# Getting / cleaning data 2

## Regular expressions

## Regular expressions

We've already done some things to manipulate strings. For example, if we wanted to separate "Name" into last name and first name (including title), we could actually do that with the `separate` function:

```
titanic_train %>%
  select(Name) %>%
  slice(1:3) %>%
  separate(Name, c("last_name", "first_name"), sep = ", ")
```

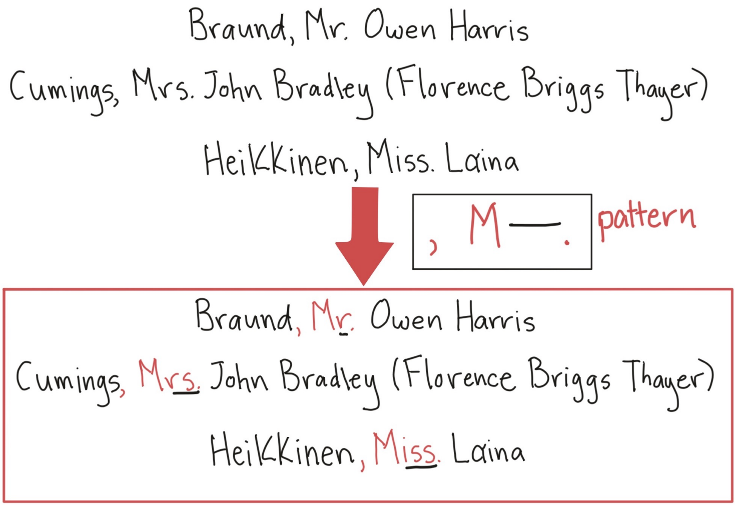
```
##   last_name                                      first_name
## 1    Braund                                 Mr. Owen Harris
## 2   Cumings Mrs. John Bradley (Florence Briggs Thayer)
## 3 Heikkinen                                     Miss. Laina
```

## Regular expressions

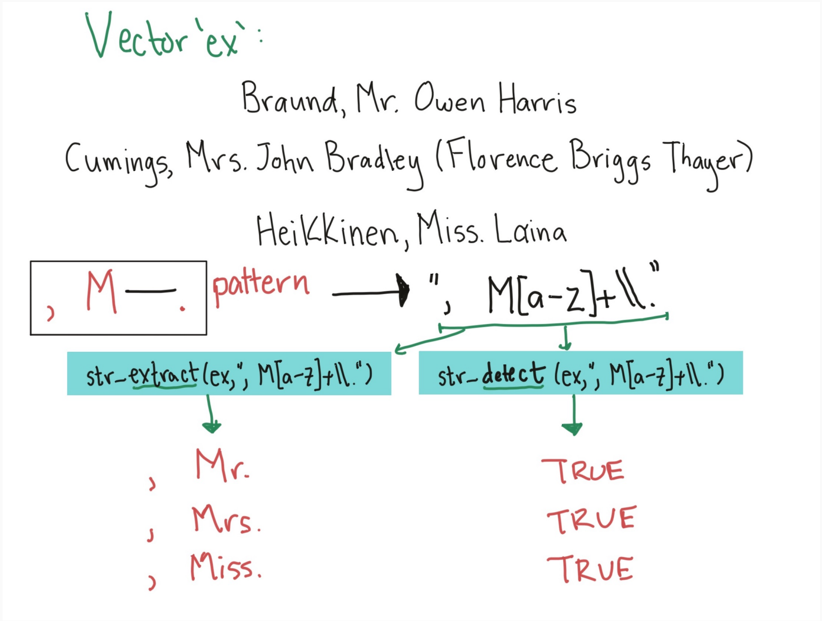
Notice that `separate` is looking for a regular pattern (“,”) and then doing something based on the location of that pattern in each string (splitting the string).

There are a variety of functions in R that can perform manipulations based on finding regular patterns in character strings.

