

Poisson regression

Overview

Quick review of logistic regression

Poisson regression

Data cleaning and wrangling using the tidyverse (time permitting)

- Processing dates (lubridate)
- Manipulating strings (stringr)
- Reshaping data (tidyr)
- Joining data frames (dplyr)

Announcement

Homework 9 is due Sunday November 17th

Keep thinking about your final project!

- Find data ASAP that you can use in your project and make sure you can load it into R!



Very quick review of logistic regression

In **logistic regression** we try to predict if a case belongs to one of two categories

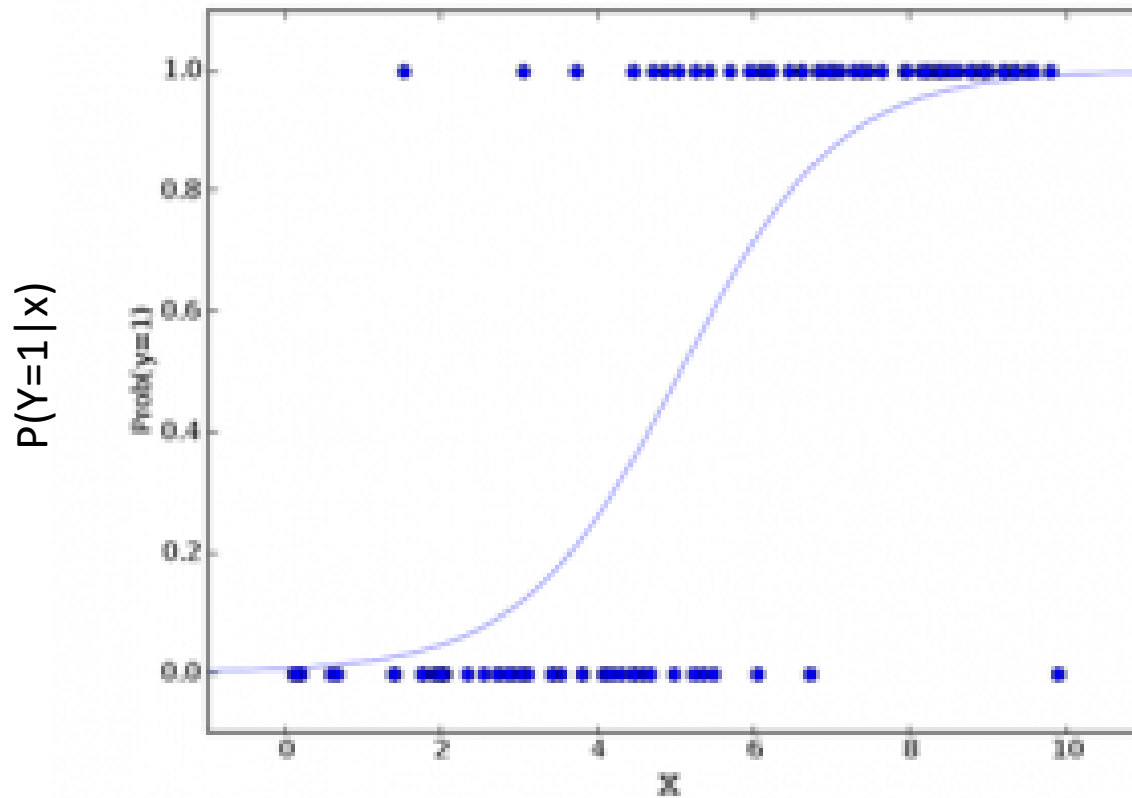
To do this we model the log-odds as a linear function of predictor variables:

$$\log\left(\frac{P(Y=1|x)}{1-P(Y=1|x)}\right) = \beta_0 + \beta_1 \cdot x$$

If we write the above equation in terms of the probability of being in class *1* we get the logistic function:

$$P(Y = 1|x) = \frac{e^{\beta_0 + \beta_1 \cdot x}}{1 + e^{\beta_0 + \beta_1 \cdot x}}$$

Very quick review of logistic regression



$$P(Y = 1|x) = \frac{e^{\beta_0 + \beta_1 \cdot x}}{1 + e^{\beta_0 + \beta_1 \cdot x}}$$

Very quick review: assessing model fit

To assess model fit, we can use proportions as probability estimates

For example, if we want to estimate the probability a professor who makes \$75,000 is a Full professor we can use

- $P(Y = \text{Full} \mid \text{salary} = \$75,000) \approx \hat{p}_{\text{Full} \mid \$75k}$

In our data, we don't have that many professors who make exactly \$75,000 so we could estimate this proportion by looking in the range of \$70,000-\$80,000

We can do this for intervals [\$30k \$40k), [\$40k \$50k), ..., [\$130k \$140k]

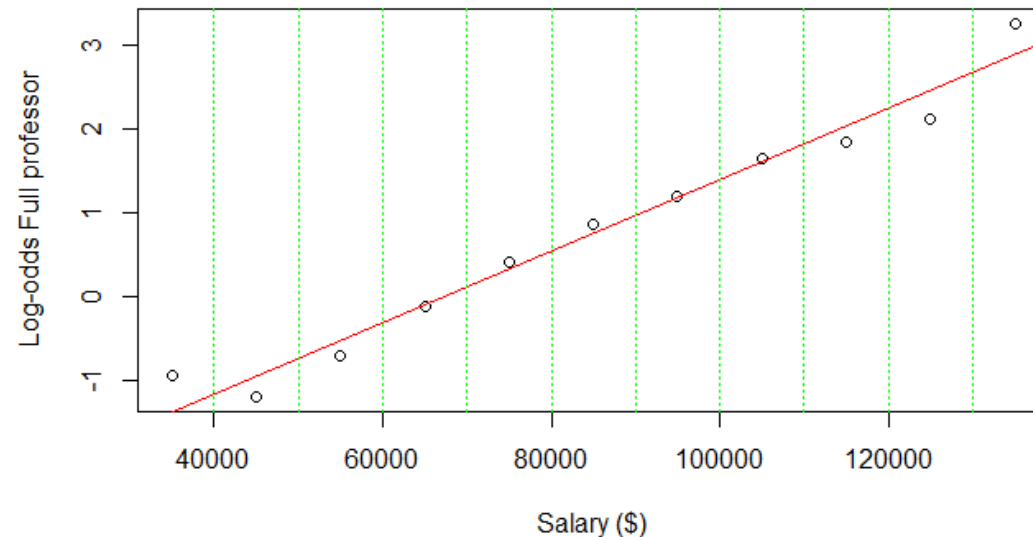
- $\hat{p}_{\text{Full} \mid \$35k}, \hat{p}_{\text{Full} \mid \$45k}, \dots, \hat{p}_{\text{Full} \mid \$135k}$

