

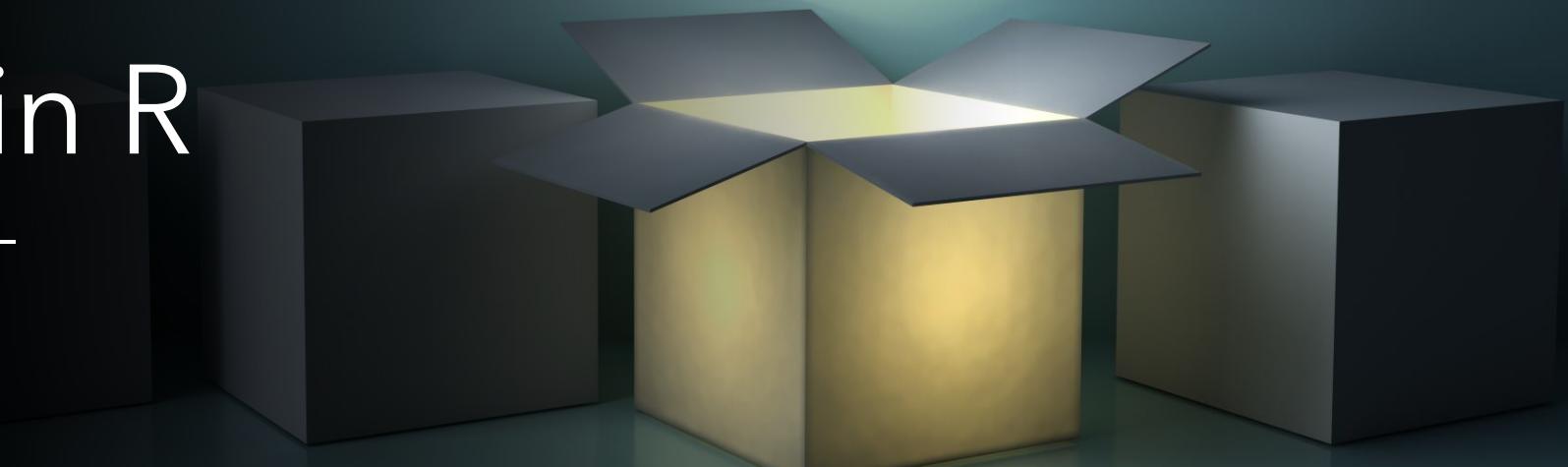


TSA Package in R

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:

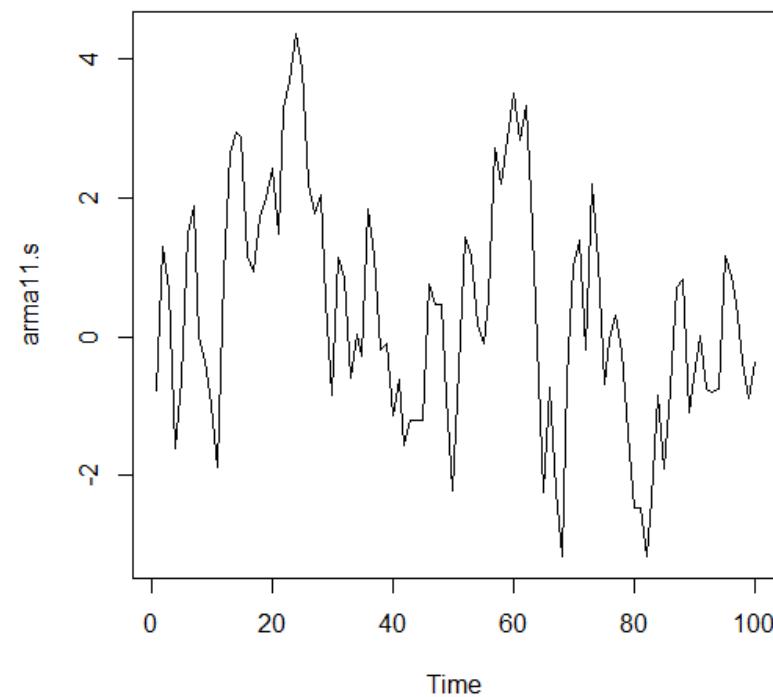
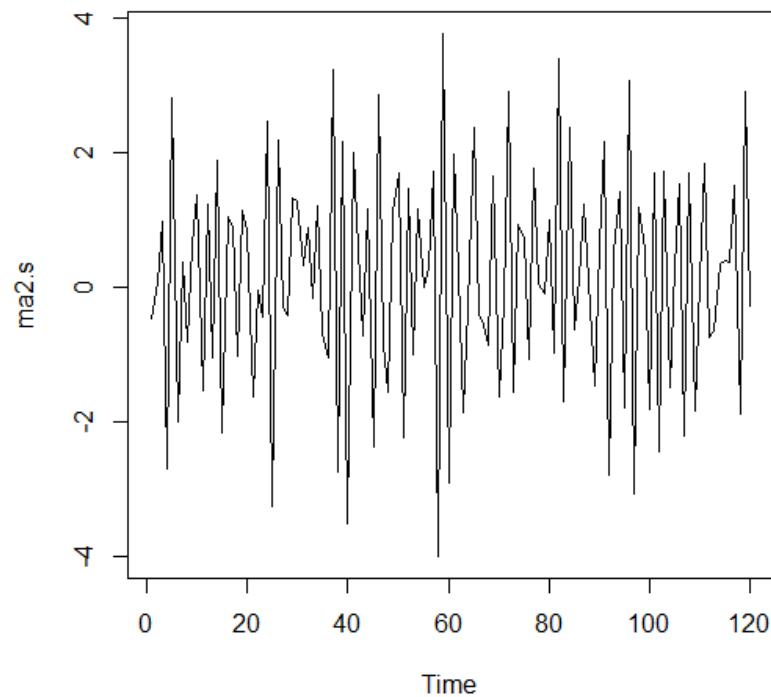
There are lots of time series data available with this package which could be used to do some analysis and test some functions. For example:

- **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:

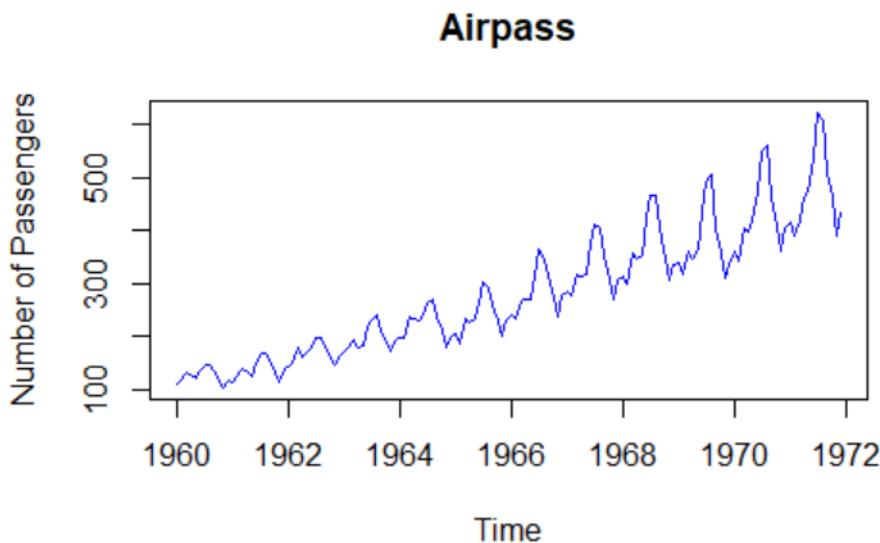
- **ar1.2.s** : A simulated AR(1) series with the AR coefficient equal to 0.4. The model is $Y(t)=0.4*Y(t-1)+e(t)$ where the e's are iid standard normal.
- **ar1.s** : A simulated AR(1) series with the AR coefficient equal to 0.9. The model is $Y(t)=0.9*Y(t-1)+e(t)$ where the e's are iid standard normal.
- **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 normal.
- **ma2.s** : A simulated MA(2) series with MA coefficients being 1 and -0.6. The model is $Y(t) = e(t)-e(-1)+0.6e(t-2)$ where the e's are iid standard normal random variables.