## Regular expressions



## Regular expressions



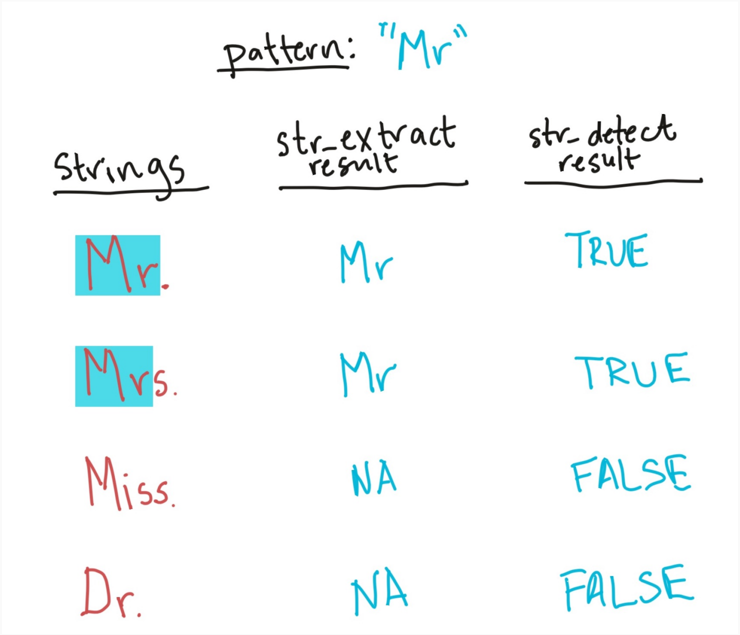
```
str_extract(ex, ", M[a-z]+\\.")
```

```
, Mr.
, Mrs.
, Miss.
```

```
str_detect (ex, ", M[a-z]+\\.")
```

```
TRUE
TRUE
TRUE
```

## Regular expressions



pattern: "Mr"

| strings | str_extract result | str_detect result |
|---|---|---|
| Mr. | Mr | TRUE |
| Mrs. | Mr | TRUE |
| Miss. | NA | FALSE |
| Dr. | NA | FALSE |

## Regular expression patterns

The easiest regular expression patterns are literal text. For example, the regular expression pattern if you're trying to match "Mr" is just "Mr":

```r
ex_names <- c("Braund, Mr. Owen Harris",
              "Cumings, Mrs. John Bradley",
              "Heikkinen, Miss. Laina")
str_extract(ex_names, pattern = "Mr")
```

```
## [1] "Mr" "Mr" NA
```

## Regular expression patterns

Regular expression patterns are case sensitive, so you won't match "Mr" with the pattern "mr":

```
ex_names <- c("Braund, Mr. Owen Harris",
              "Cumings, Mrs. John Bradley",
              "Heikkinen, Miss. Laina")
str_extract(ex_names, pattern = "mr")
```

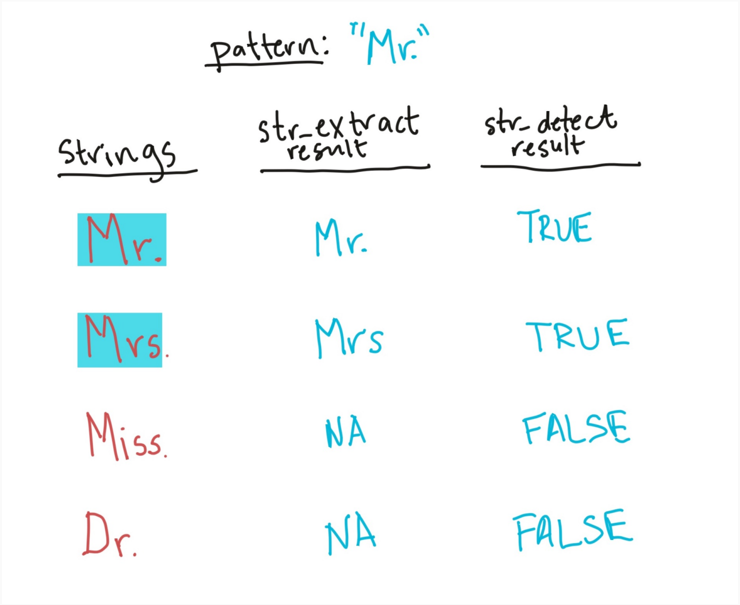
```
## [1] NA NA NA
```

## Regular expression patterns

There are a few characters called **metacharacters** that mean something special in regular expression patterns.

To use any of these literally in a regular expression, you need to “protect” them with two backslashes.