Very quick review: assessing model fit

We can then convert these probability estimates into **log-odds** using:

$$\log\left(\frac{P(Y=Full|salary)}{1-P(Y=Full|salary)}\right)$$

If logistic regression is appropriate, there should be a **linear** relationship between our estimated **log-odds** and **salary**



Very quick review: multiple logistic regression

We can easily extend our logistic regression model to include multiple explanatory variables

$$\log\left(\frac{P(Y=1|x)}{1-P(Y=1|x)}\right) = \hat{\beta}_0 + \hat{\beta}_1 \cdot x_1 + \hat{\beta}_2 \cdot x_2 + \dots + \hat{\beta}_k \cdot x_k$$

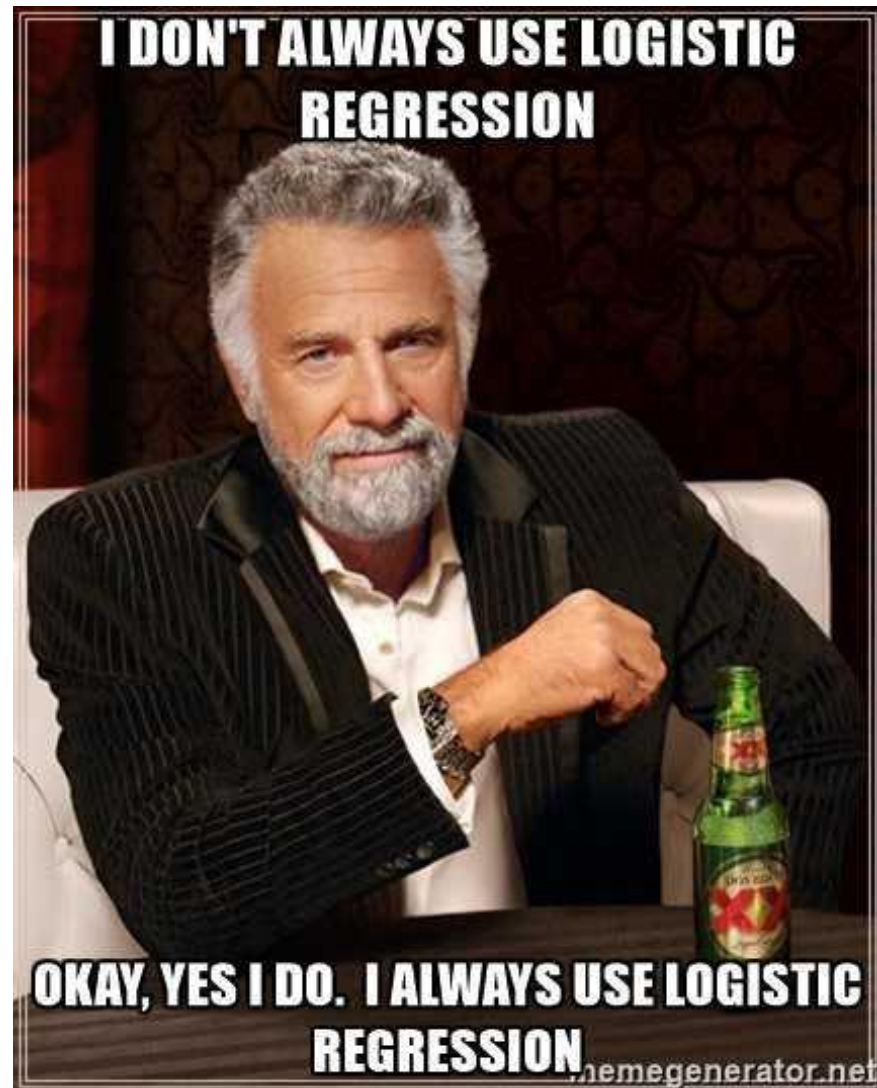
When using a categorical predictor, x_2 , in a logistic regression model, the exponential of the regression coefficient $e^{\hat{\beta}_2}$ is the **odds ratio**

- Tells us how many times greater the odds are when $x_2 = 1$ vs. when $x_2 = 0$

We can fit logistic regression models in R using the `glm()` function

```
> lr_fit <- glm(rank_name ~ salary_tot, data = assistant_full_data, family = "binomial")
```


Questions?



