is the min/max of x[i], y[i], . . .
cumsum(x), cummin(x), cummax(x), cumprod(x) a vector which ith element is the sum/min/max from x[1] to x[i]
union(x,y), intersect(x,y), setdiff(x,y), setequal(x,y), is.element(el,set) "set" functions
Re(x) real part of a complex number
Im(x) imaginary part
Mod(x) modulus; **abs(x)** is the same
Arg(x) angle in radians of the complex number
Conj(x) complex conjugate
convolve(x,y) compute convolutions of sequences
fft(x) Fast Fourier Transform of an array
mvfft(x) FFT of each column of a matrix
filter(x,filter) applies linear filtering to a univariate time series or to each series separately of a multivariate time series

Correlation and variance

cor(x) correlation matrix of x if it is a matrix or a data frame (1 if x is a vector)

cor(x, y) linear correlation (or correlation matrix) between x and y

var(x) or **cov(x)** variance of the elements of x (calculated on n − 1); if x is a matrix or a data frame, the variance-covariance matrix is calculated

var(x, y) or **cov(x, y)** covariance between x and y, or between the columns of x and those of y if they are matrices or data frames

Matrices

t(x) transpose

diag(x) diagonal

%*% matrix multiplication

solve(a,b) solves a %*% x = b for x solve(a) matrix inverse of a

rowsum(x), colsum(x) sum of rows/cols for a matrix-like object (consider **rowMeans(x), colMeans(x)**)

Distributions

Family of distribution functions, depending on first letter either provide: r(andom sample) ; p(robability density), c(umulative probability density),or q(uantile):

rnorm(n, mean=0, sd=1) Gaussian (normal)
rexp(n, rate=1) exponential
rgamma(n, shape, scale=1) gamma
rpois(n, lambda) Poisson
rweibull(n, shape, scale=1) Weibull
rcauchy(n, location=0, scale=1) Cauchy
rbeta(n, shape1, shape2) beta
rt(n, df) 'Student' (t)
rf(n, df1, df2) Fisher-Snedecor (*F*) (!!!²)
rchisq(n, df) Pearson
rbinom(n, size, prob) binomial
rgeom(n, prob) geometric
rhyper(nn, m, n, k) hypergeometric
rlogis(n, location=0, scale=1) logistic
rlnorm(n, meanlog=0, sdlog=1) lognormal
rnbinom(n, size, prob) negative binomial
runif(n, min=0, max=1) uniform
rwilcox(nn, m, n), rsignrank(nn, n) Wilcoxon

Descriptive statistics

mean(x) mean of the elements of x

median(x) median of the elements of x

quantile(x,probs=) sample quantiles corresponding to the given probabilities (defaults to 0,.25,.5,.75,1)

weighted.mean(x, w) mean of x with weights w

rank(x) ranks of the elements of x

describe(x) statistical description of data (in *Hmisc* package)

describe(x) statistical description of data useful for psychometrics (in *psych* package)

sd(x) standard deviation of x

density(x) kernel density estimates of x

Some statistical tests

cor.test(a,b) test correlation; **t.test()** t test; **prop.test(), binom.test()** sign test; **chisq.test()** chi-square test; **fisher.test()** Fisher exact test; **friedman.test()** Friedman test; **ks.test()** Kolmogorov-Smirnov test... use **help.search("test")**

Models

Model formulas

Formulas use the form: response ~ termA + termB ...
Other formula operators are:

1	intercept, meaning depdendent variable has its mean value when independent variables are zeros or have no influence
:	interaction term
*	factor crossing, a*b is same as a+b+a:b
^	crossing to the specified degree, so (a+b+c)^2 is same as (a+b+c)*(a+b+c)
-	removes specified term, can be used to remove intercept as in resp ~ a - 1
%in%	left term nested within the right: a + b %in% a is same as a + a:b
I()	operators inside parens are used literally: I(a*b) means a multiplied by b
|	conditional on, should be parenthetical

Formula-based modeling functions commonly take the arguments: data, subset, and na.action.

