



Data Wrangling

Live 04 - Data Science Bootcamp

R

Project 02 - Casino Royale



You go into a casino. Play 10 games.

Each game costs \$50.

Win chance 30% to earn \$150.

```
casino_royal <- function() {  
  prize <- vector(length = 10, mode = "numeric")  
  for (i in 1:10) {  
    result = sample(c("win", "loss"), size=1, prob = c(0.30, 0.70))  
    if (result == "win") {  
      prize[i] = prize[i] + 150  
    } else {  
      prize[i] = 0  
    }  
  }  
  sum(prize)  
}
```

R

Project 02 - Casino Royale



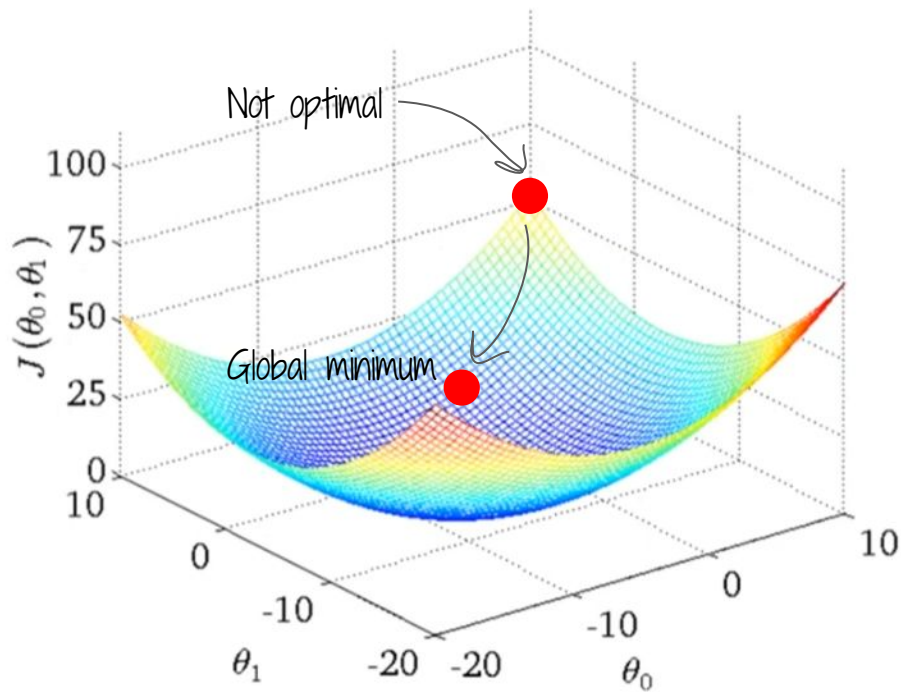
You go into a casino. Play 10 games.

Each game costs \$50.

Win chance 30% to earn \$150.

$$\text{Expected Return} = \$150 * 0.3 = \$45$$

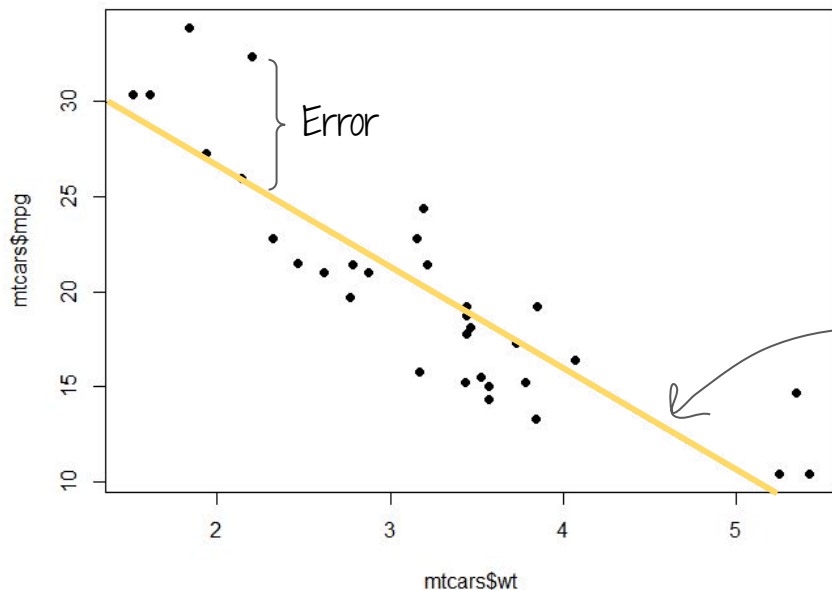
Project 03 - Gradient Descent Algorithm



How ML algorithm works

R

Revisit our linear regression model



$$\text{mpg} = f(\text{wt})$$

$$\text{mpg} = 37.285 - 5.344 * \text{wt}$$

Intercept

slope



Revisit our linear regression model

```
# we can run a simple linear regression model  
using lm()
```

```
lm(mpg ~ wt, data = mtcars)
```