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



EXAMPLE

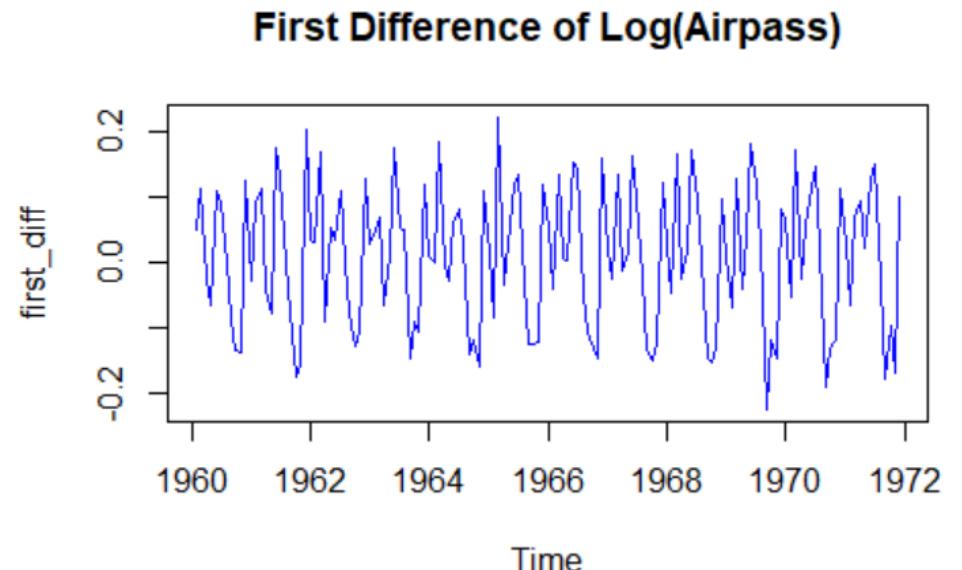
- The monthly airline passenger time series, first investigated in Box and Jenkins (1976), is considered a classic time series. The data are in the file named airpass.
 - Load the data: `data(airpass)`
 - Plot the data: `plot(airpass, main = 'Airpass', ylab = 'Number of Passengers', col='blue')`



- Trend
- Seasonality
- Increasing variance

Diff() - Remove Trend

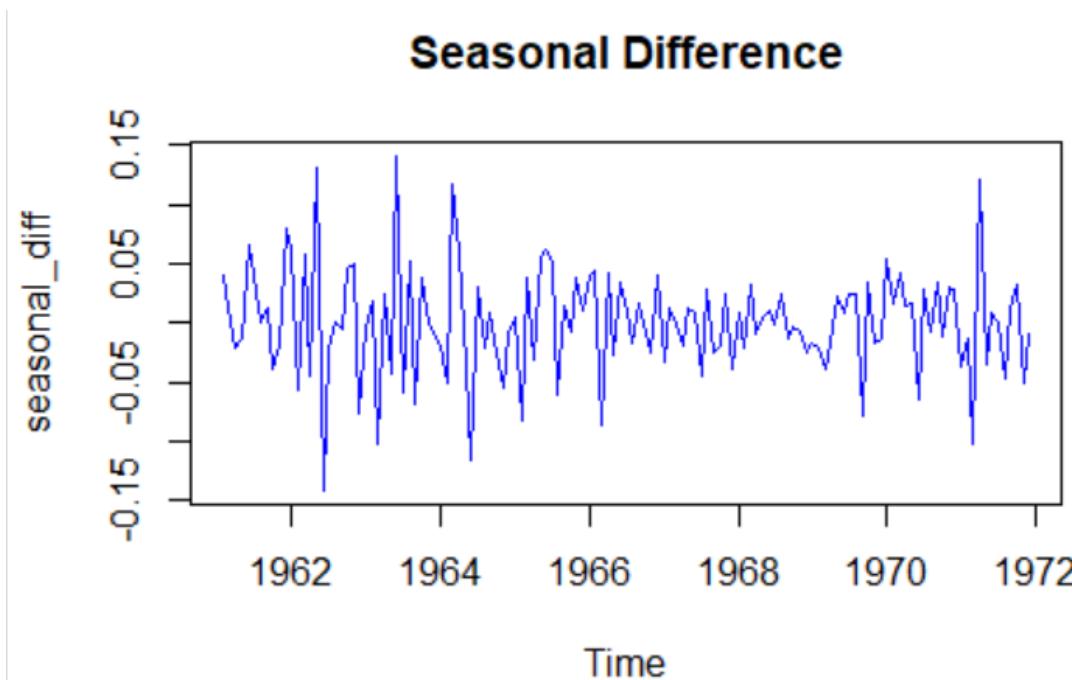
- `diff(x, lag)`
 - `x`: a numeric vector or matrix containing the values to be differenced
 - `lag`: an integer indicating which lag to use.
- Value: $x[(1+lag):n] - x[1:(n-lag)]$



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

Diff() - Remove Seasonality

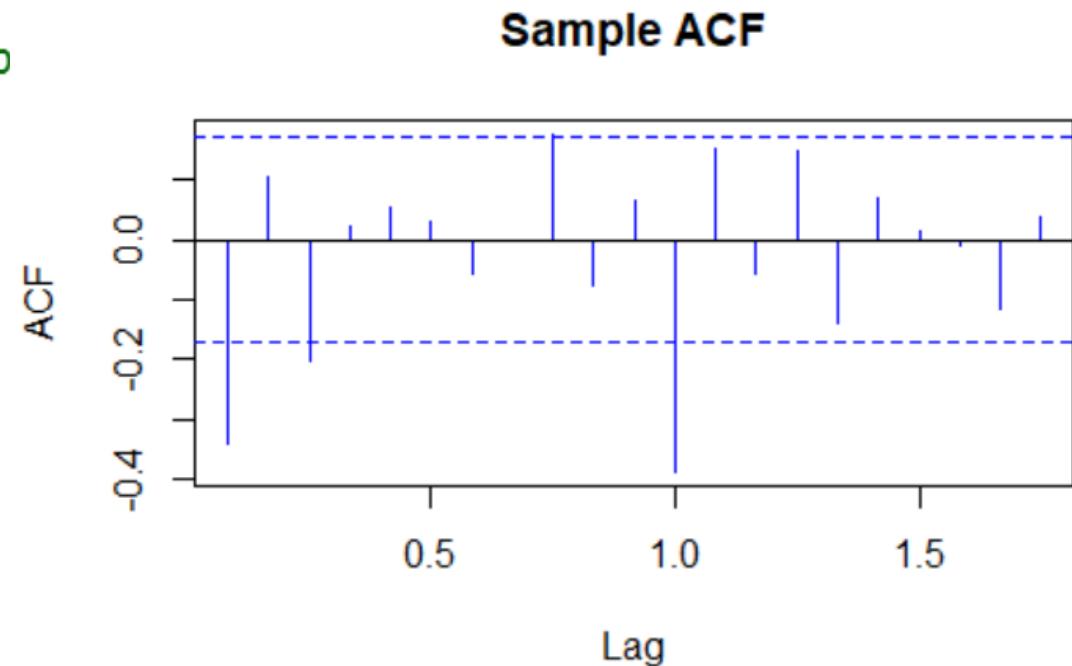
```
seasonal_diff <- diff(first_diff, lag = 12)
plot(seasonal_diff, main = 'Seasonal Difference', col='blue')
```



Sample ACF

- `acf()`
- Produces the sample autocorrelation function.

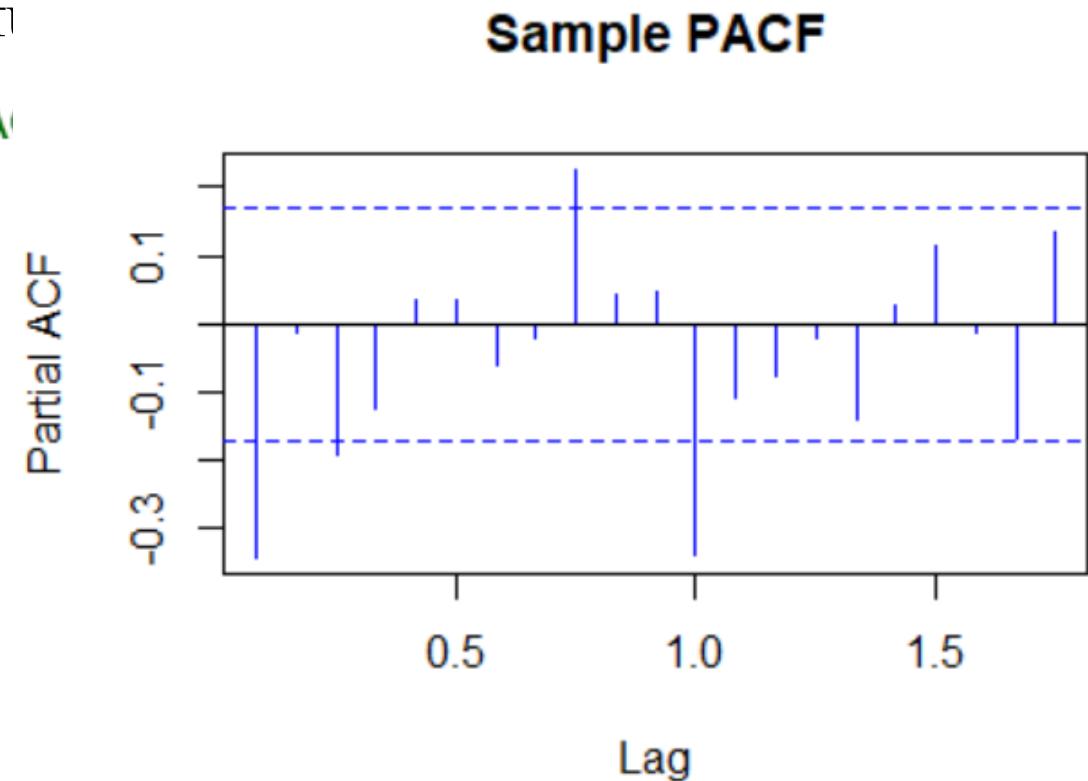
```
acf(seasonal_diff, main = 'Sample ACF', col='b'
```



