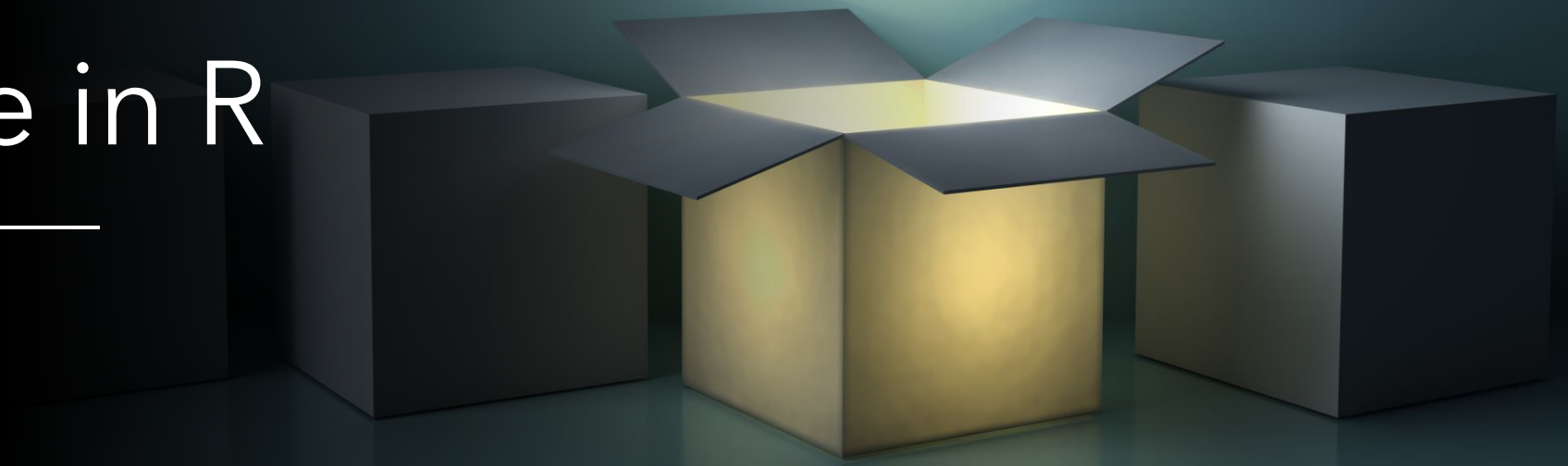
TSA Package in R



CS 510

Hanna Lu

Hesam Sadri

Introduction

- Contains R functions and datasets detailed in the book "Time Series Analysis with Applications in R (second edition)" by Jonathan Cryer and Kung-Sik Chan.
- What is time series?

Time series refers to any group of statistical information collected at regular intervals of time. Time series analysis is used to detect the changes in patterns in these collected data.

How to Install?

- install.packages("TSA")
- library(TSA)

What is the objective of time series analysis?

- Time series analysis is used to identify the fluctuation in economics and business.
- Time series is used to predict future values based on previously observed values.
- It helps in the evaluation of current achievements.
- Time series is used in pattern recognition, signal processing, weather forecasting and earthquake prediction.

Any time series data consists of trend , seasonality , cycle and an independent random values we should try to separate out these components and use the past and present data points to predict the oncoming values.

What Is an Example of Time Series Data?

Field	Example topics	Example dataset
Economics	Gross Domestic Product (GDP), Consumer Price Index (CPI), S&P 500 Index, and unemployment rates	U.S. GDP from the Federal Reserve Economic Data
Social sciences	Birth rates, population, migration data, political indicators	Population without citizenship from Eurostat
Epidemiology	Disease rates, mortality rates, mosquito populations	U.S. Cancer Incidence rates from the Center for Disease Control
Medicine	Blood pressure tracking, weight tracking, cholesterol measurements, heart rate monitoring	MRI scanning and behavioral test dataset
Physical sciences	Global temperatures, monthly sunspot observations, pollution levels.	Global air pollution from the Our World in Data

Time series Data:

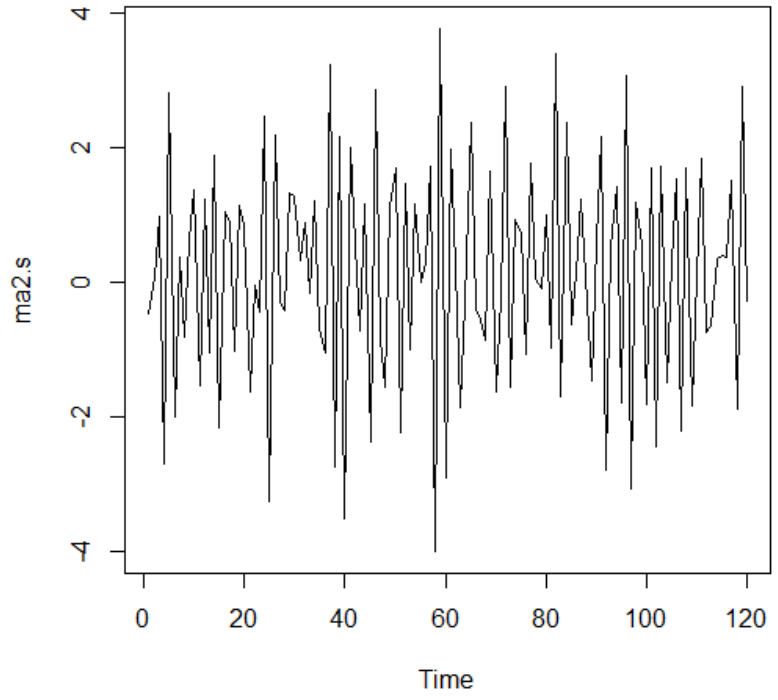
There are lots of time series data available with this package which could be used to do some analysis and test some functions. For example:

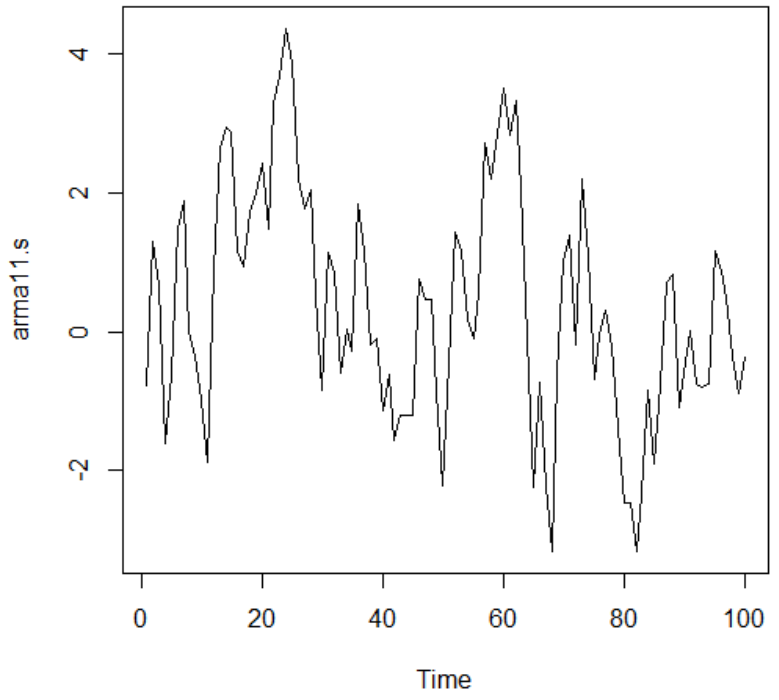
- **Airmiles** : Monthly U.S. airline passenger-miles: 01/1996 - 05/2005.
- **Airpass** : Monthly total international airline passengers from 01/1960- 12/1971.
- **Beersales** : Monthly beer sales in millions of barrels, 01/1975 - 12/1990.
- **Bluebird** : Weekly unit sales (log-transformed) of Bluebird standard potato chips (New Zealand) and their price for 104 weeks.
- **Co2**: Monthly CO2 level at Alert, Northwest Territories, Canada, near the Artic Circle, 01/1994 - 12/2004.
- **Milk**: Average monthly milk production per cow in the US, 01/1994 - 12/2005
- And so many other data are available.