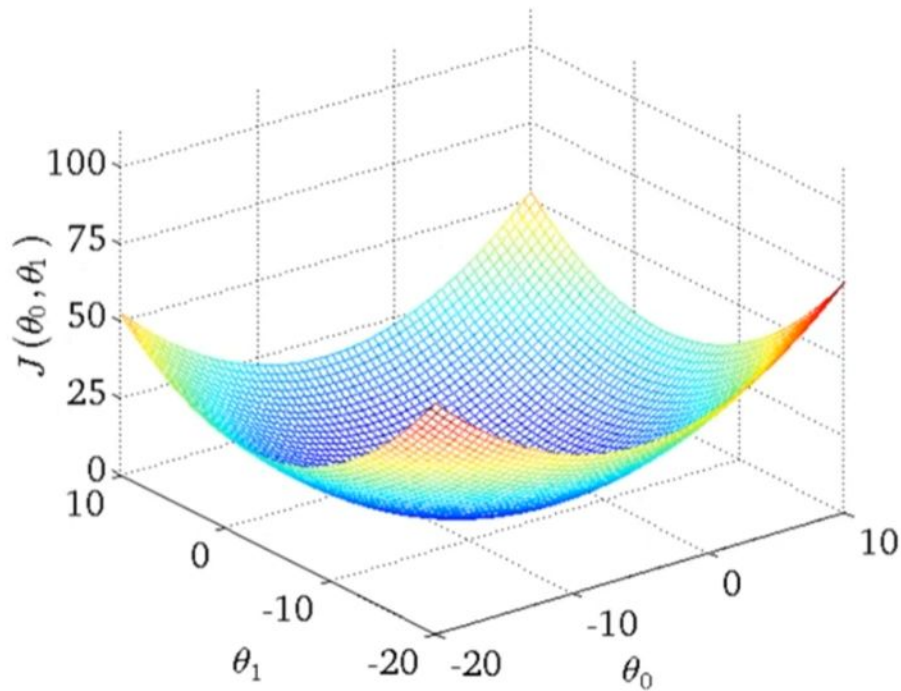


Very easy to use, perform
regression in R

R

Cost function (sum of squared error)

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

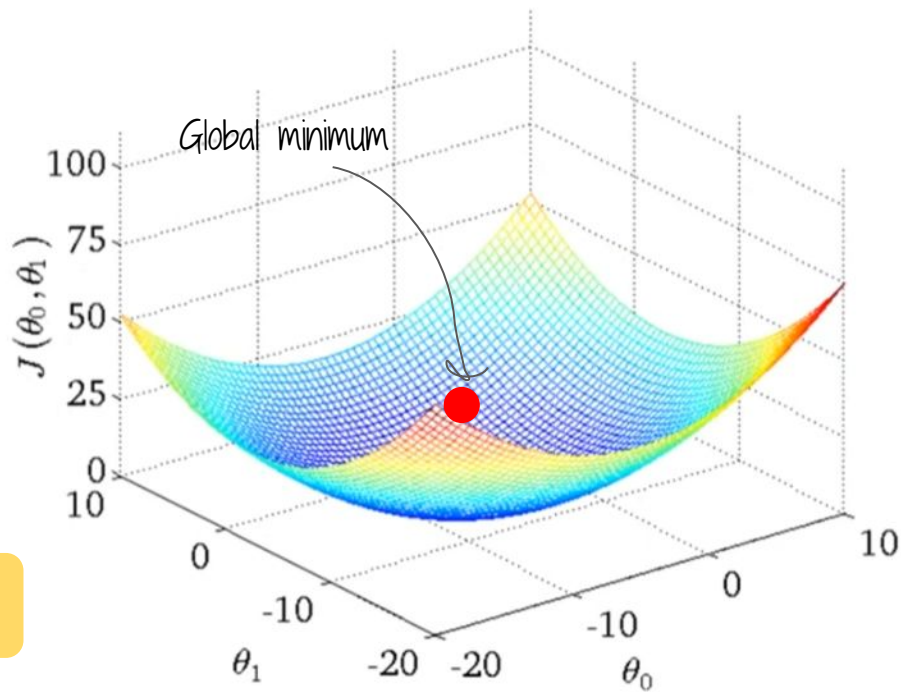


R

Cost function (sum of squared error)