Model functions

aov(formula, data) analysis of variance model
lm(formula, data) fit linear models;
glm(formula, family, data) fit generalized linear models; family is description of error distribution and link function to be used; see ?family
nls(formula, data) nonlinear least-squares estimates of the nonlinear model parameters
lmer(formula, data) fit mixed effects model (*lme4*); see also **lme()** (*nlme*)
anova(fit, data...) provides sequential sums of squares and corresponding F-test for objects
contrasts(fit, contrasts = TRUE) view contrasts associated with a factor; to set use: **contrasts(fit, how.many) <- value**
glht(fit, linfct) makes multiple comparisons using a linear function linfct (*mutcomp*)
summary(fit) summary of model, often w/ t-values
confint(parameter) confidence intervals for one or more parameters in a fitted model.
predict(fit,...) predictions from fit
df.residual(fit) returns residual degrees of freedom
coef(fit) returns the estimated coefficients (sometimes with standard-errors)
residuals(fit) returns the residuals
deviance(fit) returns the deviance
fitted(fit) returns the fitted values
logLik(fit) computes the logarithm of the likelihood and the number of parameters
AIC(fit), BIC(fit) compute Akaike or Bayesian information criterion
influence.measures(fit) diagnostics for lm & glm
approx(x,y) linearly interpolate given data points; x can be an xy plotting structure
spline(x,y) cubic spline interpolation
loess(formula) fit polynomial surface using local fitting
optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN") general-purpose optimization; par is initial values, fn is function to optimize (normally minimize)
nlm(f,p) minimize function f using a Newton-type algorithm with starting values p

Flow control

if(cond) expr
if(cond) cons.expr else alt.expr
for(var in seq) expr
while(cond) expr repeat expr
break
next
switch
Use braces {} around statements
ifelse(test, yes, no) a value with the same shape as test filled with elements from either yes or no
do.call(funname, args) executes a function call from the name of the function and a list of arguments to be passed to it

Writing functions

function(arglist) expr function definition,
missing test whether a value was specified as an argument to a function
require load a package within a function
<<- attempts assignment within parent environment before search up thru environments
on.exit(expr) executes an expression at function end
return(value) or **invisible**

Strings

paste(vectors, sep, collapse) concatenate vectors after converting to character; sep is a string to separate terms; collapse is optional string to separate "collapsed" results; see also str_c below
substr(x,start,stop) get or assign substrings in a character vector. See also str_sub below
strsplit(x,split) split x according to the substring split
grep(pattern,x) searches for matches to pattern within x; see **?regex**
gsub(pattern,replacement,x) replace pattern in x using regular expression matching; sub() is similar but only replaces the first occurrence.
tolower(x), toupper(x) convert to lower/uppercase
match(x,table) a vector of the positions of first matches for the elements of x among table
x %in% table as above but returns a logical vector
pmatch(x,table) partial matches for the elements of x among table
nchar(x) # of characters. See also str_length below

***stringr* package** provides a nice interface for string functions:
str_detect detects the presence of a pattern; returns a logical vector
str_locate locates the first position of a pattern; returns a numeric matrix with col start and end. (str_locate_all locates all matches)
str_extract extracts text corresponding to the first match; returns a character vector (str_extract_all extracts all matches)
str_match extracts "capture groups" formed by () from the first match; returns a character matrix with one column for the complete match and one column for each group
str_match_all extracts "capture groups" from all matches ; returns a list of character matrices
str_replace replaces the first matched pattern; returns a character vector
str_replace_all replaces all matches.
str_split_fixed splits string into a fixed number of pieces based on a pattern; returns character matrix
str_split splits a string into a variable number of pieces; returns a list of character vectors
str_c joins multiple strings, similar to paste
str_length gets length of a string, similar to nchar
str_sub extracts substrings from character vector, similar to substr

Dates and Times

Class **Date** is dates without times. Class **POSIXct** is dates and times, including time zones. Class **timeDate** in *timeDate*includes financial centers. ***lubridate* package** is great for manipulating time/dates and has 3 new object classes:

- **interval class:** time between two specific instants. Create with **new_interval()** or subtract two times. Access with **int_start()** and **int_end()**
- **duration class:** time spans with exact lengths **new_duration()** creates generic time span that can be added to a date; other functions that create duration objects start with d: dyears(), dweeks()…
- **period class:** time spans that may not have a consistent lengths in seconds; functions include: years(), months(), weeks(), days(), hours(), minutes(), and seconds()
- **ymd(date, tz), mdy(date, tz), dmy(date, tz)** transform character or numeric dates to POSIXct object using timezone tz (*lubridate*)

Other time packages: *zoo, xts, its* do irregular time series; *TimeWarp* has a holiday database from 1980+; *timeDate* also does holidays; *tseries* for analysis and computational finance; *forecast* for modeling univariate time series forecasts; *fts* for faster operations; *tis* for time indexes and time indexed series, compatible with FAME frequencies.

