

# DATA MANIPULATION WITH DPLYR



# DPLYR RATIONALE

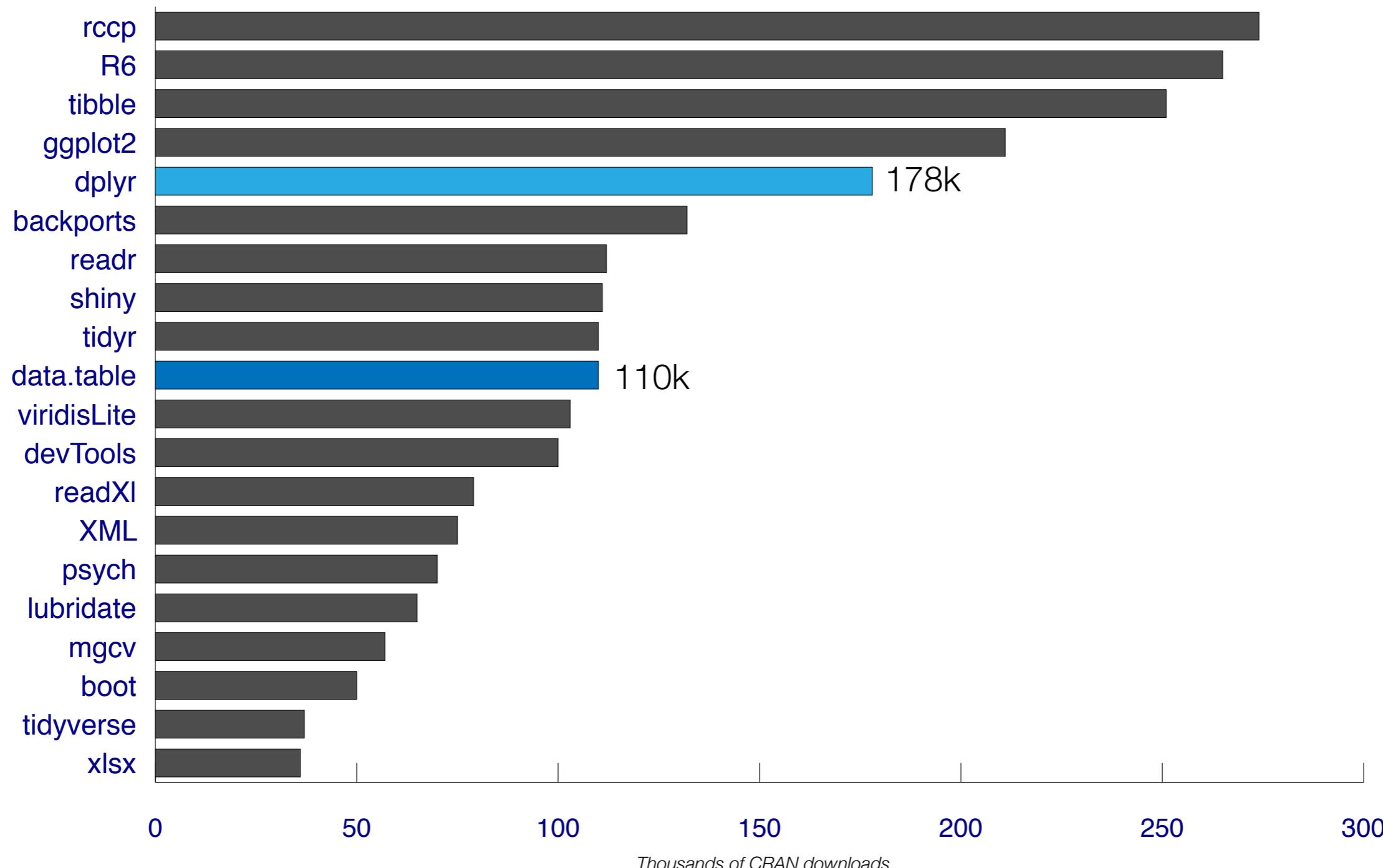
- Data science means manipulating data
- R is a flexible environment

# IT STARTS WITH DATA MANIPULATION



# POPULAR R LIBRARIES

Top 20 CRAN downloads (direct and indirect) September 2017



source: <http://makemeanalyst.com/20-most-popular-r-packages/>

# DPLYR Vs DATA.TABLE

## *dplyr advantages*

- SQL like syntax..
- Popular

## *data.table advantages*

- Extremely fast
- Perhaps more robust..

# Data Wrangling with dplyr and tidyverse

## Cheat Sheet



### Syntax - Helpful conventions for wrangling

#### dplyr::tbl\_df(iris)

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

```
Source: local data frame [150 x 5]
  Sepal.Length Sepal.Width Petal.Length
1          5.1        3.5         1.4
2          4.9        3.0         1.4
3          4.7        3.2         1.3
4          4.6        3.1         1.5
5          5.0        3.6         1.4
...
Variables not shown: Petal.Width (dbl), Species (fctr)
```

#### dplyr::glimpse(iris)

Information dense summary of `tbl` data.

