Introduction to Data Science: Operations

Héctor Corrada Bravo

University of Maryland, College Park, USA

2020-01-29

Principles: Basic Operations

Now that we have a data frame describing our data, let's learn a few fundamental operations we perform on data frames on almost any analysis.

We divide these first set of operations into two groups:

- operations on *attributes*
- operations on *entitites*.

Operations that subset attributes

`select`

Suppose we only want to study patterns in these arrests based on a smaller number of attributes.

In that case we would like to create a data frame that contains only those attributes of interest.

Operations that subset attributes

Let's create a data frame containing only the `age`, `sex` and `district` attributes

```
select(arrest_tab, age, sex, district)
```

```
## # A tibble: 104,528 x 3
##      age sex   district
##    <dbl> <chr> <chr>
##  1    23 M     <NA>
##  2    37 M     SOUTHERN
##  3    46 M     NORTHEASTERN
##  4    50 M     WESTERN
```

Operations that subset attributes

We can use an operator to describe ranges. E.g., `1:5` would be attributes 1 through 5:

```
select(arrest_tab, 1:5)
```

```
## # A tibble: 104,528 x 5
##      arrest   age race  sex   arrestDate
##       <dbl> <dbl> <chr> <chr> <chr>
##  1 11126858    23 B     M     01/01/2011
##  2 11127013    37 B     M     01/01/2011
##  3 11126887    46 B     M     01/01/2011
##  4 11126873    50 B     M     01/01/2011
```

Operations that subset entities

`slice`

We can choose specific entities by their row position. For instance, to choose entities in rows 1,3 and 10, we would use the following:

```
slice(arrest_tab, c(1, 3, 10))
```

Operations that subset entities

As before, the first argument is the data frame to operate on.

The second argument is a *vector* of indices.

We used the `c` function (for concatenate) to create a vector of indices.

Operations that subset entities

We can also use the range operator here:

```
slice(arrest_tab, 1:5)
```

```
## # A tibble: 5 x 15
##   arrest   age race  sex   arrestDate arrestTime arrestLocation
##    <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
## 1 1.11e7    23 B     M     01/01/2011 00'00"     <NA>
## 2 1.11e7    37 B     M     01/01/2011 01'00"     2000 Wilkens …
## 3 1.11e7    46 B     M     01/01/2011 01'00"     2800 Mayfield…
## 4 1.11e7    50 B     M     01/01/2011 04'00"     2100 Ashburto…
## 5 1.11e7    33 B     M     01/01/2011 05'00"     4000 Wilsby A…
```

Operations that subset entities

To create general sequences of indices we would use the `seq` function. For example, to select entities in even positions we would use the following:

```
slice(arrest_tab, seq(2, nrow(arrest_tab), by=2))
```

```
## # A tibble: 52,264 x 15
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.11e7    37 B     M     01/01/2011 01'00"     2000 Wilkens …
##  2 1.11e7    50 B     M     01/01/2011 04'00"     2100 Ashburto…
##  3 1.11e7    41 B     M     01/01/2011 05'00"     2900 Spellman…
```

Operations that subset entities

`filter`

We can also select entities based on attribute properties. For example, to select arrests where age is less than 18 years old, we would use the following:

```
filter(arrest_tab, age < 18)
```

```
## # A tibble: 463 x 15
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.11e7    17 B     M     01/03/2011 15:00      <NA>
```

Operations that subset entities

The second argument is an expression that evaluates to a vector of logical values (`TRUE` or `FALSE`), if the expression evaluates to TRUE for a given entity (row) then that entity (row) is part of the resulting data frame.