Poisson regression

Summary of linear regression

We can summarize the linear regression model as:

$$Y_i = \mu_i + \varepsilon_i \quad \text{where } \varepsilon_i \sim N(0, \sigma_\varepsilon)$$

$$\mu_i = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

Equivalently, $Y_i \sim N(\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k, \sigma_\varepsilon)$

Generalized linear models

We can summarize the linear regression model as:

$$Y_i = \mu_i + \varepsilon_i \quad \text{where } \varepsilon_i \sim N(0, \sigma_\varepsilon)$$
$$\mu_i = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

In generalized linear models, we generalize the model to:

$$Y_i \sim f(y|\theta_i) \quad \text{where } f(y|\theta_i) \text{ is some probability distribution}$$
$$\theta_i = g^{-1}(\beta_0 + \beta_1 x_1 + \dots + \beta_k x_k)$$

g^{-1} is called an "inverse link function"
Links "linear predictor" to parameters

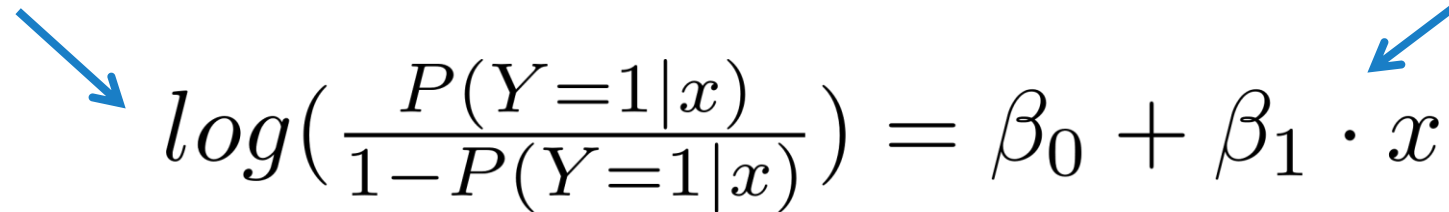
We choose a particular "family" of distributions (e.g., Poisson, binomial, etc.)

Example: logistic regression

In logistic regression we model whether a case belongs to one of two categories

- $P(Y = 0 \mid \mathbf{x})$ or $P(Y = 1 \mid \mathbf{x})$

The logit function (log-odds) is a "link function"

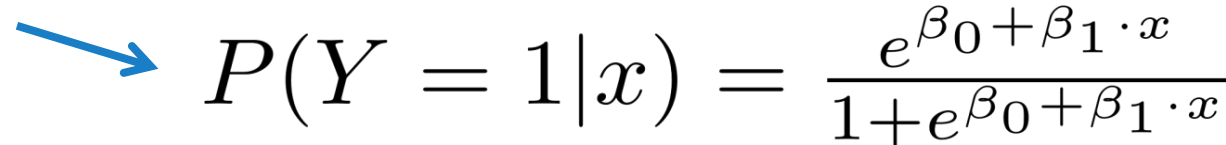

$$\log\left(\frac{P(Y=1|x)}{1-P(Y=1|x)}\right) = \beta_0 + \beta_1 \cdot x$$

The diagram shows the equation with a blue arrow pointing from the text 'The logit function (log-odds) is a "link function"' to the log function, and another blue arrow pointing from the text '"Linear predictor"' to the expression $\beta_0 + \beta_1 \cdot x$.

"Linear predictor"

Inverse link function
(logistic function)

Solving for $P(Y = 1 \mid x)$


$$P(Y = 1|x) = \frac{e^{\beta_0 + \beta_1 \cdot x}}{1 + e^{\beta_0 + \beta_1 \cdot x}}$$

The diagram shows the equation with a blue arrow pointing from the text 'Solving for $P(Y = 1 \mid x)$ ' to the left side of the equation, and another blue arrow pointing from the text 'Inverse link function (logistic function)' to the right side of the equation.

Family is Bernoulli distribution
(binomial with $n = 1$)


$$Y_i \sim \text{Bernoulli}(P(Y = 1|x))$$

The diagram shows the equation with a blue arrow pointing from the text 'Family is Bernoulli distribution (binomial with $n = 1$)' to the equation.

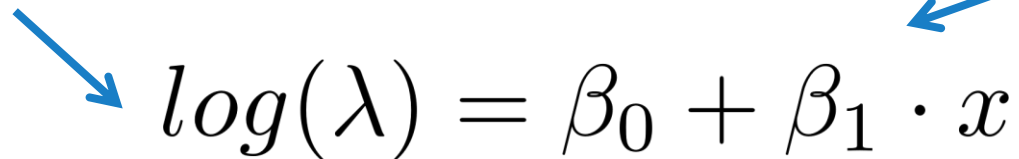
R: `glm_fit <- glm(y ~ x, family = binomial(link = logit))`

Poisson regression

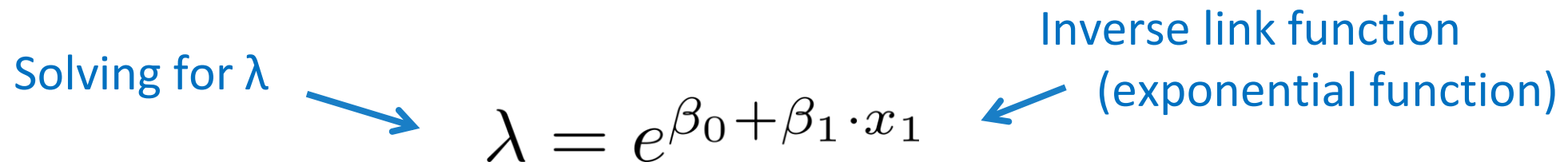
In Poisson regression we model counts

- i.e., integer values: 0, 1, 2, 3, ...

