Week3_Notes

March 5, 2024

1 Week 3 Notes - Getting and Cleaning Data

1.1 Subsetting and Sorting

Time to wrangle and mould our datasets to our desires. Create a basic dataframe, reshjuffle the contents of the columns and then insert some NA values too.

```
set.seed(1345)
X <- data.frame("var1"=sample(1:5), "var2"=sample(6:10), "var3"=sample(11:15))
X <- X[sample(1:5),]; X$var2[c(1,3)] = NA
```

In Julia

[1]:
```
using DataFrames, CSV, Pkg
```

[2]:
```
X = DataFrame(col1=rand(1:20, 5), col2=rand(2:30, 5), col3=[missing, missing,␣
↪5, 4, 21])
```

[2]:

	col1	col2	col3
	Int64	Int64	Int64?
1	19	5	*missing*
2	6	17	*missing*
3	4	15	5
4	9	9	4
5	10	22	21

Let's subset the dataframe and take specific columns and row combinations

In R `X[,1]` to take the first column. We can also take the first column by passing the column name as a string `X[,"var1]`. Let's take the first two rows of column 2 `X[1:2,"var2"]`

With Julia we can use the basic .dot syntax

[3]:
```
X.col1
```

[3]:
```
5-element Vector{Int64}:
 19
  6
  4
  9
 10
```

[4]:
```
# This string based extraction is a bit slower
# Julia converts the String to a Symbol type
X."col1"
```

[4]:
```
5-element Vector{Int64}:
 19
  6
  4
  9
 10
```

We can also use indexing and the column names

[5]:
```
X[:, 2]
```

[5]:
```
5-element Vector{Int64}:
  5
 17
 15
  9
 22
```

[6]:
```
X[:, "col1"]
```

[6]:
```
5-element Vector{Int64}:
 19
  6
  4
  9
 10
```

[7]:
```
X[:, :col1]
```

[7]:
```
5-element Vector{Int64}:
 19
  6
  4
  9
 10
```

So to summarise it all, if we're using indexing, we can use colnumber, "colname", :Colname. If we're using .dot syntax we can use df.number, df."colname" - wooooooo

In Julia, to check the column index of a certain column e.g. is it in the 10th column? etc., we can use the **columnindex()** function

[8]:
```
columnindex(X, "col3")
```

[8]:
```
3
```

To test whether a specific column is in the dataframe, based on it's name, we can do **hasproperty()**

[9]:
```
hasproperty(X, "col3")
```

[9]:
```
true
```

Subset the dataframe using conditions, such as, print the dataframe in which the first column has values over 18

[10]:
```
filter(row -> row.col1 > 18, X)
```

[10]:

	col1	col2	col3
	Int64	Int64	Int64?
1	19	5	*missing*

If we have multiple conditons

[11]:
```
filter(row -> row.col1 > 1 || row.col2 > 1, X)
```

[11]:

	col1	col2	col3
	Int64	Int64	Int64?
1	19	5	*missing*
2	6	17	*missing*
3	4	15	5
4	9	9	4
5	10	22	21

In R we could do

```
X[(X$var1 <= 3 & X$var3 > 11),]
X[(X$var1 <= 3 | X$var3 > 11)]
```

1.1.1 Sorting

In R;

```
sort(X$var1)
# Sort in reverse
sort(X$var1, decreasing=TRUE)
```

In Julia to just get a vector of a specific dataframes column https://dataframes.juliadata.org/stable/man/sorting/

[12]:
```
sort(X.col1, rev=true)
```

[12]:
```
5-element Vector{Int64}:
 19
 10
  9
  6
  4
```

To print out the entire dataframe for viewing

[13]:
```
sort(X, "col1")
# or
sort(X, 1)
# or - this is the slowest one
sort(X, [:1])
```

[13]:

	col1	col2	col3
	Int64	Int64	Int64?
1	4	15	5
2	6	17	*missing*
3	9	9	4
4	10	22	21
5	19	5	*missing*

1.1.2 Ordering

Ordering is used in conjunction with sorting, as it will allow us to specificy the sorting order of the columns in the DataFrame, e.g. first X and reverse sort it

In R;

```
X[order(X$var1, X$var3),]
```

Now in Julia, based on the help information

[14]:
```
sort(X, order("col1", rev=true))
```

[14]:

	col1	col2	col3
	Int64	Int64	Int64?
1	19	5	*missing*
2	10	22	21
3	9	9	4
4	6	17	*missing*
5	4	15	5

We can pass multiple order functions within a single dataframe in order to handle the other columns

[15]:
```
sort!(X, ["col1", "col2"], rev=[true, false])
```

[15]:

	col1	col2	col3
	Int64	Int64	Int64?
1	19	5	*missing*
2	10	22	21
3	9	9	4
4	6	17	*missing*
5	4	15	5

1.1.3 Adding rows and columns

Adding rows and columns is a very common procedure - it should become as comfortable as adding sides to the playdough structure that we've made.

In R;

```
X$var4 <- rnorm(5)
```

In Julia, a very basic way to do this is via indexing, we can index into a column which doesn't exist yet, but soon will, and provide the data which will fill the column

[16]:
```
X.col4 = rand(100:200, 5)
```

[16]:
```
5-element Vector{Int64}:
 162
 175
 148
 162
 151
```

[17]:
```
X
```

[17]:

	col1	col2	col3	col4
	Int64	Int64	Int64?	Int64
1	19	5	*missing*	162
2	10	22	21	175
3	9	9	4	148
4	6	17	*missing*	162
5	4	15	5	151

1.2 Summarising Data

We'll be looking at different ways of providing a snapshot of the general big picture of our datasets - the averages, limits, deviations and so on.

Let's do the basics, the beginning and end of the datasets; In R;

```
head(data, n=3)
tail(data, n=5)
```

In Julia

[18]:
```
first(X)
```

[18]:

	col1	col2	col3	col4
	Int64	Int64	Int64?	Int64
1	19	5	*missing*	162

[19]:
```
last(X)
```

[19]:

	col1	col2	col3	col4
	Int64	Int64	Int64?	Int64
5	4	15	5	151

To get a brief summary of the data with descriptve stats and other information such as the Types of the variables in the columns, we can use `summary(data)` in R and in Julia we can use **describe()**

[20]:
```
describe(X)
```

[20]:

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Float64	Int64	Float64	Int64	Int64	Type
1	col1	9.6	4	9.0	19	0	Int64
2	col2	13.6	5	15.0	22	0	Int64
3	col3	10.0	4	5.0	21	2	Union{Missing, Int64}
4	col4	159.6	148	162.0	175	0	Int64

In R you can also use the `str(data)` command

[21]:
```
typeof(X.col1)
```

[21]:
```
Vector{Int64} (alias for Array{Int64, 1})
```

To get the quantiles of a vector, in R we have the base function `quantile(data$column.na.rm=TRUE)` and in Julia we have to use the **Statistics.jl** package to add this functionality. Remember Julia is a more general language compared to R which was always tailored towards statistical computing - see https://www.jlhub.com/julia/manual/en/function/quantile-exclamation

[22]:
```
using Statistics
```

[333]:
```
# Print quarter quantiles
quantile!(X.col1, [0.25, 0.5, 0.75, 1], )
```

[333]:
```
4-element Vector{Float64}:
  6.0
  9.0
 10.0
 19.0
```

To skip the missing values and print the median value

[24]:
```
quantile(skipmissing(X.col3), 0.5)
```

[24]:
```
5.0
```

1.2.1 Checking for missing values

In R, count the number of missing values

```
sum(is.na(data$column))
```

Check is **any** na values are present

```
any(is.na(data$column))
```

Test to see whether all the values meet a certain condition (over 0)

```
all(data$column > 0)
```

In Julia, get the sum of missing values - using the one line iterators

[25]:
```
sum(x -> ismissing(x), X.col3)
```

[25]: 2

If any missing values are in there

[26]:
```
any(x -> ismissing(x), X.col3)
```

[26]: true

If all the values are a certain condition

[27]:
```
all(x -> ismissing(x), X.col1)
```

[27]: false

[28]:
```
all(x -> x > 0, X.col1)
```

[28]: true

A cool little function in Julia to only extract the dataframes rows which contain missing values

[29]:
```
filter(x -> any(ismissing, x), X)
```

[29]:

	col1	col2	col3	col4
	Int64	Int64	Int64?	Int64
1	4	5	*missing*	162
2	10	17	*missing*	162

Perform a quick sum of all of the columns in a horizontal fashion - intuitively this would mean summing the entire row, and producing a new sum in the final column of the same row e.g. |1|2|3|6 (final)|. This can be a very quick way of checking whether there are any missing values as the missing values will propogate across!

In R;

```
colSums(is.na(data))
```

In Julia

[30]:
```
sum(eachcol(X))
```

[30]:
```
5-element Vector{Union{Missing, Int64}}:
    missing
 224
 170
    missing
 190
```

In Julia if we want to actually sum the entire column, meaning every value in the column vertically, we can collect the column and then sum it OR we can just performing broadcasting using the sum function - fascinating but easily confusing!