Simulated data:

- **ar1.2.s :** A simulated AR(1) series with the AR coefficient equal to 0.4. The model is Y(t)=0.4*Y(t-1)+e(t) where the e's are iid standard normal.
- **ar1.s** : A simulated AR(1) series with the AR coefficient equal to 0.9. The model is Y(t)=0.9*Y(t-1)+e(t) where the e's are iid standard normal.
- **ar2.s** : A simulated AR(2) series with AR coefficients being equal to 1.5 and -0.75. The model is Y(t)=1.5*Y(t-1)-0.75*Y(t-2)+e(t) where the e's are iid standard normal random variables.
- **arma11.s :** A simulated ARMA(1,1) series with the model given by Y(t)=0.6*Y+ e(t)+0.3e(t-1) where the e's are iid standard normal random variables.
- **ma1 .s:** A simulated MA(1) series with the MA(1) coefficient equal to 0.9. The model is Y (t) = e(t) - 0:9e(t-1) where the e's are iid standard norma
- **ma2.s :** A simulated MA(2) series with MA coefficients being 1 and -0.6.l. The model is Y (t) = e(t)-e(-1)+0.6 e(t-2) where the e's are iid standard normal random variables.

Sample chart for ma(2) and arma(1,1)



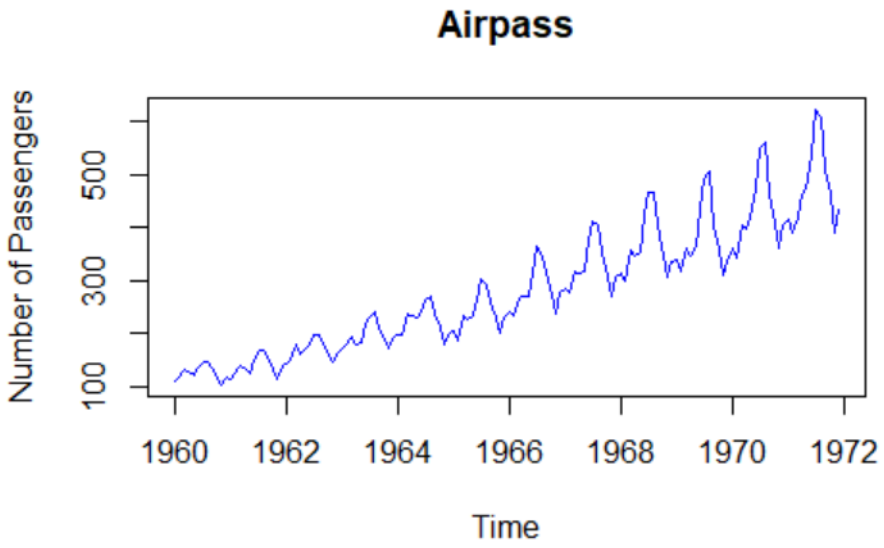


EXAMPLE

- The monthly airline passenger time series, first investigated in Box and Jenkins (1976), is considered a classic time series. The data are in the file named airpass.
  - Load the data:
  - Plot the data:

```
data(airpass)
```

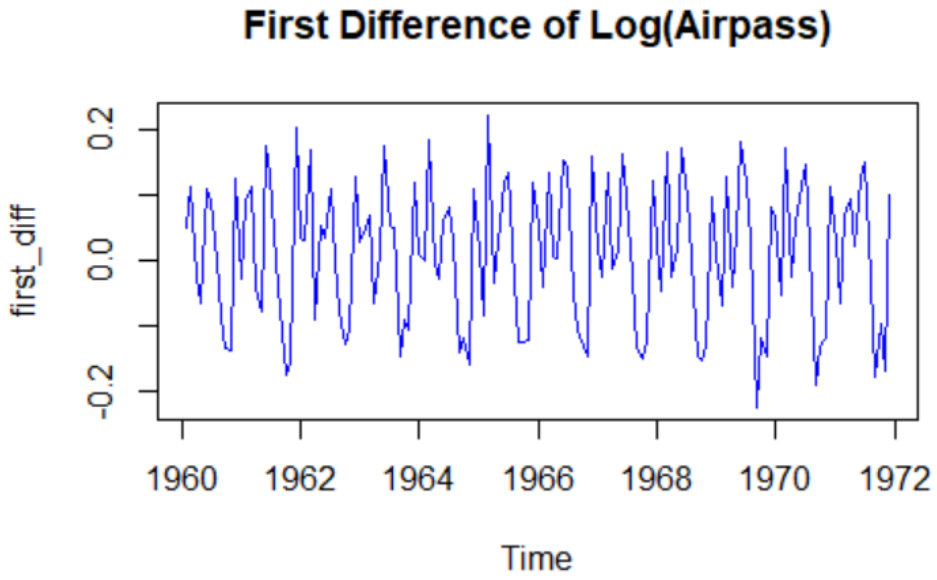
```
plot(airpass, main = 'Airpass', ylab = 'Number of Passengers',col='blue')
```



- Trend
- Seasonality
- Increasing variance

Diff() – Remove Trend

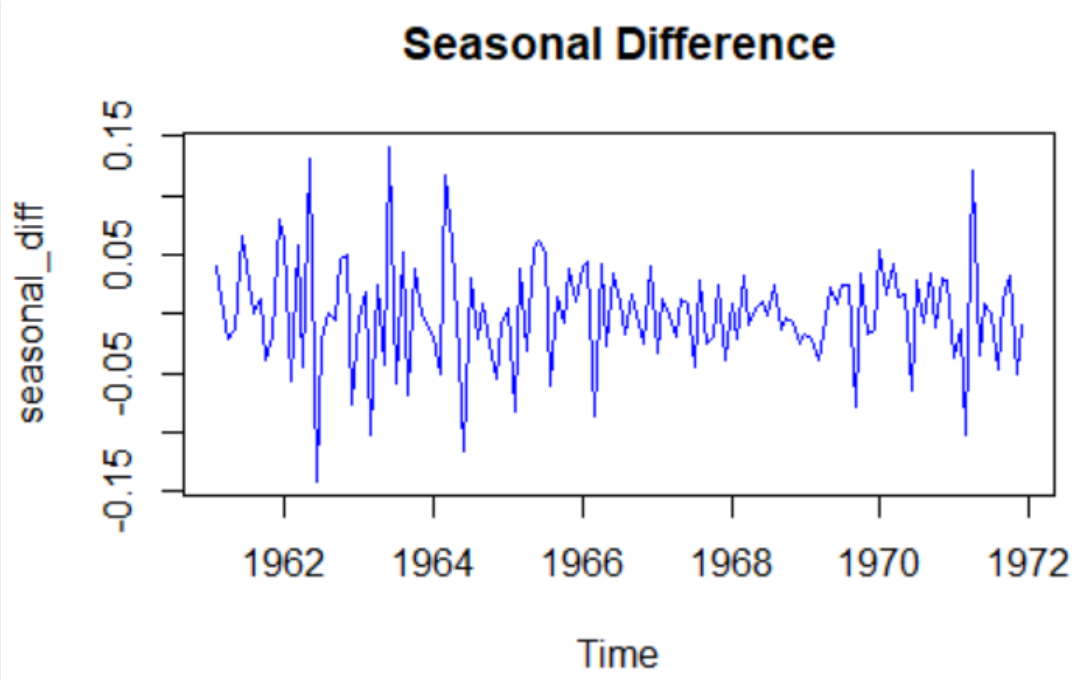
- diff(x, lag)
  - x: a numeric vector or matrix containing the values to be differenced
  - lag: an integer indicating which lag to use.
- Value: x[(1+lag):n] - x[1:(n-lag)]




```
first_diff <- diff(log(airpass))
plot(first_diff, main = 'First Difference of Log(Airpass)',col='blue')
```