#### utils::View(iris)

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

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

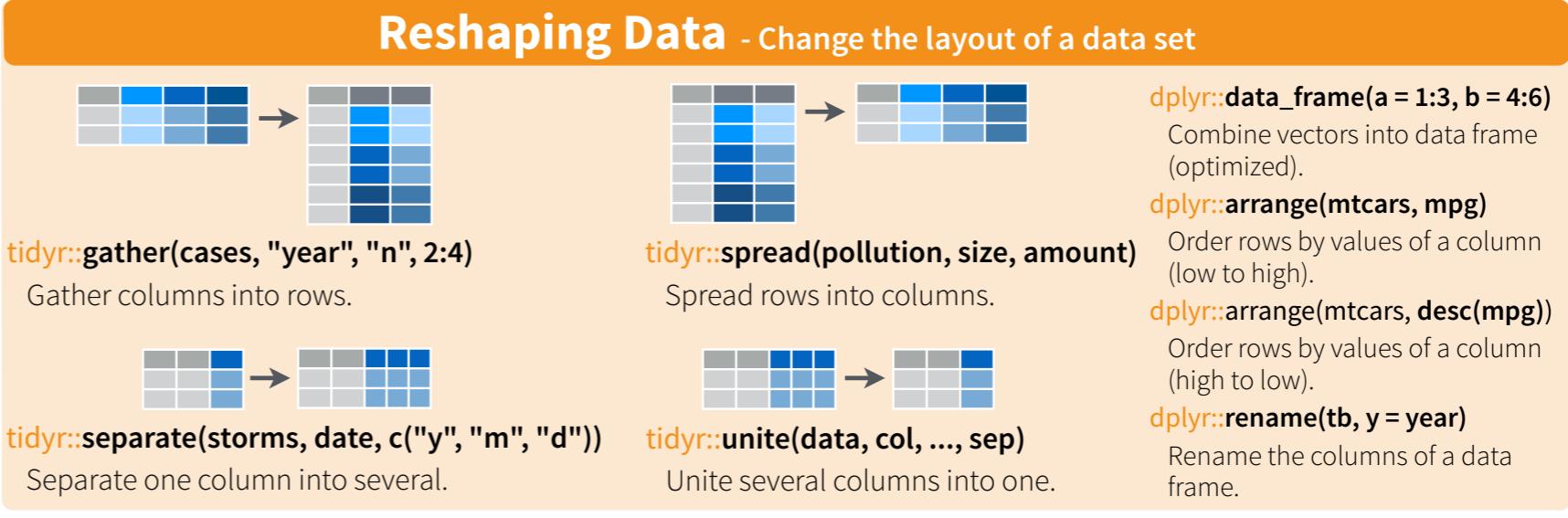
#### dplyr::%>%

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

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

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

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



### Subset Observations (Rows)



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

Extract rows that meet logical criteria.

#### dplyr::distinct(iris)

Remove duplicate rows.

#### dplyr::sample\_frac(iris, 0.5, replace = TRUE)

Randomly select fraction of rows.

#### dplyr::sample\_n(iris, 10, replace = TRUE)

Randomly select n rows.

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

Select rows by position.

#### dplyr::top\_n(storms, 2, date)

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

### Subset Variables (Columns)



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

Select columns by name or helper function.

#### Helper functions for select - ?select

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

Select columns whose name contains a character string.

##### select(iris, ends\_with("Length"))

Select columns whose name ends with a character string.

##### select(iris, everything())

Select every column.

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

Select columns whose name matches a regular expression.

##### select(iris, num\_range("x", 1:5))

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

##### select(iris, one\_of(c("Species", "Genus")))

Select columns whose names are in a group of names.

##### select(iris, starts\_with("Sepal"))

Select columns whose name starts with a character string.

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

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

##### select(iris, -Species)

Select all columns except Species.

Logic in R - ?Comparison, ?base::Logic			
<code>&lt;</code>	Less than	<code>!=</code>	Not equal to
<code>&gt;</code>	Greater than	<code>%in%</code>	Group membership
<code>==</code>	Equal to	<code>is.na</code>	Is NA
<code>&lt;=</code>	Less than or equal to	<code>!is.na</code>	Is not NA
<code>&gt;=</code>	Greater than or equal to	<code>&amp;,  , !, xor, any, all</code>	Boolean operators

## Summarise Data



**dplyr::summarise(iris, avg = mean(Sepal.Length))**

Summarise data into single row of values.

**dplyr::summarise\_each(iris, funs(mean))**

Apply summary function to each column.

**dplyr::count(iris, Species, wt = Sepal.Length)**

Count number of rows with each unique value of variable (with or without weights).



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

**dplyr::first**

First value of a vector.

**dplyr::last**

Last value of a vector.

**dplyr::nth**

Nth value of a vector.

**dplyr::n**

# of values in a vector.

**dplyr::n\_distinct**

# of distinct values in a vector.

**IQR**

IQR of a vector.

**min**

Minimum value in a vector.

**max**

Maximum value in a vector.

**mean**

Mean value of a vector.

**median**

Median value of a vector.

**var**

Variance of a vector.

**sd**

Standard deviation of a vector.

## Group Data

**dplyr::group\_by(iris, Species)**

Group data into rows with the same value of Species.

**dplyr::ungroup(iris)**

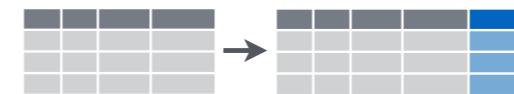
Remove grouping information from data frame.

**iris %>% group\_by(Species) %>% summarise(...)**

Compute separate summary row for each group.



## Make New Variables



**dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)**

Compute and append one or more new columns.

**dplyr::mutate\_each(iris, funs(min\_rank))**

Apply window function to each column.

**dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)**

Compute one or more new columns. Drop original columns.



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

**dplyr::lead**

Copy with values shifted by 1.

**dplyr::lag**

Copy with values lagged by 1.

**dplyr::dense\_rank**

Ranks with no gaps.

**dplyr::min\_rank**

Ranks. Ties get min rank.

**dplyr::percent\_rank**

Ranks rescaled to [0, 1].

**dplyr::row\_number**

Ranks. Ties got to first value.

**dplyr::ntile**

Bin vector into n buckets.

**dplyr::between**

Are values between a and b?

**dplyr::cume\_dist**

Cumulative distribution.

**dplyr::cumall**

Cumulative all

**dplyr::cumany**

Cumulative any

**dplyr::cummean**

Cumulative mean

**cumsum**

Cumulative sum

**cummax**

Cumulative max

**cummin**

Cumulative min

**cumprod**

Cumulative prod

**pmax**

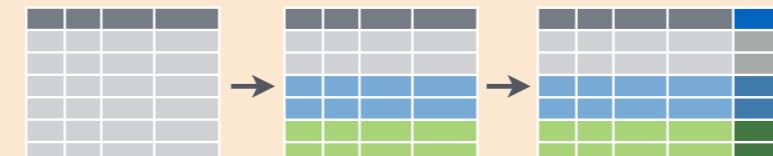
Element-wise max

**pmin**

Element-wise min

**iris %>% group\_by(Species) %>% mutate(...)**

Compute new variables by group.



## Combine Data Sets



### Mutating Joins

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

**dplyr::left\_join(a, b, by = "x1")**

Join matching rows from b to a.

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

**dplyr::right\_join(a, b, by = "x1")**

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

**dplyr::inner\_join(a, b, by = "x1")**

Join data. Retain only rows in both sets.