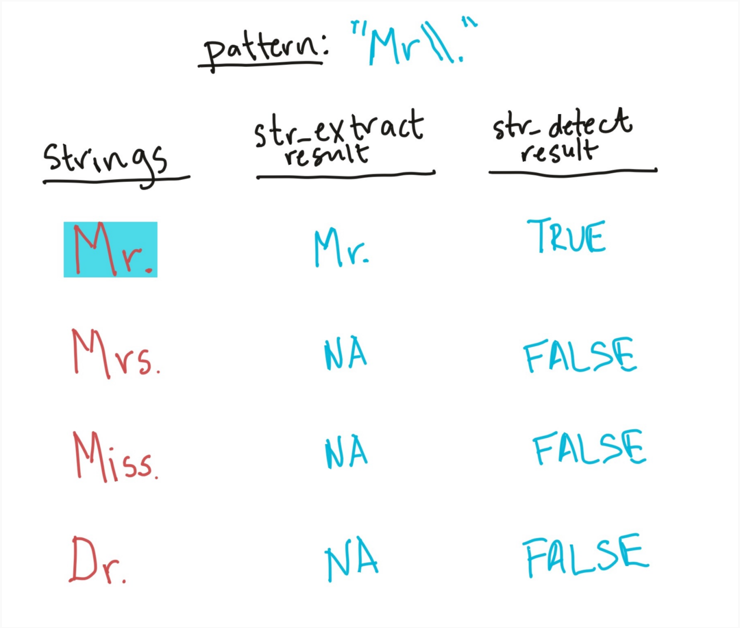
## Regular expressions



pattern: "Mr."

| Strings | str_extract result | str_detect result |
|---|---|---|
| Mr. | Mr. | TRUE |
| Mrs. | Mrs | TRUE |
| Miss. | NA | FALSE |
| Dr. | NA | FALSE |

## Regular expressions



pattern: "Mr\\."

| Strings | str_extract result | str_detect result |
|---|---|---|
| Mr. | Mr. | TRUE |
| Mrs. | NA | FALSE |
| Miss. | NA | FALSE |
| Dr. | NA | FALSE |

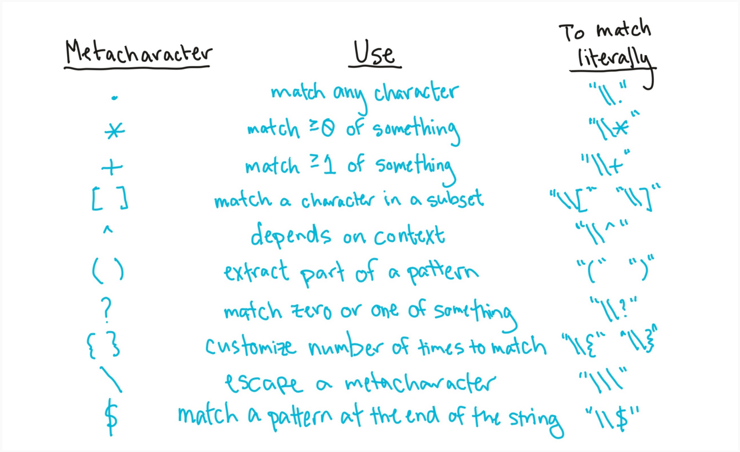
## Regular expression patterns

For example, "." is a metacharacter, so to match "Mr.", you need to use the pattern "Mr\\.":

```r
ex_names <- c("Braund, Mr. Owen Harris",
              "Cumings, Mrs. John Bradley",
              "Heikkinen, Miss. Laina")
str_extract(ex_names, pattern = "Mr\\.")
```