Sample PACF

- pacf()
- Produces the sample partial autocorrelation function

```
pacf(seasonal_diff, main = 'Sample PACF')
```



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
1 x o x x o o o x o o 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")`

```
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>   <list>
1 C-3PO     167     75 NA          gold        yellow       112 none  masculin... Tatooine Droid   <chr...
2 R2-D2      96      32 NA          white, bl... red           33 none  masculin... Naboo   Droid   <chr...
3 R5-D4     97      32 NA          white, red red           NA none  masculin... Tatooine Droid   <chr...
4 IG-88     200     140 none        metal        red           15 none  masculin... NA      Droid   <chr...
5 R4-P...     96      NA none        silver, r... red, blue     NA none  feminin... NA      Droid   <chr...
6 BB8       NA      NA none        none         black          NA none  masculin... NA      Droid   <chr...
# ... with 2 more variables: vehicles <list>, starships <list>
```

```
# 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
```

```
# 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
```

```
# 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> <chr>
1 Jabb...     175  1358 NA          green-tan... orange       600  herm... mascul... Nal Hutta Hutt    <chr...
2 Grie...     216   159 none        brown, wh... green, y... NA          male  mascul... Kalee   Kaleesh <chr...
3 IG-88      200   140 none        metal       red           15   none  mascul... NA        Droid    <chr...
4 Dart...     202   136 none        white      yellow        41.9 male  mascul... Tatooine Human   <chr...
5 Tarf...     234   136 brown       brown       blue          NA          male  mascul... Kashyyyk Wookiee <chr...
6 Owen...     178   120 brown, gr... light       blue          52   male  mascul... Tatooine Human   <chr...
7 Bossk       190   113 none        green       red           53   male  mascul... Trandosha Trando... <chr...
8 Chew...     228   112 brown       unknown     blue          200  male  mascul... Kashyyyk Wookiee <chr...
9 Jek ...     180   110 brown       fair        blue          NA          male  mascul... Bestine ... Human   <chr...
10 Dext...     198   102 none        brown      yellow        NA          male  mascul... Ojom     Besali... <chr...
# ... with 77 more rows, and 2 more variables: vehicles <list>, starships <list>
|
```

```
# group_by() allows to perform any operation by a group
# summarise() reduces multiple values 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 values 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
```


R+SQL=sqldf

By Guannan Liu

Sep. 2020

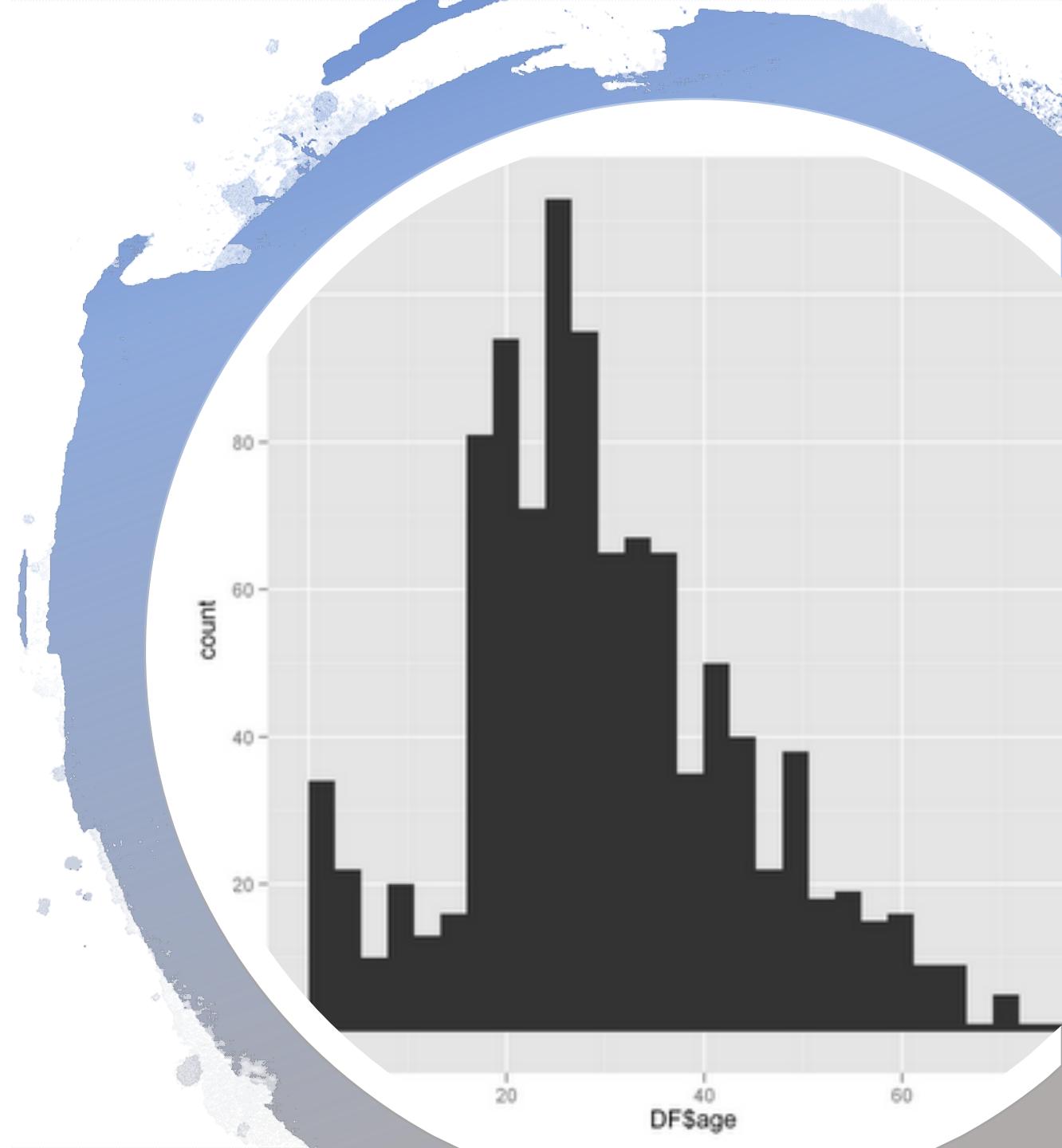
What's sqldf?

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 running 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 ?

dtJob
Work Code (PK)
Role
Status

dtWorker
Worker ID (PK)
Work Code (FK)
Hire Date
End Date
City
State

dtItemizedAmount
ID (PK)
Invoice ID (FK)
Worker ID
Worker Invoice ID
Pay Item ID
Quantity
Unit Pay
Date Perform
Itemized Amount

dtPamt
Direct Deposit Request ID (PK)
Invoice ID (FK)
Worker ID
Worker Invoice ID
Payment Date
Payment Amount

dtPayitem
Pay Item ID (PK)
Pay Item Name
Pay Item Description
Pay Item Rate
Variable Rate

dtInvoice
Invoice ID (PK)
Worker Invoice ID (Alternate PK)
WorkerID (FK)
Worker Invoice Date