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

**dplyr::full\_join(a, b, by = "x1")**

Join data. Retain all values, all rows.

### Filtering Joins

x1	x2
A	1
B	2

**dplyr::semi\_join(a, b, by = "x1")**

All rows in a that have a match in b.

x1	x2
C	3

**dplyr::anti\_join(a, b, by = "x1")**

All rows in a that do not have a match in b.



x1	x2
A	1
B	2
C	3

**dplyr::intersect(y, z)**

Rows that appear in both y and z.



**dplyr::union(y, z)**

Rows that appear in either or both y and z.

x1	x2
A	1

**dplyr::setdiff(y, z)**

Rows that appear in y but not z.

x1	x2
B	2
C	3

x1	x2
D	4

x1	x2
A	1
B	2

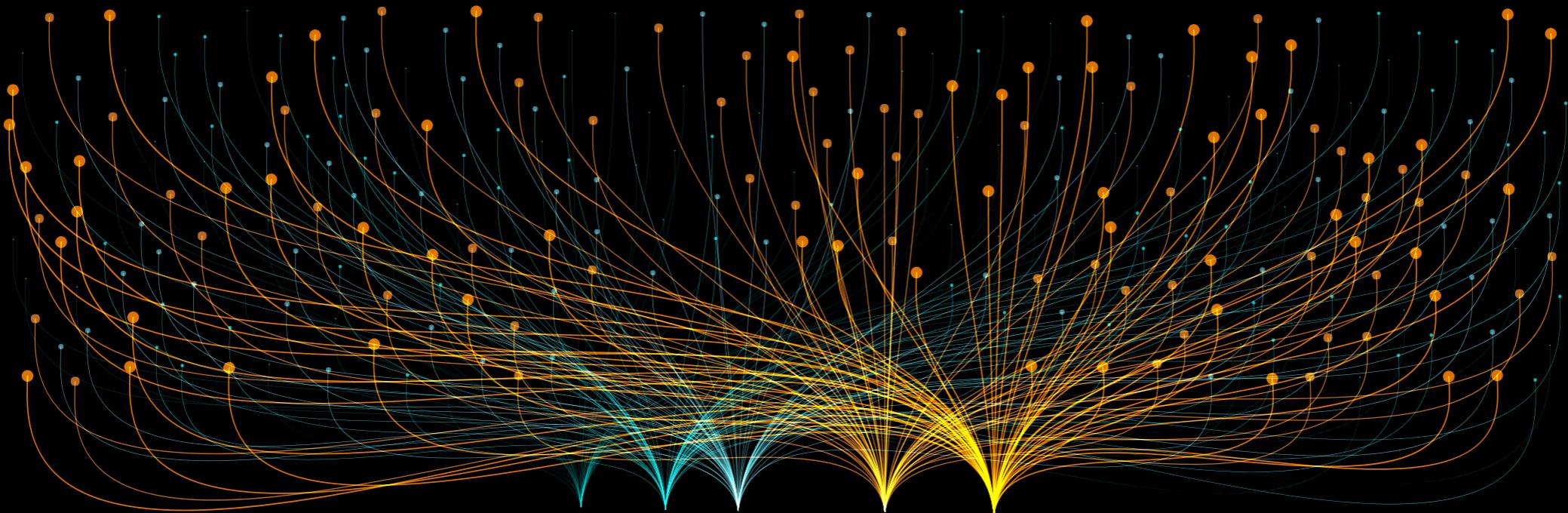
x1	x2
B	2
C	3

x1	x2
D	4

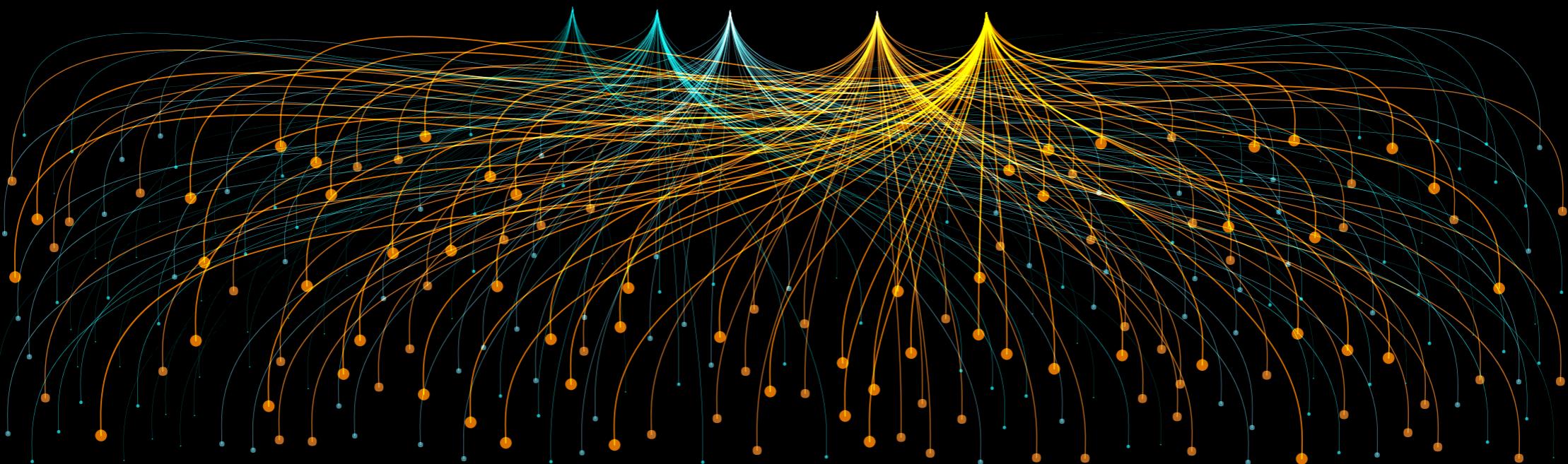
**dplyr::bind\_rows(y, z)**

Append z to y as new rows.