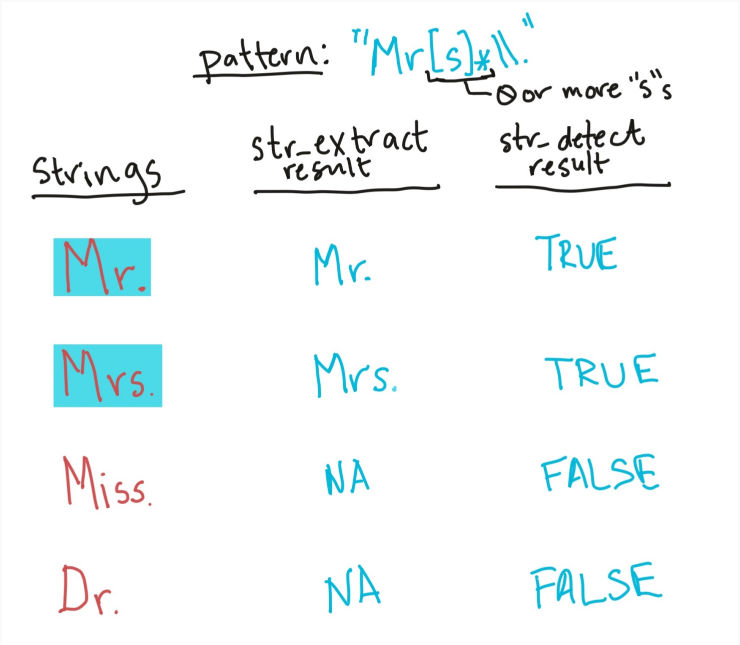
```
## [1] "Mr." NA    NA
```

## Regular expression metacharacters



| Metacharacter | Use | To match literally |
|---|---|---|
| . | match any character | "\\." |
| * | match ≥0 of something | "\*" |
| + | match ≥1 of something | "\\+" |
| [ ] | match a character in a subset | "\\[" "\\]" |
| ^ | depends on context | "\\^" |
| ( ) | extract part of a pattern | "(" ")" |
| ? | match zero or one of something | "\\?" |
| { } | customize number of times to match | "\\{" "\\}" |
| \ | escape a metacharacter | "\\\" |
| $ | match a pattern at the end of the string | "\\$" |

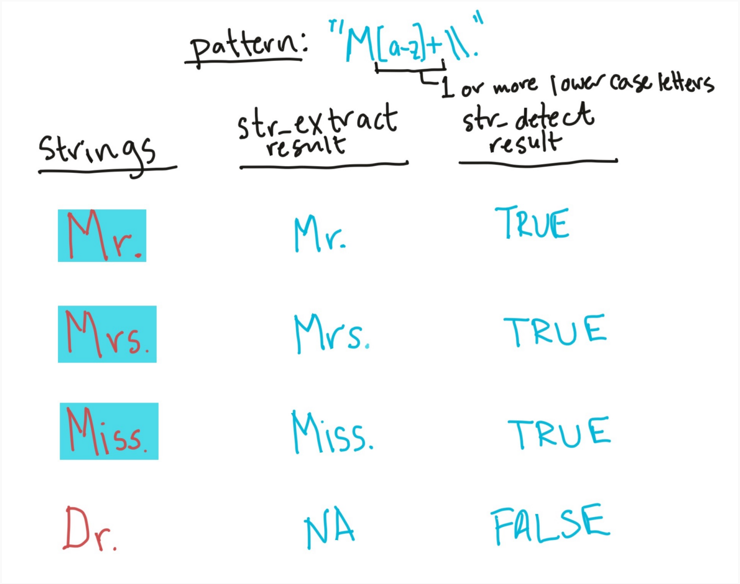
## Regular expression patterns


pattern: "Mr[s]*\\."

0 or more "s"s

| Strings | str_extract result | str_detect result |
|---|---|---|
| Mr. | Mr. | TRUE |
| Mrs. | Mrs. | TRUE |
| Miss. | NA | FALSE |
| Dr. | NA | FALSE |

## Regular expression patterns



pattern: "M[a-z]+\\."

$\underbrace{\text{[a-z]}}_{\text{1 or more lower case letters}}$

| Strings | str_extract result | str_detect result |
|---|---|---|
| Mr. | Mr. | TRUE |
| Mrs. | Mrs. | TRUE |
| Miss. | Miss. | TRUE |
| Dr. | NA | FALSE |