Diff() – Remove Seasonality

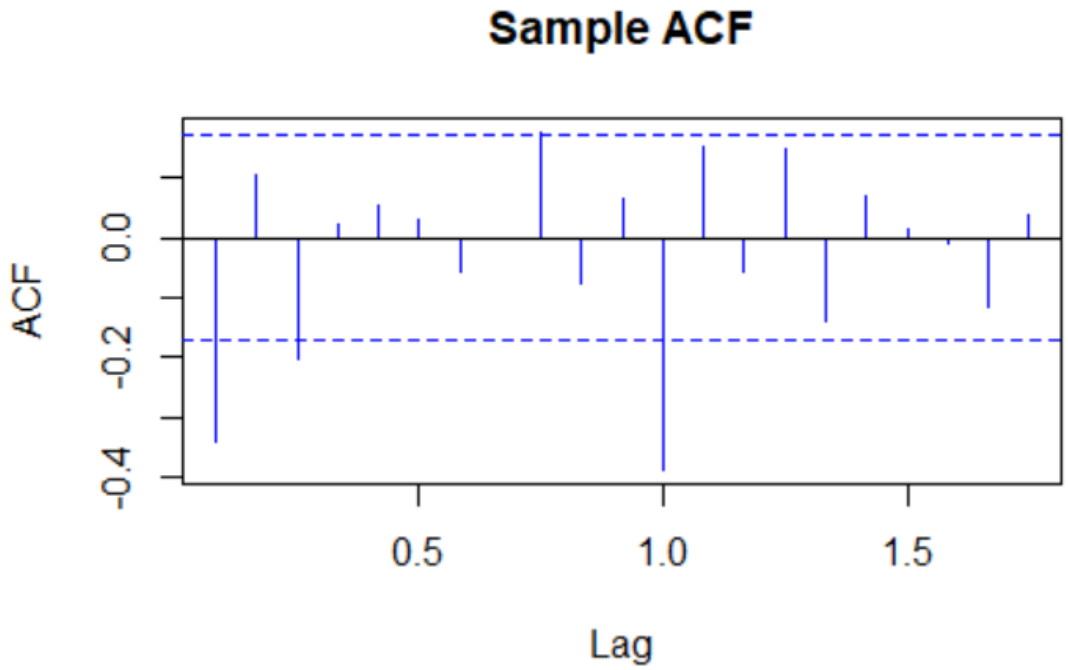
```
seasonal_diff <- diff(first_diff, lag = 12)
plot(seasonal_diff, main = 'Seasonal Difference', col='blue')
```



Sample ACF

- acf()
- Produces the sample autocorrelation function.

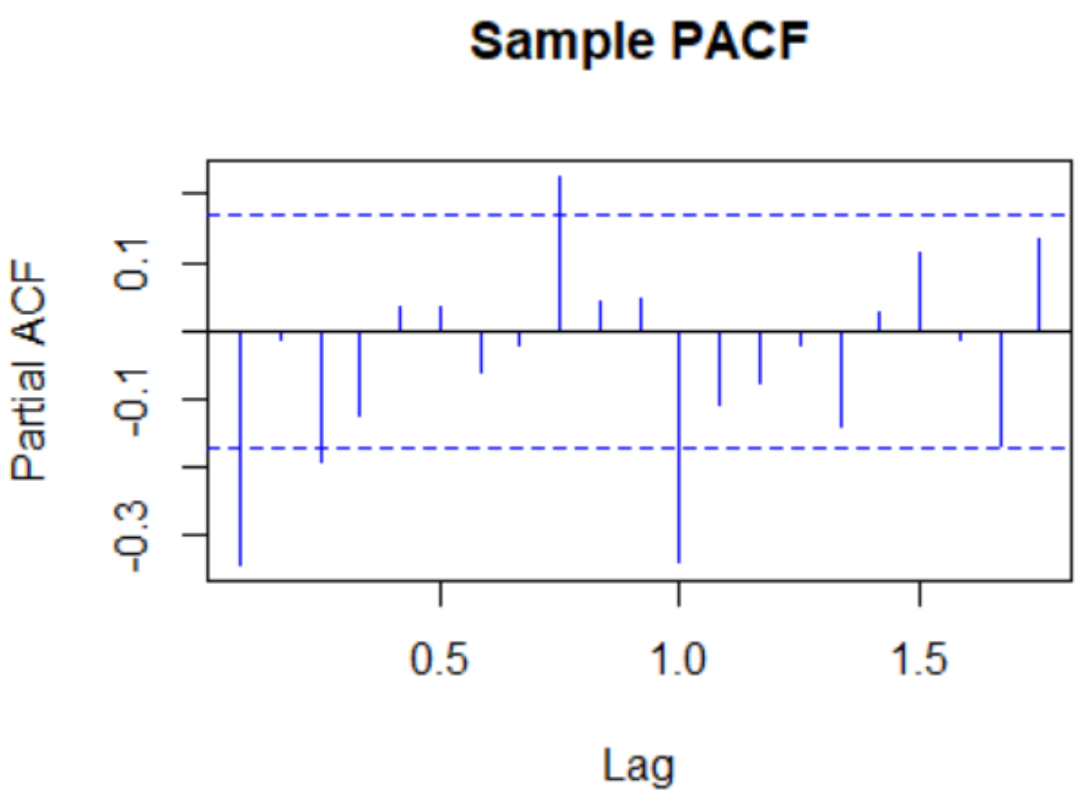
```
acf(seasonal_diff, main = 'Sample ACF', col='b
```



Sample PACF

- pacf()
- Produces the sample partial autocorrelation fi

```
pacf(seasonal_diff, main = 'Sample PA
```



Fit an ARIMA Model

- arima()
- Fits a SARIMA model of order (p,d,q)x(P,D,Q), with period s.

```
mod <- arima(log(airpass), order=c(0, 1, 1), seasonal=list(order=c(0, 1, 1), period=12))
```

```
Call:
arima(x = log(airpass), order = c(0, 1, 1), seasonal = list(order = c(0, 1,
    1), period = 12))

Coefficients:
          ma1     sma1
      -0.4018  -0.5569
s.e.   0.0896   0.0731

sigma^2 estimated as 0.001348:  log likelihood = 244.7,  aic = -485.4
```

Eacf():

- **Description**: Computes the sample extended acf (ESACF) for the time series. The matrix of ESACF with the AR order up to ar.max and the MA order up to ma.max is stored in the matrix EACFM.
- **Usage**: eacf(z, ar.max = 7, ma.max = 13)
- Pick the simplest suggested model
- For this data: arma(1,1)

```
> eacf(airpass)
AR/MA
  0 1 2 3 4 5 6 7 8 9 10 11 12 13
0 x x x x x x x x x x x  x  x  x
1 x o x x o o o x o o x  x  x  o
2 x o o x o o o x o o o  x  x  o
3 x o o x o o o x o o o  x  x  o
4 x x x x o o o o o o o  x  x  x
5 o x x x o o o o o o o  x  o  x
6 x x o x o o o o o o o  x  x  x
7 x x o x o o o o o o o  x  x  x
```