[31]:
```
sum.(collect(eachcol(X)))
```

[31]:
```
4-element Vector{Union{Missing, Int64}}:
  48
  68
    missing
 798
```

[32]:
```
sum.(eachcol(X))
```

[32]:
```
4-element Vector{Union{Missing, Int64}}:
  48
  68
    missing
 798
```

There is an equivalent operation by using the broadcasting over **eachrow()**

[33]:
```
sum.(eachrow(X))
```

[33]:
```
5-element Vector{Union{Missing, Int64}}:
    missing
 224
 170
    missing
 190
```

If we want to skip missing values when doing these operations we would broadcasting **skipmissing()** across

[34]:
```
sum.(skipmissing.(eachrow(X)))
```

[34]:
```
5-element Vector{Int64}:
 171
 224
 170
 189
 190
```

1.2.2 Subsetting the dataframe based upon values in the columns

Say for example that we only want the data which have a specific zipcode (generic value) in a column, what can we do? In R;

```
data[data$zipCode %in% c("4109", "4110"),]
```

In Julia - get a dataframe in which the values in the first column are 1

[35]:
```
filter(row -> row.col1 == 1, X)
```

[35]:

	col1	col2	col3	col4
	Int64	Int64	Int64?	Int64

Now another one wherein the values are larger and 1 and smaller than 15

[36]:
```
filter(row -> row.col1 > 1 && row.col1 < 15, X)
```

[36]:

	col1	col2	col3	col4
	Int64	Int64	Int64?	Int64
1	4	5	*missing*	162
2	6	22	21	175
3	9	9	4	148
4	10	17	*missing*	162

1.3 Cross Tabulation aka Frequency Tables

In order provide small snapshots of potential interactions and relations, we can see cross-tabulation or frequency comparisons between variables, say, male and female and acceptance rates to university

In R we have some base functions;

```
xt <- xtabs(Freq ~ Gender + Admit, data=DF)
```

In Julia we have to load a specific package called **FreqTables** https://github.com/nalimilan/FreqTables.jl

[37]:
```
using Pkg; Pkg.add("FreqTables") ; using FreqTables
```

```
    Updating registry at `~/.julia/registries/General.toml`
   Resolving package versions…
  No Changes to `~/.julia/environments/v1.10/Project.toml`
  No Changes to `~/.julia/environments/v1.10/Manifest.toml`
```

Do a frequency table between columns 1 and 4 - clearly there is not much here to see given both are randomly generated vectors

[38]:
```
freqtable(X, :col1, :col4)
```

[38]:
```
5×4 Named Matrix{Int64}
col1   col4   148  151  162  175

4               0    0    1    0
6               0    0    0    1
9               1    0    0    0
10              0    0    1    0
19              0    1    0    0
```

1.4 Size of the data in human readable form

Very simple and yet very informative information - how big is our data? In R;

```
object.size(data), units="Mb")
```

In Julia, we can use **varinfo()**

1.5 Creating New Variables

Often our datasets may consist of variables and values which we need to prune, transform and mould to our likings - perhaps they have broken delimiters, or are a combination of two values in ones, or are better represented in a different form - and so on. These tasks require us to create new variables and add these to the dataset - remember that we should always keep copies of the original dataset and not simply mutate it into oblivion.

Let's take some restaurant data from Baltimore city to use as the sample dataset

[39]:
```
download("https://gist.githubusercontent.com/slowteetoe/528c78213fcd80f05419/
↪raw/e0a4a89476fca79e692df1a373e5025f5112a5f6/restaurants.csv", "restaurants.
↪csv")
```

[39]:
```
"restaurants.csv"
```

[40]:
```
rest_data = CSV.File("restaurants.csv") |> DataFrame
# We can also do
# CSV.read(open("file.csv"), DataFrame)
```

[40]:

	name	zipCode	neighborhood	councilDistrict	policeDistrict
	String	Int64	String	Int64	String15
1	410	21206	Frankford	2	NORTHEASTER
2	1919	21231	Fells Point	1	SOUTHEASTER
3	SAUTE	21224	Canton	1	SOUTHEASTER
4	#1 CHINESE KITCHEN	21211	Hampden	14	NORTHERN
5	#1 chinese restaurant	21223	Millhill	9	SOUTHWESTER
6	19TH HOLE	21218	Clifton Park	14	NORTHEASTER
7	3 KINGS	21205	McElderry Park	13	SOUTHEASTER
8	3 MILES HOUSE, INC.	21211	Remington	7	NORTHERN
9	3 W'S TAVERN	21205	McElderry Park	13	SOUTHEASTER
10	300 SOUTH ANN STREET	21231	Upper Fells Point	1	SOUTHEASTER
11	438 CLUB	21226	Curtis Bay	10	SOUTHERN
12	5-MILE HOUSE	21215	Woodmere	5	NORTHWESTER
13	743 S. MONTFORD,INC.	21224	Canton	1	SOUTHEASTER
14	A & W RESTAURANT	21224	Pulaski Industrial Area	1	SOUTHEASTER
15	A TASTE OF CHINA	21202	Downtown	11	CENTRAL
16	ABACROMBIE FINE FOODS	21201	Mid-Town Belvedere	11	CENTRAL
17	ABC SUSHI	21205	Middle East	13	EASTERN
18	ACROPOLIS RESTAURANT	21224	Greektown	2	SOUTHEASTER
19	ADMIRAL FELL INN	21231	Fells Point	1	SOUTHEASTER
20	AG & HARP'S	21230	Morrell Park	10	SOUTHWESTER
21	AIRPORT BAR & GRILL, INC.	21222	Saint Helena	1	SOUTHEASTER
22	AKBAR RESTAURANT	21201	Mount Vernon	11	CENTRAL
23	AKDEY SUBWAY, INC.	21223	Millhill	9	SOUTHWESTER
24	ALDO'S RESTAURANT	21202	Little Italy	1	SOUTHEASTER
25	ALE MARY'S	21231	Fells Point	1	SOUTHEASTER
26	ALEXANDER'S TAVERN	21231	Fells Point	1	SOUTHEASTER
27	ALFEO'S	21215	Mondawmin	7	WESTERN
28	AL-HO CLUB	21223	Booth-Boyd	9	SOUTHWESTER
29	BAY ATLANTIC CLUB	21212	Downtown	11	CENTRAL
30	BAY CAFE	21224	Canton	1	SOUTHEASTER
...	...	...	...	...	...

1.5.1 Creating sequences

We can create sequences to use as indexes for extracting data e.g. `rest_data[index sequence]`

In R we can create a sequence of intergers which a step size of 2 by ;

```
seq_one <- seq(1,10, by=2)
```

or also using the **c()** collect function

```
seq_c <- c(1,3,8,25,100) ; seq(along = x)
```

In Julia we can do it using **collect()** function as well - with the ranges on either side and the step size in the middle

[41]:
```
seq_one = collect(1:3:10)
```

```
[41]: 4-element Vector{Int64}:
        1
        4
        7
       10
```

Or just create a vector directly

```
[42]: seq_vec = Vector(1:3:10)
```

```
[42]: 4-element Vector{Int64}:
        1
        4
        7
       10
```

Say we want to create a new variable which indicates whether a certain data meet a certain condition, say, they are in the neighbourhoods of Sunnybank or Sunnybank Hills, and thus they are close to me - and if they meet this condition, they are assigned a "TRUE" value or a "FALSE" value in a new row - how would we do this?

In R;

```
restData$nearMe = restData$neighbourhood %in% c("Roland Park", "Homeland")
```

In Julia - this is a new one for me! - make sure we use the double encapsulation in the array [[]] otherwise broadcasting won't work

```
[43]: rest_data.nearMe = in.(rest_data.neighborhood, [["Roland Park", "Homeland"]])
```

```
[43]: 1327-element BitVector:
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0

 0
 0
 0
 0
 1
```

```
0
0
0
0
0
0
0
```

R base has the very handy function table() which creates a nice tally of ones variables and counts them in a neat table output - Julia base doesn't really have this single item function so we have to use the **StatsBase** package and the **countmap()** function

[44]:
```
Pkg.add("StatsBase") ; using StatsBase
```

```
  Resolving package versions…
 No Changes to `~/.julia/environments/v1.10/Project.toml`
 No Changes to `~/.julia/environments/v1.10/Manifest.toml`
```

[45]:
```
# 0 = true, 1 = false
# perhaps we can convert the 0,1 to TRUE/FALSE?
qz = countmap(rest_data.nearMe)
println(qz)
```

```
Dict{Bool, Int64}(0 => 1314, 1 => 13)
```

1.5.2 Creating binary variables

We can use the **ifelse()** function with broadcasting to evaluate a conditonal statement and print true or false in a new variables based upon the answer of the condition. Here we'll do it in R first;

```
restData$wrongZip = ifelse(restData$zipCode) < -, TRUE, FALSE)
```

And then in Julia

[46]:
```
rest_data.wrongZip = ifelse.(rest_data.zipCode .< 0, true, false)
```

[46]:
```
1327-element BitVector:
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
```