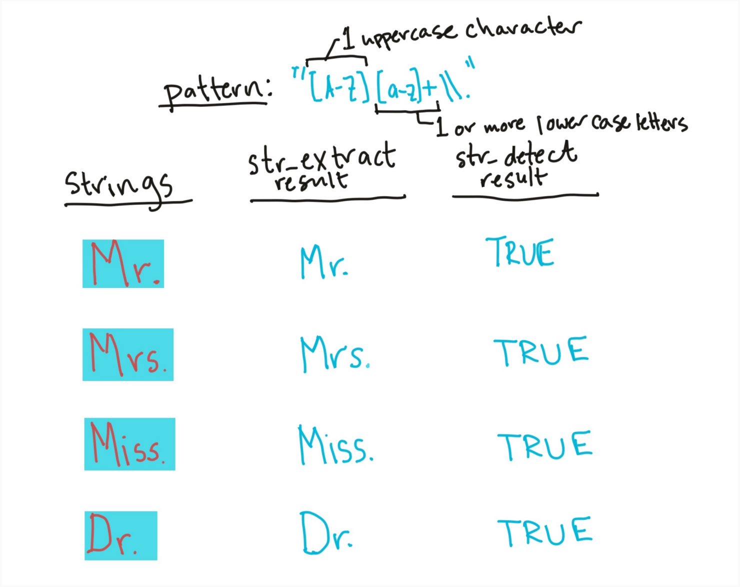
## Regular expressions

The last pattern used `[a-z]+` to match one or more lowercase letters. The `[a-z]`is a **character class**.

You can also match digits (`[0-9]`), uppercase letters (`[A-Z]`), just some letters (`[aeiou]`), etc.

You can negate a character class by starting it with `^`. For example, `[^0-9]` will match anything that **isn't** a digit.

# Regular expression patterns



pattern: "[A-Z][a-z]+\\."

| Strings | str_extract result | str_detect result |
|---|---|---|
| Mr. | Mr. | TRUE |
| Mrs. | Mrs. | TRUE |
| Miss. | Miss. | TRUE |
| Dr. | Dr. | TRUE |

## Regular expressions

The `str_detect` function will look through each element of a character vector for a designated pattern. If the pattern is there, it will return `TRUE`, and otherwise `FALSE`. The convention is:

```
## Generic code
str_detect(string = [vector you want to check],
           pattern = [pattern you want to check for])
```

For example, to create a logical vector specifying which of the Titanic passenger names include “Mrs.”, you can call:

```r
mrs <- str_detect(titanic_train$Name, "Mrs\\.")
head(mrs)
```

```
## [1] FALSE  TRUE FALSE  TRUE FALSE FALSE
```

## Regular expressions

The result is a logical vector, so `str_detect` can be used in `filter` to subset data to only rows where the passenger's name includes "Mrs.":

```r
titanic_train %>%
  filter(str_detect(Name, "Mrs\\.")) %>%
  select(Name) %>%
  slice(1:3)
```

```
##                                                    Name
## 1 Cumings, Mrs. John Bradley (Florence Briggs Thayer)
## 2        Futrelle, Mrs. Jacques Heath (Lily May Peel)
## 3   Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
```

## Regular expressions

The str_extract function can be used to extract a string (if it exists) from each value in a character vector. It follows similar conventions to str_detect:

```
## Generic code
str_extract(string = [vector you want to check],
           pattern = [pattern you want to check for])
```

## Regular expressions

For example, you might want to extract "Mrs." if it exists in a passenger's name:

```
titanic_train %>%
  mutate(mrs = str_extract(Name, "Mrs\\.")) %>%
  select(Name, mrs) %>%
  slice(1:3)
```

```
##                                                  Name  mrs
## 1                             Braund, Mr. Owen Harris <NA>
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer) Mrs.
## 3                              Heikkinen, Miss. Laina <NA>
```

Notice that now we're creating a new column (`mrs`) that either has "Mrs." (if there's a match) or is missing (`NA`) if there's not a match.

## Regular expressions

For this first example, we were looking for an exact string ("Mrs"). However, you can use patterns that match a particular pattern, but not an exact string. For example, we could expand the regular expression to find "Mr." or "Mrs.":

```
titanic_train %>%
  mutate(title = str_extract(Name, "Mr[s]*\\.")) %>%
  select(Name, title) %>%
  slice(1:3)
```

```
##                                                   Name title
## 1                              Braund, Mr. Owen Harris   Mr.
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer)  Mrs.
## 3                               Heikkinen, Miss. Laina  <NA>
```

This pattern uses `[s]*` to match zero or more "s"s at this spot in the pattern.