**dplyr::bind\_cols(y, z)**



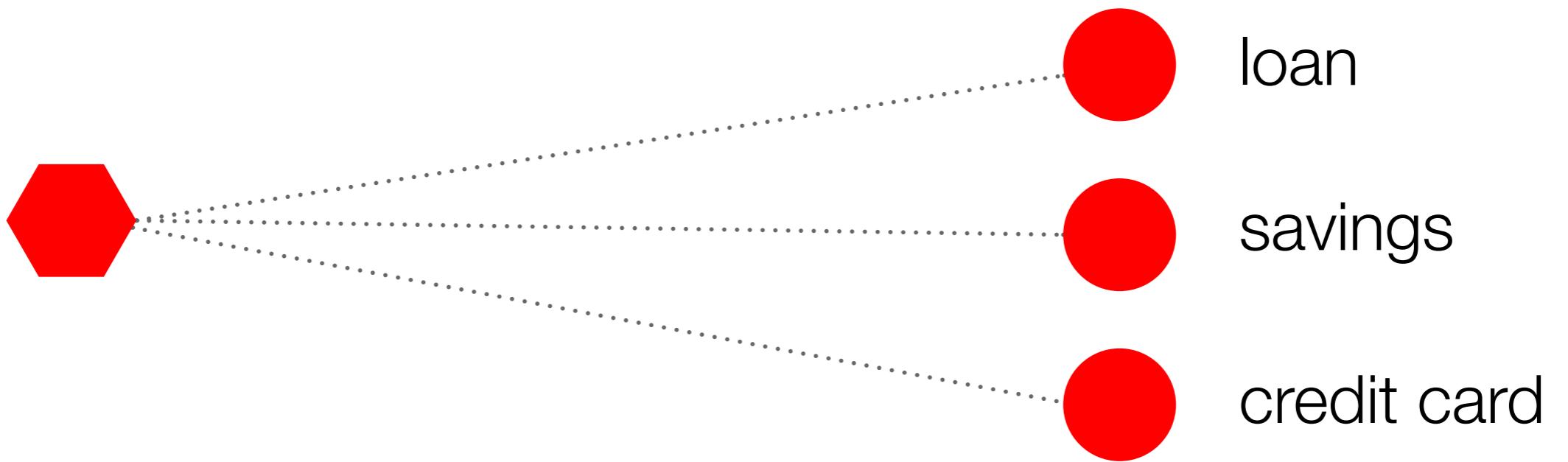
# EXAMPLE DATA



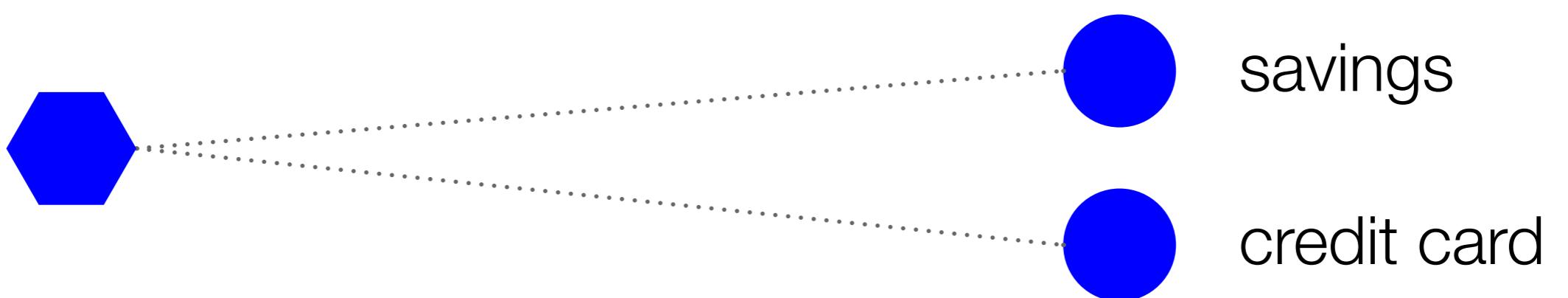
# ONE TO MANY

*One customer to many accounts*

Ellen



Mark



# CUSTOMER TABLE

cust_no (pk)	name	birthdate	gender	address
10	andrew	1980	m	north
15	andrew	1970	m	south
20	mark	1985	m	north
30	ellen	1990	f	south
40	cathy	1987	f	north
50	megan	1978	f	south
60	colin	1992	m	north
70	trung	1990	m	north

# ACCOUNTS TABLE

acct_no (pk)	cust_num (fk)	type	balance	name	birthdate
A	30	loan	400	ellen	1990
B	40	savings	80	cathy	1987
C	30	savings	200	ellen	1990
D	50	savings	400	megan	1978
E	60	loan	90	colin	1992
F	10	loan	200	andrew	1980
G	60	credit_card	120	colin	1992
H	60	savings	40	colin	1992
I	20	credit_card	50	mark	1985
J	40	term_deposit	200	cathy	1987
K	50	loan	300	megan	1978
L	20	savings	300	mark	1985
M	30	credit_card	50	ellen	1990
N	15	savings	20	andrew	1970

# CODE EXAMPLES

```
<div class="inner half-sw">
<h2 class="title">Send <span class="grayed">Withdrawing funds</span></h2>
<h3 class="subtitle">Transfer funds to another bank account</h3>
<ul class="equal">
    <li>Transfer funds to another bank account</li>
    <li>Send bank transfers to more than one bank account</li>
    <li>Get started with a quick, simple and secure transfer</li>
</ul>
<a href="MassPayoutServices.aspx" class="read-more">Read More</a>
<div class="clearfix"></div>
<div class="buttons">
    <div class="button"><a href="https://www.usps.com/usps-transfer-funds-to-another-bank-account">Transfer funds</a></div>
    <div id="generalDemo" class="demo-button">
        Watch<br>
        Demo
    </div>
</div>
<div class="demoContainer" id="generalDemoContainer">
    <iframe width="653" height="480" src="https://www.usps.com/usps-transfer-funds-to-another-bank-account"></iframe>
</div>
</div>
```

```
<div class="receive">
    <div class="inner half-sw">
        <h2 class="title">Receive <span class="grayed">Withdrawing funds</span></h2>
        <h3 class="subtitle">Global solution to receive funds</h3>
        <ul class="equal">
            <li>Receive funds from any <strong>U.S. based company</strong></li>
            <li>Withdraw funds to your <strong>local bank account</strong></li>
            <li>Spend funds with a <strong>Prepaid card</strong></li>
        </ul>
        <a href="ReceiveWithdraw.aspx" class="read-more">Read More</a>
    <div class="clearfix"></div>
    <div class="buttons">
        <div class="button"><a href="https://www.usps.com/usps-receive-withdraw-funds">Receive & Withdraw</a></div>
        <div id="uspsDemo" class="demo-button">
            Watch<br>
            Demo
        </div>
    </div>
</div>
```