[47]:
```
countmap(rest_data.wrongZip)
```

[47]:
```
Dict{Bool, Int64} with 2 entries:
  0 => 1326
  1 => 1
```

1.5.3 Creating categorical variables

We may want to summarise certain aspects of our dataset by chunking them into categorical blocks - similar to percentiles. Say we want to get a look at the distribution of gene lengths or zip codes in our dataset, at quartile ranges - we can turn to categorical variables

Create them in Julia https://categoricalarrays.juliadata.org/v0.1/using.html

[48]:
```
Pkg.add("CategoricalArrays") ; using CategoricalArrays
```

```
  Resolving package versions…
 No Changes to `~/.julia/environments/v1.10/Project.toml`
 No Changes to `~/.julia/environments/v1.10/Manifest.toml`
```

[49]:
```
cut(X.col1, 4)
```

[49]:
```
5-element CategoricalArray{String,1,UInt32}:
 "Q1: [4.0, 6.0)"
 "Q2: [6.0, 9.0)"
 "Q3: [9.0, 10.0)"
 "Q4: [10.0, 19.0]"
 "Q4: [10.0, 19.0]"
```

[50]:
```
countmap(cut(rest_data.zipCode, 4))
```

[50]:
```
Dict{CategoricalValue{String, UInt32}, Int64} with 4 entries:
  "Q4: [21225.5, 21287.0]"  => 332
  "Q2: [21202.0, 21218.0)"  => 507
  "Q3: [21218.0, 21225.5)"  => 351
```

```
"Q1: [-21226.0, 21202.0)" => 137
```

In R let's use the **Hmisc** library

```
library(Hmisc)
restData$zipGroups = cut2(restData$zipCode, g=4)
table(restData$zipGroups)
```

When we turn our data into Categorical variables, we turn them into factors

1.5.4 Reshaping data

The notions of split-apply-combine are prevalent in data analytics, as they are terrific guiding principles for approaching data frames. They were originally popularised in R and built around R packages, but are now supported by plenty of packages in most high level programming languages.

We'll start with R and the basic **reshape2()** library

```
library(reshape2)
head(mtcars) #standard dataset in R
```

We can use some of the popular R datasets in Julia using the **RDatasets.jl** package - as we can see almost everything is supported in Julia

[51]:
```
Pkg.add("RDatasets") ; using RDatasets
```

```
  Resolving package versions…
 No Changes to `~/.julia/environments/v1.10/Project.toml`
 No Changes to `~/.julia/environments/v1.10/Manifest.toml`
```

Load the mtcars dataset from "datasets"

[52]:
```
mtcars = dataset("datasets", "mtcars")
```

[52]:

	Model	MPG	Cyl	Disp	HP	DRat	WT	QSec	
	String31	Float64	Int64	Float64	Int64	Float64	Float64	Float64	
1	Mazda RX4	21.0	6	160.0	110	3.9	2.62	16.46	...
2	Mazda RX4 Wag	21.0	6	160.0	110	3.9	2.875	17.02	...
3	Datsun 710	22.8	4	108.0	93	3.85	2.32	18.61	...
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	...
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.44	17.02	...
6	Valiant	18.1	6	225.0	105	2.76	3.46	20.22	...
7	Duster 360	14.3	8	360.0	245	3.21	3.57	15.84	...
8	Merc 240D	24.4	4	146.7	62	3.69	3.19	20.0	...
9	Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	...
10	Merc 280	19.2	6	167.6	123	3.92	3.44	18.3	...
11	Merc 280C	17.8	6	167.6	123	3.92	3.44	18.9	...
12	Merc 450SE	16.4	8	275.8	180	3.07	4.07	17.4	...
13	Merc 450SL	17.3	8	275.8	180	3.07	3.73	17.6	...
14	Merc 450SLC	15.2	8	275.8	180	3.07	3.78	18.0	...
15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.25	17.98	...
16	Lincoln Continental	10.4	8	460.0	215	3.0	5.424	17.82	...
17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	...
18	Fiat 128	32.4	4	78.7	66	4.08	2.2	19.47	...
19	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	...
20	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9	...
21	Toyota Corona	21.5	4	120.1	97	3.7	2.465	20.01	...
22	Dodge Challenger	15.5	8	318.0	150	2.76	3.52	16.87	...
23	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.3	...
24	Camaro Z28	13.3	8	350.0	245	3.73	3.84	15.41	...
25	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	...
26	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.9	...
27	Porsche 914-2	26.0	4	120.3	91	4.43	2.14	16.7	...
28	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	...
29	Ford Pantera L	15.8	8	351.0	264	4.22	3.17	14.5	...
30	Ferrari Dino	19.7	6	145.0	175	3.62	2.77	15.5	...
...	...	...	...	...	...	...	...	...	

[53]:
```
first(mtcars, 5)
```

[53]:

	Model	MPG	Cyl	Disp	HP	DRat	WT	QSec	VS	
	String31	Float64	Int64	Float64	Int64	Float64	Float64	Float64	Int64	
1	Mazda RX4	21.0	6	160.0	110	3.9	2.62	16.46	0	...
2	Mazda RX4 Wag	21.0	6	160.0	110	3.9	2.875	17.02	0	...
3	Datsun 710	22.8	4	108.0	93	3.85	2.32	18.61	1	...
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	...
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.44	17.02	0	...

[54]:
```
# descriptive stats
describe(mtcars)
```

[54]:

	variable	mean	min	median	max	nmissing	eltype
	Symbol	Union…	Any	Union…	Any	Int64	DataType
1	Model		AMC Javelin		Volvo 142E	0	String31
2	MPG	20.0906	10.4	19.2	33.9	0	Float64
3	Cyl	6.1875	4	6.0	8	0	Int64
4	Disp	230.722	71.1	196.3	472.0	0	Float64
5	HP	146.688	52	123.0	335	0	Int64
6	DRat	3.59656	2.76	3.695	4.93	0	Float64
7	WT	3.21725	1.513	3.325	5.424	0	Float64
8	QSec	17.8487	14.5	17.71	22.9	0	Float64
9	VS	0.4375	0	0.0	1	0	Int64
10	AM	0.40625	0	0.0	1	0	Int64
11	Gear	3.6875	3	4.0	5	0	Int64
12	Carb	2.8125	1	2.0	8	0	Int64

[55]:
```
# column names
names(mtcars)
```

[55]:
```
12-element Vector{String}:
 "Model"
 "MPG"
 "Cyl"
 "Disp"
 "HP"
 "DRat"
 "WT"
 "QSec"
 "VS"
 "AM"
 "Gear"
 "Carb"
```

1.5.5 "Melting" dataframes in R / "Reshaping" in Julia

The main idea behind melting is the presence of "id" and "value" variables - the ID variables will represent the consistent identifiers we want to retain, and the value variables will be "stacked" to form a group with its own column - almost like a sub-data frame - it's hard to imagine but easier to understand once we play with it a little bit ourselves - so when performing the melt and stack functions, we need to always designate these two variables. In R, id variables come first and the measure variables seconds, whereas in Julia it is the other way around. Let's take a look in R;

```
mtcars$carname <- rownames(mtcars) # looks redunant based on julia code
carMelt <- melt(mtcares, id=c("carname", "gear", "cyl"), measure.vars=c("mpg", "hp"))
```

In Julia this is perform with the DataFrames package and **stack()**

[56]:
```
mt_stack = stack(mtcars, [:MPG, :HP], [:Model, :Gear, :Cyl])
```

[56]:

	Model	Gear	Cyl	variable	value
	String31	Int64	Int64	String	Float64
1	Mazda RX4	4	6	MPG	21.0
2	Mazda RX4 Wag	4	6	MPG	21.0
3	Datsun 710	4	4	MPG	22.8
4	Hornet 4 Drive	3	6	MPG	21.4
5	Hornet Sportabout	3	8	MPG	18.7
6	Valiant	3	6	MPG	18.1
7	Duster 360	3	8	MPG	14.3
8	Merc 240D	4	4	MPG	24.4
9	Merc 230	4	4	MPG	22.8
10	Merc 280	4	6	MPG	19.2
11	Merc 280C	4	6	MPG	17.8
12	Merc 450SE	3	8	MPG	16.4
13	Merc 450SL	3	8	MPG	17.3
14	Merc 450SLC	3	8	MPG	15.2
15	Cadillac Fleetwood	3	8	MPG	10.4
16	Lincoln Continental	3	8	MPG	10.4
17	Chrysler Imperial	3	8	MPG	14.7
18	Fiat 128	4	4	MPG	32.4
19	Honda Civic	4	4	MPG	30.4
20	Toyota Corolla	4	4	MPG	33.9
21	Toyota Corona	3	4	MPG	21.5
22	Dodge Challenger	3	8	MPG	15.5
23	AMC Javelin	3	8	MPG	15.2
24	Camaro Z28	3	8	MPG	13.3
25	Pontiac Firebird	3	8	MPG	19.2
26	Fiat X1-9	4	4	MPG	27.3
27	Porsche 914-2	5	4	MPG	26.0
28	Lotus Europa	5	4	MPG	30.4
29	Ford Pantera L	5	8	MPG	15.8
30	Ferrari Dino	5	6	MPG	19.7
...	...	...	...	...	...