## Regular expressions

In the previous code, we found “Mr.” and “Mrs.”, but missed “Miss.”. We could tweak the pattern again to try to capture that, as well. For all three, we have the pattern that it starts with “M”, has some lowercase letters, and then ends with “.”.

```
titanic_train %>%
  mutate(title = str_extract(Name, "M[a-z]+\\.")) %>%
  select(Name, title) %>%
  slice(1:3)
```

```
##                                                  Name title
## 1                             Braund, Mr. Owen Harris   Mr.
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer)  Mrs.
## 3                              Heikkinen, Miss. Laina Miss.
```

## Regular expressions

Sometimes, you want to match a pattern, but then only subset a part of it. For example, each passenger seems to have a title (“Mr.”, “Mrs.”, etc.) that comes after “,” and before “.”. We can use this pattern to find the title, but then we get some extra stuff with the match:

```
titanic_train %>%
  mutate(title = str_extract(Name, ", [A-Z][a-z]*\\.")) %>%
  select(title) %>%
  slice(1:3)
```

```
##     title
## 1   , Mr.
## 2  , Mrs.
## 3 , Miss.
```

## Regular expressions

We are getting things like ", Mr. ", when we really want "Mr". We can use the `str_match` function to do this. We group what we want to extract from the pattern in parentheses, and then the function returns a matrix. The first column is the full pattern match, and each following column gives just what matches within the groups.

```
head(str_match(titanic_train$Name,
          pattern = ", ([A-Z][a-z]*)\\."))
```

```
##      [,1]      [,2]
## [1,] ", Mr."   "Mr"
## [2,] ", Mrs."  "Mrs"
## [3,] ", Miss." "Miss"
## [4,] ", Mrs."  "Mrs"
## [5,] ", Mr."   "Mr"
## [6,] ", Mr."   "Mr"
```

## Regular expressions

To get just the title, then, we can run:

```
titanic_train %>%
  mutate(title =
           str_match(Name, ", ([A-Z][a-z]*)\\.")[ , 2]) %>%
  select(Name, title) %>%
  slice(1:3)
```

```
##                                                  Name title
## 1                             Braund, Mr. Owen Harris    Mr
## 2 Cumings, Mrs. John Bradley (Florence Briggs Thayer)   Mrs
## 3                              Heikkinen, Miss. Laina  Miss
```

The `[ , 2]` pulls out just the second column from the matrix returned by `str_match`.

## Regular expressions

Here are some of the most common titles:

```
titanic_train %>%
  mutate(title =
           str_match(Name, ", ([A-Z][a-z]*)\\.")[ , 2]) %>%
  group_by(title) %>% summarize(n = n()) %>%
  arrange(desc(n)) %>% slice(1:5)
```

```
## `summarise()` ungrouping output (override with `.groups` argu
```

```
## # A tibble: 5 x 2
##   title      n
##   <chr>  <int>
## 1 Mr       517
## 2 Miss     182
## 3 Mrs      125
## 4 Master    40
## 5 Dr         7
```

## Regular expressions

The following slides have a few other examples of regular expressions in action with this dataset.

Get just names that start with ("^") the letter "A":

```
titanic_train %>%
  filter(str_detect(Name, "^A")) %>%
  select(Name) %>%
  slice(1:3)
```

```
##                                                    Name
## 1                              Allen, Mr. William Henry
## 2                           Andersson, Mr. Anders Johan
## 3 Asplund, Mrs. Carl Oscar (Selma Augusta Emilia Johansson)
```

## Regular expressions

Get names with “II” or “III” (`{2,}` says to match at least two times):

```
titanic_train %>%
  filter(str_detect(Name, "I{2,}")) %>%
  select(Name) %>%
  slice(1:3)
```

```
##                                    Name
## 1  Carter, Master. William Thornton II
## 2 Roebling, Mr. Washington Augustus II
```

## Regular expressions

Get names with "Andersen" or "Anderson" (alternatives in square brackets):

```
titanic_train %>%
  filter(str_detect(Name, "Anders[eo]n")) %>%
  select(Name)
```

```
##                                                  Name
## 1 Andersen-Jensen, Miss. Carla Christine Nielsine
## 2                             Anderson, Mr. Harry
## 3                   Walker, Mr. William Anderson
## 4                     Olsvigen, Mr. Thor Anderson
## 5      Soholt, Mr. Peter Andreas Lauritz Andersen
```

