R for Business 1

1201 Data Science: Tim Raiswell

2018/10/29

Quick Update

Last week's polling exercise revealed that you want:

- More training in basic programming for data science.
- Guidance on how what we cover can be related to your work.
- A curriculum.

A is for Accuracy

Definition

The proportion of correct predictions by a machine learning model.

$$Accuracy = \frac{Correct\ Predictions}{Total\ Results}$$

Definition

The proportion of correct predictions by a machine learning model.

$$Accuracy = \frac{Correct\ Predictions}{Total\ Results}$$

Higher accuracy is better than lower accuracy but it can be misleading. For example: consider a machine learning model that achieves 73% accuracy on a binary classification task. Sounds goods right?

(10 + 100) / (10 + 100 + 25 + 15 = 73.3% accuracy 🙌

	Fraud	No Fraud
Predicted Fraud	10	25
Predicted No Fraud	15	100

The Accuracy Paradox

What if we made a naive algorithm that just predicted 'no fraud' 100% of the time?

(0 + 125)/(0 + 125 + 0 + 25) = 83.3% accuracy 🤔

	Fraud	No Fraud
Predicted Fraud	0	0
Predicted No Fraud	25	125

The Takeaway

Be careful with accuracy as a model performance metric. Also consider precision and recall.

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives}$$

Basic Calculations in R

Standard Arithmetic

Open up R-Studio.

Try these entries in the console:

```
1 + 3
```

```
## [1] 4
```

```
52 - 23
```

```
## [1] 29
```

```
5 / 67
```

```
## [1] 0.07462687
```

```
72 * 51
```

```
## [1] 3672
```

Some New Functions

The round function gives you control of decimal places in your output.

```
round(5/67, 4)
```

```
## [1] 0.0746
```

```
round(5/67, 2)
```

```
## [1] 0.07
```

The modulo function gives you the remainder of the second number into the first for integer division:

```
67 %% 5
```

```
## [1] 2
```

Some New Functions

Let's check it.

```
(5*13) + 2 # 5x13 = 65 + 2 = 67
```

```
## [1] 67
```

Press Ctrl + L to clear your console.

To allocate an exponent to a number:

```
6 ^ 2 # 6-power-2, six-squared
```

```
# [1] 36
```

```
6 ^ 9 # 6-power-9
```

```
# [1] 10077696
```

Logic

Try this.

```
8 == 8 # the double-equality sign is R's version of "is equal to".
```

```
# [1] TRUE
```

Now this. What happens? What do you think `!=` does?

```
8 != 9 #
```

```
# [1] TRUE
```

Logic Operations

Operation	Operator	Example	Answer
Less than	<	4 < 10	TRUE
Less than or equal to	<=	4 <= 4	TRUE
Greater than	>	11 > 12	FALSE
Greater than or equal to	>=	4 >= 4	TRUE
Equal to	==	3 == 2	FALSE
Not equal to	!=	3 != 2	TRUE
Not	!	!(3 == 3)	FALSE
Or	|	(3==3) | (4==7)	TRUE
And	&	(3==3) & (4 == 7)	FALSE

Assignment

You will use the `<-` function more than any other in R.

The `<-` function is used here to assign values to a variable `cash_flow`:

```
cash_flow <- 1000000 # we assign an integer value
risk <- "high" # we assign a character string descriptive value

(cash_flow > 500000 & risk == "low") # logical tests can be used...
```

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

```
# ...to filter results
(cash_flow > 500000 & risk == "high")
```

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

What Are We Assigning?

The data and the type of data is coded within the assignment.

```
class(cash_flow)
```

```
## [1] "numeric"
```

```
class(risk)
```

```
## [1] "character"
```

Assign a logical value to the variable 'deep'. What kind of variable is it?

```
deep <- TRUE # note the cases. Try it with mixed case letters.
class(deep)
```

```
## [1] "logical"
```

More Examples

This is a categorical variable or a factor. Before it is passed to the factor function, it is a character string 'green'.

```
color <- as.factor('green')
class(color)
```

```
## [1] "factor"
```

```
color <- 'green'
class(color)
```

```
## [1] "character"
```

We have overwritten the factor variable `green` with the character string.

More Examples

More Examples

What type of variable is `colors`? What is the little `c`?

```
colors <- c('green', 'purple', 'blue')
```

More Examples

What type of variable is `colors`? What is the little `c`?

```r
colors <- c('green', 'purple', 'blue')
```

It is a vector of character strings. The `c` stands for concatenate.