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

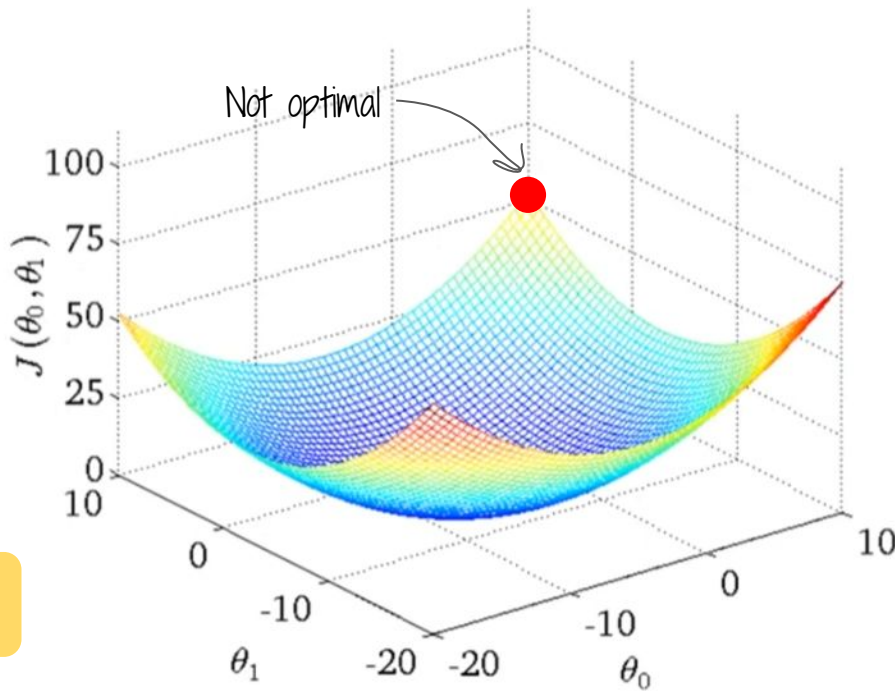
$$\text{mpg} = 37.285 - 5.344 * \text{wt}$$



R What if the intercept and slope are not OPTIMAL.

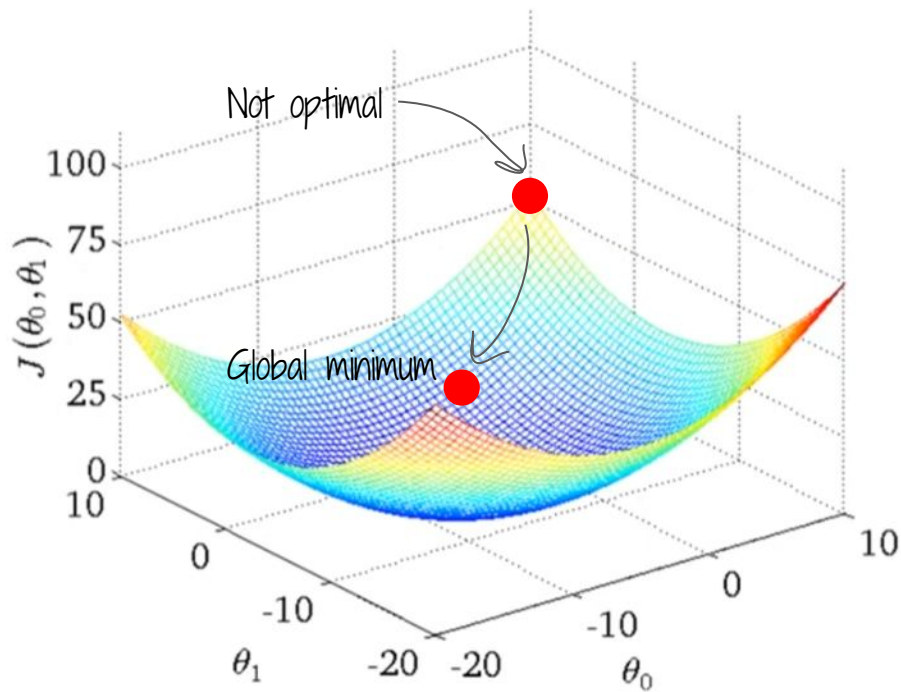
$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$\text{mpg} = 20.213 - 8.256 \cdot \text{wt}$$



