stringr practical

Jumping Rivers

Question 1

We'll start by loading the necessary packages and data sets

```
library("stringr")
library("dplyr")
data(names, package = "jrTidyverse2")
```

Here we have a data set containing 800 people with the names [1] "Isabella" "Emily" "Olivia" "Ava" "Mia"
[6] "Charlotte" "Sophia" "Emma" "Abigail" "Harper"
[11] "Alexander" "William" "James" "Noah" "Michael"
[16] "Mason" "Ethan" "Liam" "Jacob" "Benjamin" We want to count how many of each name there is. Try

```
names %>%
  count(name)
```

What's wrong? All of the the names are messed up. Use `mutate()`, `str_trim()`, `str_to_title()` and `count()` to count the frequency of each name.

Question 2

We'll start off by loading the data

```
data(beer, package = "jrTidyverse2")
```

Let's inspect the data

```
head(beer)
```

Here we have a data set of beers with their alcohol percentage and colour. Colour is ranked from 1-50 with 1 being pale and 50 being black. The only problem is that the names of the beers have been scraped from the web and so are contained in an url. To do any analysis on the beers we are going to need to extract the names. The names of the beers are always after the last forward slash in the url. For example, the first url,
`/homebrew/recipe/view/61925/the-devil-is-in-the-details-duvel-clone-`
would become
`The Devil Is In The Details Duvel Clone` It's going to be a bit easier to extract the vector of urls, work with it that way, then reattach it once we are done instead of getting lost in mutate hell

```
url = beer$URL
```

1) Extract the last part of the url. Use `str_extract()` to do so. Hint: Grab everything between the last forward slash (i.e. your regex should start with \) and the end of the line (i.e. your regex should end with a $).
2) Going with the first example, your beer name should now look like
`/the_devil_is_in_the_details_duvel_clone_`
Get rid of the forward slash using `str_replace()`
3) The beer names should now look like
`the_devil_is_in_the_details_duvel_clone_`
Replace the underscores with spaces (Careful, some of the urls have dashes inbetween them!)
Hint: Use `str_replace_all()` and a group, `()`, in your regex

4) The beer names will now look like
`the devil is in the details duvel clone`
Trim the surrounding whitespace and give all words capital letters. Once that is complete, overwrite the urls with the extracted names within the data. Hint: Use `str_trim()` and `str_to_title()`

5) We want to do some analysis on the beers based on whether they are an IPA, stout or pale ale. To do this we're going to introduce a new function called `if_else()` from **dplyr**. For example

```
df = data.frame(x = c(2,4,6,8))
```

Here we have made a data frame called `df` containing a column of numbers called `x`.

```
df = df %>%
  mutate(y = if_else(condition = x > 5, true = 1, false = 0))
df
```

In this step we are mutating a new column called `y` that will be the value `1` when `x > 5` and the value `0` otherwise. We can do the same for the beers. Notice that if we run the code

```
str_detect(beer$URL, "Ale")
```

We get a `TRUE` when the beer name contains `Ale` and `FALSE` otherwise. We can use this inside `if_else()` as a condition

So here we would be creating a column called `Ale`, that is `1` when the beer name contains `Ale` and `0` otherwise. Create a column called `Ipa` that is `1` when the name contains `Ipa` and `0` otherwise. Do the same for `Stout`.

6) Under the principles of tidy data, this is no longer tidy, we should have one column containing whether the beer is an "Ale", "Ipa" or "Stout". We can do this using `gather()` from **tidyr**

```
library("tidyr")
beer = beer %>%
  gather(Type, Yes, Ale, Ipa, Stout) %>%
  filter(Yes != 0) %>%
  select(-Yes)
```

Here we are gathering the Ale, Ipa and Stout columns into two columns called Type and Yes. We're not initerested in the beers with a value of 0 so we filter them out. Then we delete the Yes column using `select()`. Which type of beer has the highest average alcohol percentage and color? Hint: Use `group_by()` and `summarise()`

7) Now we can plot the data

```
library("ggplot2")
beer %>%
ggplot(aes(x = ABV, y = Color)) +
  geom_point(aes(colour = Type))
```