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

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

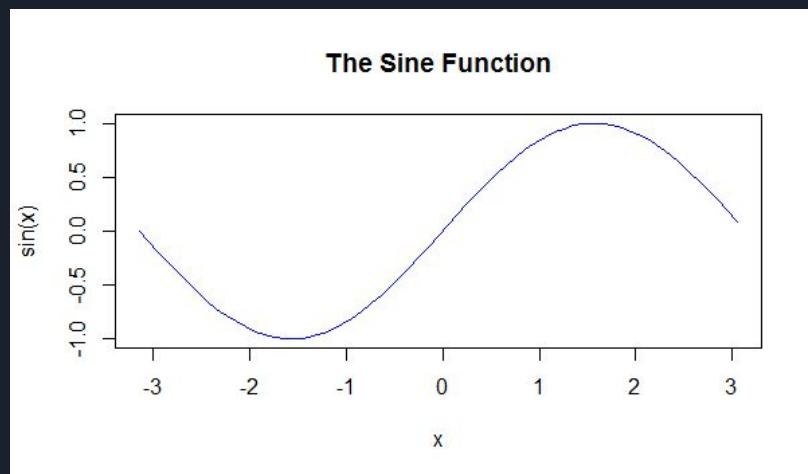
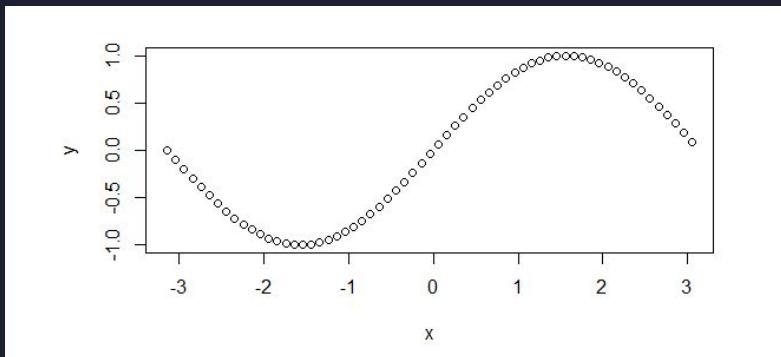


Grammar of Graphics - Layers

- 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 zoom
- 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

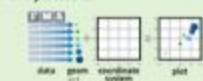


Basics

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 **shape** locations.



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

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

`ggplot(data = mpg, aes(x = cyl, y = hwy))`

Begins a plot that you finish by adding layers to. No defaults, but provides more control than `qplot()`.

```
ggplot(mpg, aes(hwy, cyl)) +  
  geom_point(aes(color = cyl)) +  
  geom_smooth(method = "lm") +  
  coord_cartesian(xlim = c(0, 8),  
    ylim = c(10, 50)) +  
  scale_color_gradient() +  
  theme_bw()  
+ 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 5x5' 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")
```

```
a + geom_density(alpha = .2, stat = "bin")
```

```
a + geom_hex(stat = "density")
```

```
a + geom_dotplot()
```

```
a + geom_freqpoly()
```

```
a + geom_histogram()
```

```
a + geom_hex(stat = "density")
```

```
a + geom_rug()
```

```
a + geom_smooth()
```

```
a + geom_text(label = cyl)
```

```
b <- ggplot(mpg, aes(fct))
```

```
b + geom_bar()
```

```
c <- ggplot(mpg, aes(class, hwy))
```

```
c + geom_bar(stat = "identity")
```

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

```
d + geom_path()
```

```
d + geom_rect()
```

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

```
e + geom_violin()
```

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

```
f + geom_bar(stat = "identity")
```