1.5.6 Casting dataframes

Imagine we want to get a quick overview of how many values of a certain variable we have in our dataset - say, the distribution of miles per gallon and horse power based upon the cylinders a car has - the relation between cylinders and engine performance : Cylinders ~ engine. In R we would use **dcast()**;

```
cylData <- dcast(carMelt, cyl ~ variable)
```

This is a littler trickier in Julia as there's not as much supporting documentation yet, but this is called a "pivot table" in Julia – bogumil (who else?!) has a great post here https://www.juliabloggers.com/pivot-tables-in-dataframes-jl/

[57]:
```
unstack(mt_stack, :Cyl, :variable, :Cyl, combine=length)
```

[57]:

	Cyl	MPG	HP
	Int64	Int64?	Int64?
1	6	7	7
2	4	11	11
3	8	14	14

Now let's get the mean of these measures, rather than their length?

[58]:
```
unstack(mt_stack, :Cyl, :variable, :value, combine=mean)
```

[58]:

	Cyl	MPG	HP
	Int64	Float64?	Float64?
1	6	19.7429	122.286
2	4	26.6636	82.6364
3	8	15.1	209.214

1.5.7 Averaging values

[59]:
```
countmap(sum.(mtcars.AM))
```

[59]:
```
Dict{Int64, Int64} with 2 entries:
  0 => 19
  1 => 13
```

[60]:
```
groupby(mtcars, :Cyl)
```

[60]:

GroupedDataFrame with 3 groups based on key: Cyl

First Group (11 rows): Cyl = 4

	Model	MPG	Cyl	Disp	HP	DRat	WT	QSec	VS	
	String31	Float64	Int64	Float64	Int64	Float64	Float64	Float64	Int64	
1	Datsun 710	22.8	4	108.0	93	3.85	2.32	18.61	1	...
2	Merc 240D	24.4	4	146.7	62	3.69	3.19	20.0	1	...
3	Merc 230	22.8	4	140.8	95	3.92	3.15	22.9	1	...
4	Fiat 128	32.4	4	78.7	66	4.08	2.2	19.47	1	...
5	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	...
6	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9	1	...
7	Toyota Corona	21.5	4	120.1	97	3.7	2.465	20.01	1	...
8	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.9	1	...
9	Porsche 914-2	26.0	4	120.3	91	4.43	2.14	16.7	0	...
10	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9	1	...
11	Volvo 142E	21.4	4	121.0	109	4.11	2.78	18.6	1	...

...

Last Group (14 rows): Cyl = 8

	Model	MPG	Cyl	Disp	HP	DRat	WT	QSec	
	String31	Float64	Int64	Float64	Int64	Float64	Float64	Float64	
1	Hornet Sportabout	18.7	8	360.0	175	3.15	3.44	17.02	...
2	Duster 360	14.3	8	360.0	245	3.21	3.57	15.84	...
3	Merc 450SE	16.4	8	275.8	180	3.07	4.07	17.4	...
4	Merc 450SL	17.3	8	275.8	180	3.07	3.73	17.6	...
5	Merc 450SLC	15.2	8	275.8	180	3.07	3.78	18.0	...
6	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.25	17.98	...
7	Lincoln Continental	10.4	8	460.0	215	3.0	5.424	17.82	...
8	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	...
9	Dodge Challenger	15.5	8	318.0	150	2.76	3.52	16.87	...
10	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.3	...
11	Camaro Z28	13.3	8	350.0	245	3.73	3.84	15.41	...
12	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	...
13	Ford Pantera L	15.8	8	351.0	264	4.22	3.17	14.5	...
14	Maserati Bora	15.0	8	301.0	335	3.54	3.57	14.6	...

1.6 DataFrames.jl

Almost everything covered herein appears to be described somewhere on the DataFrames.jl package site https://dataframes.juliadata.org/stable/man/split_apply_combine/ and/or on the DataFramesMeta site https://juliadata.org/DataFramesMeta.jl/stable/dplyr/ . It may take some googling and forum scouring, but rest assured that the developers have considered many many procedures and functions. As such it is best to work through Bogumil's book and create your own projects that you're interested in - passion and curiosity as the backbones of science and inquiry!

1.7 Merging/Joining data

Joining datasets together is a common operation, especially within SQL - we can think of the common terminology of inner join, outer join and so on that's almost synonymous with databases. In R lets merge some data;

```
mergedData = merge(reviews, solution, by.x="solution_id", by.y="yd", all=TRUE)
```

In Julia let's create some mock data

[61]:
```
people = DataFrame(ID=[20, 40], Name=["John Doe", "Jane Doe"])
```

[61]:

	ID	Name
	Int64	String
1	20	John Doe
2	40	Jane Doe

[62]:
```
jobs = DataFrame(ID=[20, 40], Job=["Lawyer", "Doctor"])
```

[62]:

	ID	Job
	Int64	String
1	20	Lawyer
2	40	Doctor

Say there is a common ID shared between datasets, we can take advantage of this ID and 'sink'

our data around it, in some sense this is like performing a union operation on a set - the common element is never duplicated.

Inner joins are "the output contains rows for values of the key that exist in all passed data frames."

[63]:
```
innerjoin(people, jobs, on = :ID)
```

[63]:

	ID	Name	Job
	Int64	String	String
1	20	John Doe	Lawyer
2	40	Jane Doe	Doctor

[64]:
```
intersect(names(people), names(jobs))
```

[64]:
```
1-element Vector{String}:
 "ID"
```

[65]:
```
breaks = DataFrame(ID=[20, 60], Fruit=["Apple", "Watermelon"])
```

[65]:

	ID	Fruit
	Int64	String
1	20	Apple
2	60	Watermelon

To merge all of the datasets together based upon a shared variable, even if not all of the rows are present in the other datasets, we can use the **outerjoin()** method

[66]:
```
outerjoin(breaks, people, jobs, on = :ID)
```

[66]:

	ID	Fruit	Name	Job
	Int64	String?	String?	String?
1	20	Apple	John Doe	Lawyer
2	40	*missing*	Jane Doe	Doctor
3	60	Watermelon	*missing*	*missing*

In R we provide all of the column names this time;

```
mergedData2 <- merge(reviews, solutions, all=TRUE)
```

2 Quiz

2.1 1.

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv

and load the data into R. The code book, describing the variable names is here:

https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FPUMSDataDict06.pdf

Create a logical vector that identifies the households on greater than 10 acres who sold more than $10,000 worth of agriculture products. Assign that logical vector to the variable agricultureLogical. Apply the which() function like this to identify the rows of the data frame where the logical vector is TRUE.

```
which(agricultureLogical)
```

What are the first 3 values that result?

[67]:
```
download("https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv",␣
↪"quiz3_q1.csv")
```

[67]:
```
"quiz3_q1.csv"
```

[94]:
```
first_q = CSV.File("quiz3_q1.csv") |> DataFrame
```

[94]:

	RT	SERIALNO	DIVISION	PUMA	REGION	ST	ADJUST	WGTP	NP	TYPE	
	String1	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	
1	H	186	8	700	4	16	1015675	89	4	1	...
2	H	306	8	700	4	16	1015675	310	1	1	...
3	H	395	8	100	4	16	1015675	106	2	1	...
4	H	506	8	700	4	16	1015675	240	4	1	...
5	H	835	8	800	4	16	1015675	118	4	1	...
6	H	989	8	700	4	16	1015675	115	4	1	...
7	H	1861	8	700	4	16	1015675	0	1	2	...
8	H	2120	8	200	4	16	1015675	35	1	1	...
9	H	2278	8	400	4	16	1015675	47	2	1	...
10	H	2428	8	500	4	16	1015675	51	2	1	...
11	H	2677	8	400	4	16	1015675	114	2	1	...
12	H	2928	8	800	4	16	1015675	51	2	1	...
13	H	3283	8	400	4	16	1015675	67	3	1	...
14	H	3331	8	600	4	16	1015675	0	1	2	...
15	H	3341	8	500	4	16	1015675	92	1	1	...
16	H	3827	8	300	4	16	1015675	51	2	1	...
17	H	4331	8	800	4	16	1015675	71	3	1	...
18	H	4425	8	400	4	16	1015675	349	1	1	...
19	H	4457	8	900	4	16	1015675	270	4	1	...
20	H	4551	8	500	4	16	1015675	83	3	1	...
21	H	4609	8	600	4	16	1015675	78	1	1	...
22	H	4720	8	200	4	16	1015675	115	1	1	...
23	H	5191	8	600	4	16	1015675	314	2	1	...
24	H	5465	8	100	4	16	1015675	22	1	1	...
25	H	5590	8	100	4	16	1015675	73	2	1	...
26	H	5763	8	800	4	16	1015675	73	4	1	...
27	H	5890	8	800	4	16	1015675	39	1	1	...
28	H	6517	8	800	4	16	1015675	63	1	1	...
29	H	6581	8	300	4	16	1015675	88	1	1	...
30	H	6789	8	600	4	16	1015675	321	2	1	...
...	...	...	...	...	...	...	...	...	...	...	