Other Functions

- **predict()**
  - Predicts n steps ahead, from any fitted model, including a time series fitted using the command arima
- **arima.sim()**
  - Simulates an ARIMA model of stated length of the order (p,d,q), with innovations having a stated variance
- **tsdiag()**
  - Produces 3 standard diagnostic charts for a fitted ARIMA model:
    - Plot of residuals from the model
    - Sample autocorrelation function of the residuals from the fitted model
    - Ljung-Box portmanteau statistic for stated maximum number of lags.
- **And more …**

Other Commonly Used Time Series Packages

- tseries
  - Functions to diagnose time series models with tests
- forecast
  - Functions to make forecast for time series models

dplyr package

Overview

dplyr

- Is a data manipulation package
- Provides a consistent set of verbs that solve many common data manipulation challenges

To install: install.packages(“dplyr”)

```r
library(dplyr)
starwars

# filter() picks cases based on their values
starwars %>%
filter(species == "Droid")
```

```
> # filter() picks cases based on their values
> starwars %>%
+ filter(species == "Droid")
# A tibble: 6 x 14
  name   height  mass hair_color skin_color  eye_color  birth_year sex   gender homeworld species films
  <chr>   <int> <dbl> <chr>      <chr>       <chr>           <dbl> <chr> <chr>  <chr>     <chr>   <lis>
1 C-3PO     167    75 NA         gold        yellow            112 none  mascu… Tatooine  Droid   <chr…
2 R2-D2      96    32 NA         white, bl…  red                33 none  mascu… Naboo     Droid   <chr…
3 R5-D4      97    32 NA         white, red  red                NA none  mascu… Tatooine  Droid   <chr…
4 IG-88     200   140 none       metal       red                15 none  mascu… NA        Droid   <chr…
5 R4-P…      96    NA none       silver, r…  red, blue          NA none  femin… NA        Droid   <chr…
6 BB8        NA    NA none       none        black              NA none  mascu… NA        Droid   <chr…
# … with 2 more variables: vehicles <list>, starships <list>
```

```r
# select() picks variables based on their name
starwars %>%
  select(name, ends_with("color"))
```

```
> # select() picks variables based on their name
> starwars %>%
+   select(name, ends_with("color"))
# A tibble: 87 x 4
   name               hair_color    skin_color  eye_color
   <chr>              <chr>         <chr>       <chr>
 1 Luke Skywalker     blond         fair        blue
 2 C-3PO              NA            gold        yellow
 3 R2-D2              NA            white, blue red
 4 Darth Vader        none          white       yellow
 5 Leia Organa        brown         light       brown
 6 Owen Lars          brown, grey   light       blue
 7 Beru Whitesun lars brown         light       blue
 8 R5-D4              NA            white, red  red
 9 Biggs Darklighter  black         light       brown
10 Obi-Wan Kenobi     auburn, white fair        blue-gray
# … with 77 more rows
```

```r
# mutate() adds new variables that are functions of existing variables
starwars %>%
  mutate(name, bmi = mass / ((height / 100)  ^ 2)) %>%
  select(name:mass, bmi)
```

```
> starwars %>%
+   mutate(name, bmi = mass / ((height / 100)  ^ 2)) %>%
+   select(name:mass, bmi)
# A tibble: 87 x 4
   name               height  mass   bmi
   <chr>               <int> <dbl> <dbl>
 1 Luke Skywalker        172    77  26.0
 2 C-3PO                 167    75  26.9
 3 R2-D2                  96    32  34.7
 4 Darth Vader           202   136  33.3
 5 Leia Organa           150    49  21.8
 6 Owen Lars             178   120  37.9
 7 Beru Whitesun lars    165    75  27.5
 8 R5-D4                  97    32  34.0
 9 Biggs Darklighter     183    84  25.1
10 Obi-Wan Kenobi        182    77  23.2
# … with 77 more rows
```

```r
# arrange() changes the ordering of the rows
starwars %>%
  arrange(desc(mass))
```

```
> # arrange() changes the ordering of the rows
> starwars %>%
+   arrange(desc(mass))
# A tibble: 87 x 14
   name   height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
 1 Jabb…     175  1358 NA         green-tan… orange           600  herm… mascu… Nal Hutta Hutt    <chr…
 2 Grie…     216   159 none       brown, wh… green, y…         NA  male  mascu… Kalee     Kaleesh <chr…
 3 IG-88     200   140 none       metal      red               15  none  mascu… NA         Droid   <chr…
 4 Dart…     202   136 none       white      yellow            41.9 male  mascu… Tatooine  Human   <chr…
 5 Tarf…     234   136 brown      brown      blue              NA  male  mascu… Kashyyyk  Wookiee <chr…
 6 Owen…     178   120 brown, gr… light      blue              52  male  mascu… Tatooine  Human   <chr…
 7 Bossk     190   113 none       green      red               53  male  mascu… Trandosha Trando… <chr…
 8 Chew…     228   112 brown      unknown    blue             200  male  mascu… Kashyyyk  Wookiee <chr…
 9 Jek …     180   110 brown      fair       blue              NA  male  mascu… Bestine … Human   <chr…
10 Dext…     198   102 none       brown      yellow            NA  male  mascu… Ojom      Besali… <chr…
# … with 77 more rows, and 2 more variables: vehicles <list>, starships <list>
```

```r
# group_by() allows to perform any operation by a group
# summarise() reduces multiple alues down to a single summary
starwars %>%
  group_by(species) %>%
  summarise(n = n(),mass = mean(mass, na.rm = TRUE)) %>%
  filter(n > 1,mass > 50)
```

```
> # summarise() reduces multiple alues down to a single summary
> starwars %>%
+   group_by(species) %>%
+   summarise(n = n(),mass = mean(mass, na.rm = TRUE)) %>%
+   filter(n > 1,mass > 50)
`summarise()` ungrouping output (override with `.groups` argument)
# A tibble: 8 x 3
  species      n  mass
  <chr>    <int> <dbl>
1 Droid        6  69.8
2 Gungan       3  74
3 Human       35  82.8
4 Kaminoan     2  88
5 Mirialan     2  53.1
6 Twi'lek      2  55
7 Wookiee      2 124
8 Zabrak       2  80
```

Data Transformation with dplyr : : CHEAT SHEET

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

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

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

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

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

summary function

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

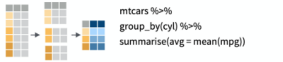
count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

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

Group Cases

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



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

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

Manipulate Cases

EXTRACT CASES

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

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

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

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

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

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

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

Logical and boolean operators to use with filter()

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

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

ARRANGE CASES

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

ADD CASES

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

Manipulate Variables

EXTRACT VARIABLES

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

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

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

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

contains(match)
ends_with(match)
matches(match)
num_range(prefix, range)
one_of(…)
starts_with(match)
:, e.g. mpg:cyl
-, e.g, -Species

MAKE NEW VARIABLES

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

vectorized function

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

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

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

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

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

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

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
irs %>% mutate(Species = case_when(Species == "versicolor" ~ "versi", Species == "virginica" ~ "virgi", TRUE ~ Species))

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2019-08

R+SQL=sqldf

By Guannan Liu

Sep. 2020

What’s sqldf?