Date and time formats are specified with:

%a, %A Abbreviated and full weekday name.
%b, %B Abbreviated and full month name.
%d Day of the month (01-31)
%H Hours (00-23)
%I Hours (01-12)
%j Day of year (001-366)
%m Month (01-12)
%M Minute (00-59)
%p AM/PM indicator
%S Second as decimal number (00-61)
%U Week (00-53); first Sun is day 1 of wk 1
%w Weekday (0-6, Sunday is 0)
%W Week (00-53); 1st Mon is day 1 of wk 1
%y Year without century (00-99) Don't use
%Y Year with century
%z (output only) signed offset from Greenwich; -0800 is 8 hours west of
%Z (output only) Time zone as a character string

Graphs

There are three main classes of plots in R: base plots, grid & lattice plots, and *ggplot2* package. They have limited interoperability. Base, grid, and lattice are covered here. *ggplot2* needs its own reference sheet.

Base graphics

Common arguments for base plots:

add=FALSE if TRUE superposes the plot on the previous one (if it exists)
axes=TRUE if FALSE does not draw the axes and the box
type="p" specifies the type of plot, "p": points, "l": lines, "b": points connected by lines, "o": same as previous but lines are over the points, "h": vertical lines, "s": steps, data are represented by the top of the vertical lines, "S": same as previous but data are represented by the bottom of the vertical lines
xlim=, ylim= specifies the lower and upper limits of the axes, for example with xlim=c(1, 10) or xlim=range(x)
xlab=, ylab= annotates the axes, must be variables of mode character main= main title, must be a variable of mode character
sub= sub-title (written in a smaller font)

Base plot functions

plot(x) plot of the values of x (on the y-axis) ordered on the x-axis
plot(x, y) bivariate plot of x (on the x-axis) and y (on the y-axis)
hist(x) histogram of the frequencies of x
barplot(x) histogram of the values of x; use horiz=TRUE for horizontal bars
dotchart(x) if x is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)
boxplot(x) "box-and-whiskers" plot
stripplot(x) plot of the values of x on a line (an alternative to boxplot() for small sample sizes)
coplot(x~y | z) bivariate plot of x and y for each value or interval of values of z
interaction.plot (f1, f2, y) if f1 and f2 are factors, plots the means of y (on the y-axis) with respect to the values of f1 (on the x-axis) and of f2 (different curves); the option fun allows to choose the summary statistic of y (by default fun=mean)
matplot(x,y) bivariate plot of the first column of x vs. the first one of y, the second one of x vs. the second one of y, etc.
fourfoldplot(x) visualizes, with quarters of circles, the association between two dichotomous variables for different populations (x must be an array with dim=c(2, 2, k), or a matrix with dim=c(2, 2) if k=1)
assocplot(x) Cohen-Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table
mosaicplot(x) 'mosaic' graph of the residuals from a log-linear regression of a contingency table
pairs(x) if x is a matrix or a data frame, draws all possible bivariate plots between the columns of x
plot.ts(x) if x is an object of class "ts", plot of x with respect to time, x may be multivariate but the series must have the same frequency and dates
ts.plot(x) same as above but if x is multivariate the series may have different dates and must have the same frequency
qqnorm(x) quantiles of x with respect to the values expected under a normal distribution
qqplot(x, y) diagnostic plotr of quantiles of y vs. quantiles of x; see also qqPlot in *cars* package and distplot in *vcd* package
contour(x, y, z) contour plot (data are interpolated to draw the curves), x and y must be vectors and z must be a matrix so that dim(z)= c(length(x), length(y)) (x and y may be omitted). See also filled.contour, image, and persp
symbols(x, y, ...) draws, at the coordinates given by x and y, symbols (circles, squares, rectangles, stars, thermometers or "boxplots") with sizes, colours . . . are specified by supplementary arguments
termplot(mod.obj) plot of the (partial) effects of a regression model (mod.obj)
colorRampPalette creates a color palette (use: colfunc <- colorRampPalette(c("black", "white")); colfunc(10)

Low-level base plot arguments

points(x, y) adds points (the option type= can be used)
lines(x, y) same as above but with lines
text(x, y, labels, ...) adds text given by labels at

coordinates (x,y); a typical use is: plot(x, y, type="n"); text(x, y, names)

mtext(text, side=3, line=0, ...) adds text given by text in the margin specified by side (see axis() below); line specifies the line from the plotting area segments(x0, y0, x1, y1) draws lines from points (x0,y0) to points (x1,y1)

arrows(x0, y0, x1, y1, angle= 30, code=2) same as above with arrows at points (x0,y0) if code=2, at points (x1,y1) if code=1, or both if code=3; angle controls the angle from the shaft of the arrow to the edge of the arrow head