You will use vectors and their close cousins matrices a lot as a data scientist.

```r
class(colors)
```

```
## [1] "character"
```

More Examples

What about this?

```
things <- c('purple', 15, 27, 'lilac', TRUE, as.factor('yellow'))
```

More Examples

What about this?

```
things <- c('purple', 15, 27, 'lilac', TRUE, as.factor('yellow'))
```

The vector function `c()` has coerced everything to a character string.

```
class(things)
```

```
## [1] "character"
```

There is a way of retaining the original variable types without coercing them into different classes.

Lists in R

```
new_things <- list('purple', 15, 'lilac', TRUE, as.factor('yellow'))
class(new_things)
```

```
## [1] "list"
```

Note that `list` is a class in and of itself, unlike a vector. We can access list items to see if they have retained their original classes.

```
item_3 <- new_things[[3]]
print(item_3)
```

```
## [1] "lilac"
```

```
class(item_3)
```

```
## [1] "character"
```

More Lists

We can name list elements.

```
names(new_things) <- c("fave color", "number",
                       "another color", "logical thingy", "a factor")
print(new_things)
```

```
## $`fave color`
## [1] "purple"
##
## $number
## [1] 15
##
## $`another color`
## [1] "lilac"
##
## $`logical thingy`
## [1] TRUE
##
## $`a factor`
## [1] yellow
## Levels: yellow
```

Catch Your Breath

We've covered:

- Accuracy
- Arithmetic
- Logical operators
- Vectors
- Lists

Now for:

- Matrices
- Data frames
- Pipes
- Data transformation

The Matrix 🥋

Definition

A matrix is a rectangular array of numbers or other mathematical objects for which operations such as addition and multiplication are defined.

```
example_matrix <- matrix(data = c(2,8,12,16), nrow = 2, ncol = 2)
print(example_matrix)
```

```
##      [,1] [,2]
## [1,]    2   12
## [2,]    8   16
```

```
print(t(example_matrix)) # t is matrix transpose
```

```
##      [,1] [,2]
## [1,]    2    8
## [2,]   12   16
```

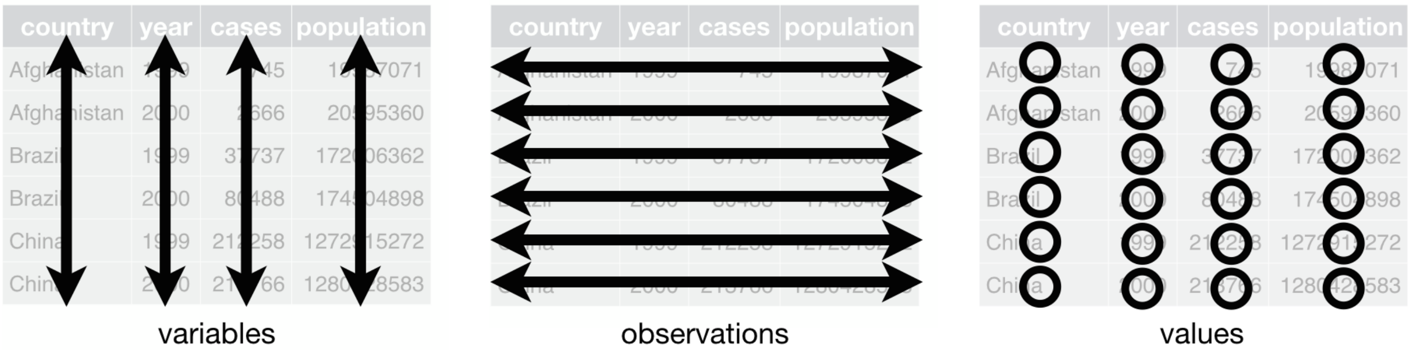
The Matrix Continued

For the most part you don't need to worry about matrices unless you're running certain machine learning types that require matrix manipulation, or natural language processing. It's important to know what matrices are because machine learning is based on them[1].

[1]Because linear algebra

The Almighty Data Frame

Tidy Data



Source: R for Data Science

Is This Tidy Data?

```r
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────── t

## ✔ ggplot2 3.1.0     ✔ purrr   0.2.5
## ✔ tibble  1.4.2     ✔ dplyr   0.7.7
## ✔ tidyr   0.8.2     ✔ stringr 1.3.1
## ✔ readr   1.1.1     ✔ forcats 0.3.0

## ── Conflicts ──────────────────────────────────────────────────── tidyver
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```r
table4b
```

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
## * <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

How About This?