The log is the "link function"


$$\log(\lambda) = \beta_0 + \beta_1 \cdot x$$

Solving for λ


$$\lambda = e^{\beta_0 + \beta_1 \cdot x_1}$$

Family is Poisson distributions


$$Y_i \sim \text{Poisson}(\lambda)$$

R: `glm_fit <- glm(y ~ x, family = Poisson(link = log))`

Poisson distributions

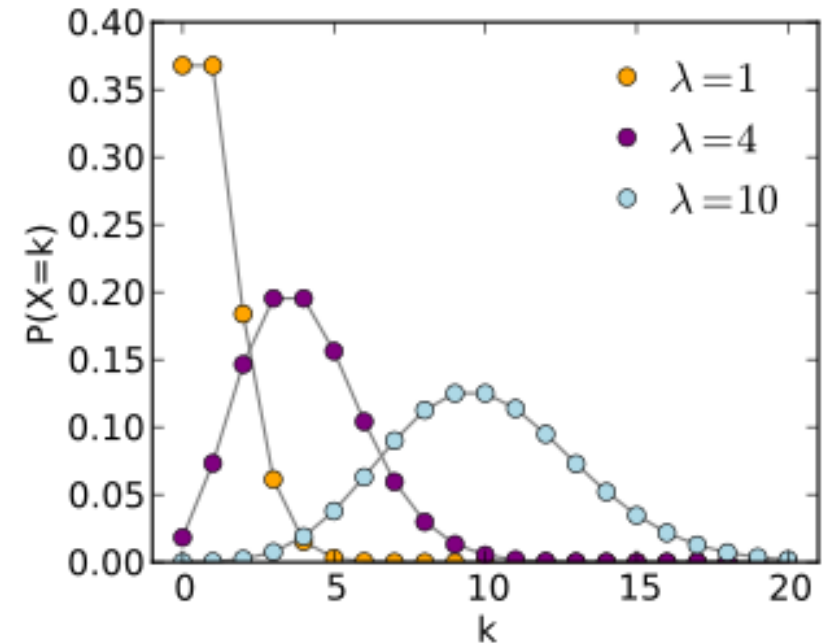
A Poisson distribution is a probability distribution over non-negative integers

- i.e., over values 0, 1, 2, 3, ...

Poisson distributions have a single parameter λ

$$X \sim \text{Pois}(\lambda)$$

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad k = 0, 1, 2, \dots$$



- Density: `dpois()`
- Cumulative distribution: `ppois()`
- Random number: `rpois()`

Poisson processes

Poisson distributions models the number of outcomes that have occurred from a **Poisson process**

A **Poisson process** is a stochastic process where:

- Events (random outcome) occur at a fixed rate (λ)
- Every event is independent of the other events

Examples of Poisson processes?

Side note: Maximum likelihood estimate (MLE)

When building regression models, we need a way to estimate parameters

The "true" underlying model is:

$$y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \dots + \beta_k \cdot x_k + \epsilon$$

We estimate coefficients using a data set to make predictions \hat{y}

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \cdot x_1 + \hat{\beta}_2 \cdot x_2 + \dots + \hat{\beta}_k \cdot x_k$$

For GLMs, the maximum likelihood estimates (MLE) is used to estimate the regression coefficients:

- MLEs find the parameters that make the data as likely as possible
 - (For linear regression with normal errors, MLE gives the same coefficient estimates as least squares)

Example: Roy Kent saying f#ck

Ted Lasso was a Apple TV+ series that aired from July 2021 to March 2023

One of the main characters on the show was Roy Kent, who tended to say f#ck frequently

In different episodes of the show Roy was:

- A coach
- Dated Keeley Jones

Let's use Poisson regression to assess if Roy had a tendency to say f#ck more when he was **coaching** and/or when he was **dating** Keeley



Example from season 2

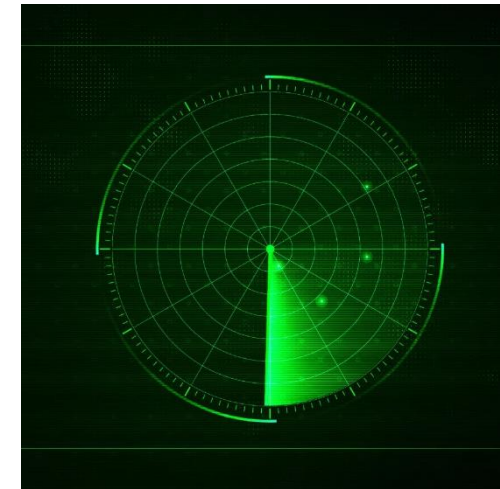
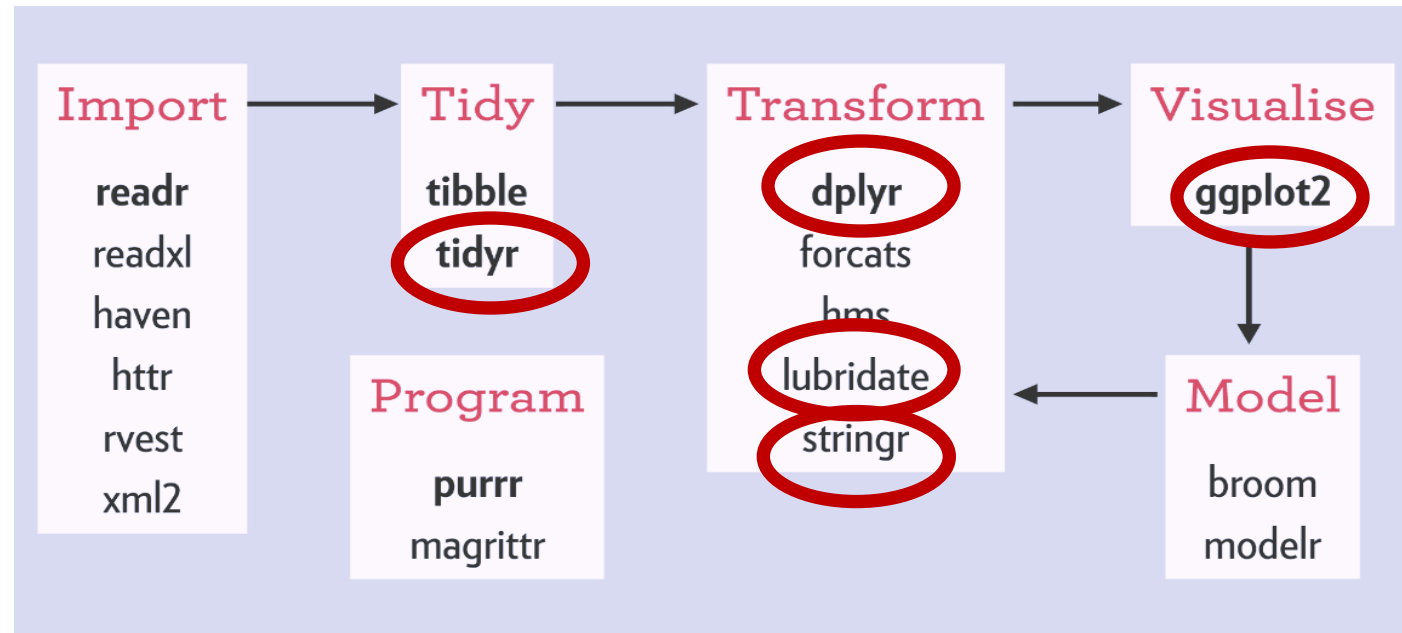
Let's try it in R...