# MAGRITTR REVISION [a]

*replace letters “a” with “\*” – demonstrates “.” operator*

*# syntax without magrittr*

```
gsub("a", "*", df_customer$name)
```

*# syntax with magrittr*

*# using the “.” if not the first argument*

```
df_customer$name %>% gsub("a", "*", .)
```



# MAGRITTR REVISION [b]

*add “\_suffix” to each customer name*

*# following statements are equivalent*

```
paste(df_customer$name, "_suffix")
```

*# syntax with magrittr - first argument inserted automatically*

```
df_customer$name %>% paste("_suffix")
```

# DPLYR VERB STRUCTURE

*dplyr verbs have the table name as **first** argument*

*# following statements are equivalent*

```
dplyr::select(df_customer, name, birthdate)
```

*# equivalent with magrittr*

```
df_customer %>% dplyr::select(name, birthdate)
```

# RESULTS

birthdate	name
1980	andrew
1970	andrew
1985	mark
1990	ellen
1987	cathy
1978	megan
1992	colin
1990	trung

# COMPARISON TO SQL

# *SQL Syntax*

SELECT name, birthdate

FROM df\_customer

ORDER BY birthdate

# DPLYR EQUIVALENT

# *Equivalent using dplyr*

```
df_customer %>%  
  select(name, birthdate) %>%  
  arrange(birthdate)
```

# FLEXIBLE OPERATION ORDER

# *equivalent operation - flip verb order*

```
df_customer %>%  
  arrange(birthdate) %>%  
  select(name, birthdate)
```

# RESULTS

name	birthdate
andrew	1970
megan	1978
andrew	1980
mark	1985
cathy	1987
ellen	1990
trung	1990
colin	1992

# ADD A CALCULATED COLUMN

*Using mutate()*

```
# add a calculated 'age' column and then sort by  
# descending order  
  
df_customer %>%  
  mutate(age = 2018 - birthdate) %>%  
  arrange(desc(age))
```

# RESULTS

<b>cust_no</b>	<b>name</b>	<b>birthdate</b>	<b>gender</b>	<b>address</b>	<b>age</b>
15	andrew	1970	m	south	48
50	megan	1978	f	south	40
10	andrew	1980	m	north	38
20	mark	1985	m	north	33
40	cathy	1987	f	north	31
30	ellen	1990	f	south	28
70	trung	1990	m	north	28
60	colin	1992	m	north	26

# AGGREGATION - NO GROUPS

# calculate average birthdate for all rows

```
df_customer %>%  
  summarise(av_birth_date = mean(birthdate))
```

# RESULTS

av\_birth\_date

1984

# AGGREGATION WITH GROUPS

# calculate mean age by gender

```
df_customer %>%
```

```
  mutate(age = 2018 - birthdate) %>%
```

```
  group_by(gender) %>%
```

```
  summarise(av_age = mean(age))
```

# RESULTS

gender	av_age
f	33.0
m	34.6

# COERCION TO TIBBLE

*# Some dplyr verbs implicitly coerce to a Tibble. Tibble is a type of data.frame that has some better print functions especially for large data.frames. The following explicitly converts to Tibble and then back to data.frame*

```
df_customer %>%  
  tibble::as_tibble() %>%  
  base::as_data.frame()
```

# TWO TABLE VERBS – INNER JOIN

```
# Use one primary key and foreign key to join the two  
# tables: df_customer and df_account  
  
df_customer %>%  
  select(-c(name, birthdate)) %>%  
  inner_join(df_account,  
             by = c("cust_no" = "cust_num"))
```

# RESULTS

cust_no	name.x	birthdate.x	gender	address	account_no	type	balance	name.y	birthdate.y
10	andrew	1980	m	north	F	loan	200	andrew	1980
15	andrew	1970	m	south	N	savings	20	andrew	1970
20	mark	1985	m	north	I	credit_card	50	mark	1985
20	mark	1985	m	north	L	savings	300	mark	1985
30	ellen	1990	f	south	A	loan	400	ellen	1990
30	ellen	1990	f	south	C	savings	200	ellen	1990
30	ellen	1990	f	south	M	credit_card	50	ellen	1990
40	cathy	1987	f	north	B	savings	80	cathy	1987
40	cathy	1987	f	north	J	term_deposit	200	cathy	1987
50	megan	1978	f	south	D	savings	400	megan	1978
50	megan	1978	f	south	K	loan	300	megan	1978
60	colin	1992	m	north	E	loan	90	colin	1992
60	colin	1992	m	north	G	credit_card	120	colin	1992
60	colin	1992	m	north	H	savings	40	colin	1992

# DESELECTING COLUMNS

```
# We want to exclude two columns from df_customer  
  
df_customer %>%  
  select(-c(name, birthdate)) %>%  
  inner_join(df_account,  
             by = "cust_no" = “cust_num”)
```

# RESULTS

cust_no	gender	address	account_no	type	balance	name	birthdate
10	m	north	F	loan	200	andrew	1980
15	m	south	N	savings	20	andrew	1970
20	m	north	I	credit_card	50	mark	1985
20	m	north	L	savings	300	mark	1985
30	f	south	A	loan	400	ellen	1990
30	f	south	C	savings	200	ellen	1990
30	f	south	M	credit_card	50	ellen	1990
40	f	north	B	savings	80	cathy	1987
40	f	north	J	term_deposit	200	cathy	1987
50	f	south	D	savings	400	megan	1978
50	f	south	K	loan	300	megan	1978
60	m	north	E	loan	90	colin	1992
60	m	north	G	credit_card	120	colin	1992
60	m	north	H	savings	40	colin	1992

# ANTI-JOIN

# What customers do not have accounts.

# Joining using multiple columns

```
df_customer %>%  
  anti_join(df_account,  
            by = c("name" = "name",  
                  "birthdate" = "birthdate"))
```

# RESULTS

cust_no	name	birthdate	gender	address
70	trung	1990	m	north

# SELECTING DISTINCT ROWS

*# The following two statements are equivalent*

```
df_account %>% distinct(name, birthdate)
```

```
df_account %>% select(name, birthdate) %>%  
distinct()
```