## Regular expressions

Get names that start with ("^" outside of brackets) the letters "A" and "B":

```r
titanic_train %>%
  filter(str_detect(Name, "^[AB]")) %>%
  select(Name) %>%
  slice(1:3)
```

```
##                        Name
## 1  Braund, Mr. Owen Harris
## 2 Allen, Mr. William Henry
## 3 Bonnell, Miss. Elizabeth
```

## Regular expressions

Get names that end with ("$") the letter "b" (either lowercase or uppercase):

```
titanic_train %>%
  filter(str_detect(Name, "[bB]$")) %>%
  select(Name)
```

```
##                           Name
## 1   Emir, Mr. Farred Chehab
## 2 Goldschmidt, Mr. George B
## 3             Cook, Mr. Jacob
## 4            Pasic, Mr. Jakob
```

## Regular expressions

There is a family of older, base R functions called `grep` that does something very similar.

You may see these functions in example code.

## Regular expressions

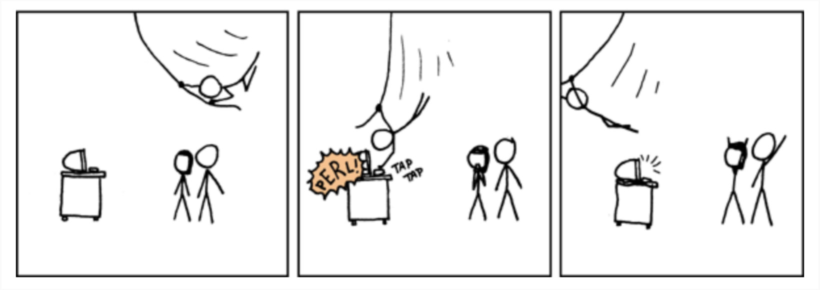
WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!

Souce: xkcd

## Regular expressions

Souce: xkcd

## Regular expressions



Souce: xkcd

## Regular expressions

For more on these patterns, see:

- Help file for the `stringi-search-regex` function in the `stringi` package (which should install when you install `stringr`)
- Chapter 14 of R For Data Science
- http://gskinner.com/RegExr: Interactive tool for helping you build regular expression pattern strings

## Tidy select

There are `tidyverse` functions to make selecting variables more straightforwards. You can call these functions as arguments of the `select` function to streamline variable selection. Examples include: `starts_with()`, `ends_with()`, and `contains()`.

## Tidy select (helpers)

Here we use `starts_with("t")` to select all variables that begin with `t`.

```
titanic_train %>%
  select(starts_with("t")) %>%
  slice(1:3)
```

```
##              Ticket
## 1         A/5 21171
## 2          PC 17599
## 3 STON/O2. 3101282
```

## Tidy select

The are also tidyverse functions that allow us to easily operate on a selection of variables. These functions are called `scoped varients`. You can identity these functions by these `_all`, `_at`, and `_if` suffixes.

## Tidy select (*_if)

Here we use `select_if` to select all the numeric variables in a dataframe and covert their names to lower case (a handy function to tidy the variable names).

```
titanic_train %>%
  select_if(is.numeric, tolower) %>%
  slice(1:3)
```

```
##   passengerid survived pclass age sibsp parch    fare
## 1           1        0      3  22     1     0  7.2500
## 2           2        1      1  38     1     0 71.2833
## 3           3        1      3  26     0     0  7.9250
```

## Tidy select (*_if)

The `select_if` function takes the following form.

```r
## Generic code
new_df <- select_if(old_df,
                    .predicate [selects the variable to keep],
                    .funs = [the function to apply to
                            the selected column names])
```

## Tidy select (*_at)

Here we use `select_at` to select all the variables that contain `ss` in their name and then covert their names to lower case (a handy function to tidy the variable names).

```r
titanic_train %>%
  select_at(vars(contains("ss")), tolower) %>%
  slice(1:3)
```

```
##   passengerid pclass
## 1           1      3
## 2           2      1
## 3           3      3
```