Tidyverse packages useful for your projects

Tidyverse packages useful for your projects

The packages share a common design philosophy

- Most written by Hadley Wickham



The [posit cheat sheets](#) can be very useful

lubridate



lubridate is a package that makes it easier to process dates

```
library(lubridate)    # install.packages(lubridate)
```

There are functions to parse strings and numbers into dates

```
some_date <- ymd(20240712)  
current_date <- mdy("11/14/24")
```

There are functions to extract information from dates:

```
year(current_date)
```

stringr

stringr is a package for manipulate character strings

```
library(stringr)
```



There are many useful functions in the stringr package

- Perhaps we can discuss stringr in more detail later in the semester...

Particularly useful is a function for replacing parts of a string:

```
str_replace_all("one fish, two fish, red fish, blue fish", "fish", "cat")
```

You can use **regular expressions** to make the string matching much more powerful

Let's try it in R...

tidyr for pivoting data

Wide vs. Long data

Plotting data using ggplot requires that data is in the right format

- i.e., requires data transformations

Often this involves converting data from a **wide format** to **long format**

Wide data

Person	Age	Height
Bob	32	72
Alice	24	65
Steve	64	70

Narrow data

Person	name	value
Bob	Age	32
Bob	Height	72
Alice	Age	24
Alice	Height	65
Steve	Age	64
Steve	Height	70

`library(tidyr)`

tidyr::pivot_longer()

pivot_longer(df, cols) converts data from **wide** to **long**

- Takes multiple columns and converts them into two columns: name and value
 - Column names become categorical variable levels of a new variable called **name**
 - The data in rows become entries in a variable called **value**

Wide data

Person	Age	Height
Bob	32	72
Alice	24	65
Steve	64	70



Long data

Person	name	value
Bob	Age	32
Bob	Height	72
Alice	Age	24
Alice	Height	65
Steve	Age	64
Steve	Height	70

tidyr::pivot_wider()

`pivot_wider(df, names_from, values_from)` converts data from long to wide

- Turns the levels of categorical data into columns in a data frame

Narrow data

person	name	value
Bob	Age	32
Bob	Height	72
Alice	Age	24
Alice	Height	65
Steve	Age	64
Steve	Height	70

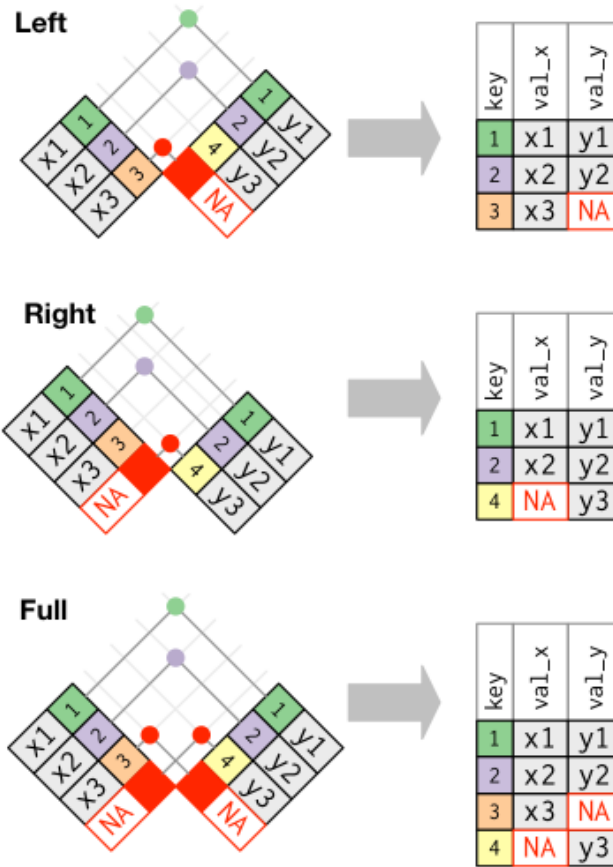


Wide data

Person	Age	Height
Bob	32	72
Alice	24	65
Steve	64	70

Let's try it in R...