# RESULTS

name	birthdate
ellen	1990
cathy	1987
megan	1978
colin	1992
andrew	1980
mark	1985
andrew	1970

# FILTERING USING EXPRESSIONS

```
# northern men  
df_customer %>%  
  filter(address == "north" & gender == "m")
```

```
# this is the same as the above  
df_customer %>%  
  filter(address == ) %>%  
  filter(gender == "m")
```

# FILTERING USING VECTORS

*# What is the total account balance for  
Cathy, Mark & Megan*

```
df_account %>%  
  filter(name %in% c("cathy", "mark", "megan")) %>%  
  summarise(total_balance = sum(balance))
```

# RESULTS

total\_balance

1330

# VECTOR NEGATION

# What is the total account balance for the rest

```
df_account %>%  
  filter(!name %in% c("cathy", "mark", "megan")) %>%  
  summarise(total_balance = sum(balance))
```

# RESULTS

total\_balance

1120

# SLICING GROUPS

```
# For each customer what is the account with  
# the highest value  
# n() counts the number of rows in each group
```

```
df_account %>%  
  group_by(cust_num) %>%  
    arrange(balance) %>%  
    slice(n())
```

# RESULTS

account_no	cust_num	type	balance	name	birthdate
F	10	loan	200	andrew	1980
N	15	savings	20	andrew	1970
L	20	savings	300	mark	1985
A	30	loan	400	ellen	1990
J	40	term_deposit	200	cathy	1987
D	50	savings	400	megan	1978
G	60	credit_card	120	colin	1992

# RENAMING COLUMNS

# *Select two columns*

# *and rename them.*

```
df_customer %>%
```

```
  select(name, birthdate) %>%
```

```
  rename(new_name = name, new_birthdate = birthdate))
```

# ADVANCED FUNCTIONS

# ADVANCED SLICING

```
# For customers with 3 or more accounts  
# list the smallest and largest accounts
```

```
df_account %>%  
  group_by(cust_num) %>%  
  filter(n() >= 3) %>%  
  arrange(balance) %>%  
  slice(c(1, n()))
```

# RESULTS

account_no	cust_num	type	balance	name	birthdate
M	30	credit_card	50	ellen	1990
A	30	loan	400	ellen	1990
H	60	savings	40	colin	1992
G	60	credit_card	120	colin	1992

# ADVANCED GROUP OPERATIONS

```
# Get all accounts associated with a customer who has  
# a loan account
```

```
df_account %>%  
  group_by(cust_num) %>%  
  filter(any(type == "loan")) %>%  
  arrange(cust_num)
```

# RESULTS

account_no	cust_num	type	balance	name	birthdate
F	10	loan	200	andrew	1980
A	30	loan	400	ellen	1990
C	30	savings	200	ellen	1990
M	30	credit_card	50	ellen	1990
D	50	savings	400	megan	1978
K	50	loan	300	megan	1978
E	60	loan	90	colin	1992
G	60	credit_card	120	colin	1992
H	60	savings	40	colin	1992

# WIDE AND LONG FORMATS

*Yes! you can even process LIDAR data using dplyr*



*Black mountain in Canberra, Australia. Rendered using 12,000,000 3D points*

# LONG TO WIDE

*And back again using `tidyr::spread()` and `tidyr::gather()`*

---

```
# lets use account type as column headings
library(tidyr)

df_account_wide <-
  df_account %>%
  mutate(uniq_name = paste0(name, "_", birthdate)) %>%
  select(uniq_name, type, balance) %>%
  tidyr::spread(key = type, value = balance, fill = 0)
```

# RESULTS

uniq_name	credit_card	loan	savings	term_deposit
andrew_1970	0	0	20	0
andrew_1980	0	200	0	0
cathy_1987	0	0	80	200
colin_1992	120	90	40	0
ellen_1990	50	400	200	0
mark_1985	50	0	300	0
megan_1978	0	300	400	0

# WIDE TO LONG

*Using `tidyr::gather()`*

```
# go from wide format to long; sample 10 rows
```

```
set.seed(123)  
df_account_wide %>%  
  tidyr::gather(key = type,  
                value = value,  
                -unique_name) %>%  
  sample_n(10)
```

# RESULTS

uniq_name	type	value
andrew_1980	loan	200
andrew_1970	term_deposit	0
colin_1992	loan	90
andrew_1980	term_deposit	0
colin_1992	term_deposit	0
andrew_1980	credit_card	0
ellen_1990	loan	400
ellen_1990	savings	200
mark_1985	term_deposit	0
megan_1978	term_deposit	0

# NON STANDARD EVALUATION

# NON STANDARD EVALUATION

Saves *typing* do not have to use quotes

```
# Non standard evaluation - no quotes
```

```
df_customer %>%  
  select(name, birthdate)
```

```
# Standard evaluation - uses quotes (note the underscore)
```

```
df_customer %>%  
  select_("name", "birthdate")
```

# NON STANDARD Vs STANDARD EVALUATION

*Non standard evaluation is useful for interactive programming as it saves time.*

*But what about programmatically specifying dplyr's arguments....*

# STANDARD EVALUATION

*A simple example*

```
# Create a simple list of column names
lst_names <- c("name", "birthdate") %>% lapply(as.name)

# Set the .dots argument equal to the list
df_customer %>% select_(.dots = lst_names)
```

# RESULT

name	birthdate
andrew	1980
andrew	1970
mark	1985
ellen	1990
cathy	1987
megan	1978
colin	1992
trung	1990

# A MORE USEFUL EXAMPLE [1]

**Firstly** create a more useful data set by  
joining the two tables

```
df_data <- df_customer %>%  
  select(-c(name, birthdate)) %>%  
  inner_join(df_account,  
             by = c("cust_no" = "cust_num"))
```