R

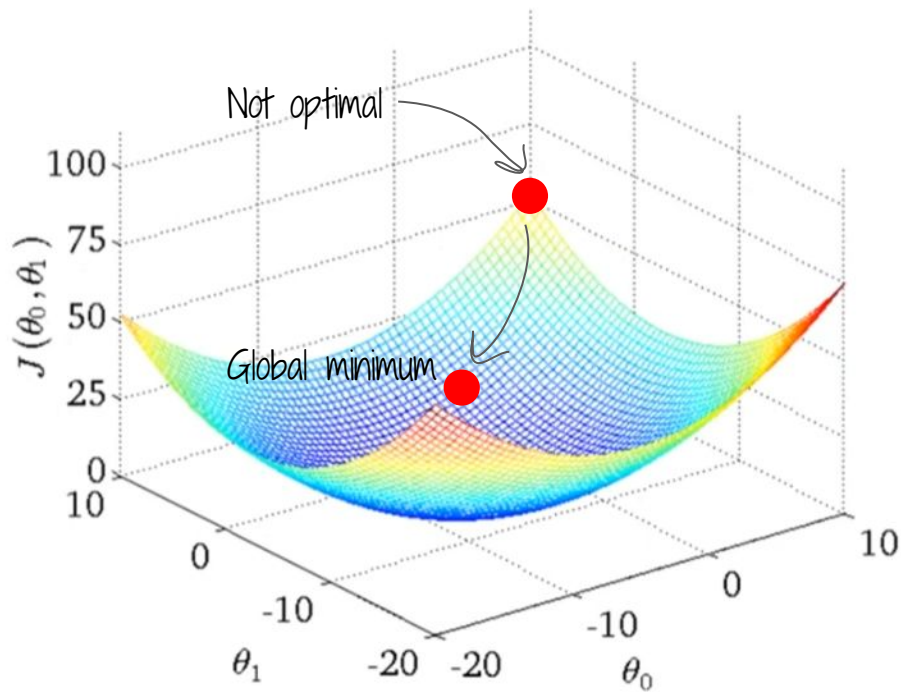
We want to go down the hill



We want to find the best b_0 and b_1 that make SSE (our cost) lowest possible.