Joining data frames



Left and right tables

Suppose we have two data frames called *x* and *y*

- *x* have two variables called *key_x*, and *val_x*
- *y* has two variables called *key_y* and *val_y*

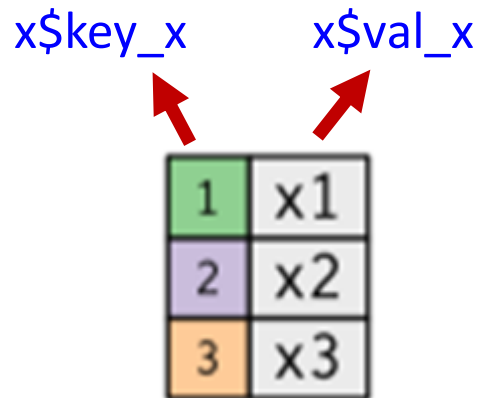


Diagram illustrating Data frame x. The table has two columns: *key_x* (labeled *x\$key_x*) and *val_x* (labeled *x\$val_x*). The rows are indexed 1, 2, and 3.

<i>x\$key_x</i>	<i>x\$val_x</i>
1	x1
2	x2
3	x3

Data frame x

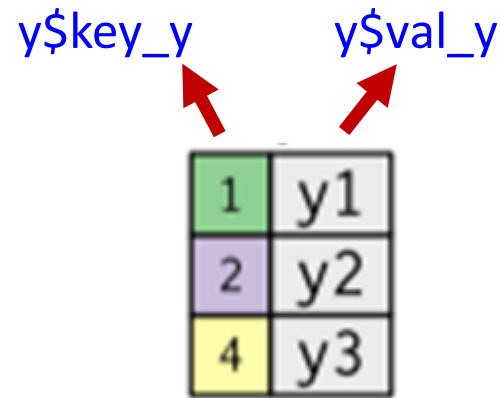


Diagram illustrating Data frame y. The table has two columns: *key_y* (labeled *y\$key_y*) and *val_y* (labeled *y\$val_y*). The rows are indexed 1, 2, and 4.

<i>y\$key_y</i>	<i>y\$val_y</i>
1	y1
2	y2
4	y3

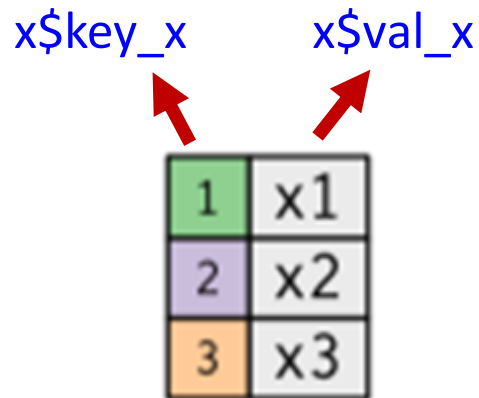
Data frame y

`SDS230:download_data('x_y_join.rda')`

Left and right tables

Suppose we have two data frames called x and y

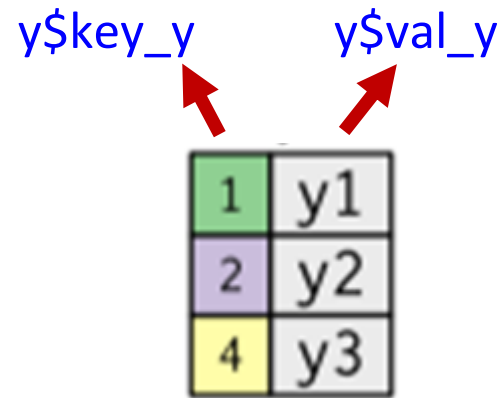
- x have two variables called `key_x`, and `val_x`
- y has two variables called `key_y` and `val_y`



A diagram of Data frame x. It consists of a 3x2 grid of cells. The first column contains the values 1, 2, and 3, each in a colored box (green, purple, and orange respectively). The second column contains the values x1, x2, and x3, each in a grey box. Above the first column, the text `x$key_x` has a red arrow pointing to the first cell. Above the second column, the text `x$val_x` has a red arrow pointing to the first cell.

1	x1
2	x2
3	x3

Data frame x



A diagram of Data frame y. It consists of a 3x2 grid of cells. The first column contains the values 1, 2, and 4, each in a colored box (green, purple, and yellow respectively). The second column contains the values y1, y2, and y3, each in a grey box. Above the first column, the text `y$key_y` has a red arrow pointing to the first cell. Above the second column, the text `y$val_y` has a red arrow pointing to the first cell.

1	y1
2	y2
4	y3

Data frame y

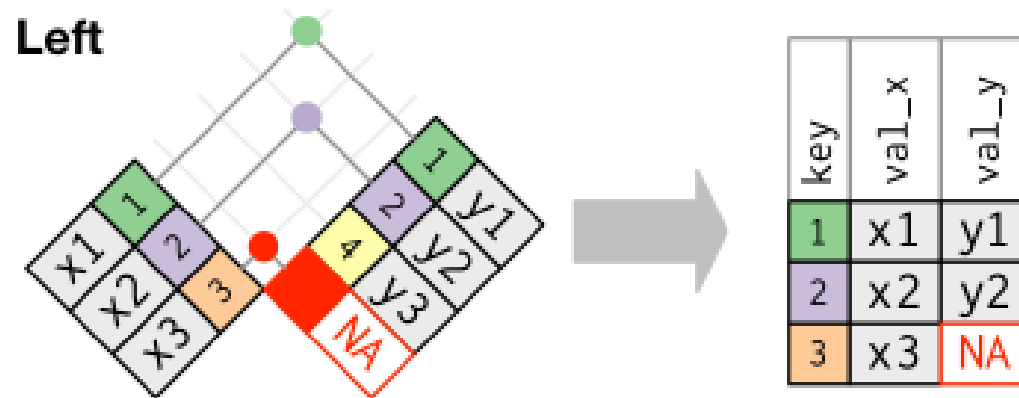
Joins have the general form:

```
join(x, y, by = c("key_x" = "key_y"))
```

Left joins

Left joins keep all rows in the left table.

Data from right table is added when there is a matching key, otherwise NA is added.