This package is incredibly simple

- SQL is a database query language - a language designed specifically for interacting with a database. It offers syntax for extracting data, updating data, replacing data, creating data, etc. For our purposes, it will typically be used when accessing data off a server database. If the database isn’t too large, you can grab the entire data set and stick it in a data.frame.
- **sqldf is an R package for runing SQL statements on R data frames, optimized for convenience.**
  - The user simply specifies an SQL statement in R using data frame names in place of table names and a database with appropriate table layouts/schema is automatically created, the data frames are automatically loaded into the database, the specified SQL statement is performed, the result is read back into R and the database is deleted all automatically behind the scenes making the database's existence transparent to the user who only specifies the SQL statement.

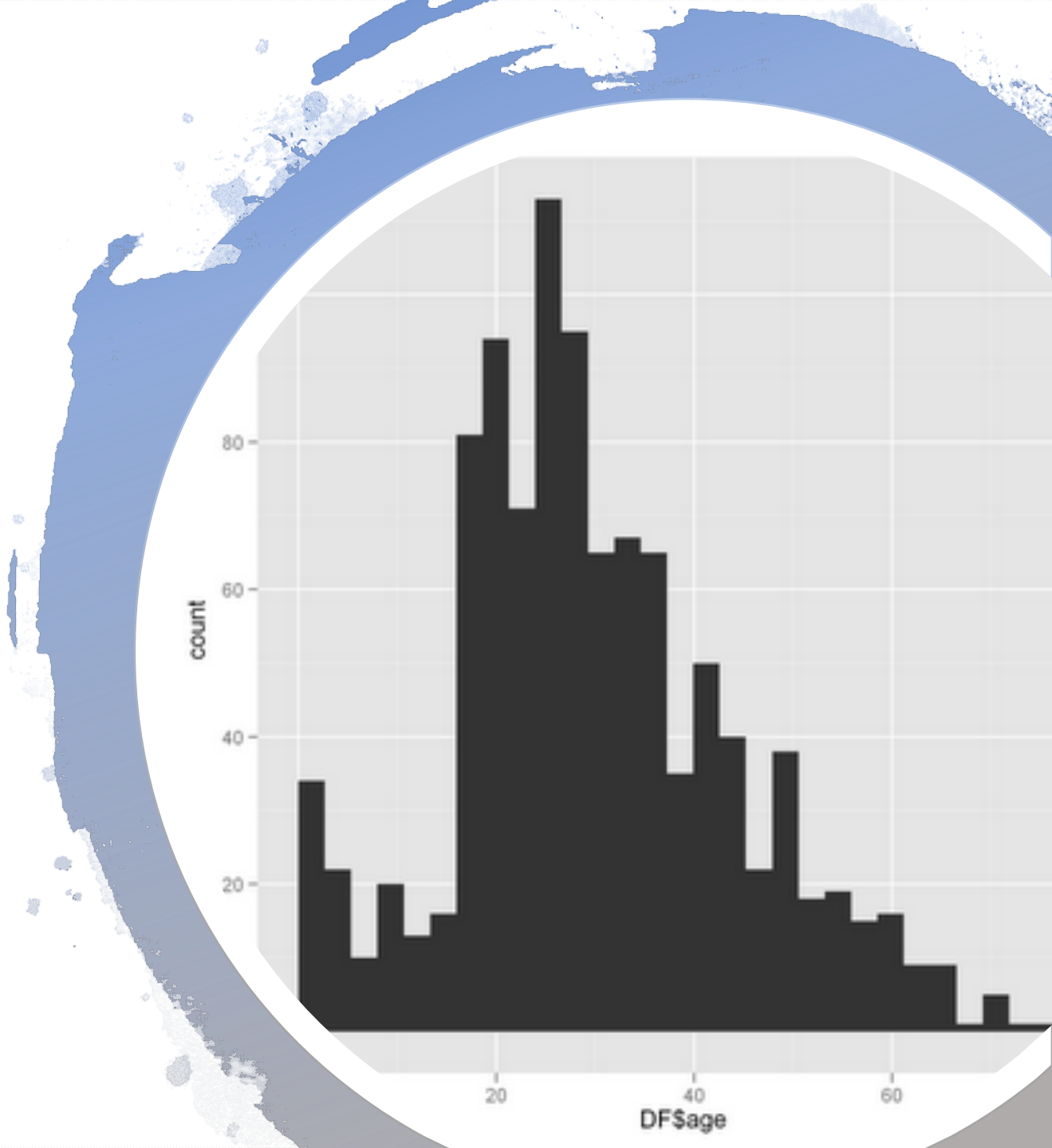
An example of Titanic dataset

```
# Load the package
library(sqldf)

# Use the titanic data set
```

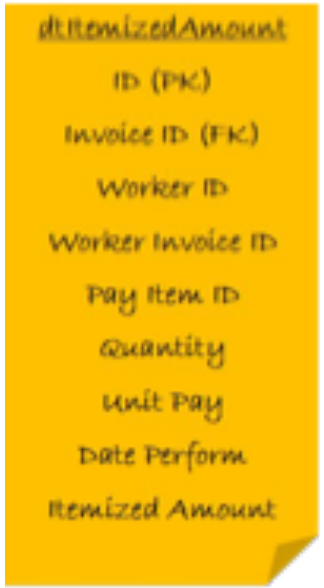
```
data(titanic3, package="PASWR")
```

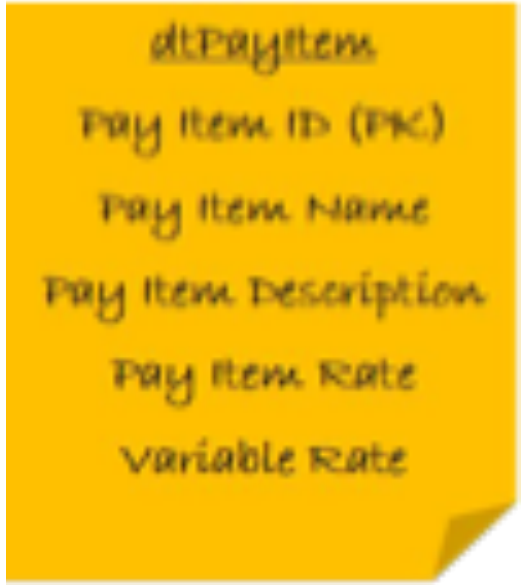
```
DF=sqldf('select age from titanic3 where age != "NA"')
qplot(DF$age,data=DF, geom="histogram")
```



How to know the list of

Generate subset of independent contractors that have submitted entertainment related charges ?





Function	Use
sqldf("...") SELECT ... FROM	sqldf("**SELECT** variable1, variable2 **FROM** dt")
sqldf("...") SELECT ... AS ... FROM	sqldf("SELECT variable1, variable2 **AS** var2 FROM dt")
sqldf("...") SELECT ... DISTINCT ... FROM	sqldf("SELECT **DISTINCT** variable1, variable2 FROM dt")
sqldf("...") ... WHERE	sqldf("SELECT variable1, variable2 FROM dt WHERE variable1**=numericValue**")
sqldf("...") ... WHERE	sqldf("SELECT variable1, variable2 FROM dt WHERE variable2**='textValue'**")
sqldf("...") ... WHERE ... AND	sqldf("SELECT variable1, variable2, variable3 FROM dt **WHERE** variable1>numericValue **AND** variable3<numericvalue ")
sqldf("...") ... WHERE ... OR	sqldf("SELECT variable1, variable2 FROM dt **WHERE** variable1>numericValue **OR** variable2='textValue'")
sqldf("...") ... WHERE ... BETWEEN	sqldf("SELECT variable1, variable2 FROM dt **WHERE** variable1 **BETWEEN** numericValue1 **AND** numericValue2")
sqldf("...") ... WHERE ... IS NULL or IS NOT NULL	sqldf("SELECT variable1, variable2 FROM dt **WHERE** variable2 **IS NULL**")
sqldf("...") CASE WHEN ... THEN ... WHEN ... THEN ... ELSE END AS variableName	sqldf("SELECT variable1, variable2, **CASE** **WHEN** variable1 > 65 **THEN** 'senior' **WHEN** variable1 < 18 **THEN** 'young' **ELSE** 'adult' **END AS** categoricalAge FROM dt")