abline(a,b) draws a line of slope b and intercept a
abline(h=y) draws a horizontal line at ordinate y
abline(v=x) draws a vertical line at abcissa x

abline(lm.obj) draws the regression line given by lm.obj

rect(x1, y1, x2, y2) draws a rectangle with left, right, bottom, and top limits of x1, x2, y1, and y2, respectively

polygon(x, y) draws a polygon linking the points with coordinates given by x and y

legend(x, y, legend) adds the legend at the point (x,y) with the symbols given by legend

title() adds a title and optionally a sub-title

axis(side, vect) adds an axis at the bottom (side=1), on the left (2), at the top (3), or on the right (4); vect (optional) gives the abcissa (or ordinates) where tick-marks are drawn

rug(x) draws the data x on the x-axis as small vertical lines

locator(n, type="n", ...) returns the coordinates (x, y) after the user has clicked n times on the plot with the mouse; also draws symbols (type="p") or lines (type="l") with respect to optional graphic parameters (...); by default nothing is drawn (type="n")

Plot parameters

These can be set globally with par(...); many can be passed as parameters to plotting commands.

adj controls text justification (0 left-justified, 0.5 centred, 1 right-justified)

bg specifies the colour of the background (ex. : bg="red", bg="blue", . . the list of the 657 available colours is displayed with colors())

bty controls the type of box drawn around the plot, allowed values are: "o", "l", "7", "c", "u" ou "]" (the box looks like the corresponding character); if bty="n" the box is not drawn

cex a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, cex.axis, the axis labels, cex.lab, the title, cex.main, and the sub-title, cex.sub

col controls the color of symbols and lines; use color names: "red", "blue" see colors() or as "#RRGGBB"; see rgb(), hsv(), gray(), and rainbow(); as for cex there are: col.axis, col.lab, col.main, col.sub

font an integer that controls the style of text (1: normal, 2: italics, 3: bold, 4: bold italics); as for cex there are: font.axis, font.lab, font.main, font.sub

las an integer that controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical)

lty controls the type of lines, can be an integer or string (1: "solid", 2: "dashed", 3: "dotted", 4: "dotdash", 5: "longdash", 6: "twodash", or a string of up to eight characters (between "0" and "9") that specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example lty="44" will have the same effect than lty=2

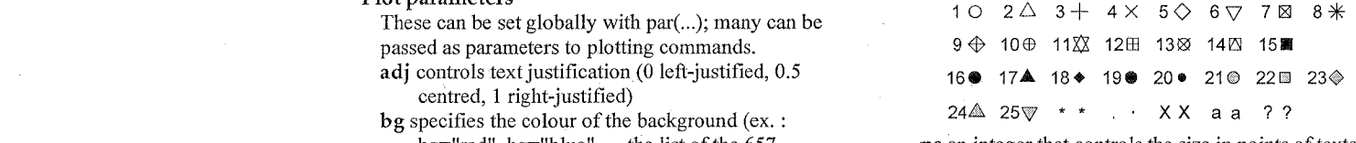
lwd numeric that controls the width of lines, default 1

mar a vector of 4 numeric values that control the space between the axes and the border of the graph of the form c(bottom, left, top, right), the default values are c(5.1, 4.1, 4.1, 2.1)

mfcol a vector of the form c(nr,nc) that partitions the graphic window as a matrix of nr lines and nc columns, the plots are then drawn in columns

mfrow same as above but the plots are drawn by row

pch controls the type of symbol, either an integer between 1 and 25, or any single char within ""



ps an integer that controls the size in points of texts and symbols

pty a character that specifies the type of the plotting region, "s": square, "m": maximal

tck a value that specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if tck=1 a grid is drawn

tcl a value that specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default tcl=-0.5)

xaxt if xaxt="n" the x-axis is set but not drawn (useful in conjonction with
axis(side=1, ...))

yaxt if yaxt="n" the y-axis is set but not drawn (useful in conjonction with axis(side=2, ...))