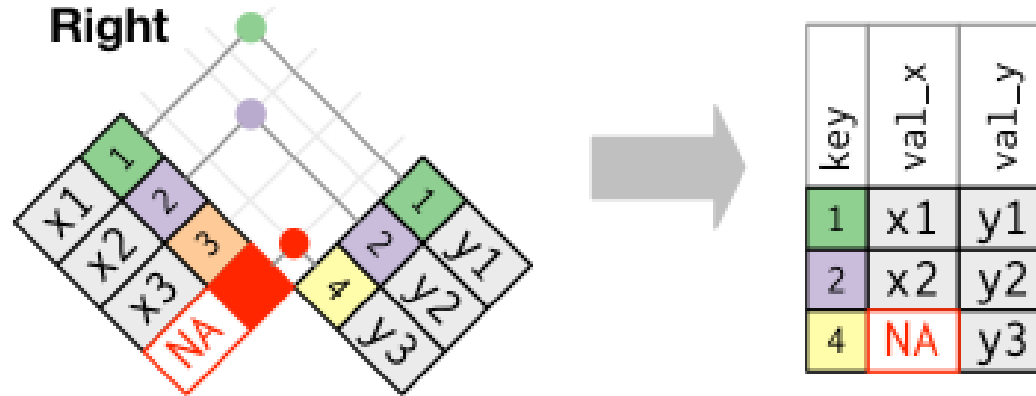


```
> left_join(x, y, by = c("key_x" = "key_y"))
```

Right joins

Right joins keep all rows in the right table.

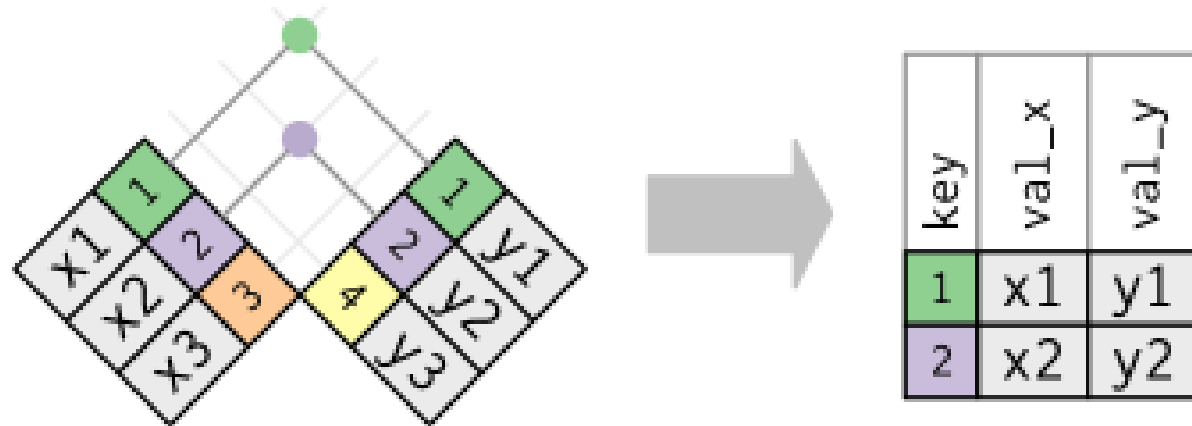
Data from left table added when there is a matching key, otherwise NA as added.



```
> right_join(x, y, by = c("key_x" = "key_y"))
```


Inner joins

Inner joins only keep rows in which there are matches between the keys in both tables.

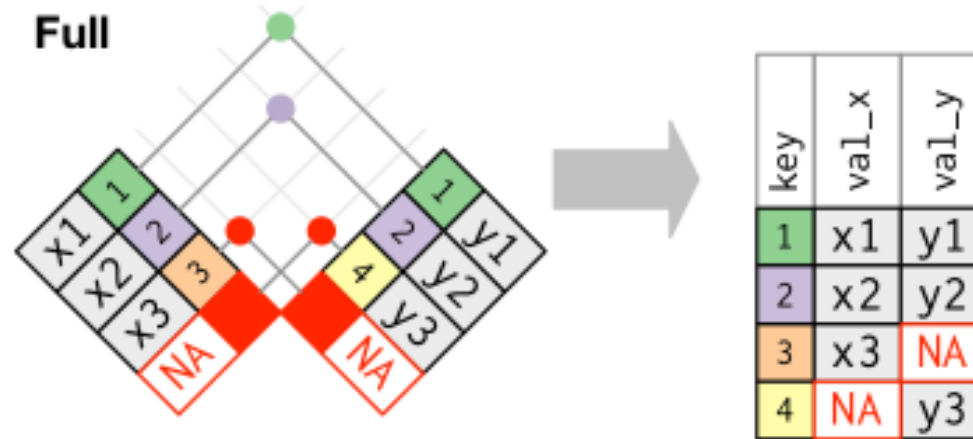


```
> inner_join(x, y, by = c("key_x" = "key_y"))
```

Full joins

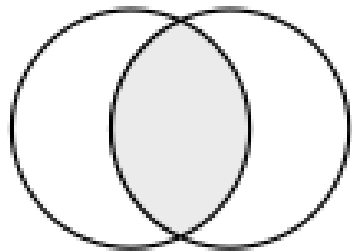
Full joins keep all rows in both table.

NAs are added where there are no matches.