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

```
g + geom_boxplot()
```

```
h <- ggplot(mpg, aes(class, hwy))
```

```
h + geom_dotplot()
```

```
i <- ggplot(mpg, aes(class, hwy))
```

```
i + geom_hex()
```

```
j <- ggplot(mpg, aes(class, hwy))
```

```
j + geom_jitter()
```

```
k <- ggplot(mpg, aes(class, hwy))
```

```
k + geom_line()
```

```
l <- ggplot(mpg, aes(class, hwy))
```

```
l + geom_point()
```

```
m <- ggplot(mpg, aes(class, hwy))
```

```
m + geom_rect()
```

```
n <- ggplot(mpg, aes(class, hwy))
```

```
n + geom_raster()
```

```
o <- ggplot(mpg, aes(class, hwy))
```

```
o + geom_hex()
```

```
p <- ggplot(mpg, aes(class, hwy))
```

```
p + geom_jitter()
```

```
q <- ggplot(mpg, aes(class, hwy))
```

```
q + geom_line()
```

```
r <- ggplot(mpg, aes(class, hwy))
```

```
r + geom_point()
```

```
s <- ggplot(mpg, aes(class, hwy))
```

```
s + geom_rect()
```

```
t <- ggplot(mpg, aes(class, hwy))
```

```
t + geom_raster()
```

```
u <- ggplot(mpg, aes(class, hwy))
```

```
u + geom_hex()
```

```
v <- ggplot(mpg, aes(class, hwy))
```

```
v + geom_jitter()
```

```
w <- ggplot(mpg, aes(class, hwy))
```

```
w + geom_line()
```

```
x <- ggplot(mpg, aes(class, hwy))
```

```
x + geom_point()
```

```
y <- ggplot(mpg, aes(class, hwy))
```

```
y + geom_rect()
```

```
z <- ggplot(mpg, aes(class, hwy))
```

```
z + geom_raster()
```

```
aa <- ggplot(mpg, aes(class, hwy))
```

```
aa + geom_hex()
```

```
bb <- ggplot(mpg, aes(class, hwy))
```

```
bb + geom_jitter()
```

```
cc <- ggplot(mpg, aes(class, hwy))
```

```
cc + geom_line()
```

```
dd <- ggplot(mpg, aes(class, hwy))
```

```
dd + geom_point()
```

```
ee <- ggplot(mpg, aes(class, hwy))
```

```
ee + geom_rect()
```

```
ff <- ggplot(mpg, aes(class, hwy))
```

```
ff + geom_raster()
```

```
gg <- ggplot(mpg, aes(class, hwy))
```

```
gg + geom_hex()
```

```
hh <- ggplot(mpg, aes(class, hwy))
```

```
hh + geom_jitter()
```

```
ii <- ggplot(mpg, aes(class, hwy))
```

```
ii + geom_line()
```

```
jj <- ggplot(mpg, aes(class, hwy))
```

```
jj + geom_point()
```

```
kk <- ggplot(mpg, aes(class, hwy))
```

```
kk + geom_rect()
```

```
ll <- ggplot(mpg, aes(class, hwy))
```

```
ll + geom_raster()
```

```
mm <- ggplot(mpg, aes(class, hwy))
```

```
mm + geom_hex()
```

```
nn <- ggplot(mpg, aes(class, hwy))
```

```
nn + geom_jitter()
```

```
oo <- ggplot(mpg, aes(class, hwy))
```

```
oo + geom_line()
```

```
pp <- ggplot(mpg, aes(class, hwy))
```

```
pp + geom_point()
```

```
qq <- ggplot(mpg, aes(class, hwy))
```

```
qq + geom_rect()
```

```
rr <- ggplot(mpg, aes(class, hwy))
```

```
rr + geom_raster()
```

```
ss <- ggplot(mpg, aes(class, hwy))
```

```
ss + geom_hex()
```

```
tt <- ggplot(mpg, aes(class, hwy))
```

```
tt + geom_jitter()
```

```
uu <- ggplot(mpg, aes(class, hwy))
```

```
uu + geom_line()
```

```
vv <- ggplot(mpg, aes(class, hwy))
```

```
vv + geom_point()
```

```
ww <- ggplot(mpg, aes(class, hwy))
```

```
ww + geom_rect()
```

```
xx <- ggplot(mpg, aes(class, hwy))
```

```
xx + geom_raster()
```

```
yy <- ggplot(mpg, aes(class, hwy))
```

```
yy + geom_hex()
```

```
zz <- ggplot(mpg, aes(class, hwy))
```

```
zz + geom_jitter()
```

```
aa <- ggplot(mpg, aes(class, hwy))
```

```
aa + geom_line()
```

```
bb <- ggplot(mpg, aes(class, hwy))
```

```
bb + geom_point()
```

```
cc <- ggplot(mpg, aes(class, hwy))
```

```
cc + geom_rect()
```

```
dd <- ggplot(mpg, aes(class, hwy))
```

```
dd + geom_raster()
```

```
ee <- ggplot(mpg, aes(class, hwy))
```

```
ee + geom_hex()
```

```
ff <- ggplot(mpg, aes(class, hwy))
```

```
ff + geom_jitter()
```

```
gg <- ggplot(mpg, aes(class, hwy))
```

```
gg + geom_line()
```

```
hh <- ggplot(mpg, aes(class, hwy))
```

```
hh + geom_point()
```

```
ii <- ggplot(mpg, aes(class, hwy))
```

```
ii + geom_rect()
```

```
jj <- ggplot(mpg, aes(class, hwy))
```

```
jj + geom_raster()
```

```
kk <- ggplot(mpg, aes(class, hwy))
```

```
kk + geom_hex()
```

```
ll <- ggplot(mpg, aes(class, hwy))
```

```
ll + geom_jitter()
```

```
mm <- ggplot(mpg, aes(class, hwy))
```

```
mm + geom_line()
```

```
nn <- ggplot(mpg, aes(class, hwy))
```

```
nn + geom_point()
```

```
oo <- ggplot(mpg, aes(class, hwy))
```

```
oo + geom_rect()
```

```
pp <- ggplot(mpg, aes(class, hwy))
```

```
pp + geom_raster()
```

```
qq <- ggplot(mpg, aes(class, hwy))
```

```
qq + geom_hex()
```

```
rr <- ggplot(mpg, aes(class, hwy))
```

```
rr + geom_jitter()
```

```
ss <- ggplot(mpg, aes(class, hwy))
```

```
ss + geom_line()
```

```
tt <- ggplot(mpg, aes(class, hwy))
```

```
tt + geom_point()
```

```
uu <- ggplot(mpg, aes(class, hwy))
```

```
uu + geom_rect()
```

```
vv <- ggplot(mpg, aes(class, hwy))
```

```
vv + geom_raster()
```

```
ww <- ggplot(mpg, aes(class, hwy))
```

```
ww + geom_hex()
```

```
xx <- ggplot(mpg, aes(class, hwy))
```

```
xx + geom_jitter()
```

```
yy <- ggplot(mpg, aes(class, hwy))
```

```
yy + geom_line()
```

```
zz <- ggplot(mpg, aes(class, hwy))
```

```
zz + geom_point()
```

```
aa <- ggplot(mpg, aes(class, hwy))
```

```
aa + geom_rect()
```

```
bb <- ggplot(mpg, aes(class, hwy))
```

```
bb + geom_raster()
```

```
cc <- ggplot(mpg, aes(class, hwy))
```

```
cc + geom_hex()
```

```
dd <- ggplot(mpg, aes(class, hwy))
```

```
dd + geom_jitter()
```

```
ee <- ggplot(mpg, aes(class, hwy))
```

```
ee + geom_line()
```

```
ff <- ggplot(mpg, aes(class, hwy))
```

```
ff + geom_point()
```

```
gg <- ggplot(mpg, aes(class, hwy))
```

```
gg + geom_rect()
```

```
hh <- ggplot(mpg, aes(class, hwy))
```

```
hh + geom_raster()
```

```
ii <- ggplot(mpg, aes(class, hwy))
```

```
ii + geom_hex()
```

```
jj <- ggplot(mpg, aes(class, hwy))
```

```
jj + geom_jitter()
```

```
kk <- ggplot(mpg, aes(class, hwy))
```

```
kk + geom_line()
```

```
ll <- ggplot(mpg, aes(class, hwy))
```

```
ll + geom_point()
```

```
mm <- ggplot(mpg, aes(class, hwy))
```

```
mm + geom_rect()
```

```
nn <- ggplot(mpg, aes(class, hwy))
```

```
nn + geom_raster()
```

```
oo <- ggplot(mpg, aes(class, hwy))
```

```
oo + geom_hex()
```

```
pp <- ggplot(mpg, aes(class, hwy))
```

```
pp + geom_jitter()
```

```
qq <- ggplot(mpg, aes(class, hwy))
```

```
qq + geom_line()
```

```
rr <- ggplot(mpg, aes(class, hwy))
```

```
rr + geom_point()
```

```
ss <- ggplot(mpg, aes(class, hwy))
```

```
ss + geom_rect()
```

```
tt <- ggplot(mpg, aes(class, hwy))
```

```
tt + geom_raster()
```

```
uu <- ggplot(mpg, aes(class, hwy))
```

```
uu + geom_hex()
```

```
vv <- ggplot(mpg, aes(class, hwy))
```

```
vv + geom_jitter()
```

```
ww <- ggplot(mpg, aes(class, hwy))
```

```
ww + geom_line()
```

```
xx <- ggplot(mpg, aes(class, hwy))
```

```
xx + geom_point()
```

```
yy <- ggplot(mpg, aes(class, hwy))
```

```
yy + geom_rect()
```

```
zz <- ggplot(mpg, aes(class, hwy))
```

```
zz + geom_raster()
```

```
aa <- ggplot(mpg, aes(class, hwy))
```

```
aa + geom_hex()
```

```
bb <- ggplot(mpg, aes(class, hwy))
```

```
bb + geom_jitter()
```

```
cc <- ggplot(mpg, aes(class, hwy))
```

```
cc + geom_line()
```

```
dd <- ggplot(mpg, aes(class, hwy))
```

```
dd + geom_point()
```

```
ee <- ggplot(mpg, aes(class, hwy))
```

```
ee + geom_rect()
```

```
ff <- ggplot(mpg, aes(class, hwy))
```

```
ff + geom_raster()
```

```
gg <- ggplot(mpg, aes(class, hwy))
```

```
gg + geom_hex()
```

```
hh <- ggplot(mpg, aes(class, hwy))
```

```
hh + geom_jitter()
```

```
ii <- ggplot(mpg, aes(class, hwy))
```

```
ii + geom_line()
```

```
jj <- ggplot(mpg, aes(class, hwy))
```

```
jj + geom_point()
```

```
kk <- ggplot(mpg, aes(class, hwy))
```

```
kk + geom_rect()
```

```
ll <- ggplot(mpg, aes(class, hwy))
```

```
ll + geom_raster()
```

```
mm <- ggplot(mpg, aes(class, hwy))
```

```
mm + geom_hex()
```

```
nn <- ggplot(mpg, aes(class, hwy))
```

```
nn + geom_jitter()
```

```
oo <- ggplot(mpg, aes(class, hwy))
```

```
oo + geom_line()
```

```
pp <- ggplot(mpg, aes(class, hwy))
```

```
pp + geom_point()
```

```
qq <- ggplot(mpg, aes(class, hwy))
```

```
qq + geom_rect()
```

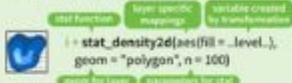
```
rr <- ggplot(mpg, aes(class, hwy))
```

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")`



```
a + stat_bin(binwidth = 1, origin = 0) # 1D binning  
x, y, count, density, intensity.  
+ stat_density2d(aes(fill = ..level..),  
geom = "polygon", n = 100)  
geom for layer + parameters for stat
```

```
a + stat_bin2d(binwidth = 10, drop = TRUE) # 2D binning  
x, y, fill, count, density.  
+ stat_binnedbins(bins = 10)  
x, y, fill, count, density.  
+ stat_density2d(binwidth = 1, bins = 100)  
x, y, color, size, level.
```

```
+ stat_contour(binwidth = 1) # 2D contours  
x, y, z, color, size.
```

```
+ stat_hexbin(binwidth = 1, angle = 45)  
angle, radius, x, mind, yend = 1, max = 1, ..., pmin,  
+ stat_summary_hexbin(binwidth = 1, bins = 30, fun = mean)  
x, y, fill, value.
```

```
+ stat_summary2d(binwidth = 1, bins = 30, fun = mean)  
x, y, fill, value.
```

```
+ stat_boxplot(binwidth = 1) # Comparisons  
x, y, lower, middle, upper, outliers.
```

```
+ stat_violin(binwidth = 1, kernel = "gaussian", scale = "ave")  
x, y, density, scaled, count, n, violinwidth, width.
```

```
+ stat_ecdf(n = 40) # Functions  
x, y, ..., p.
```

```
+ stat_quantile(q = c(0.25, 0.5, 0.75), formula = y ~ log(x),  
method = "rqf")  
x, y, quantile, ..., x, y.
```

```
+ stat_smooth(method = "auto", formula = y ~ x, stat = TRUE, n = 10,  
fullrange = FALSE, level = 0.95)  
x, y, ..., x, y, geom, geoms.
```

```
+ stat_function(fun = 2.5, # General Purpose  
hex = hex, n = 200, arg = list(b = 0.5))  
x, y.
```

```
+ stat_identity()  
geom + stat_identity(bins = 100, distribution = qt,  
dparams = list(t = 0))  
sample, x, y, ...  
+ stat_sum()  
x, y, size, ...  
+ stat_summary(fun.data = "mean_se", bootstrap = TRUE)  
x, y, ..., x, y.
```

See [stat_functions](#)