Lattice graphics

Lattice functions return objects of class trellis and must be printed. Use print(xyplot(...)) inside functions where automatic printing doesn't work. Use lattice.theme and lset to change Lattice defaults.
In the normal Lattice formula, y x|g1*g2 has combinations of optional conditioning variables g1 and g2 plotted on separate panels. Lattice functions take many of the same args as base graphics plus also data= the data frame for the formula variables and subset= for subsetting. Use panel= to define a custom panel function (see apropos("panel") and ?llines).

xyplot(y~x) bivariate plots (with many functionalities)

barchart(y~x) histogram of the values of y with respect to those of x

dotplot(y~x) Cleveland dot plot (stacked plots line-by-line and column-by-column)

densityplot(~x) density functions plot histogram(~x) histogram of the frequencies of x bwplot(y~x) "box-and-whiskers" plot

qqmath(~x) quantiles of x with respect to the values expected under a theoretical distribution

stripplot(y~x) single dimension plot, x must be numeric, y may be a factor

qq(y~x) quantiles to compare two distributions, x must be numeric, y may be numeric, character, or factor but must have two 'levels'

splom(~x) matrix of bivariate plots

parallel(~x) parallel coordinates plot

levelplot(z~x*y|g1*g2) coloured plot of the values of z at the coordinates given by x and y (x, y and z are all of the same length)

wireframe(z~x*y|g1*g2) 3d surface plot

cloud(z~x*y|g1*g2) 3d scatter plot

Geoms

`geom_abline`(aes(intercept, slope), alpha, colour, linetype, size)
The abline geom adds a line with specified slope and intercept to the plot.

`geom_area`(aes(x, ymin=0, ymax), alpha, colour, fill, linetype, size)
An area plot is the continuous analog of a stacked bar chart (see `geom_bar`), and can be used to show how composition of the whole varies over the range of x. Choosing the order in which different components is stacked is very important, as it becomes increasing hard to see the individual pattern as you move up the stack.

`geom_bar`(aes(x), alpha, colour, fill, linetype, size, weight)
The bar geom is used to produce 1d area plots: bar charts for categorical x, and histograms for continuous y. `stat_bin` explains the details of these summaries in more detail. In particular, you can use the weight aesthetic to create weighted histograms and barcharts where the height of the bar no longer represent a count of observations, but a sum over some other variable.

`geom_bin2d`(aes(xmax, xmin, ymax, ymin), alpha, colour, fill, linetype, size, weight)
Add heatmap of 2d bin counts.

`geom_blank`(aes(),)
The blank geom draws nothing, but can be a useful way of ensuring common scales between different plots.

`geom_boxplot`(aes(lower, middle, upper, x, ymax, ymin), alpha, colour, fill, linetype, shape, size, weight, notch=FALSE,)
The upper and lower "hinges" correspond to the first and third quartiles (the 25th and 75th percentiles). This differs slightly from the method used by the `boxplot` function, and may be apparent with small samples. See `boxplot.stats` for for more information on how hinge positions are calculated for `boxplot`.

`geom_contour`(aes(x, y), alpha, colour, linetype, size, weight)
Display contours of a 3d surface in 2d. See `stat_contour`.

`geom_crossbar`(aes(x, y, ymax, ymin), alpha, colour, fill, linetype, size)
Hollow bar with middle indicated by horizontal line. See `geom_linerange`.

`geom_density`(aes(x, y), alpha, colour, fill, linetype, size, weight)
A smooth density estimate calculated by `stat_density`.

`geom_density2d`(aes(x, y), alpha, colour, fill, linetype, size)
Perform a 2D kernel density estimatation using `kde2d` and display the results with contours.

`geom_dotplot`(aes(x, y), alpha, colour, fill)
In a dot plot, the width of a dot corresponds to the bin width (or maximum width, depending on the binning algorithm), and dots are stacked, with each dot representing one observation.

`geom_errorbar`(aes(x, ymax, ymin), alpha, colour, linetype, size, width)
Error bars.

`geom_freqpoly`(aes(x), alpha, colour, linetype, size)
Frequency polygon.

`geom_hex`(aes(x, y), alpha, colour, fill, size)
Hexagon binning.

`geom_histogram`(aes(x), alpha, colour, fill, linetype, size, weight)
`geom_histogram` is an alias for `geom_bar` plus `stat_bin` (look there to see parameters).

`geom_hline`(aes(yintercept), alpha, colour, linetype, size)
This geom allows you to annotate the plot with horizontal lines.

`geom_jitter`(aes(x, y), alpha, colour, fill, shape, size)
The jitter geom is a convenient default for `geom_point` with `position = 'jitter'`. See `position_jitter` to see how to adjust amount of jittering.

`geom_line`(aes(x, y), alpha, colour, linetype, size)
Connect observations, ordered by x value.

`geom_linerange`(aes(x, ymin, ymax),)
An interval represented by a vertical line

`geom_map`(aes(`map_id`), alpha, colour, fill, linetype, size)
Need data frame with map coordinates, with columns `x` or `long`, `y` or `lat`, and `region` or `id`. With `geom_polygon` will need two data frames - coordinates of the polygon (positions) and values for each polygon (values) linked by an id variable. `expand_limits`() may also be helpful.