# RESULT

cust_no	gender	address	account_no	type	balance	name	birthdate
10	m	north	F	loan	200	andrew	1980
15	m	south	N	savings	20	andrew	1970
20	m	north	I	credit_card	50	mark	1985
20	m	north	L	savings	300	mark	1985
30	f	south	A	loan	400	ellen	1990
30	f	south	C	savings	200	ellen	1990
30	f	south	M	credit_card	50	ellen	1990
40	f	north	B	savings	80	cathy	1987
40	f	north	J	term_deposit	200	cathy	1987
50	f	south	D	savings	400	megan	1978
50	f	south	K	loan	300	megan	1978
60	m	north	E	loan	90	colin	1992
60	m	north	G	credit_card	120	colin	1992
60	m	north	H	savings	40	colin	1992

# A MORE USEFUL EXAMPLE [2]

**Secondly** create a standard evaluation function

```
fn_demo_dots <- function(. . .) {  
  
  lst_group_by <- list(. . .) %>% lapply(as.name)  
  
  df_data %>%  
    group_by_(.dots = list_group_by) %>%  
    summarise(total = sum(balance))  
}
```

# A MORE USEFUL EXAMPLE [3]

**Finally** apply the function

```
# group_by address and gender
```

```
fn_demo_dots("address","gender")
```

# RESULT

address	gender	total
north	f	280
north	m	800
south	f	1350
south	m	20

# A MORE USEFUL EXAMPLE [4]

*Apply the function again*

```
# group_by type  
fn_demo_dots("type")
```

*For more complex usage of using standard evaluation  
with dplyr see the lazyeval library*

# RESULT

type	total
credit_card	220
loan	990
savings	1040
term_deposit	200

# LAZYEVAL EXAMPLE [1]

```
# filter table where (loan) type == str_type
```

```
library(lazyeval)
```

```
fn_filter_account <- function(df, str_type) {  
  filter_condition <- lazyeval::interp(~x == y, x =  
    as.name("type"), y = str_type)  
  df_rtn <- df %>% filter_(filter_condition)  
  return(df_rtn)  
}
```

# LAZYEVAL EXAMPLE [2]

# filter the accounts table by: type == loan

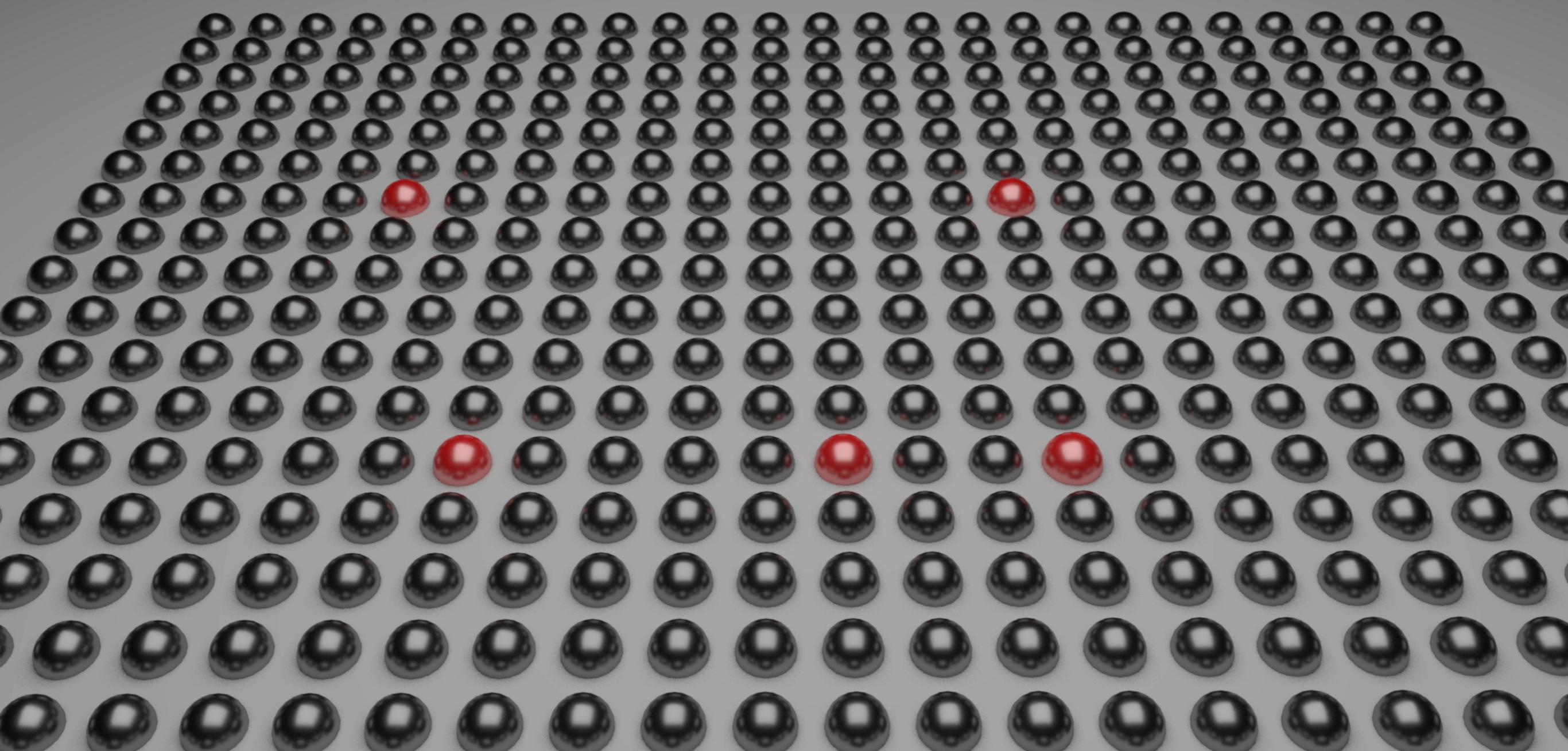
# call the function we created previously

```
fn_filter_account(df_account, "loan")
```

# RESULT

account_no	cust_num	type	balance	name	birthdate
A	30	loan	400	ellen	1990
E	60	loan	90	colin	1992
F	10	loan	200	andrew	1980
K	50	loan	300	megan	1978

# MULTIPLE ROWS FROM GROUPS



# MULTIPLE ROWS FROM A GROUP

*Using do()*

```
# get the first two rows from each group
```

```
df_data %>%  
  group_by(type) %>%  
  do(head(., 2))
```