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Data Frames Versus Tibbles

1. A tibble IS a data frame created for the Tidyverse, with several unique properties:

2. It displays simply.

3. It's harder to make errors with a tibble when you subset it.

Data Frame Examples

```
class(cars)
```

```
## [1] "data.frame"
```

```
cars # uh oh, where'd our data go?
```

```
##    speed dist
## 1      4    2
## 2      4   10
## 3      7    4
## 4      7   22
## 5      8   16
## 6      9   10
## 7     10   18
## 8     10   26
## 9     10   34
## 10    11   17
## 11    11   28
## 12    12   14
## 13    12   20
## 14    12   24
```

Data Frame Examples

```
new_cars <- as_data_frame(cars)
class(new_cars)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
new_cars
```

```
## # A tibble: 50 x 2
##    speed  dist
##    <dbl> <dbl>
##  1     4     2
##  2     4    10
##  3     7     4
##  4     7    22
##  5     8    16
##  6     9    10
##  7    10    18
##  8    10    26
##  9    10    34
## 10    11    17
## # ... with 40 more rows
```

And Then...

%>%

```
table1 %>% # and then...
  mutate(per_capita = cases / population) #...create a per_capita co
```

```
## # A tibble: 6 x 5
##   country      year  cases population per_capita
##   <chr>       <int>  <int>      <int>      <dbl>
## 1 Afghanistan  1999    745   19987071  0.0000373
## 2 Afghanistan  2000   2666   20595360  0.000129
## 3 Brazil       1999  37737  172006362  0.000219
## 4 Brazil       2000  80488  174504898  0.000461
## 5 China        1999 212258 1272915272  0.000167
## 6 China        2000 213766 1280428583  0.000167
```

The Alternative

```
(mutate(table1, per_capita = cases / population))
```

```
## # A tibble: 6 x 5
##   country      year  cases population per_capita
##   <chr>       <int>  <int>      <int>      <dbl>
## 1 Afghanistan  1999    745   19987071  0.0000373
## 2 Afghanistan  2000   2666   20595360  0.000129
## 3 Brazil       1999  37737  172006362  0.000219
## 4 Brazil       2000  80488  174504898  0.000461
## 5 China        1999 212258 1272915272  0.000167
## 6 China        2000 213766 1280428583  0.000167
```

Coming Into Its Own

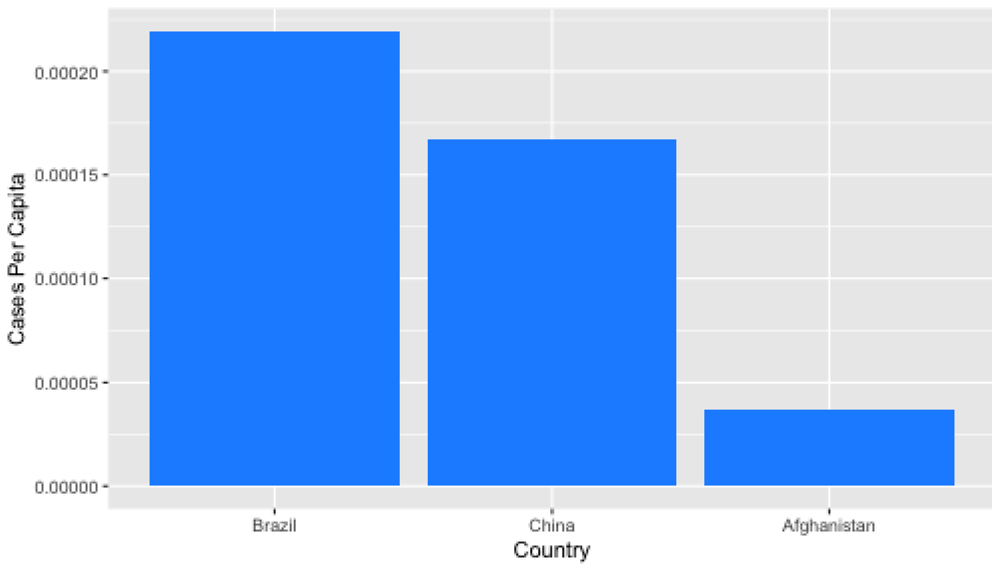
```
table1 %>%
  filter(year == 1999)
```

```
table1 %>% # building an analyitc pipeline!
  filter(year == 1999) %>%
  mutate(per_capita = cases / population)
```

```
## # A tibble: 3 x 5
##   country      year  cases population per_capita
##   <chr>       <int>  <int>      <int>      <dbl>
## 1 Afghanistan  1999    745   19987071  0.0000373
## 2 Brazil       1999  37737  172006362  0.000219
## 3 China        1999 212258 1272915272  0.000167
```

Culminating in Data Visualization

```
table1 %>%
  filter(year == 1999) %>%
  mutate(per_capita = cases / population) %>%
  ggplot(aes(reorder(country, -per_capita), per_capita)) + # call to
  geom_bar(stat = "identity", fill = "dodgerblue") + # bar chart
  labs(x = "Country", y = "Cases Per Capita") # axis labels
```



Hold On a Minute

Call	Function Name	Purpose
filter()	Filter	Filters a data frame based on single or multiple named variables and accompanying logical operators.
mutate()	Mutate	Creates a new variable with parameters specified, e.g. sum of existing variables, or altered variable type.
ggplot()	ggplot	Calls a chart plotting function.
geom_bar()	Bar chart geom	Geoms add layers to charts. They tell ggplot which data to use in which format.
labs()	Labels	Tells ggplot how you want to label everything.