Function	Use	Description
sqldf("...") SUM ... GROUP BY	sqldf("SELECT x1, sum(y1) AS sum_of_y1 FROM dt GROUP BY x1")	Create an aggregate variable - for example sum, count, min, max, average - by grouping observations (GROUP BY) based on the values of another variable (typically a categorical variable). In the example shown we take the SUM of y1 rename it AS sum_of_y1 for each group of the x1. The output will have as many values as the number of groups in x1.
sqldf("...") COUNT ... GROUP BY	sqldf("SELECT x1, count(y1) AS count_of_y1 FROM dt GROUP BY x1")	In the example shown we find the COUNT of y1 rename it AS count_of_y1 for each group of the x1. The output will have as many values as the number of groups in x1.
sqldf("...") MEAN ... GROUP BY	sqldf("SELECT x1, mean(y1) AS mean_of_y1 FROM dt GROUP BY x1")	In the example shown we find the MEAN of y1 rename it AS mean_of_y1 for each group of the x1. The output will have as many values as the number of groups in x1.
sqldf("...") SUM ... GROUP BY ... HAVING	sqldf("SELECT x1, sum(y1) AS sum_of_y1 FROM dt GROUP BY x1 HAVING sum(y1)> someValue")	Create an aggregate variable by grouping observations based on the values of a variable and limit output to groups meeting (HAVING) a condition. For example, we can limit the results to only groups that have a sum which is greater than 0, i.e., ... HAVING sum(y1)> 0.
sqldf("...") WHERE ... GROUP BY ... HAVING	sqldf("SELECT x1, x2, sum(y1) AS sum_of_y1 FROM dt WHERE x2 > someValue GROUP BY x1 HAVING sum(y1)> someOtherValue")	Limit the data set to only WHERE a variable meets some condition, and create an aggregate variable by grouping observations based on the values of another variable. Limit the final output to groups meeting (HAVING) an aggregate condition. For example, we limit the data set to observations WHERE the variable x2 > 100, we generate sum for y1 GROUPED BY x2. In the end we retain only groups HAVING sum(y1) greater than 0.

Detailed document:

https://cran.r-project.org/web/packages/sqldf/sqldf.pdf

GGPlot2-
Grammar of Graphics

Grammar of Graphics

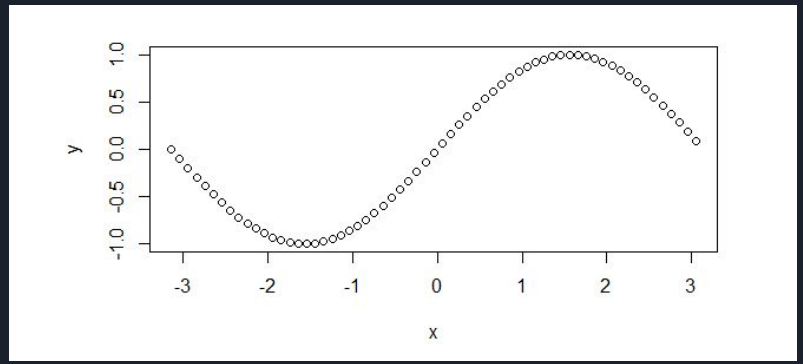
- Book by Leiland Wilkinson
- Considered one of the seminal texts of Data Science
- Organized types of graphics and how they're related.
  - Mentions that Pie charts are just bar charts on a polar axis
- Chart mentality is bad
- Graphics are represented by their parts (layers)
- Uses principles of OOP to build graphics
- Disclaimer: I did not read the book yet, but I plan to

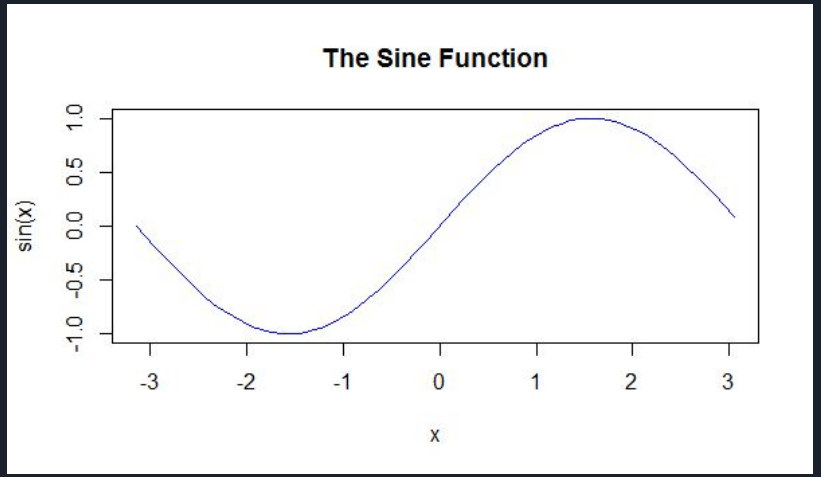
Grammar of Graphics - Layers

- Variables: The data you're representing
- Geom: the visuals and aesthetics of the graphic
- Coordinate system: Polar, cartesian, fixed, flipped, maps
- Theme: Aesthetic stuff
- Zooming: How does it look when you zom
- Faceting: How the grids are divided
- Legends: typical chart stuff
- Labels: more typical chart stuff
- Stat: transformations of the data

Plots without ggplot - plot()

- Base R already has functions
- Very basic, not very developed
- Not custom





Using GGplot2

- Two ways to plot
  - qplot() - quick plot function. Lots of good defaults. 99% of your work will be done using this
  - ggplot() - no defaults have to add in each layer more custom.
- Powerful conditional control. If we think of things based off of the layers.
- Able to change the geom based on the value of data
  - Size of dot, color of dot, type of dot
-

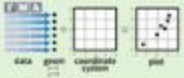
Data Visualization with ggplot2

Cheat Sheet

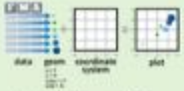
RStudio

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

Learn more at **docs.ggplot2.org** • ggplot2 0.9.3.1 • Updated: 3/15

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 mapping | 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.

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 | Continuous

Shape scales

Size scales

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.)

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))

Themes

+ theme_bw()
White background with grid lines

+ theme_grey()
Grey background (default theme)

+ theme_classic()
White background no gridlines

+ theme_minimal()
Minimal theme

ggthemes - Package with additional ggplot2 themes

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

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.