R

How gradient descent algorithm works



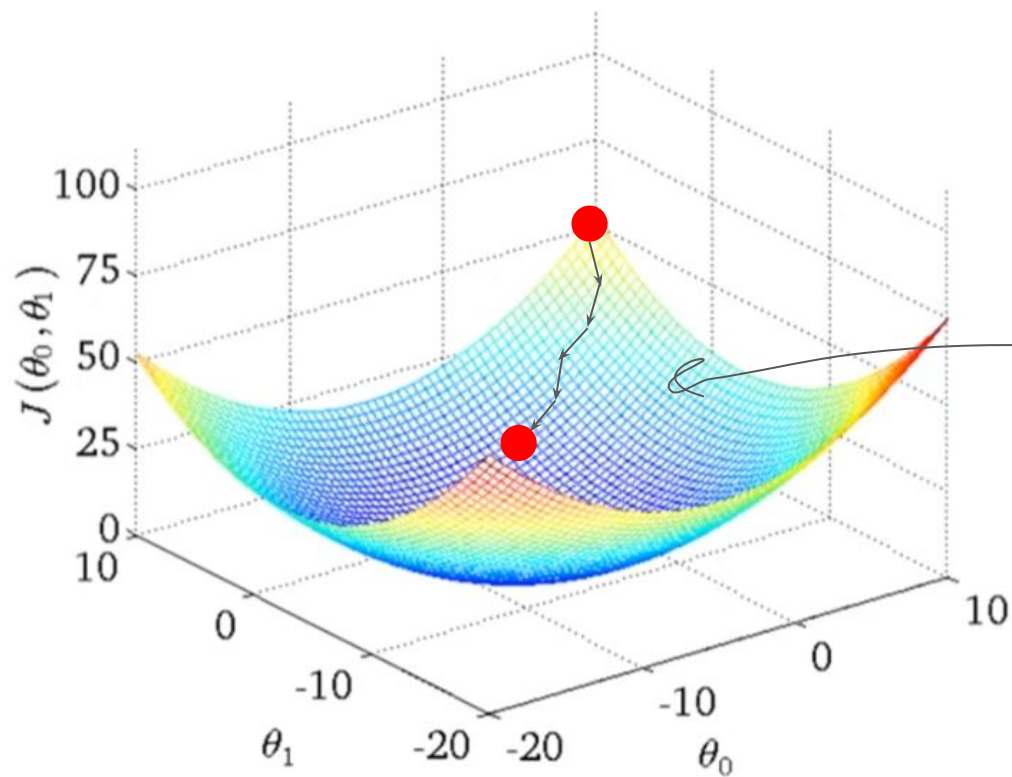
Completed in 4 steps

1. Randomize b_0 and b_1
2. Compute cost and find slope (Gradient)
3. Update b_0 and b_1
4. Repeat 2-3 until **convergence**

Lowest cost possible

R

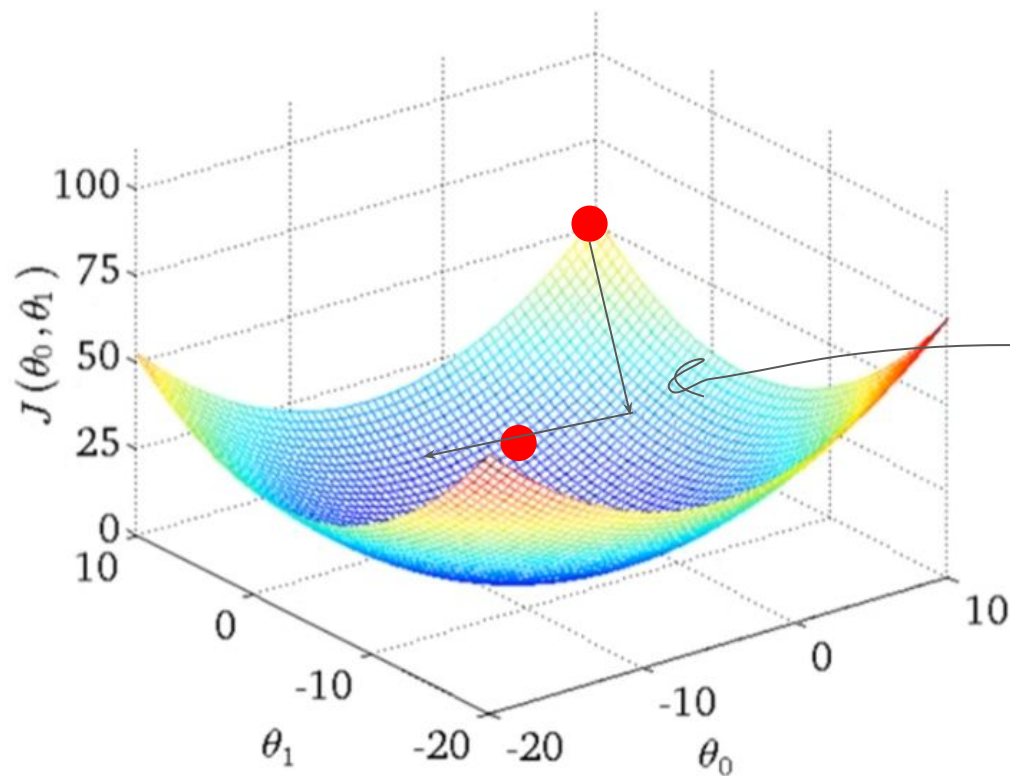
Learning Rate (Choose it wisely)



Learning rate is how fast you want to learn (i.e. lower cost)

R

Too Large Learning Rate



Overshooting

Algorithm will never find the global minimum

R Algorithm to update intercept (b0) and slope (b1)

Gradient descent algorithm

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

}

Learning rate (alpha) = how fast you want to go down the hill



[Gradient Descent For Linear Regression | Coursera](#)