`geom_path`(aes(x, y), alpha, colour, linetype, size)
Connect observations in original order.

`geom_point`(aes(x, y), alpha, colour, fill, shape, size)
Used to create scatterplots.

`geom_pointrange`(aes(x, y, ymax, ymin), alpha, colour, fill, linetype, shape, size)
An interval represented by a vertical line with a point in the middle. See `geom_linerange`.

`geom_polygon`(aes(x, y), alpha, colour, fill, linetype, size)
Polygon, a filled path.

`geom_quantile`(aes(x, y), alpha, colour, linetype, size, weight)
A continuous analogue of `geom_boxplot`.

`geom_raster`(aes(x, y), alpha, fill)
This is a special case of `geom_tile` where all tiles are the same size. It is implemented highly efficiently using the internal `rasterGrob` function.

`geom_rect`(aes(xmax, xmin, ymax, ymin), alpha, colour, fill, linetype, size)
2d rectangles.

`geom_ribbon`(aes(x, ymax, ymin), alpha, colour, fill, linetype, size)
Ribbons, y range with continuous x values.

`geom_rug`(aes(), alpha, colour, linetype, size)
Marginal rug plots.

`geom_segment`(aes(x, xend, y, yend), alpha, colour, linetype, size)
Single line segments.

`geom_smooth`(aes(x, y), alpha, colour, fill, linetype, size, weight)
Add a smoothed conditional mean. See `stat_smooth()`

`geom_step`(aes(x, y), alpha, colour, linetype, size)
Connect observations by stairs.

`geom_text`(aes(label, x, y), alpha, angle, colour, family, fontface, hjust, lineheight, size, vjust)
Textual annotations.

`geom_tile`(aes(x, y), alpha, colour, fill, linetype, size)
Similar to `levelplot` and `image`.

`geom_violin`(aes(x, y), alpha, colour, fill, linetype, size, weight)
Violin plot

`geom_vline`(aes(xintercept), alpha, colour, linetype, size)
This geom allows you to annotate the plot with vertical lines (see `geom_hline` and `geom_abline` for other types of lines.

Positions

`position_dodge`(width = NULL, height = NULL)
Adjust position by dodging overlaps to the side.

`position_fill`(width = NULL, height = NULL)
Stack overlapping objects on top of one another, and standardise to have equal height.

`position_identity`(width = NULL, height = NULL)
Don't adjust position

`position_stack`(width = NULL, height = NULL)
Stack overlapping objects on top of one another

`position_jitter`(width=NULL, height=NULL)
Jitter points to avoid overplotting.

Statistics

stat_bin(binwidth, breaks, origin, width, right=TRUE, drop=FALSE, ...)

stat_bin2d(bins, drop=FALSE, ...)

stat_bindot(binaxis="x", method="dotdensity", binwidth, binpositions, origin, right=TRUE, drop=FALSE, na.rm=FALSE, aes(), geom, position)

stat_binhex(bins=c(30, 30), na.rm=FALSE, ...)
Bin 2d plane into hexagons

stat_boxplot(coef=1.5, na.rm=FALSE, ...)
Calculates components of box and whisker plot.

stat_contour(na.rm=FALSE, ..., bins, binwidth)
Calculates contours of 3d data; bins gives number of contours, binwidth specifies the same thing by contour width. Also possible to map size or color to contour level by =..level..

stat_density(adjust, kernel, trim=TRUE, na.rm=FALSE, ...)
1d kernel density estimate.

stat_density2d(contour=TRUE, n, kde2d(...), na.rm=TRUE, ...)
2d density estimation. `kde2d(...)` is for other arguments to be passed to `kde2d`.

stat_ecdf(n, ...)
Empirical CDF of x. If n is NULL, do not interpolate, otherwise, interpolate over n points.

stat_function(fun, n, args, ...)
Superimpose a function, `fun`, n points to interpolate along, with `args()` to pass to `fun`.

stat_identity(width, height)
Identity statistic - width and height describe the width and height of the tiles.

stat_qq(distribution, dparams, ..., na.rm=FALSE, ...)
Calculation for quantile-quantile plot. distribution function `dist` with parameters `dparams` and other arguments ...

stat_quantiles(quantiles, formula, method="rq", na.rm=FALSE, ...)
quantiles of y to calculate, using `formula` and currently only supports method `rq`

stat_smooth(method, formula, se=TRUE, fullrange, level=0.95, n, na.rm=FALSE, ...)
Uses a smoother fit by one of `lm`, `glm`, `gam`, `loess`, or `rlm`.