Let's Practice a Little

Let's Practice a Little

Filter `table1` to show only the observations for China.

Let's Practice a Little

Filter `table1` to show only the observations for China.

```
table1 %>%
  filter(country == "China")
```

```
## # A tibble: 2 x 4
##   country  year  cases population
##   <chr>   <int>  <int>      <int>
## 1 China    1999 212258 1272915272
## 2 China    2000 213766 1280428583
```

Let's Practice a Little

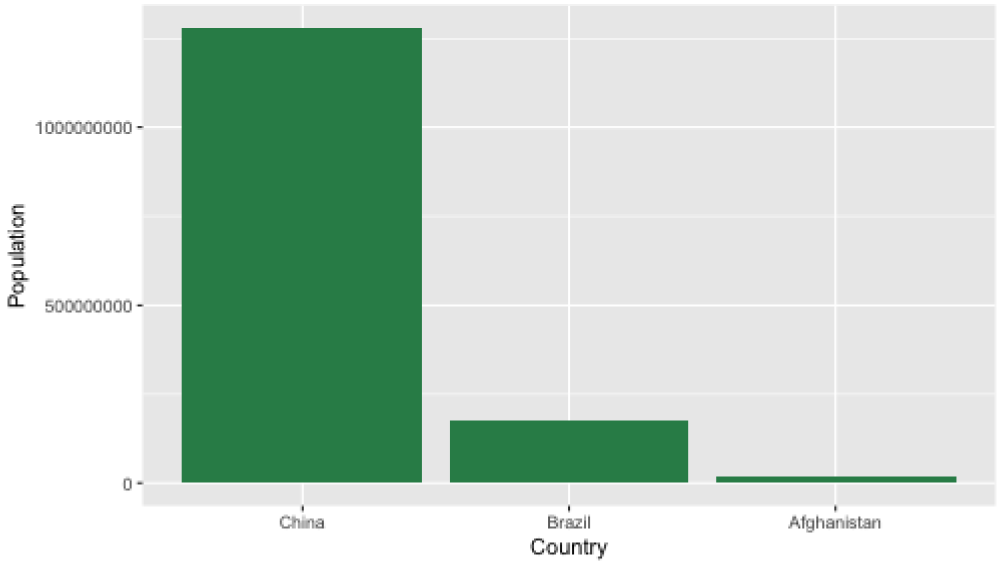
Let's Practice a Little

Create a chart with `table1` that shows the population of the three countries for the year 2000, and use the color `seagreen` to fill the bar chart.

Let's Practice a Little

Create a chart with `table1` that shows the population of the three countries for the year 2000, and use the color `seagreen` to fill the bar chart.

```
table1 %>%
  filter(year == 2000) %>%
  ggplot(aes(reorder(country, -population), population)) + # call to
  geom_bar(stat = "identity", fill = "seagreen") + # bar chart
  labs(x = "Country", y = "Population") # axis labels
```



Let's Practice a Little More

Use the `names()` function to get the names of the variables in `table1`.

Let's Practice a Little More

Use the `names()` function to get the names of the variables in `table1`.

```
names(table1)
```

```
## [1] "country"    "year"       "cases"      "population"
```

Where did our `per_capita` variable go?

Let's Practice a Little More

We didn't assign it to a new output, so it only existed in the output momentarily.

Recreate the variable for all years in the dataset (i.e. no need to filter for years), but this time assign the updated tibble to an object called `per_capita_calc`. Then print it.

Let's Practice a Little More

We didn't assign it to a new output, so it only existed in the output momentarily.

Recreate the variable for all years in the dataset (i.e. no need to filter for years), but this time assign the updated tibble to an object called `per_capita_calc`. Then print it.

```
per_capita_calc <- table1 %>%
  mutate(per_capita = cases / population)

print(per_capita_calc)
```

```
## # A tibble: 6 x 5
##   country      year  cases population per_capita
##   <chr>       <int>  <int>      <int>      <dbl>
## 1 Afghanistan  1999    745   19987071  0.0000373
## 2 Afghanistan  2000   2666   20595360  0.000129
## 3 Brazil       1999  37737  172006362  0.000219
## 4 Brazil       2000  80488  174504898  0.000461
## 5 China        1999 212258 1272915272  0.000167
## 6 China        2000 213766 1280428583  0.000167
```