Take a closer look

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

}

Prediction

Actual

Error

R

Take a closer look

repeat until convergence {

$$b_0 := b_0 - \alpha \frac{1}{m} \sum_{i=1}^m (\text{error})$$

$$b_1 := b_1 - \alpha \frac{1}{m} \sum_{i=1}^m (\text{error}) \cdot x^{(i)}$$

}

you choose this learning rate yourself



Solution - Gradient Descent

```
## define a function
gradient_descent <- function(data, x, y, alpha=0.05, iter=10000) {

  ## randomize intercept and slope
  (b0 <- runif(1,0,1))
  (b1 <- runif(1,0,1))

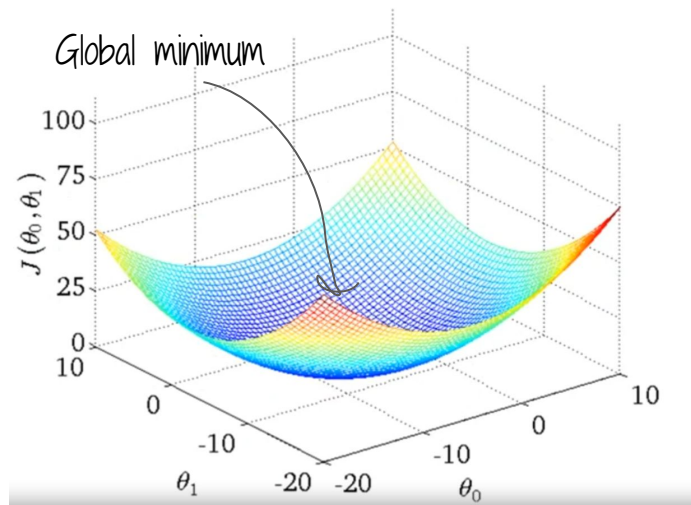
  n <- nrow(data)
  x <- scale(data[[x]])
  y <- data[[y]]

  for (i in 1:iter) {

    intercept <- b0 - alpha * (1/n) * sum(b0 + b1*x - y)
    slope <- b1 - alpha * (1/n) * sum((b0 + b1*x - y) * x)

    b0 <- intercept
    b1 <- slope

  }
  return(list(intercept = b0, slope = b1))
}
```



R

You can search for pattern with Regular Expression

Google Sheets is the best! You can use it for free, 0\$ cost.

R

Find G__S__

$G[a-z]^+ S[a-z]^+$




Google Sheets is the best! You can use it for free, 0\$ cost.

R

Find 0-9 number

Google Sheets is the best! You can
use it for free, 0\$ cost.

[0-9]





Regular Expression Basics

^A Ant, Amsterdam, America

s\$ Toys, SNSDs, APPLEs

c.t cat, cot, cet, cCt, c8t



Regular Expression Character Class

<code>[ABC]</code>	match A B or C
<code>[A-Z]</code>	match all capital letters
<code>[A-z]</code>	match all letters
<code>[a-z]</code>	match all lowercase letters
<code>[0-9]</code>	match digits

R

Regular Expression Quantifiers

*	match zero or more
+	match one or more
?	match zero or one
{5}	match exactly 5 characters
{3,5}	match min 3, max 5 characters

[Regular expression - Wikipedia](#)

R

More Examples :)

`[0-9]{5}`

apples?

`^[AB][0-9]{4}`

match exactly 5 digits

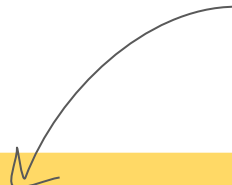
apple, apples

A1150, B2324, A3599

Data Wrangling

1. Get data into R
2. Manipulate and transform data
3. Summarise data
4. Export data

The most important R
package in 21st century



```
install.packages("tidyverse")  
library(tidyverse)
```