stat_spoke(...)
convert angle and radius to xend and yend. Requires aes(angle, radius, x, y).

stat_summary_hex(bins, drop=TRUE, fun, ..., ...)
Apply function for 2d hexagonal bins. Bins from `stat_binhex` with `fun` for summary applied to each bin. ... includes function arguments as well as standard stat arguments

stat_summary2d(bins, drop, fun, ..., ...)
Apply function for 2d rectangular bins. Bins from `stat_bin2d` with `fun` for summary applied to each bin. ... includes function arguments as well as standard stat arguments

stat_unique(...)
Removes duplicates

stat_ydensity(trim=TRUE, scale="area", na.rm=FALSE, ..., adjust, kernel, ...)
1d kernel density estimate along y axis for violin plot. If scale="count" areas are scaled proportionate to the number of observations. If scale="width", all violins have the same maximum width.

stat_sum(...)
Sum unique values - useful for overplotting on scatterplots.

stat_summary(...)
Allows flexibility in specification of summary functions - either operating on the data frame with argument name `fun.data` or on a vector `fun.y, fun.ymax, fun.ymin`.

Coordinate systems

coord_cartesian(xlim, ylim)
Setting limits on the coordinate system will zoom the plot (like you're looking at it with a magnifying glass), and will not change the underlying data like setting limits on a scale will.

coord_fixed(ratio = 1, xlim = NULL, ylim = NULL)
Forces a specified ratio between the physical representation of data units on the axes. The ratio represents the number of units on the y-axis equivalent to one unit on the x-axis.

coord_flip(...)
Flipped cartesian coordinates so horizontal becomes vertical.

coord_map(projection = "mercator", ..., orientation = c(90, 0, mean(range(x))), xlim = NULL, ylim = NULL)
This coordinate system provides the full range of map projections available in the mapproj package. Alternate projections can be found in that package.

coord_polar(theta = "x", start = 0, direction = 1)
The polar coordinate system is most commonly used for pie charts, which are a stacked bar chart in polar coordinates.

coord_trans(xtrans = "identity", ytrans = "identity", limx = NULL, limy = NULL)
Different from scale transformations in that it occurs after statistical transformation and will affect the visual appearance of geoms - there is no guarantee that straight lines will continue to be straight. Currently works only with cts values.

Faceting

facet_grid(facets, margins = FALSE, scales = "fixed", space = "fixed", shrink = TRUE, labeller = "label_value", as.table = TRUE, drop = TRUE)
Lay out panels in a grid.

facet_null(shrink=TRUE)
Specifies a single panel. If shrink=TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.

facet_wrap(facets, nrow = NULL, ncol = NULL, scales = "fixed", shrink = TRUE, as.table = TRUE, drop = TRUE)
Wrap a 1d ribbon of panels into 2d.

label_both
Passed in `facet_grid` to the labeller argument. Labels with variable name and value.

label_bquote(...)
Passed in `facet_grid` to the labeller argument. See bquote for details on the syntax of the argument. The label value is x. Useful for facet labels that are expressions

label_parsed(...)
Passed in `facet_grid` to the labeller argument. Label facets with parsed label. Useful for facet labels that are expressions.

label_value(...)
Passed in `facet_grid` to the labeller argument. Default labels.

Scales

`expand_limits`(...)
named list of aesthetics specifying the value that should be included in each scale

`guides`(...)
List of scale guide pairs

`guide_legend`(title = waiver(), title.position = NULL, title.theme = NULL, title.hjust = NULL, title.vjust = NULL, label = TRUE, label.position = NULL, label.theme = NULL, label.hjust = NULL, label.vjust = NULL, keywidth = NULL, keyheight = NULL, direction = NULL, default.unit = "line", override.aes = list(), nrow = NULL, ncol = NULL, byrow = FALSE, reverse = FALSE, order = 0, ...)
Legend type guide shows key (i.e., geoms) mapped onto values. Legend guides for various scales are integrated if possible.

`guide_colorbar`(title = waiver(), title.position = NULL, title.theme = NULL, title.hjust = NULL, title.vjust = NULL, label = TRUE, label.position = NULL, label.theme = NULL, label.hjust = NULL, label.vjust = NULL, barwidth = NULL, barheight = NULL, nbin = 20, raster = TRUE, ticks = TRUE, draw.ulim = TRUE, draw.llim = TRUE, direction = NULL, default.unit = "linc", reverse = FALSE, order = 0, ...)
Colour bar guide shows continuous color scales mapped onto values. Colour bar is available with `scale_fill` and `scale_colour`.