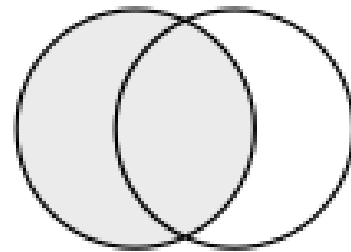


```
> full_join(x, y, by = c("key_x" = "key_y"))
```

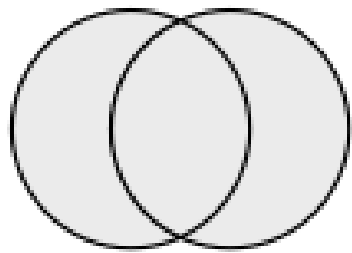
Summary



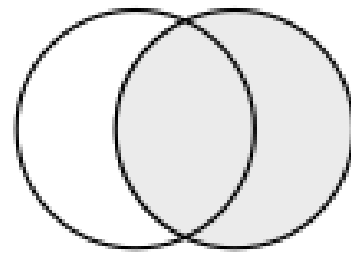
`inner_join(x, y)`



`left_join(x, y)`



`full_join(x, y)`

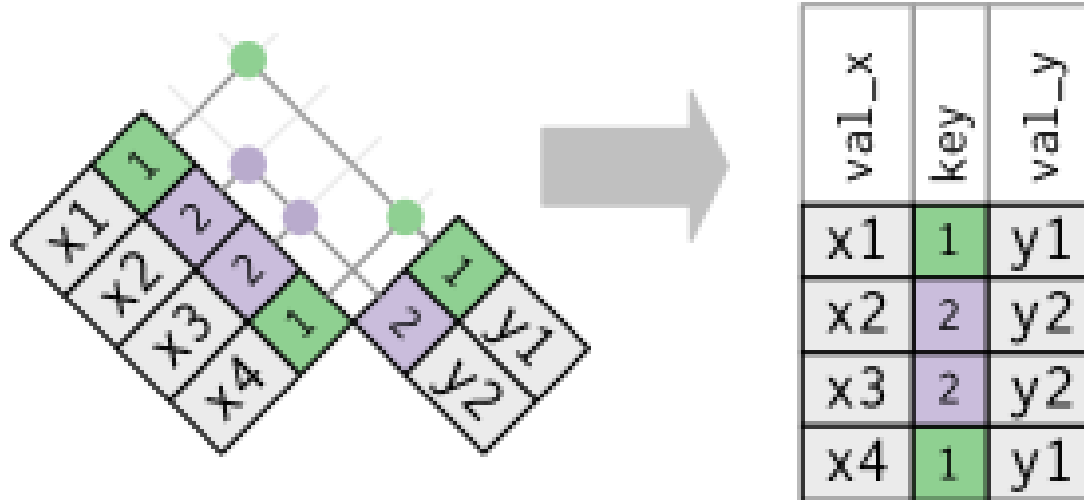


`right_join(x, y)`

Duplicate keys

Duplicate keys are useful if there is a many-to-one relationship

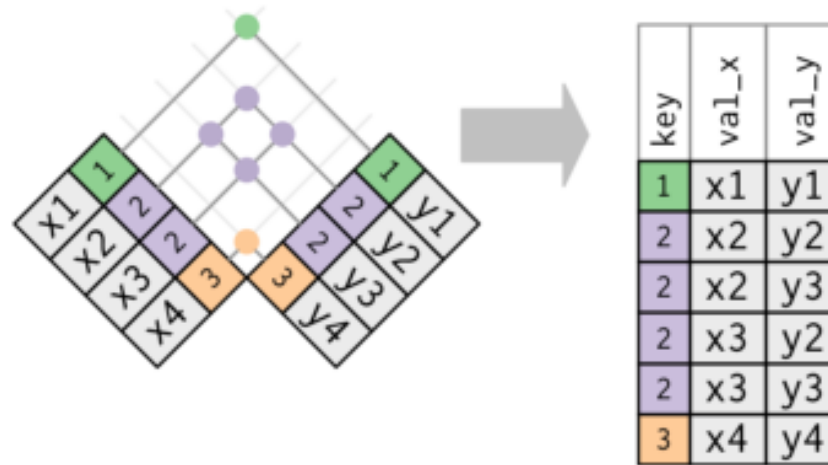
- e.g., duplicates are useful in the left table when doing a left join



Duplicate keys

If both tables have duplicate keys you get all possible combinations of joined values (Cartesian product).

- **This is usually an error!**



Always check the output size after you join a table because even if there is not a syntax error you might not get the table you are expecting!

- You can check how many rows a data frame has using the `nrow()` function

Duplicate keys

To deal with duplicate keys in both tables, we can join the tables using multiple keys in order to make sure that each row is uniquely specified.

We can do this using the syntax:

```
join(x2, y2, by = c("key1_x" = "key1_y", "key2_x" = "key2_y"))
```

Duplicate keys

```
> x2 <- data.frame(key1_x = c(1, 2, 2),  
  key2_x = c("a", "a", "b"),  
  val_x = c("y1", "y2", "y3"))  
  
> y2 <- data.frame(key1_y = c(1, 2, 2, 3, 3),  
  key2_y = c("a", "a", "b", "a", "b"),  
  val_y = c("y1", "y2", "y3", "y4", "y5"))  
  
> left_join(x2, y2, c("key1_x" = "key1_y"))  
> left_join(x2, y2, c("key1_x" = "key1_y", "key2_x" = "key2_y"))
```

Structured Query Language

Having multiple tables that can be joined together is common in Relational Database Systems (RDBS).

- A common language used by RDBS is Structured Query Language (SQL)

dplyr	SQL
<code>inner_join(x, y, by = "z")</code>	<code>SELECT * FROM x INNER JOIN y USING (z)</code>
<code>left_join(x, y, by = "z")</code>	<code>SELECT * FROM x LEFT OUTER JOIN y USING (z)</code>
<code>right_join(x, y, by = "z")</code>	<code>SELECT * FROM x RIGHT OUTER JOIN y USING (z)</code>
<code>full_join(x, y, by = "z")</code>	<code>SELECT * FROM x FULL OUTER JOIN y USING (z)</code>

Let's try it in R...