# RESULT

cust_no	gender	address	account_no	type	balance	name	birthdate
20	m	north	I	credit_card	50	mark	1985
30	f	south	M	credit_card	50	ellen	1990
10	m	north	F	loan	200	andrew	1980
30	f	south	A	loan	400	ellen	1990
15	m	south	N	savings	20	andrew	1970
20	m	north	L	savings	300	mark	1985
40	f	north	J	term_deposit	200	cathy	1987

# MORE COMPLEX EXAMPLE

*Fit a linear model to each group using do()*

```
# regress balance against birthdate  
df_models <- df_data %>%  
  group_by(type) %>%  
  do(mod = lm(balance ~ birthdate, data = .))
```

```
# problem is how to extract results  
df_models
```

# RESULT

*Each row contains an S3 model*

type	mod
credit_card	<a href="#"><code>&lt;S3: lm&gt;</code></a>
loan	<a href="#"><code>&lt;S3: lm&gt;</code></a>
savings	<a href="#"><code>&lt;S3: lm&gt;</code></a>
term_deposit	<a href="#"><code>&lt;S3: lm&gt;</code></a>

# EXTRACTING LIST INFORMATION [1]

*broom: “tidying statistical models into data.frames”*

```
# broom::glance() returns one row per data.frame
```

```
df_models %>%  
  broom::glance(mod)
```

# RESULT

*Selection of columns - output of broom::glance()*

type	r.squared	sigma	statistic	p.value	AIC	BIC	df.residual
credit_card	0.51	39.62	1.08	0.48	33.29	30.59	1
loan	0.02	160.59	0.05	0.83	55.21	53.36	2
savings	0.001	171.69	0.01	0.93	82.34	81.71	4
term_deposit	0	NA	NA	NA	NA	NA	0

# EXTRACTING LIST INFORMATION [2]

*broom: “tidying statistical models into data.frames”*

```
# broom::tidy() returns multiple rows per data.frame
```

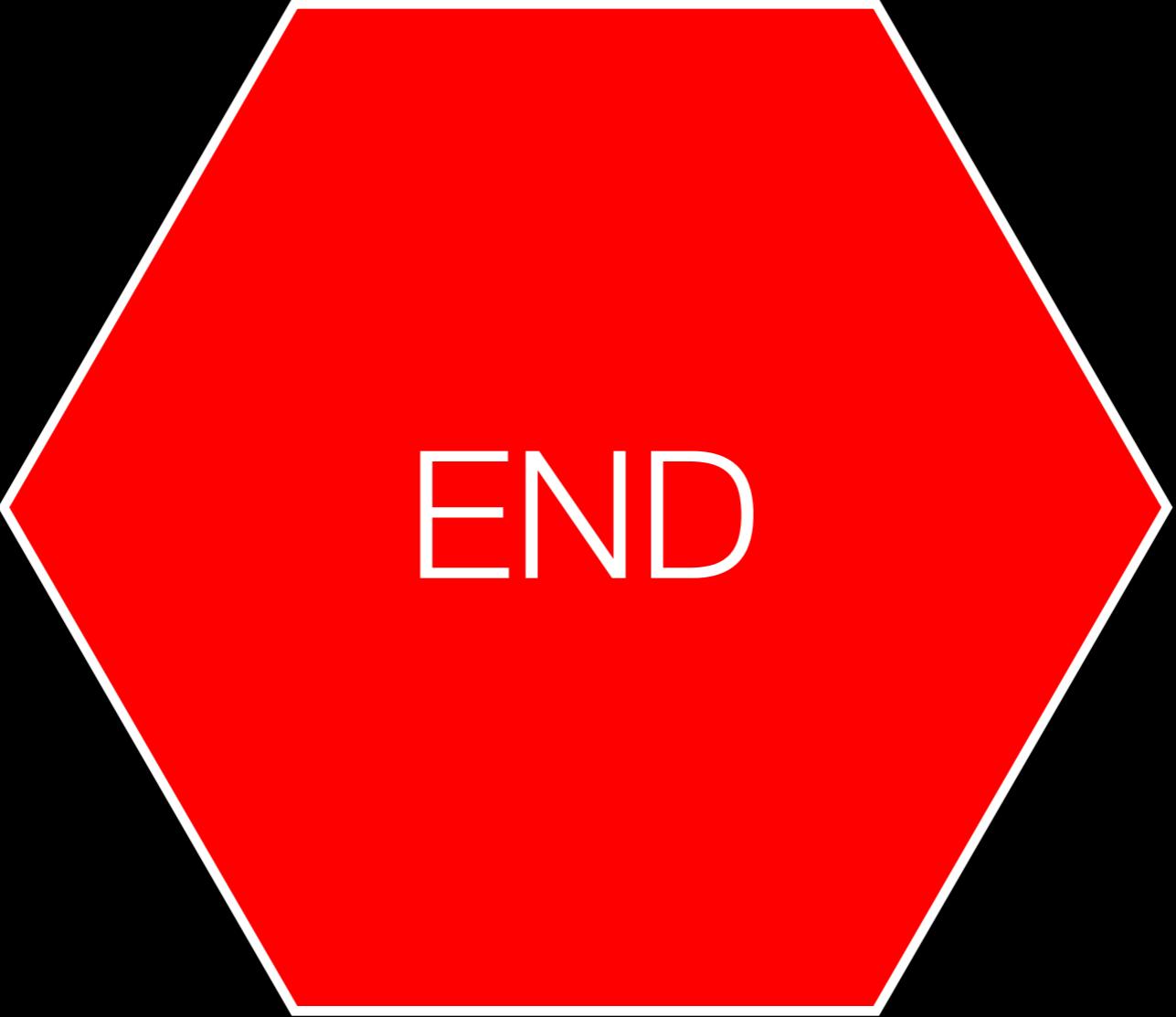
```
df_models %>%
```

```
  broom::tidy(mod)
```

# RESULT

*Output of broom::tidy()*

type	term	estimate	std.error	statistic	p.value
credit_card	(Intercept)	-15991.66	15458.57	-1.03	0.48
credit_card	birthdate	8.07	7.77	1.039	0.48
loan	(Intercept)	6551.21	26204.00	0.25	0.82
loan	birthdate	-3.17	13.20	-0.240	0.83
savings	(Intercept)	1761.81	18434.69	0.095	0.92
savings	birthdate	-0.80	9.29	-0.086	0.93
term_deposit	(Intercept)	200	NA	NA	NA



END