`scale_alpha`(..., range = c(0.1, 1))

`scale_area`(..., range=c(1,6))

`scale_colour_brewer`(..., type="seq", palette=1)
Substitute color or fill for colour. If palette is a string, will use that name, otherwise, will index the list of palettes.

`scale_colour_gradient`(..., low = "#132B43", high = "#56B1F7", space = "Lab", na.value = "grey50", guide = "colourbar")
Substitute color or fill for colour. Also aliases `scale_colour_continuous`.

`scale_colour_gradient2`(..., low = muted("red"), mid = "white", high = muted("blue"), midpoint = 0, space = "rgb", na.value = "grey50", guide = "colourbar")
Diverging color scheme. Substitute color or fill for colour.

`scale_colour_gradientn`(..., colours, values = NULL, space = "Lab", na.value = "grey50", guide = "colourbar")
Smooth color gradient between n colors. Substitute color or fill for colour.

`scale_colour_grey`(..., start = 0.2, end = 0.8, na.value = "red")

`scale_colour_hue`(..., h = c(0, 360) + 15, c = 100, l = 65, h.start = 0, direction = 1, na.value = "grey50")
Qualitative colour scale with evenly spaced hues. Substitute color or fill for colour. Also aliases `scale_colour_discrete`.

`scale_colour_identity`(..., guide="none")
Use values without scaling. Substitute fill, shape, linetype, alpha, size, color for colour.

`scale_colour_manual`(..., values)
Create your own discrete scale

`scale_linetype_discrete`(..., na.value = "blank")
Must be a discrete scale.

`scale_shape_discrete`(..., solid = TRUE)
Must be a discrete scale.

`scale_size`(..., range = c(1, 6))
Can be discrete (`scale_size_discrete`) or continuous. Range specifies minimum and maximum size of plotting symbols after transformation.

`scale_x_continuous`(..., expand=waiver())
Also works for y. Common parameters: name, breaks, labels, na.value, limits, trans. Aliases for transformations: `scale_x_log10`, `scale_x_reverse`, `scale_x_sqrt`.

`scale_x_date`(..., expand = waiver(), breaks = pretty_breaks(), minor_breaks = waiver())
Also works for y. Args: `breaks` = vector of breaks, `minor_breaks` = locations of minor breaks between labeled breaks.

`scale_x_datetime`(..., expand = waiver(), breaks = pretty_breaks(), minor_breaks = waiver())
Also works for y. Args: `breaks` = vector of breaks, `minor_breaks` = locations of minor breaks between labeled breaks.

`scale_x_discrete`(..., expand = waiver())
Also works for y. You can use continuous positions even with a discrete position scale - this allows you (e.g.) to place labels between bars in a bar chart. Continuous positions are numeric values starting at one for the first level, and increasing by one for each level (i.e. the labels are placed at integer positions). This is what allows jittering to work.

`labs`(title, x, y)

`xlab`(label)

`ggtitle`(title)

`update_labels`(p, labels)
p is the plot to modify, labels are a named list of new labels. Works for axis, legend labels.

`xlim`(...)
If numeric, will create a continuous scale, if factor or character, will create a discrete scale. Observations not in this range will be dropped completely and not passed to any other layers.

Themes

`add_theme`(t1, t2, t2name)
Modify properties of an element in a theme object. Add t1 to t2 and name it t2name.

`element_blank`()()
This theme element draws nothing, and assigns no space

`element_line`(colour = NULL, size = NULL, linetype = NULL, lineend = NULL, color = NULL)

`element_rect`(fill = NULL, colour = NULL, size = NULL, linetype = NULL, color = NULL)

`element_text`(family = NULL, face = NULL, colour = NULL, size = NULL, hjust = NULL, vjust = NULL, angle = NULL, lineheight = NULL, color = NULL)

`theme`(..., complete = FALSE)
Use this function to modify theme settings. Elements include line, rect, text, title, axis.title, axis.text, axis.ticks, axis.ticks.length, axis.ticks.margin, axis.line, legend.background, legend.box, panel.background, panel.border, panel.margin, panel.grid, plot.background, plot.title, plot.margin, strip.background, strip.text

`theme_bw`(base_size=12, base_family="")
A theme with white background and black gridlines.

`theme_grey`(base_size=12, base_family="")
A theme with grey background and white gridlines. (default theme)

`theme_classic`()
A classic-looking theme, with x and y axis lines and no gridlines.

`theme_minimal`()
A minimalistic theme with no background annotations.