Operations that subset entities

Operators used frequently include:

==, !=: tests equality and inequality respectively (categorical, numerical, datetimes, etc.)
<, >, <=, >=: tests order relationships for ordered data types (not categorical)
!, &, |: not, and, or, logical operators

Operations that subset entities

To select arrests with ages between 18 and 25 we can use

```
filter(arrest_tab, age >= 18 & age <= 25)
```

```
## # A tibble: 35,770 x 15
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.11e7    23 B     M     01/01/2011 00:00      <NA>
##  2 1.11e7    20 W     M     01/01/2011 00:05      5200 Moravia …
##  3 1.11e7    24 B     M     01/01/2011 00:07      2400 Gainsdbo…
##  4 1.11e7    25 B     M     01/01/2011 00:20      2800 Violet A…
##  5 1.11e7    24 B     M     01/01/2011 00:40      3900 Greenmou…
```

Operations that subset entities

The filter function can take multiple logical expressions. In this case they are combined with &. So the above is equivalent to

```
filter(arrest_tab, age >= 18, age <= 25)
```

```
## # A tibble: 35,770 x 15
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.11e7    23 B     M     01/01/2011 00:00      <NA>
##  2 1.11e7    20 W     M     01/01/2011 00:05      5200 Moravia …
##  3 1.11e7    24 B     M     01/01/2011 00:07      2400 Gainsdbo…
##  4 1.11e7    25 B     M     01/01/2011 00:20      2800 Violet A…
```

Operations that subset entities

`sample_n` and `sample_frac`

Frequently we will want to choose entities from a data frame at random. The `sample_n` function selects a specific number of entities at random:

```
sample_n(arrest_tab, 10)
```

```
## # A tibble: 10 x 15
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.26e7    25 B     M     09/26/2012 22:25      0 N Howard St
##  2 1.14e7    22 B     F     11/10/2011 18:00      2700 Kinsey St
```

Operations that subset entities

The `sample_frac` function selects a fraction of entitites at random:

```
sample_frac(arrest_tab, .1)
```

```
## # A tibble: 10,453 x 15
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.13e7    34 B     M     09/26/2011 19:30      <NA>
##  2 1.25e7    20 B     M     04/05/2012 04:30      1300 N Calhou…
##  3 1.11e7    26 B     M     02/04/2011 10:10      <NA>
##  4 1.25e7    20 B     M     09/05/2012 10:45      <NA>
##  5 1.26e7    32 B     M     11/08/2012 08:35      3800 Brehms Ln
```

Pipelines of operations

All of the functions implementing our first set of operations have the same argument/value structure.

They take a data frame as a first argument and return a data frame. We refer to this as the *data-->transform-->data* pattern.

This is the core a lot of what we will do in class as part of data analyses.

Specifically, we will combine operations into *pipelines* that manipulate data frames.

In R, the `dplyr` package introduces *syntactic sugar* to make this pattern explicit.

```
arrest_tab %>%

  sample_frac(.1)
```

```
## # A tibble: 10,453 x 15

##     arrest   age race  sex   arrestDate arrestTime arrestLocation

##      <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>

##  1  1.24e7    59 W     M     01/14/2012 17:50      600 Monroe St

##  2  1.12e7    44 B     M     05/01/2011 00:30      <NA>

##  3  1.24e7    26 W     M     03/28/2012 02:15      <NA>

##  4  1.26e7    23 B     M     12/05/2012 11:28      <NA>

##  5  1.26e7    31 B     M     09/24/2012 17:20      1900 E Federa…
```

The `%>%` binary operator takes the value to its **left** and inserts it as the first argument of the function call to its **right**. So the expression `LHS %>% f(another_argument)` is **equivalent** to the expression `f(LHS, another_argument)`.

In `pandas`, you can chain `.` calls.

Using the %>% operator and the *data-->transform-->data* pattern of the functions we've seen so far, we can create pipelines.