The **filter()** function seems like an obvious first choice, until we try to work with missing values, and this is where it comes short - instead we should use **subset()** in combination by **ByRow()** to parse this correctly

[69]:
```
#filter(row -> row.AGS == 6 && row.ACR == 3, first_q)
subset(first_q, :AGS => ByRow(==(6)), :ACR => ByRow(==(3)), skipmissing=true)
```

[69]:

	RT	SERIALNO	DIVISION	PUMA	REGION	ST	ADJUST	WGTP	NP	TYPE	
	String1	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	Int64	
1	H	30346	8	400	4	16	1015675	120	4	1	...
2	H	53292	8	300	4	16	1015675	26	3	1	...
3	H	56299	8	800	4	16	1015675	97	2	1	...
4	H	101282	8	800	4	16	1015675	76	2	1	...
5	H	120351	8	800	4	16	1015675	51	5	1	...
6	H	122802	8	800	4	16	1015675	63	5	1	...
7	H	133128	8	300	4	16	1015675	15	2	1	...
8	H	140896	8	400	4	16	1015675	72	2	1	...
9	H	169806	8	800	4	16	1015675	62	1	1	...
10	H	173013	8	500	4	16	1015675	77	2	1	...
11	H	176884	8	900	4	16	1015675	88	1	1	...
12	H	183434	8	500	4	16	1015675	54	2	1	...
13	H	203578	8	800	4	16	1015675	70	2	1	...
14	H	204262	8	200	4	16	1015675	24	2	1	...
15	H	223184	8	100	4	16	1015675	22	5	1	...
16	H	270844	8	300	4	16	1015675	67	2	1	...
17	H	272251	8	800	4	16	1015675	24	2	1	...
18	H	278331	8	300	4	16	1015675	163	3	1	...
19	H	293603	8	300	4	16	1015675	73	4	1	...
20	H	341269	8	900	4	16	1015675	146	2	1	...
21	H	347362	8	900	4	16	1015675	126	2	1	...
22	H	352408	8	400	4	16	1015675	52	2	1	...
23	H	395701	8	300	4	16	1015675	40	2	1	...
24	H	409401	8	800	4	16	1015675	22	2	1	...
25	H	444160	8	800	4	16	1015675	371	2	1	...
26	H	465760	8	200	4	16	1015675	24	4	1	...
27	H	510757	8	400	4	16	1015675	24	3	1	...
28	H	519912	8	800	4	16	1015675	42	2	1	...
29	H	537967	8	300	4	16	1015675	83	3	1	...
30	H	546403	8	200	4	16	1015675	27	2	1	...
...	...	...	...	...	...	...	...	...	...	...	

But we have to create construct a new variable based upon the evaluation of a conditional - if it is true then it will be represented as such in each row in the new column, and conversly as false if it is not. I was stuck for a very long time until finding this post by who else than Bogumil https://stackoverflow.com/questions/50114613/add-column-conditionally

[95]:
```
first_q.largeLand = ifelse.((ismissing.(first_q.AGS)) .| (first_q.AGS .!= 6) .|
 ↪(first_q.ACR .!= 3), false, true)
```

[95]:
```
6496-element BitVector:
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0

 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
 0
```

[96]:
```
countmap(first_q.largeLand)
```

[96]:
```
Dict{Bool, Int64} with 2 entries:
  0 => 6419
  1 => 77
```

[98]:
```
findall(first_q.largeLand .== 1)
```

[98]:
```
77-element Vector{Int64}:
  125
  238
  262
  470
  555
  568
  608
  643
  787
  808
```

```
 824
 849
 952

5236
5326
5417
5531
5574
5894
6033
6044
6089
6275
6376
6420
```

The answer is 125, 238, 262 - these represent the row numbers

2.2 2.

Using the jpeg package read in the following picture of your instructor into R

https://d396qusza40orc.cloudfront.net/getdata%2Fjeff.jpg

Use the parameter native=TRUE. What are the 30th and 80th quantiles of the resulting data? (some Linux systems may produce an answer 638 different for the 30th quantile)

[99]:
```
download("https://d396qusza40orc.cloudfront.net/getdata%2Fjeff.jpg", "leek.jpg")
```

[99]:
```
"leek.jpg"
```

[142]:
```
Pkg.add(["ImageIO", "Images", "ImageMagick", "FileIO"]) ; using ImageIO,␣
 ↪FileIO, ImageMagick
```

```
  Resolving package versions…
  Installed JpegTurbo_jll         v3.0.1+0
  Installed LERC_jll             v3.0.0+1
  Installed TiledIteration          v0.3.1
  Installed Images               v0.24.1
  Installed ImageMagick            v1.3.0
  Installed FFTW                 v1.8.0
  Installed IdentityRanges          v0.3.1
  Installed IntelOpenMP_jll         v2024.0.2+0
  Installed Netpbm               v1.0.1
  Installed StaticArrays            v1.9.2
  Installed RealDot              v0.1.0
  Installed Distances             v0.10.11
  Installed StaticArraysCore         v1.4.2
  Installed CustomUnitRanges         v1.0.2
```

```
  Installed Ratios              v0.4.5
  Installed ImageMetadata          v0.9.5
  Installed ComputationalResources    v0.3.2
  Installed ImageMagick_jll        v6.9.10-12+3
  Installed CatIndices           v0.2.2
  Installed Zstd_jll            v1.5.5+0
  Installed ImageQualityIndexes      v0.2.2
  Installed ImageTransformations     v0.8.13
  Installed MKL_jll             v2024.0.0+0
  Installed ImageShow            v0.3.1
  Installed Libtiff_jll          v4.4.0+0
  Installed CoordinateTransformations   v0.6.3
  Installed ChainRulesCore         v1.21.1
  Installed Quaternions           v0.7.6
  Installed Rotations            v1.7.0
  Installed FFTW_jll            v3.3.10+0
  Installed UnPack              v1.0.2
  Installed ImageCore            v0.8.22
  Installed IndirectArrays         v0.5.1
  Installed AxisAlgorithms         v1.0.1
  Installed ImageDistances         v0.2.13
  Installed ImageAxes            v0.6.9
  Installed FFTViews             v0.3.2
  Installed Parameters           v0.12.3
  Installed WoodburyMatrices        v0.5.6
  Installed ImageMorphology        v0.2.11
  Installed ImageContrastAdjustment    v0.3.7
  Installed ImageFiltering         v0.6.21
  Installed Interpolations         v0.13.6
    Updating `~/.julia/environments/v1.10/Project.toml`
 [6218d12a] + ImageMagick v1.3.0
 [916415d5] + Images v0.24.1
    Updating `~/.julia/environments/v1.10/Manifest.toml`
 [13072b0f] + AxisAlgorithms v1.0.1
 [aafaddc9] + CatIndices v0.2.2
 [d360d2e6] + ChainRulesCore v1.21.1
 [ed09eef8] + ComputationalResources v0.3.2
 [150eb455] + CoordinateTransformations v0.6.3
 [dc8bdbbb] + CustomUnitRanges v1.0.2
 [864edb3b] ↑ DataStructures v0.17.20  v0.18.16
 [b4f34e82] + Distances v0.10.11
 [4f61f5a4] + FFTViews v0.3.2
 [7a1cc6ca] + FFTW v1.8.0
 [bbac6d45] + IdentityRanges v0.3.1
 [2803e5a7] ↓ ImageAxes v0.6.11  v0.6.9
 [c817782e] - ImageBase v0.1.5
 [f332f351] + ImageContrastAdjustment v0.3.7
 [a09fc81d] ↓ ImageCore v0.9.4  v0.8.22
```

```
  [51556ac3] + ImageDistances v0.2.13
  [6a3955dd] + ImageFiltering v0.6.21
  [6218d12a] + ImageMagick v1.3.0
  [bc367c6b] ↓ ImageMetadata v0.9.9  v0.9.5
  [787d08f9] + ImageMorphology v0.2.11
  [2996bd0c] + ImageQualityIndexes v0.2.2
  [4e3cecfd] + ImageShow v0.3.1
  [02fcd773] + ImageTransformations v0.8.13
  [916415d5] + Images v0.24.1
  [9b13fd28] ↓ IndirectArrays v1.0.0  v0.5.1
  [a98d9a8b] + Interpolations v0.13.6
  [86f7a689] ↑ NamedArrays v0.9.4  v0.10.0
  [f09324ee] ↓ Netpbm v1.1.1  v1.0.1
  [d96e819e] + Parameters v0.12.3
  [94ee1d12] + Quaternions v0.7.6
  [c84ed2f1] + Ratios v0.4.5
  [c1ae055f] + RealDot v0.1.0
  [6038ab10] + Rotations v1.7.0
  [90137ffa] + StaticArrays v1.9.2
  [1e83bf80] + StaticArraysCore v1.4.2
  [06e1c1a7] + TiledIteration v0.3.1
  [3a884ed6] + UnPack v1.0.2
  [efce3f68] + WoodburyMatrices v0.5.6
  [f5851436] + FFTW_jll v3.3.10+0
  [c73af94c] + ImageMagick_jll v6.9.10-12+3
  [1d5cc7b8] + IntelOpenMP_jll v2024.0.2+0
  [aacddb02] + JpegTurbo_jll v3.0.1+0
  [88015f11] + LERC_jll v3.0.0+1
  [89763e89] + Libtiff_jll v4.4.0+0
  [856f044c] + MKL_jll v2024.0.0+0
  [3161d3a3] + Zstd_jll v1.5.5+0
  [8ba89e20] + Distributed
  [1a1011a3] + SharedArrays
        Info Packages marked with  and
have new versions available. Those with  may be upgradable, but those
with  are restricted by compatibility constraints from upgrading. To
see why use `status --outdated -m`
Precompiling project…
    CustomUnitRanges
    IdentityRanges
    UnPack
    IndirectArrays
    RealDot
    StaticArraysCore
    Ratios
    TiledIteration
    ComputationalResources
    WoodburyMatrices
```

```
    IntelOpenMP_jll
    LERC_jll
    ChainRulesCore
    Zstd_jll
    FFTW_jll
    JpegTurbo_jll
    CatIndices
    Parameters
    Quaternions
    Distances
    DataStructures
    Ratios → RatiosFixedPointNumbersExt
    AxisAlgorithms
    ChainRulesCore → ChainRulesCoreSparseArraysExt
    AbstractFFTs → AbstractFFTsChainRulesCoreExt
    Distances → DistancesSparseArraysExt
    Libtiff_jll
    LogExpFunctions → LogExpFunctionsChainRulesCoreExt
    Distances → DistancesChainRulesCoreExt
    NamedArrays
    ImageMagick_jll
    ImageCore
    ImageMorphology
    ImageAxes
    ImageShow
    StaticArrays
    Netpbm
    ImageDistances
    PNGFiles
    ImageMetadata
    StaticArrays → StaticArraysStatisticsExt
    StaticArrays → StaticArraysChainRulesCoreExt
    ImageMagick
    CoordinateTransformations
    Rotations
    Interpolations
MKL_jll Waiting for background task / IO / timer.
[pid 371619] waiting for IO to finish:
 Handle type        uv_handle_t->data
 timer              0x140ffc0->0x7f7c3ff8fb80
This means that a package has started a background task or event source that has
not finished running. For precompilation to complete successfully, the event
source needs to be closed explicitly. See the developer documentation on fixing
precompilation hangs for more help.
    ImageTransformations
    MKL_jll
    ImageContrastAdjustment
```

```
    FFTW
    FFTViews
    ImageFiltering
    ImageQualityIndexes
    Images
  54 dependencies successfully precompiled in 277 seconds. 136 already
precompiled. 1 skipped during auto due to previous errors.
  2 dependencies precompiled but different versions are currently
loaded. Restart julia to access the new versions
  2 dependencies had output during precompilation:

 MKL_jll

    Downloading artifact: MKL


   [pid 371619] waiting for IO to finish:

    Handle type        uv_handle_t->data

    timer              0x140ffc0->0x7f7c3ff8fb80

   This means that a package has started a background task or event source
that has not finished running. For precompilation to complete successfully, the
event source needs to be closed explicitly. See the developer documentation on
fixing precompilation hangs for more help.


 Interpolations

   WARNING: method definition for checkbounds at
/home/number25/.julia/packages/Interpolations/y4lLj/src/Interpolations.jl:454
declares type variable N but does not use it.

   WARNING: method definition for checkbounds at
/home/number25/.julia/packages/Interpolations/y4lLj/src/Interpolations.jl:457
declares type variable N but does not use it.

   WARNING: method definition for GriddedInterpolation at
/home/number25/.julia/packages/Interpolations/y4lLj/src/gridded/gridded.jl:37
declares type variable pad but does not use it.

   WARNING: method definition for GriddedInterpolation at
/home/number25/.julia/packages/Interpolations/y4lLj/src/gridded/gridded.jl:60
declares type variable pad but does not use it.

   WARNING: method definition for interpolate! at
/home/number25/.julia/packages/Interpolations/y4lLj/src/deprecations.jl:30
declares type variable TWeights but does not use it.
```