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, transparency, shape, size
`scale_<_continuous()` - map continuous values to visual values
`scale_<_discrete()` - map discrete values to visual values
`scale_<_manual()` - 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 (as shown here)
`scale_x_date()` - labels = date_format("Ym d D"),
breaks = date_breaks("2 weeks") - treat x values as dates. See [date_time](#) 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
<code>scale_color_bar(aes(y = 1))</code> y = 1, green_bar, aes(y = 1)	<code>scale_color_point(aes(size = 1))</code> size = 1, green_dotplot, aes(size = 1)
<code>scale_fill_brewer()</code> + scale_fill_discrete() + scale_fill_hex()	<code>scale_fill_gradient()</code> + scale_fill_hex() + scale_fill_gradient2() + scale_fill_hex2()
<code>scale_fill_gray()</code> start = 0.2, end = 0.8, na.value = "#F0F0F0"	<code>scale_fill_gradient2()</code> colors = terrain.colors, type = "linear", na.value = "#F0F0F0"

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

```
+ geom_bar(position = position_dodge(width = 1))
```



Coordinate Systems

```
t + b + geom_bar()
```

```
+ coord_cartesian(xlim = c(0, 5),  
x, y, min, max)
```

The default cartesian coordinate system
`coord_fixed(ratio = 1/2)`
ratio, xlim, ylim

Cartesian coordinates with fixed aspect ratio between x and y units
`coord_flip()`
x, y, min, max

Flipped Cartesian coordinates
`coord_polar(theta = "x", direction = 1)`
theta, start, direction

Polar coordinates
`coord_trans(trans = "sqrt")`
trans, xlim, ylim

Transformed cartesian coordinates. Set extras and strain to the name of a window function
`coord_map(projection = "ortho",
orientation = c(0, 0))`
projection, orientation, xlim, ylim

Map projections from the mapproj package
(mercator (default), azequidistant, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geom's that would otherwise occupy the same space:
`s + ggplot(mpg, aes(cty, hwy))`

```
+ geom_bar(position = "dodge")  
Arrange elements side by side
```

```
+ geom_bar(position = "fill")  
Stack elements on top of one another, normalize height
```

```
+ geom_bar(position = "stack")  
Stack elements on top of one another
```

```
+ geom_point(position = "jitter")  
Add random noise to X and Y position of each element to avoid overplotting
```

All of the above

Themes

```
+ theme_bw()  
white background with black grid lines
```

```
+ theme_grey()  
Grey background (defaut theme)
```

```
+ 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()
```

```
+ facet_grid(~ fl)
```

facet into columns based on fl
`+ facet_grid(~ year)`

facet into rows based on year
`+ facet_grid(~ fl * year)`

facet into both rows and columns
`+ facet_wrap(~ fl)`

wrap facets into a rectangular layout

Set scales to let axis limits vary across facets:

```
t + facet_grid(~ 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:

```
+ facet_grid(~ fl, labeller = label_both)
```

R: c, fl: f, fl: fl, fl: fl, fl: fl

```
+ facet_grid(~ fl, labeller = label_bquote(flip ~ fl))  
R: c, fl: fl, fl: fl, fl: fl, fl: fl
```

```
+ facet_grid(~ fl, labeller = label_parsed)  
R: c, fl: fl, fl: fl, fl: fl, fl: fl
```

Labels

```
+ ggtitle("New Plot Title")
```

Add a main title above the plot

```
+ xlab("New X label")
```

Change the label on the X axis

```
+ ylab("New Y label")
```

Change the label on the Y axis

```
+ labs(title = "New title", x = "New x", y = "New y")
```

All of the above

Legends

```
+ theme(legend.position = "bottom")
```

Place legend at "bottom", "top", "left", or "right"

```
+ guides(color = "none")
```

Set legend type for each aesthetic: color/shape/legend, or none (no legend)

```
+ scale_fill_discrete(name = "Title",  
labels = c("A", "B", "C"))
```

Set legend title and labels with a scale function:

Zooming

```
+ theme_bw()  
Without clipping (unzoomed)
```

```
+ coord_cartesian(xlim = c(0, 100), ylim = c(10, 200))
```

With clipping (removes unzoomed data points)