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)**)**

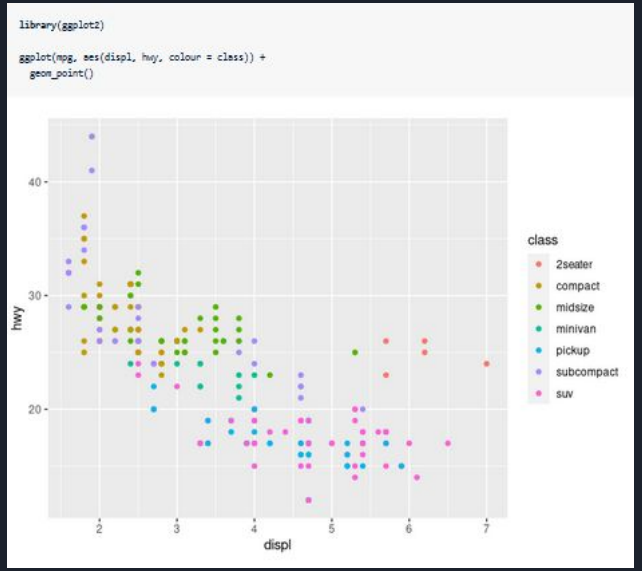
qplot()- Quick Plot

Arguments

- X,y,... = aesthetics that you pass in
- Data = data frame to use (optional)
- Facets = faceting formulat ot use
- Margins = marginal facets
- Geom = different shapes for your graphic. Defaults to point and histogram
- Xlim,ylim = x and y axis limits
- Log = log transforms

https://www.rdocumentation.org/packages/ggplot2/versions/3.3.2/topics/qplot

ggplot()- Grammar of graphics



Creates a blank graphic.

Then add on layers using +

You would decide the data with aes and go from there

Documentation and Tutorials

https://www.rdocumentation.org/packages/ggplot2/versions/3.3.2

http://r-statistics.co/ggplot2-Tutorial-With-R.html

Forcats

Joshua Anderson

Imdb_movies Dataset

```
Rows: 4,499
Columns: 31
$ title                    <chr> "AvatarÂ ", "Pirates of the Caribbean: At World...
$ year                     <int> 2009, 2007, 2015, 2012, 2012, 2007, 2010, 2015,...
$ country                  <chr> "USA", "USA", "UK", "USA", "USA", "USA", "USA",...
$ director                 <chr> "James Cameron", "Gore Verbinski", "Sam Mendes"...
$ budgetM                  <dbl> 237.0, 300.0, 245.0, 250.0, 263.7, 258.0, 260.0...
$ grossM                   <dbl> 760.50585, 309.40415, 200.07417, 448.13064, 73....
$ imdb_score               <dbl> 7.9, 7.1, 6.8, 8.5, 6.6, 6.2, 7.8, 7.5, 7.5, 6....
$ color                    <chr> "Color", "Color", "Color", "Color", "Color", "C...
$ num_critic_for_reviews   <int> 723, 302, 602, 813, 462, 392, 324, 635, 375, 67...
$ duration                 <int> 178, 169, 148, 164, 132, 156, 100, 141, 153, 18...
$ director_facebook_likes  <int> 0, 563, 0, 22000, 475, 0, 15, 0, 282, 0, 0, 395...
$ actor_3_facebook_likes   <int> 855, 1000, 161, 23000, 530, 4000, 284, 19000, 1...
$ actor_2_name             <chr> "Joel David Moore", "Orlando Bloom", "Rory Kinn...
$ actor_1_facebook_likes   <int> 1000, 40000, 11000, 27000, 640, 24000, 799, 260...
$ gross                    <int> 760505847, 309404152, 200074175, 448130642, 730...
$ genres                   <chr> "Action|Adventure|Fantasy|Sci-Fi", "Action|Adve...
$ actor_1_name             <chr> "CCH Pounder", "Johnny Depp", "Christoph Waltz"...
$ num_voted_users          <int> 886204, 471220, 275868, 1144337, 212204, 383056...
$ cast_total_facebook_likes <int> 4834, 48350, 11700, 106759, 1873, 46055, 2036, ...
$ actor_3_name             <chr> "Wes Studi", "Jack Davenport", "Stephanie Sigma...
$ facenumber_in_poster     <int> 0, 0, 1, 0, 1, 0, 1, 4, 3, 0, 0, 1, 2, 1, 0, 4,...
$ plot_keywords            <chr> "avatar|future|marine|native|paraplegic", "godd...
$ movie_imdb_link          <chr> "http://www.imdb.com/title/tt0499549/?ref_=fn_t...
$ num_user_for_reviews     <int> 3054, 1238, 994, 2701, 738, 1902, 387, 1117, 97...
$ language                 <chr> "English", "English", "English", "English", "En...
$ content_rating           <chr> "PG-13", "PG-13", "PG-13", "PG-13", "PG-13", "P...
$ budget                   <dbl> 237000000, 300000000, 245000000, 250000000, 263...
$ actor_2_facebook_likes   <int> 936, 5000, 393, 23000, 632, 11000, 553, 21000, ...
$ aspect_ratio             <dbl> 1.78, 2.35, 2.35, 2.35, 2.35, 2.35, 1.85, 2.35,...
$ movie_facebook_likes     <int> 33000, 0, 85000, 164000, 24000, 0, 29000, 11800...
$ profitM                  <dbl> 523.505847, 9.404152, -44.925825, 198.130642, -...
```

fct_lump

```
levels(movies_clean$country)
[1] ""                   "Afghanistan"        "Argentina"
[4] "Aruba"              "Australia"          "Bahamas"
[7] "Belgium"            "Brazil"             "Bulgaria"
10] "Canada"             "Chile"              "China"
13] "Colombia"           "Czech Republic"     "Denmark"
16] "Dominican Republic" "Egypt"              "Finland"
19] "France"             "Georgia"            "Germany"
22] "Greece"             "Hong Kong"          "Hungary"
25] "Iceland"            "India"              "Indonesia"
28] "Iran"               "Ireland"            "Israel"
31] "Italy"              "Japan"              "Kenya"
34] "Kyrgyzstan"         "Libya"              "Mexico"
37] "Netherlands"        "New Line"           "New Zealand"
40] "Nigeria"            "Norway"             "Official site"
43] "Pakistan"           "Panama"             "Peru"
46] "Philippines"        "Poland"             "Romania"
49] "Russia"             "Slovenia"           "South Africa"
52] "South Korea"        "Soviet Union"       "Spain"
55] "Sweden"             "Switzerland"        "Taiwan"
58] "Thailand"           "Turkey"             "UK"
61] "United Arab Emirates" "USA"              "West Germany"
```