I'm gonna put this on hold as it's a bit hard to understand at the moment

2.3 3.

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv

Load the educational data from this data set:

https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS_Country.csv

Match the data based on the country shortcode. How many of the IDs match? Sort the data frame in descending order by GDP rank (so United States is last). What is the 13th country in the resulting data frame?

Original data sources:

http://data.worldbank.org/data-catalog/GDP-ranking-table

http://data.worldbank.org/data-catalog/ed-stats

1 point

[167]:
```
download("https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv", "gdp.
↪csv") ; download("https://d396qusza40orc.cloudfront.net/
↪getdata%2Fdata%2FEDSTATS_Country.csv", "education.csv")
```

[167]:
```
"education.csv"
```

Take a look at the structure of the .csv using some bash commands - this will let us know what the delimiters look like, the header structure and any other important features to take into consideration

[178]:
```
run(`head gdp.csv -n 20`)
```

```
,Gross domestic product 2012,,,,,,,
,,,,,,,,,
,,,,(millions of,,,,,
,Ranking,,Economy,US dollars),,,,,
,,,,,,,,,
USA,1,,United States," 16,244,600 ",,,,,
CHN,2,,China," 8,227,103 ",,,,,
JPN,3,,Japan," 5,959,718 ",,,,,
DEU,4,,Germany," 3,428,131 ",,,,,
FRA,5,,France," 2,612,878 ",,,,,
GBR,6,,United Kingdom," 2,471,784 ",,,,,
BRA,7,,Brazil," 2,252,664 ",,,,,
RUS,8,,Russian Federation," 2,014,775 ",,,,,
ITA,9,,Italy," 2,014,670 ",,,,,
IND,10,,India," 1,841,710 ",,,,,
CAN,11,,Canada," 1,821,424 ",,,,,
AUS,12,,Australia," 1,532,408 ",,,,,
ESP,13,,Spain," 1,322,965 ",,,,,
MEX,14,,Mexico," 1,178,126 ",,,,,
KOR,15,,"Korea, Rep."," 1,129,598 ",,,,,
```

[178]:
```
Process(`head gdp.csv -n 20`,
ProcessExited(0))
```

Import the data into a Dataframe, start parsing the data from line 6 as the first 5 lines are headers, and the header is located on the 4th line. Since the dataframe contains many ill-formated columns and missing rows, we'll only extract the 4 columns that have been properly input, and then drop the missing rows. We'll also rename the columns to make it easier to read and work with.

[298]:
```
gdp_raw = CSV.read("gdp.csv", DataFrame, skipto=6, header=4)
gdp_raw_nomissing = dropmissing(gdp_raw[:, [:1, :2, :4, :5]])
gdp_cleaned = rename!(gdp_raw_nomissing, ["CountryCode", "Rank", "Country",␣
 ↪"US_dollars"])
```

[298]:

	CountryCode	Rank	Country	US_dollars
	String3	String	String31	String15
1	USA	1	United States	16,244,600
2	CHN	2	China	8,227,103
3	JPN	3	Japan	5,959,718
4	DEU	4	Germany	3,428,131
5	FRA	5	France	2,612,878
6	GBR	6	United Kingdom	2,471,784
7	BRA	7	Brazil	2,252,664
8	RUS	8	Russian Federation	2,014,775
9	ITA	9	Italy	2,014,670
10	IND	10	India	1,841,710
11	CAN	11	Canada	1,821,424
12	AUS	12	Australia	1,532,408
13	ESP	13	Spain	1,322,965
14	MEX	14	Mexico	1,178,126
15	KOR	15	Korea, Rep.	1,129,598
16	IDN	16	Indonesia	878,043
17	TUR	17	Turkey	789,257
18	NLD	18	Netherlands	770,555
19	SAU	19	Saudi Arabia	711,050
20	CHE	20	Switzerland	631,173
21	SWE	21	Sweden	523,806
22	IRN	22	Iran, Islamic Rep.	514,060
23	NOR	23	Norway	499,667
24	POL	24	Poland	489,795
25	BEL	25	Belgium	483,262
26	ARG	26	Argentina	475,502
27	AUT	27	Austria	394,708
28	ZAF	28	South Africa	384,313
29	VEN	29	Venezuela, RB	381,286
30	COL	30	Colombia	369,606
...	...	...	...	...

The commas in the US_dollars column are causing the numeric values to be represented as Strings, and thus the column type is String15 – in order to perform numeric operations, we need to change

this column type. We'll remove the commas and then parse the columns values as Intergers to create the new frame

[299]:
```
gdp_cleaned.US_dollars .= replace.(gdp_cleaned.US_dollars, "," => "")
gdp_cleaned.US_dollars = parse.(Int, gdp_cleaned.US_dollars)
gdp_cleaned.Rank = parse.(Int, gdp_cleaned.Rank)
#gdp_cleaned
```

[299]:
```
190-element Vector{Int64}:
   1
   2
   3
   4
   5
   6
   7
   8
   9
  10
  11
  12
  13

 178
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
```

[300]:
```
gdp_cleaned
```

[300]:

	CountryCode	Rank	Country	US_dollars
	String3	Int64	String31	Int64
1	USA	1	United States	16244600
2	CHN	2	China	8227103
3	JPN	3	Japan	5959718
4	DEU	4	Germany	3428131
5	FRA	5	France	2612878
6	GBR	6	United Kingdom	2471784
7	BRA	7	Brazil	2252664
8	RUS	8	Russian Federation	2014775
9	ITA	9	Italy	2014670
10	IND	10	India	1841710
11	CAN	11	Canada	1821424
12	AUS	12	Australia	1532408
13	ESP	13	Spain	1322965
14	MEX	14	Mexico	1178126
15	KOR	15	Korea, Rep.	1129598
16	IDN	16	Indonesia	878043
17	TUR	17	Turkey	789257
18	NLD	18	Netherlands	770555
19	SAU	19	Saudi Arabia	711050
20	CHE	20	Switzerland	631173
21	SWE	21	Sweden	523806
22	IRN	22	Iran, Islamic Rep.	514060
23	NOR	23	Norway	499667
24	POL	24	Poland	489795
25	BEL	25	Belgium	483262
26	ARG	26	Argentina	475502
27	AUT	27	Austria	394708
28	ZAF	28	South Africa	384313
29	VEN	29	Venezuela, RB	381286
30	COL	30	Colombia	369606
...	...	...	...	...