```
+ xlim(0, 100) + ylim(10, 200)
```

```
+ scale_x_continuous(limits = c(0, 100)) +  
scale_y_continuous(limits = c(0, 100))
```

Learn more at [docs.ggplot2.org](#) • `ggplot2 0.9.3.1` • Updated 3/11



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>

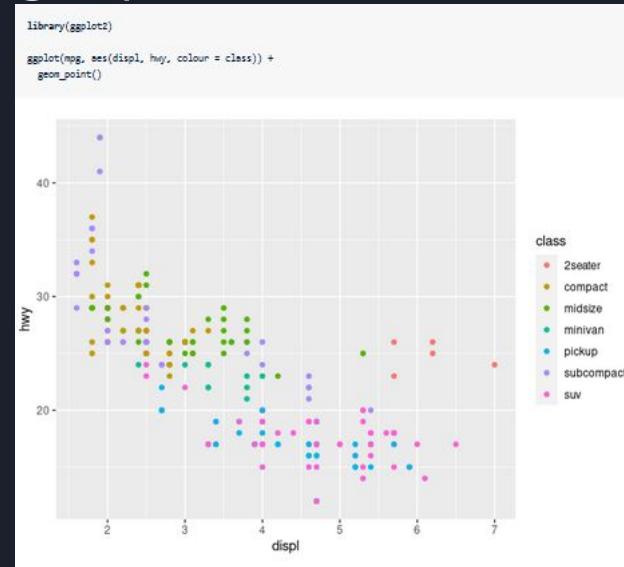
ggplot()- Grammar of graphics

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>

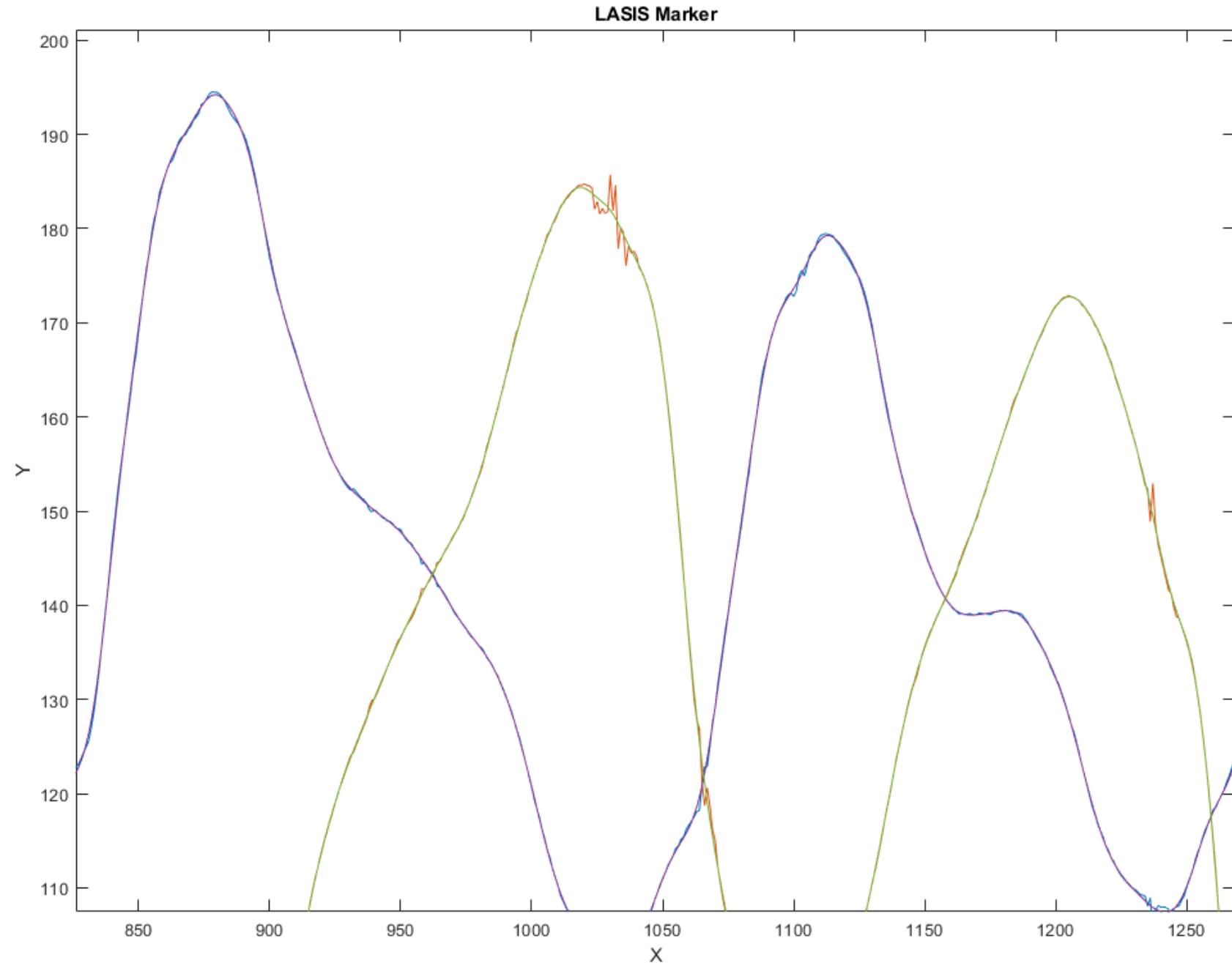
<http://r-statistics.co/ggplot2-Tutorial-With-R.html>

R: Signal Processing

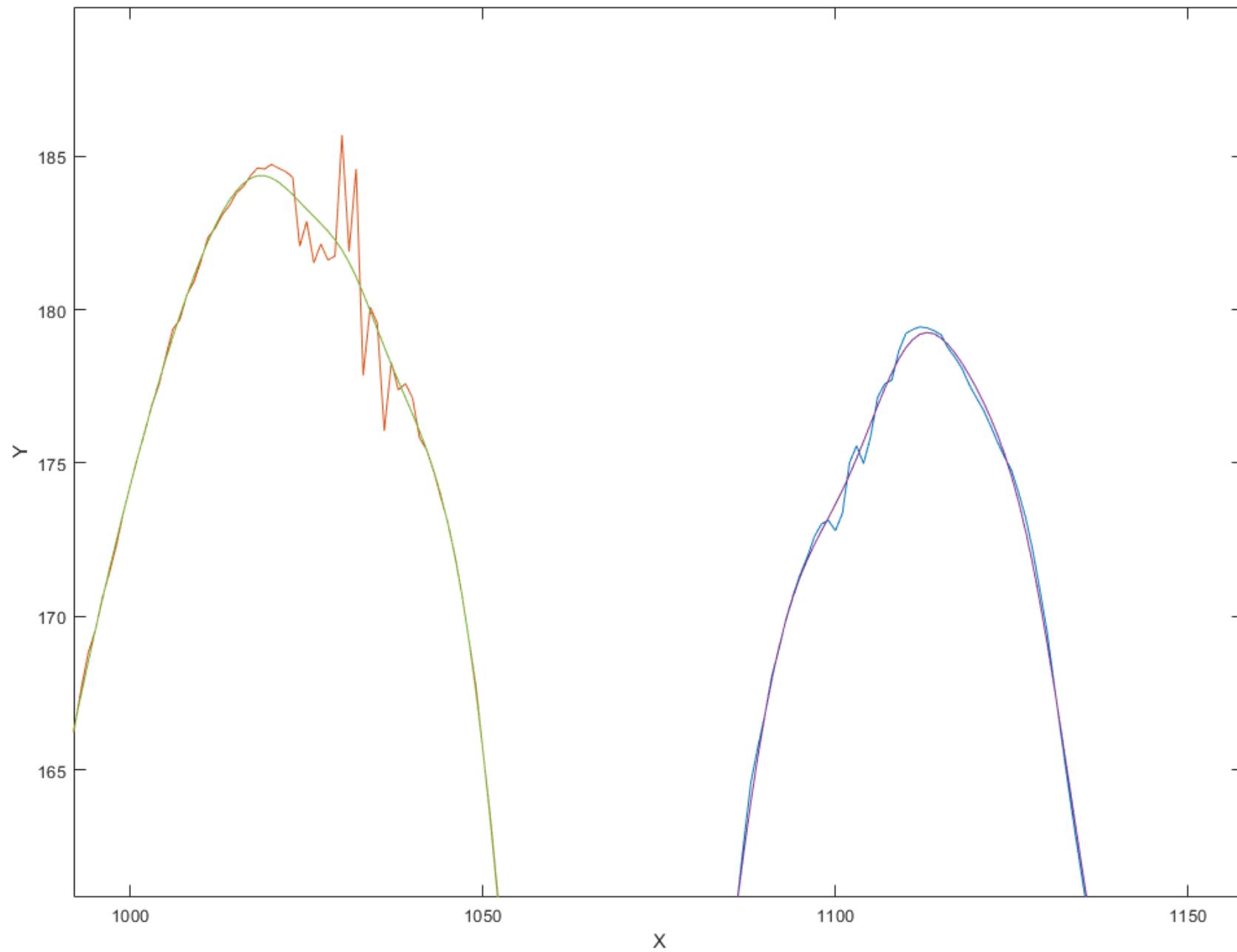
Christopher Hoang

- **Description**
- Generate Butterworth filter polynomial coefficients.
- Usage
- **## Default S3 method:**
- `butter(n, W, type = c("low", "high", "stop", "pass"),`
- `plane = c("z", "s"), ...)`
- **## S3 method for class 'FilterOfOrder'**
- `butter(n, ...)`

an	4
Arma	5
bilinear	6
butter	8
buttord	10
cheb1ord	11
chebwin	12
cheby1	14
chirp	16
conv	17
decimate	18
ellip	19
ellipord	21
fftfilt	22
filter	23
FilterOfOrder	25
filtfilt	26
fir1	27
fir2	28
freqs	29
freqz	31
grpdelay	33
ifft	35
impz	36
interp	37
interp1	38
kaiser	40
kaiserord	41
levinson	43
Ma	44
medfilt1	44
pchip	46
poly	47
polyval	48
remez	48
resample	49
roots	51
sftrans	52
sgolay	55
sgolayfilt	56



LASIS Marker

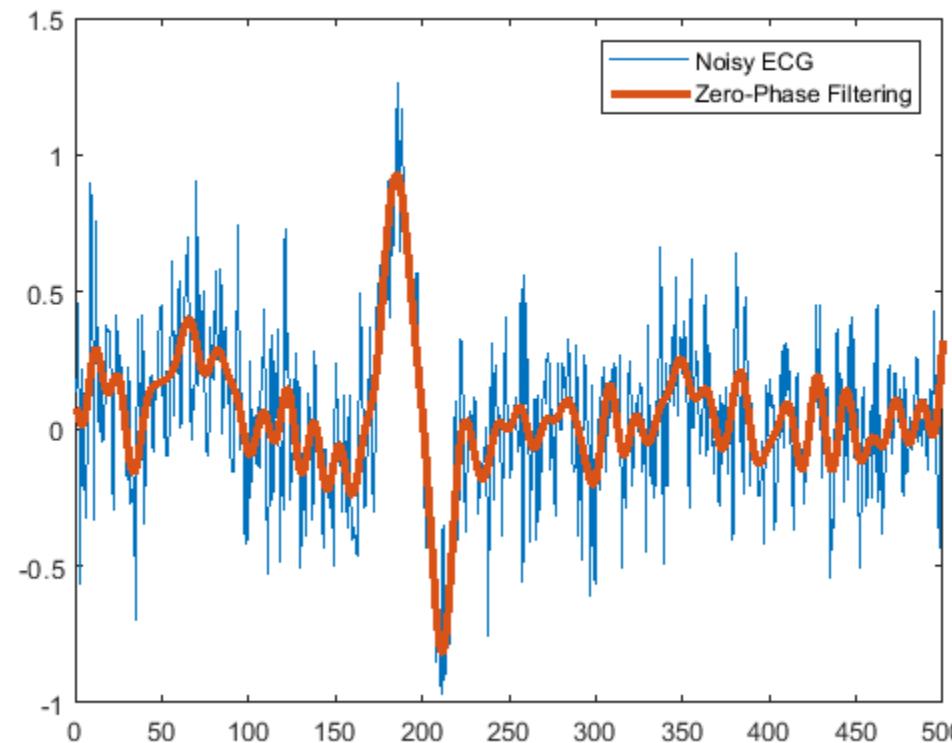


MatLab Code

- %% Filter Marker Data
- QTM_Fc=6; %Cutoff Frequency=6Hz
- % % % Butterworth filter 4th Order and cutt off frequency 6Hz
- [b,a] = butter(4,QTM_Fc/(rawMarker_freq/2), 'low');
-
- m=1;
- for i=1:length(mkrNameList)
- eval(['Marker.',mkrNameList{m}, '.filtmkrd
= ', 'filtfilt(b,a,', 'rawMarker.', mkrNameList{m}, ') ; ']);
- m=m+1;
- end
- disp('Entered Filtered Marker Data into struct Marker...');

Filtfilt

```
d1 = designfilt('lowpassiir','FilterOrder',12, ...
'HalfPowerFrequency',0.15,'DesignMethod','butter'); y = filtfilt(d1,x);
subplot(1,1,1) plot(x) hold on plot(y,'LineWidth',3) legend('Noisy
ECG','Zero-Phase Filtering')
```



Forcats

Joshua Anderson



Imdb_movies Dataset