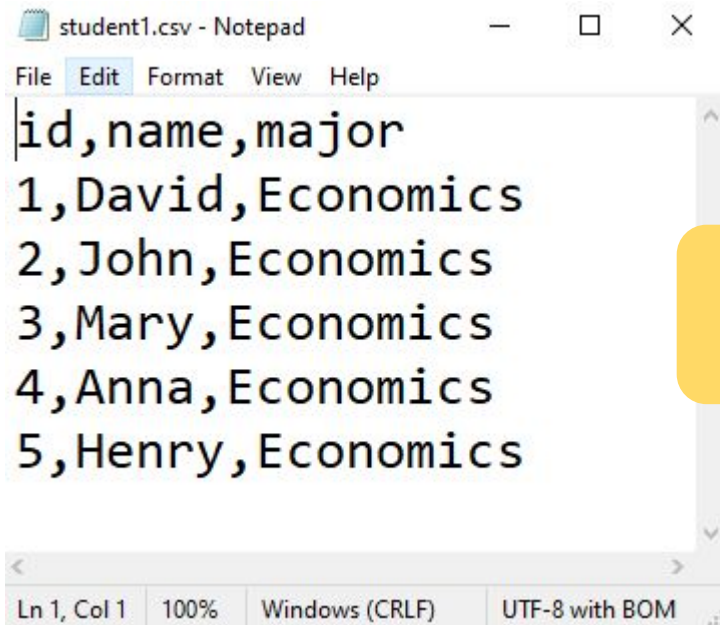


R can read data in so many formats

1. csv
2. delimited files (\t, ;)
3. excel
4. database
5. website
 - a. html
 - b. xml
6. json
7. statistics files e.g. SPSS, SAS



CSV File

A screenshot of a Notepad window titled 'student1.csv - Notepad'. The window contains a CSV file with 5 rows of data. The first row is the header: 'id,name,major'. The following rows are: '1,David,Economics', '2,John,Economics', '3,Mary,Economics', '4,Anna,Economics', and '5,Henry,Economics'. The status bar at the bottom shows 'Ln 1, Col 1', '100%', 'Windows (CRLF)', and 'UTF-8 with BOM'.

```
student1.csv - Notepad
File Edit Format View Help
id,name,major
1,David,Economics
2,John,Economics
3,Mary,Economics
4,Anna,Economics
5,Henry,Economics
Ln 1, Col 1 100% Windows (CRLF) UTF-8 with BOM
```

Modern R programming

```
library(tidyverse)
student1 <- read_csv("student1.csv")
```

R JSON

 employees_for_R.json

Raw

```
1  {
2    "ID":["1","2","3","4","5","6","7","8" ],
3    "Name":["Rick","Dan","Michelle","Ryan","Gary","Nina","Simon","Guru" ],
4    "Salary":["623.3","515.2","611","729","843.25","578","632.8","722.5" ],
5    "StartDate":["1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013","7/30/2013","6/17/2014"],
6    "Dept":["IT","Operations","IT","HR","Finance","IT","Operations","Finance"]
7  }
```

[employees_for_R.json \(github.com\)](https://github.com)



HTML

[school_website.html \(github.com\)](https://github.com/school_website.html)

school_website.html

Raw

```
1  <html>
2    <head>
3      <title>Welcome to our school website</title>
4    </head>
5    <body>
6      <h2>Our Innovative Teachers</h1>
7      <ul>
8        <li>Liam Nielson</li>
9        <li>Jack Dorsey</li>
10       <li>Reed Hastings</li>
11       <li>Mark Zuckerberg</li>
12       <li>Elon Musk</li>
13     </ul>
14     <h2>Their Companies</h2>
15     <ol>
16       <li>Hollywoods</li>
17       <li>Twitter</li>
18       <li>NetFlix</li>
19       <li>Facebook</li>
20       <li>Tesla</li>
21     </ol>
22   </body>
23 </html>
```




XML

 [test_email.xml](#)

Raw

```
1  <email>
2    <first>
3      <from>toy@datarockie</from>
4      <to>students@data-science-bootcamp.com</to>
5      <subject>Welcome to our course</subject>
6      <content>I hope you enjoy the class</content>
7      <date>16 January 2021</date>
8    </first>
9    <second>
10     <from>toy@datarockie</from>
11     <to>students@data-science-bootcamp.com</to>
12     <subject>Please install R and RStudio desktop</subject>
13     <content>Install these software before our live class on Saturday</content>
14     <date>16 January 2021</date>
15   </second>
16 </email>
```

[example email in XML format \(github.com\)](#)



Database

[SQLite \(rstudio.com\)](https://rstudio.com)

SQLite

Embeds the SQLite database engine in R, providing a DBI-compliant interface. [SQLite](#) is a public-domain, single-user, very light-weight database engine that implements a decent subset of the SQL 92 standard, including the core table creation, updating, insertion, and selection operations, plus transaction management.