Import education data

[301]:
```
edu_raw = CSV.read("education.csv", DataFrame)
```

[301]:

	CountryCode	Long Name	Income Group	Region
	String3	String	String31?	String31?
1	ABW	Aruba	High income: nonOECD	Latin America & Caribbean
2	ADO	Principality of Andorra	High income: nonOECD	Europe & Central Asia
3	AFG	Islamic State of Afghanistan	Low income	South Asia
4	AGO	People's Republic of Angola	Lower middle income	Sub-Saharan Africa
5	ALB	Republic of Albania	Upper middle income	Europe & Central Asia
6	ARE	United Arab Emirates	High income: nonOECD	Middle East & North Africa
7	ARG	Argentine Republic	Upper middle income	Latin America & Caribbean
8	ARM	Republic of Armenia	Lower middle income	Europe & Central Asia
9	ASM	American Samoa	Upper middle income	East Asia & Pacific
10	ATG	Antigua and Barbuda	Upper middle income	Latin America & Caribbean
11	AUS	Commonwealth of Australia	High income: OECD	East Asia & Pacific
12	AUT	Republic of Austria	High income: OECD	Europe & Central Asia
13	AZE	Republic of Azerbaijan	Upper middle income	Europe & Central Asia
14	BDI	Republic of Burundi	Low income	Sub-Saharan Africa
15	BEL	Kingdom of Belgium	High income: OECD	Europe & Central Asia
16	BEN	Republic of Benin	Low income	Sub-Saharan Africa
17	BFA	Burkina Faso	Low income	Sub-Saharan Africa
18	BGD	People's Republic of Bangladesh	Low income	South Asia
19	BGR	Republic of Bulgaria	Upper middle income	Europe & Central Asia
20	BHR	Kingdom of Bahrain	High income: nonOECD	Middle East & North Africa
21	BHS	Commonwealth of The Bahamas	High income: nonOECD	Latin America & Caribbean
22	BIH	Bosnia and Herzegovina	Upper middle income	Europe & Central Asia
23	BLR	Republic of Belarus	Upper middle income	Europe & Central Asia
24	BLZ	Belize	Lower middle income	Latin America & Caribbean
25	BMU	The Bermudas	High income: nonOECD	North America
26	BOL	Plurinational State of Bolivia	Lower middle income	Latin America & Caribbean
27	BRA	Federative Republic of Brazil	Upper middle income	Latin America & Caribbean
28	BRB	Barbados	High income: nonOECD	Latin America & Caribbean
29	BRN	Brunei Darussalam	High income: nonOECD	East Asia & Pacific
30	BTN	Kingdom of Bhutan	Lower middle income	South Asia
...	...	...	...	...

Perform an inner join on the "Country Code" columns

[302]:
```
gdp_edu = innerjoin(gdp_cleaned, edu_raw, on=:CountryCode)
```

[302]:

	CountryCode	Rank	Country	US_dollars	Long Name	
	String3	Int64	String31	Int64	String	
1	ABW	161	Aruba	2584	Aruba	...
2	AFG	105	Afghanistan	20497	Islamic State of Afghanistan	...
3	AGO	60	Angola	114147	People's Republic of Angola	...
4	ALB	125	Albania	12648	Republic of Albania	...
5	ARE	32	United Arab Emirates	348595	United Arab Emirates	...
6	ARG	26	Argentina	475502	Argentine Republic	...
7	ARM	133	Armenia	9951	Republic of Armenia	...
8	ATG	172	Antigua and Barbuda	1134	Antigua and Barbuda	...
9	AUS	12	Australia	1532408	Commonwealth of Australia	...
10	AUT	27	Austria	394708	Republic of Austria	...
11	AZE	68	Azerbaijan	66605	Republic of Azerbaijan	...
12	BDI	162	Burundi	2472	Republic of Burundi	...
13	BEL	25	Belgium	483262	Kingdom of Belgium	...
14	BEN	140	Benin	7557	Republic of Benin	...
15	BFA	128	Burkina Faso	10441	Burkina Faso	...
16	BGD	59	Bangladesh	116355	People's Republic of Bangladesh	...
17	BGR	76	Bulgaria	50972	Republic of Bulgaria	...
18	BHR	93	Bahrain	29044	Kingdom of Bahrain	...
19	BHS	138	Bahamas, The	8149	Commonwealth of The Bahamas	...
20	BIH	111	Bosnia and Herzegovina	17466	Bosnia and Herzegovina	...
21	BLR	69	Belarus	63267	Republic of Belarus	...
22	BLZ	169	Belize	1493	Belize	...
23	BMU	149	Bermuda	5474	The Bermudas	...
24	BOL	96	Bolivia	27035	Plurinational State of Bolivia	...
25	BRA	7	Brazil	2252664	Federative Republic of Brazil	...
26	BRB	153	Barbados	4225	Barbados	...
27	BRN	113	Brunei Darussalam	16954	Brunei Darussalam	...
28	BTN	167	Bhutan	1780	Kingdom of Bhutan	...
29	BWA	117	Botswana	14504	Republic of Botswana	...
30	CAF	165	Central African Republic	2184	Central African Republic	...
...	...	...	...	...	...	

Sort the dataframe column "US_dollars"

[279]:
```
sort!(gdp_edu, :US_dollars, rev=false)
```

[279]:

	CountryCode	Rank	Country	US_dollars	Long Name
	String3	String	String31	Int64	String
1	TUV	190	Tuvalu	40	Tuvalu
2	KIR	189	Kiribati	175	Republic of Kirib
3	MHL	188	Marshall Islands	182	Republic of the Marsha
4	PLW	187	Palau	228	Republic of Pala
5	STP	186	S\xe3o Tom\xe9 and Principe	263	Democratic Republic of S\xe3o To
6	FSM	185	Micronesia, Fed. Sts.	326	Federated States of Mi
7	TON	184	Tonga	472	Kingdom of Ton
8	DMA	183	Dominica	480	Commonwealth of Do
9	COM	182	Comoros	596	Union of the Com
10	WSM	181	Samoa	684	Samoa
11	VCT	180	St. Vincent and the Grenadines	713	St. Vincent and the Gr
12	GRD	178	Grenada	767	Grenada
13	KNA	178	St. Kitts and Nevis	767	St. Kitts and Ne
14	VUT	177	Vanuatu	787	Republic of Vanu
15	GNB	176	Guinea-Bissau	822	Republic of Guinea-
16	GMB	175	Gambia, The	917	Republic of The Ga
17	SLB	174	Solomon Islands	1008	Solomon Island
18	SYC	173	Seychelles	1129	Republic of Seych
19	ATG	172	Antigua and Barbuda	1134	Antigua and Barb
20	LCA	171	St. Lucia	1239	St. Lucia
21	TMP	170	Timor-Leste	1293	Democratic Republic of T
22	BLZ	169	Belize	1493	Belize
23	LBR	168	Liberia	1734	Republic of Libe
24	BTN	167	Bhutan	1780	Kingdom of Bhut
25	CPV	166	Cape Verde	1827	Republic of Cape V
26	CAF	165	Central African Republic	2184	Central African Rep
27	MDV	164	Maldives	2222	Republic of Mald
28	LSO	163	Lesotho	2448	Kingdom of Leso
29	BDI	162	Burundi	2472	Republic of Buru
30	ABW	161	Aruba	2584	Aruba
...	...	...	...	...	...

The answer is ! 189 matches, 13th country is St. Kitts and Nevis

2.4 4.

What is the average GDP ranking for the “High income: OECD” and “High income: nonOECD” group?