```
Rows: 4,499
Columns: 31
$ title <chr> "Avatar", "Pirates of the Caribbean: At World...
$ year <int> 2009, 2007, 2015, 2012, 2012, 2007, 2010, 2015, ...
$ country <chr> "USA", "USA", "UK", "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] ""
[4] "Aruba"
[7] "Belgium"
[10] "Canada"
[13] "Colombia"
[16] "Dominican Republic"
[19] "France"
[22] "Greece"
[25] "Iceland"
[28] "Iran"
[31] "Italy"
[34] "Kyrgyzstan"
[37] "Netherlands"
[40] "Nigeria"
[43] "Pakistan"
[46] "Philippines"
[49] "Russia"
[52] "South Korea"
[55] "Sweden"
[58] "Thailand"
[61] "United Arab Emirates" "USA"
[4] "Australia"
[7] "Brazil"
[10] "Chile"
[13] "Czech Republic"
[16] "Egypt"
[19] "Georgia"
[22] "Hong Kong"
[25] "India"
[28] "Ireland"
[31] "Japan"
[34] "Libya"
[37] "New Line"
[40] "Norway"
[43] "Panama"
[46] "Poland"
[49] "Slovenia"
[52] "Soviet Union"
[55] "Switzerland"
[58] "Turkey"
[61] "USA"
[4] "Argentina"
[7] "Bahamas"
[10] "Bulgaria"
[13] "China"
[16] "Denmark"
[19] "Finland"
[22] "Germany"
[25] "Hungary"
[28] "Indonesia"
[31] "Israel"
[34] "Kenya"
[37] "Mexico"
[40] "New Zealand"
[43] "Official site"
[46] "Peru"
[49] "Romania"
[52] "South Africa"
[55] "Spain"
[58] "Taiwan"
[61] "UK"
[4] "West Germany"
```

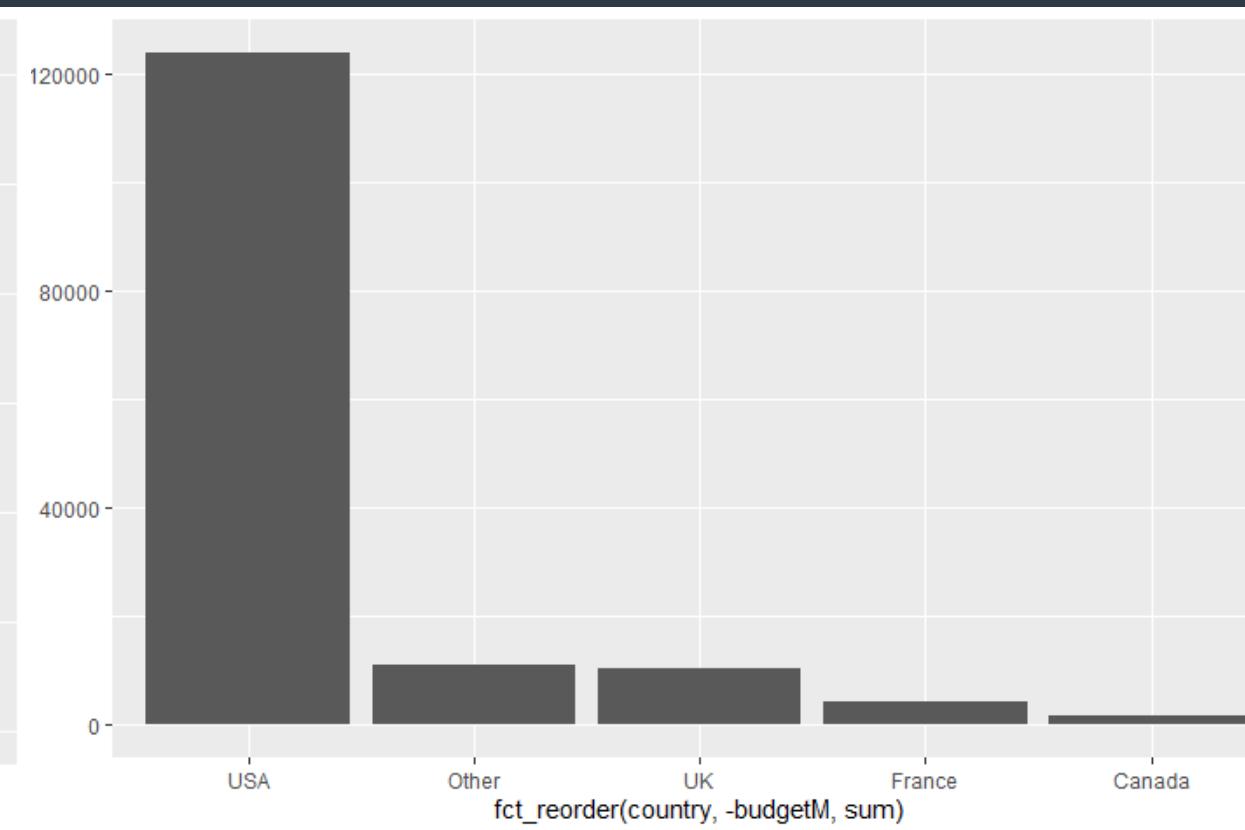
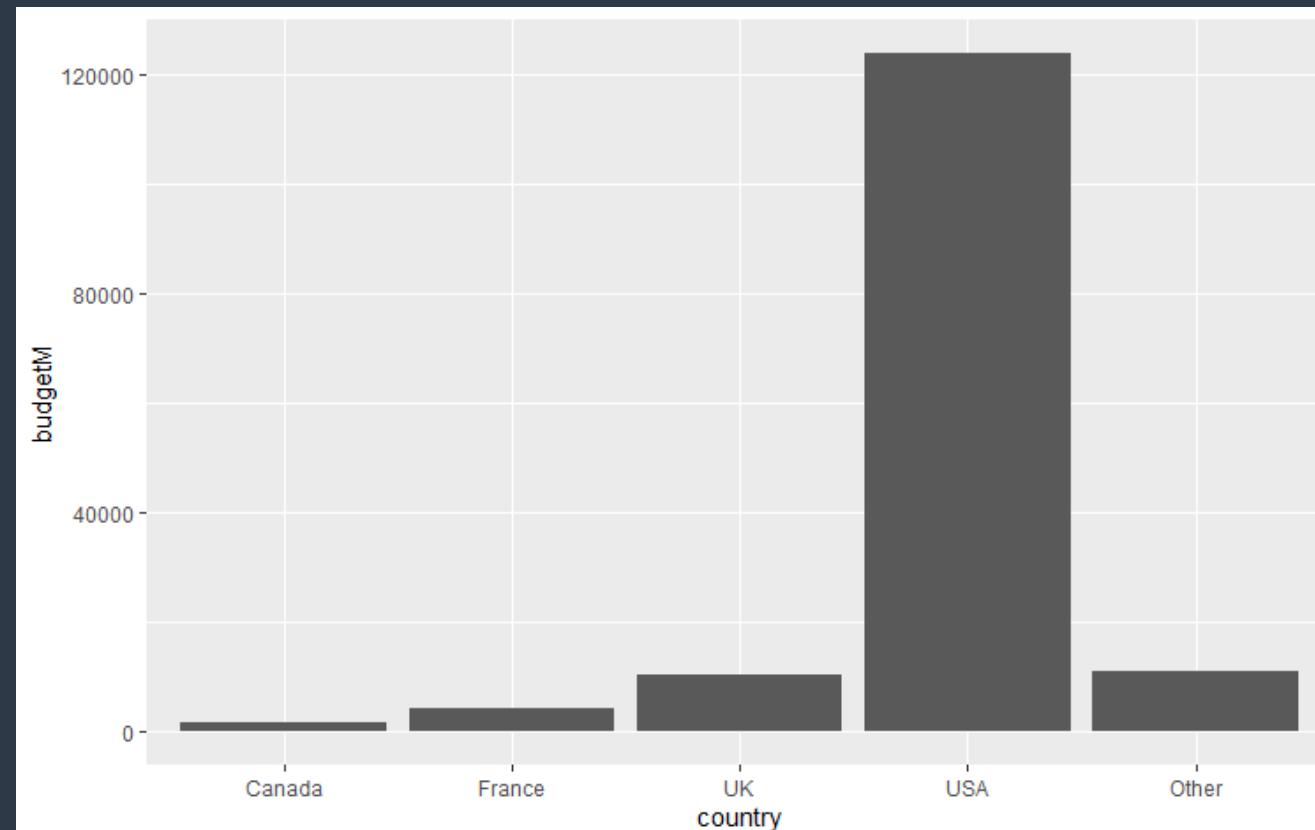
```
levels(fct_lump(movies_clean$country, n = 4))
1] "Canada" "France" "UK"      "USA"      "Other"
```

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.