You can install the latest released version of RSQLite from CRAN with:

```
install.packages("RSQLite")
```

Or install the latest development version from GitHub with:

```
# install.packages("devtools")
devtools::install_github("rstats-db/RSQLite")
```

To install from GitHub, you'll need a [development environment](#).

Basic usage

```
library(DBI)
# Create an ephemeral in-memory RSQLite database
con <- dbConnect(RSQLite::SQLite(), ":memory:")

dbListTables(con)
```

Create connection to
SQLite file



```
library(RSQLite)
con <- dbConnect(SQLite(), "chinook.db")
```



Data Pipeline in R

```
data %>%  
  step1() %>%  
  step2() %>%  
  step3() %>%  
  step4() %>%  
  step5()
```

Pipe operator
6-7 steps is okay!





Wide format

[worldphone.csv \(github.com\)](https://github.com/worldphone.csv)

Year	N.Amer	Europe	Asia	S.Amer	Oceania	Africa	Mid.Amer
1951	45939	21574	2876	1815	1646	89	555
1956	60423	29990	4708	2568	2366	1411	733
1957	64721	32510	5230	2695	2526	1546	773
1958	68484	35218	6662	2845	2691	1663	836
1959	71799	37598	6856	3000	2868	1769	911
1960	76036	40341	8220	3145	3054	1905	1008
1961	79831	43173	9053	3338	3224	2005	1076





Long format

[worldphone_long.csv \(github.com\)](https://github.com)

	Year	Region	Sales
1	1951	N.Amer	45939
2	1951	Europe	21574
3	1951	Asia	2876
4	1951	S.Amer	1815
5	1951	Oceania	1646
6	1951	Africa	89
7	1951	Mid.Amer	555
8	1956	N.Amer	60423
9	1956	Europe	29990
10	1956	Asia	4708
11	1956	S.Amer	2568
12	1956	Oceania	2366
13	1956	Africa	1411
14	1956	Mid.Amer	733
15	1957	N.Amer	64721
16	1957	Europe	32510
17	1957	Asia	5230
18	1957	S.Amer	2695
19	1957	Oceania	2526
20	1957	Africa	1546
21	1957	Mid.Amer	773
22	1958	N.Amer	68484
23	1958	Europe	35218
24	1958	Asia	6662
25	1958	S.Amer	2845
26	1958	Oceania	2691
27	1958	Africa	1663
28	1958	Mid.Amer	836
29	1959	N.Amer	71799
30	1959	Europe	37598
31	1959	Asia	6856
32	1959	S.Amer	3000
33	1959	Oceania	2868
34	1959	Africa	1769

Good format for data analysis
(R likes this very much!)





Tidy data transformation

Year	N.Amer	Europe	Asia	S.Amer	Oceania	Africa	Mid.Amer
1951	45939	21574	2876	1815	1646	89	555
1956	60423	29990	4708	2568	2366	1411	733
1957	64721	32510	5230	2695	2526	1546	773
1958	68484	35218	6662	2845	2691	1663	836
1959	71799	37598	6856	3000	2868	1769	911
1960	76036	40341	8220	3145	3054	1905	1008
1961	79831	43173	9053	3338	3224	2005	1076

Good for report



	Year	Region	Sales
1	1951	N.Amer	45939
2	1951	Europe	21574
3	1951	Asia	2876
4	1951	S.Amer	1815
5	1951	Oceania	1646
6	1951	Africa	89
7	1951	Mid.Amer	555
8	1956	N.Amer	60423
9	1956	Europe	29990
10	1956	Asia	4708
11	1956	S.Amer	2568
12	1956	Oceania	2366
13	1956	Africa	1411
14	1956	Mid.Amer	733
15	1957	N.Amer	64721
16	1957	Europe	32510
17	1957	Asia	5230
18	1957	S.Amer	2695
19	1957	Oceania	2526
20	1957	Africa	1546
21	1957	Mid.Amer	773
22	1958	N.Amer	68484
23	1958	Europe	35218
24	1958	Asia	6662
25	1958	S.Amer	2845
26	1958	Oceania	2691
27	1958	Africa	1663
28	1958	Mid.Amer	836
29	1959	N.Amer	71799
30	1959	Europe	37598
31	1959	Asia	6856
32	1959	S.Amer	3000
33	1959	Oceania	2868
34	1959	Africa	1769

Good for data analysis

Bootcamp Live 04 Data Wrangling with R



Website: <https://datarockie.com>