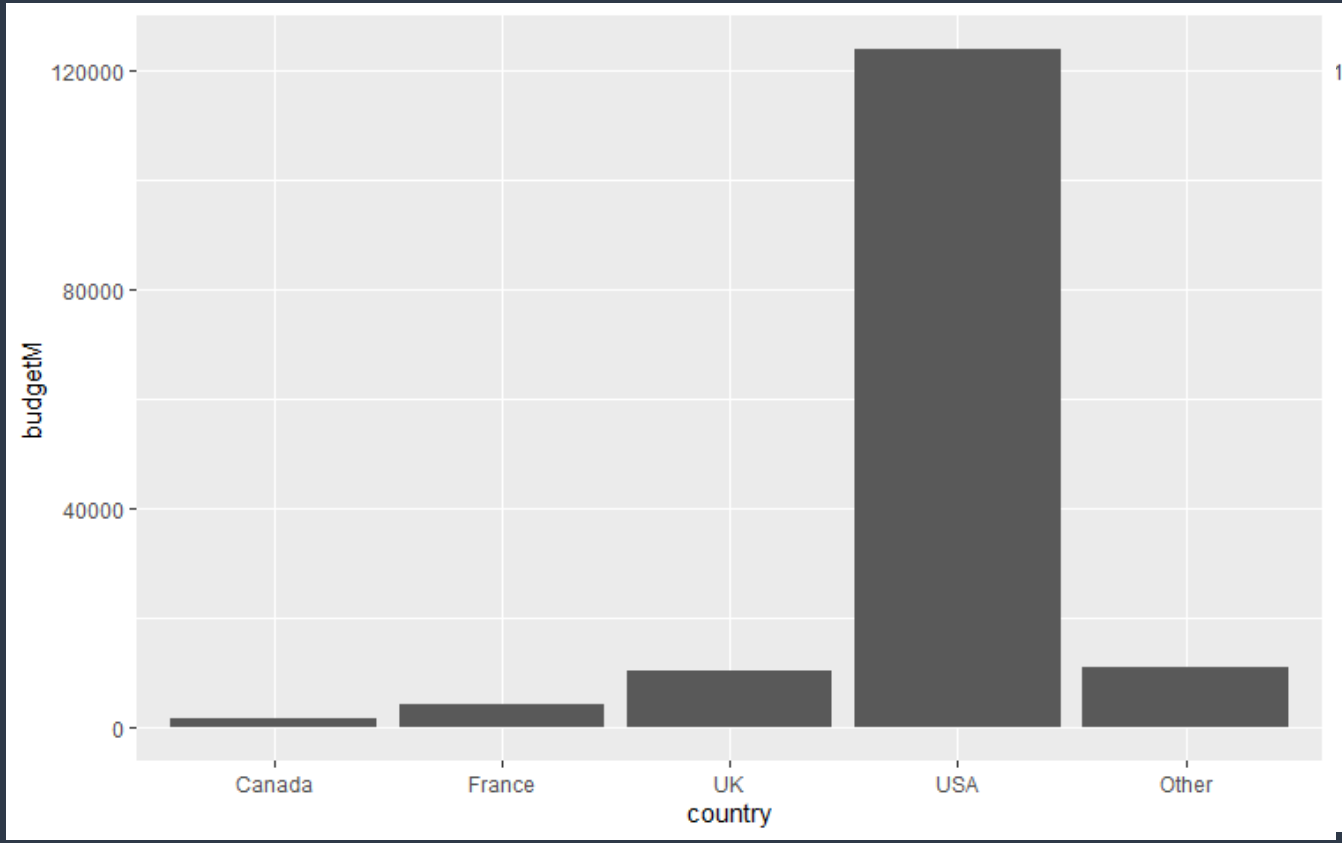
```
levels(fct_lump(movies_clean$country, n = 4))
1] "Canada" "France" "UK"     "USA"    "Other"
```

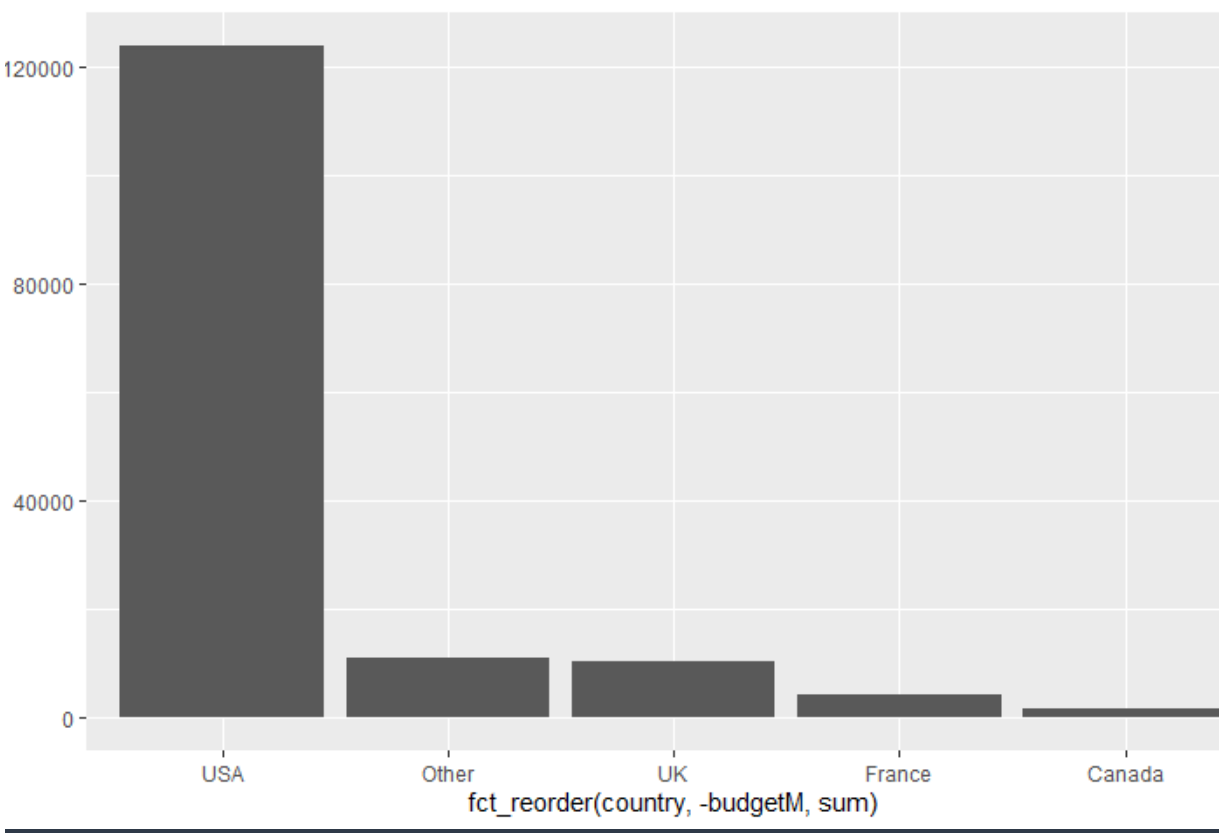
fct_lump cont.

- fct_lump_min() – lump together factors that appear less than minimum specified number of times
- fct_lump_lowfreq() – lump together the least common factors
- fct_lump_n() – lump all outside of the n most common factors

fct_reorder

```
ggplot(movies_clean, aes(fct_reorder(country, -budgetM, sum), budgetM)) +
  geom_bar(stat="identity")
```





fct_count

```
> movies_clean$content_rating %>% fct_count()
# A tibble: 17 x 2
   f               n
   <fct>       <int>
 1 ""            204
 2 "Approved"     50
 3 "G"           107
 4 "GP"            4
 5 "M"             4
 6 "NC-17"         7
 7 "Not Rated"    89
 8 "Passed"        9
 9 "PG"          654
10 "PG-13"      1362
11 "R"          1934
12 "TV-14"         5
13 "TV-G"          5
14 "TV-MA"         1
15 "TV-PG"         3
16 "Unrated"      48
17 "X"            13
```

fct_recode

```
> movies_clean %<>% mutate(country = fct_recode(country,
+   "United States of America" = "USA"
+ ))
> movies_clean$country %>% levels()
[1] "United States of America" "Canada"                    "France"
[4] "UK"                       "Other"
```

The “random” package:

- Generates random numbers for your use, similar to features found in Python, Lua, etc.
- Doesn’t actually generate pseudo-random numbers on your machine.
  - This package in fact connects to www.random.org and uses their service to generate them for you.
  - The numbers generated in that fashion are, in fact, truly random.
- Per the description of the site:
  - The `http://random.org` services uses atmospheric noise sample via a radio tuned to an unused broadcast frequency together with a skew correction originally due to John von Neumann.

All functions return matrices as follows (repeated parameters omitted):

- `randomNumbers(n:int, min:int, max:int, col:int, base:10, check:bool)` generates a random integer between min and max with the given radix and the number of columns you want. If “check” is set to “true”, it will make sure the site will actually allow you to use the service to prevent abuse.
- `randomSequence(…)` generates a sequence of random numbers where “col” is the length of the sequence.
- `randomStrings(len:int, digits:bool, upperalpha:bool,loweralpha:bool, unique:bool)` generates a random string with the parameters indicating what can be included.