[280]:
```
filter(x -> x == "High income: nonOECD", gdp_edu.var"Income Group")
```

[280]:
```
23-element PooledArrays.PooledVector{Union{Missing, String31}, UInt32,
Vector{UInt32}}:
 "High income: nonOECD"
 "High income: nonOECD"
 "High income: nonOECD"
```

```
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
"High income: nonOECD"
```

[304]:
```
high_non = filter(row -> row.var"Income Group" == "High income: nonOECD",␣
↪gdp_edu)
```

[304]:

	CountryCode	Rank	Country	US_dollars	Long Nam
	String3	Int64	String31	Int64	String
1	ABW	161	Aruba	2584	Aruba
2	ARE	32	United Arab Emirates	348595	United Arab Em
3	BHR	93	Bahrain	29044	Kingdom of Ba
4	BHS	138	Bahamas, The	8149	Commonwealth of Th
5	BMU	149	Bermuda	5474	The Bermud
6	BRB	153	Barbados	4225	Barbados
7	BRN	113	Brunei Darussalam	16954	Brunei Daruss
8	CYP	102	Cyprus	22767	Republic of Cy
9	EST	103	Estonia	22390	Republic of Es
10	GNQ	110	Equatorial Guinea	17697	Republic of Equator
11	HKG	37	Hong Kong SAR, China	263259	Hong Kong Special Administrative Region
12	HRV	71	Croatia	59228	Republic of Cr
13	KWT	56	Kuwait	160913	State of Kuw
14	LVA	94	Latvia	28373	Republic of La
15	MAC	82	Macao SAR, China	43582	Macao Special Administrative Region of
16	MCO	147	Monaco	6075	Principality of M
17	MLT	137	Malta	8722	Republic of M
18	OMN	66	Oman	69972	Sultanate of O
19	PRI	61	Puerto Rico	101496	Puerto Ric
20	QAT	54	Qatar	171476	State of Qat
21	SAU	19	Saudi Arabia	711050	Kingdom of Saud
22	SGP	35	Singapore	274701	Republic of Sing
23	TTO	101	Trinidad and Tobago	23320	Republic of Trinidad

[306]:
```
mean.(eachcol(high_non.Rank))
```

[306]:
```
1-element Vector{Float64}:
 91.91304347826087
```

Answer is 32.96667, 91.91304

2.5 5.

Cut the GDP ranking into 5 separate quantile groups. Make a table versus Income.Group. How many countries are Lower middle income but among the 38 nations with highest GDP?

Let's see how the quantiles look - combine countmap() with cut(df, n)

[323]:
```
countmap(cut(gdp_edu.Rank, 5))
```

[323]:
```
Dict{CategoricalValue{String, UInt32}, Int64} with 5 entries:
  "Q5: [152.4, 190.0]" => 38
  "Q4: [113.8, 152.4)" => 38
  "Q3: [76.2, 113.8)"  => 37
  "Q2: [38.6, 76.2)"   => 38
  "Q1: [1.0, 38.6)"    => 38
```

Add a new variable containing the quantile category that each row belongs to

[312]:
```
gdp_edu.quartile = cut(gdp_edu.Rank, 5)
```

[312]:
```
189-element CategoricalArray{String,1,UInt32}:
 "Q5: [152.4, 190.0]"
 "Q3: [76.2, 113.8)"
 "Q2: [38.6, 76.2)"
 "Q4: [113.8, 152.4)"
 "Q1: [1.0, 38.6)"
 "Q1: [1.0, 38.6)"
 "Q4: [113.8, 152.4)"
 "Q5: [152.4, 190.0]"
 "Q1: [1.0, 38.6)"
 "Q1: [1.0, 38.6)"
 "Q2: [38.6, 76.2)"
 "Q5: [152.4, 190.0]"
 "Q1: [1.0, 38.6)"

 "Q1: [1.0, 38.6)"
 "Q2: [38.6, 76.2)"
 "Q5: [152.4, 190.0]"
 "Q1: [1.0, 38.6)"
 "Q2: [38.6, 76.2)"
 "Q5: [152.4, 190.0]"
 "Q5: [152.4, 190.0]"
 "Q3: [76.2, 113.8)"
 "Q1: [1.0, 38.6)"
 "Q3: [76.2, 113.8)"
 "Q3: [76.2, 113.8)"
 "Q4: [113.8, 152.4)"
```

Stack the variables - make a table, so that the new value column is differentiated by the income group

[316]:
```
stacked_income_quartile = stack(gdp_edu, :"Income Group", :quartile)
```

[316]:

	quartile	variable	value
	Cat…	String	String31?
1	Q5: [152.4, 190.0]	Income Group	High income: nonOECD
2	Q3: [76.2, 113.8)	Income Group	Low income
3	Q2: [38.6, 76.2)	Income Group	Lower middle income
4	Q4: [113.8, 152.4)	Income Group	Upper middle income
5	Q1: [1.0, 38.6)	Income Group	High income: nonOECD
6	Q1: [1.0, 38.6)	Income Group	Upper middle income
7	Q4: [113.8, 152.4)	Income Group	Lower middle income
8	Q5: [152.4, 190.0]	Income Group	Upper middle income
9	Q1: [1.0, 38.6)	Income Group	High income: OECD
10	Q1: [1.0, 38.6)	Income Group	High income: OECD
11	Q2: [38.6, 76.2)	Income Group	Upper middle income
12	Q5: [152.4, 190.0]	Income Group	Low income
13	Q1: [1.0, 38.6)	Income Group	High income: OECD
14	Q4: [113.8, 152.4)	Income Group	Low income
15	Q4: [113.8, 152.4)	Income Group	Low income
16	Q2: [38.6, 76.2)	Income Group	Low income
17	Q2: [38.6, 76.2)	Income Group	Upper middle income
18	Q3: [76.2, 113.8)	Income Group	High income: nonOECD
19	Q4: [113.8, 152.4)	Income Group	High income: nonOECD
20	Q3: [76.2, 113.8)	Income Group	Upper middle income
21	Q2: [38.6, 76.2)	Income Group	Upper middle income
22	Q5: [152.4, 190.0]	Income Group	Lower middle income
23	Q4: [113.8, 152.4)	Income Group	High income: nonOECD
24	Q3: [76.2, 113.8)	Income Group	Lower middle income
25	Q1: [1.0, 38.6)	Income Group	Upper middle income
26	Q5: [152.4, 190.0]	Income Group	High income: nonOECD
27	Q3: [76.2, 113.8)	Income Group	High income: nonOECD
28	Q5: [152.4, 190.0]	Income Group	Lower middle income
29	Q4: [113.8, 152.4)	Income Group	Upper middle income
30	Q5: [152.4, 190.0]	Income Group	Low income
…	…	…	…

Subset the dataframe so that only the lower middle income entries are reatined

[328]:
```
lower_middle_income_quartile = filter(row -> row.value == "Lower middle␣
↪income", stacked_income_quartile)
```

[328]:

	quartile	variable	value
	Cat…	String	String31?
1	Q2: [38.6, 76.2)	Income Group	Lower middle income
2	Q4: [113.8, 152.4)	Income Group	Lower middle income
3	Q5: [152.4, 190.0]	Income Group	Lower middle income
4	Q3: [76.2, 113.8)	Income Group	Lower middle income
5	Q5: [152.4, 190.0]	Income Group	Lower middle income
6	Q1: [1.0, 38.6)	Income Group	Lower middle income
7	Q3: [76.2, 113.8)	Income Group	Lower middle income
8	Q3: [76.2, 113.8)	Income Group	Lower middle income
9	Q4: [113.8, 152.4)	Income Group	Lower middle income
10	Q5: [152.4, 190.0]	Income Group	Lower middle income
11	Q2: [38.6, 76.2)	Income Group	Lower middle income
12	Q1: [1.0, 38.6)	Income Group	Lower middle income
13	Q5: [152.4, 190.0]	Income Group	Lower middle income
14	Q4: [113.8, 152.4)	Income Group	Lower middle income
15	Q3: [76.2, 113.8)	Income Group	Lower middle income
16	Q5: [152.4, 190.0]	Income Group	Lower middle income
17	Q3: [76.2, 113.8)	Income Group	Lower middle income
18	Q1: [1.0, 38.6)	Income Group	Lower middle income
19	Q1: [1.0, 38.6)	Income Group	Lower middle income
20	Q2: [38.6, 76.2)	Income Group	Lower middle income
21	Q3: [76.2, 113.8)	Income Group	Lower middle income
22	Q5: [152.4, 190.0]	Income Group	Lower middle income
23	Q4: [113.8, 152.4)	Income Group	Lower middle income
24	Q2: [38.6, 76.2)	Income Group	Lower middle income
25	Q5: [152.4, 190.0]	Income Group	Lower middle income
26	Q2: [38.6, 76.2)	Income Group	Lower middle income
27	Q4: [113.8, 152.4)	Income Group	Lower middle income
28	Q5: [152.4, 190.0]	Income Group	Lower middle income
29	Q5: [152.4, 190.0]	Income Group	Lower middle income
30	Q4: [113.8, 152.4)	Income Group	Lower middle income
…	…	…	…

Now perform a final countmap() on the newly added quartile variable to answer our question - how many lower middle income countries belong to the upper ranges of the GDP quartile?!

[329]:
```
countmap(lower_middle_income_quartile.quartile)
```

[329]:
```
Dict{CategoricalValue{String, UInt32}, Int64} with 5 entries:
  "Q4: [113.8, 152.4)" => 9
  "Q5: [152.4, 190.0]" => 16
  "Q3: [76.2, 113.8)"  => 11
  "Q2: [38.6, 76.2)"   => 13
  "Q1: [1.0, 38.6)"    => 5
```

The answer is 5 !