For example, let's create a pipeline that:

1) filters our dataset to arrests between the ages of 18 and 25 2) selects attributes `sex`, `district` and `arrestDate` (renamed as `arrest_date`) 3) samples 50% of those arrests at random

We will assign the result to variable `analysis_tab`

```
analysis_tab <- arrest_tab %>%
  filter(age >= 18, age <= 25) %>%
  select(sex, district, arrest_date=arrestDate) %>%
  sample_frac(.5)
analysis_tab
```

```
## # A tibble: 17,885 x 3
##    sex   district     arrest_date
##    <chr> <chr>        <chr>
##  1 M     EASTERN      12/14/2012
##  2 F     <NA>         08/10/2011
##  3 M     <NA>         03/09/2012
##  4 M     <NA>         04/06/2012
##  5 M     WESTERN      01/06/2011
```

Exercise: Create a pipeline that:

1) filters dataset to arrests from the "SOUTHERN" district occurring before "12:00" (`arrestTime`)

2) selects attributes, `sex`, `age`

3) samples 10 entities at random

Principles: More Operations

Next, we learn a few more fundamental data operations: sorting, creating new attributes, summarizing and grouping.

Finally we will take a short detour through a discussion on vectors.

Operations that sort entities

Re-order entities based on the value of their `age` attribute, and then `slice` to create a data frame with just the entities of interest

```
arrest_tab %>%
  arrange(age) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 15
##     arrest   age race  sex   arrestDate arrestTime arrestLocation
##      <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1  1.11e7     0 B     F     01/24/2011 12:45      3700 Garrison…
##  2  1.12e7     0 W     M     03/22/2011 08:00      <NA>
```

Operations that sort entities

The `arrange` operation sorts entities by increasing attribute values. Use `desc` helper to sort by decreasing value. E.g., find the arrests with the 10 *oldest* subjects:

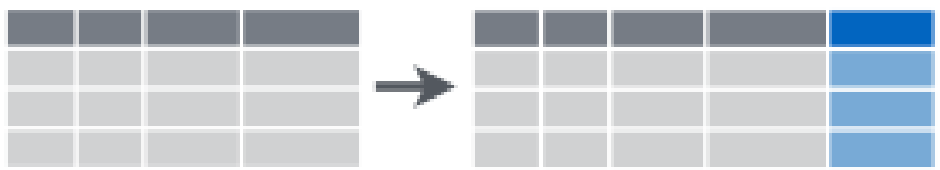
```
arrest_tab %>%
  arrange(desc(age)) %>%
  slice(1:10)
```

```
## # A tibble: 10 x 15
##     arrest   age race  sex   arrestDate arrestTime arrestLocation
##      <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1  1.13e7    87 B     M     08/28/2011 15:00      3200 E Baltim…
```

Operations that create new attributes

We will often see that for many analyses we will create new attributes based on existing attributes in a dataset.

- This is helpful for interpretation, visualization and/or statistical modeling.



Operations that create new attributes

Suppose I want to represent age in months rather than years in our dataset. To do so I would multiply 12 to the existing age attribute. The function `mutate` creates new attributes based on the result of a given expression:

```
arrest_tab %>%
  mutate(age_months = 12 * age) %>%
  select(arrest, age, age_months)
```

```
## # A tibble: 104,528 x 3
##      arrest   age age_months
##       <dbl> <dbl>      <dbl>
```

Operations that summarize (aggregate) attribute values over entities

Collapse a data frame to a single row containing the desired attribute summaries.

Operations that summarize (aggregate) attribute values over entities

Find minmum, maximum and average age in the dataset:

```
summarize(arrest_tab, min_age=min(age), mean_age=mean(age), max_age=max(age))
```

```
## # A tibble: 1 x 3
##   min_age mean_age max_age
##     <dbl>    <dbl>   <dbl>
## 1       0     33.2      87
```

Operations that summarize (aggregate) attribute values over entities

Operation(s)	Result
`mean`, `median`	average and median attribute value
`sd`	standard deviation of attribute values
`min`, `max`	minimum and maximum attribute values
`n`, `n_distinct`	number of attribute values and number of *distinct* attribute values
`any`, `all`	is `any` attribute value TRUE, or are `all` attribute values TRUE

Operations that summarize (aggregate) attribute values over entities

Let's see the number of distinct districts in our dataset:

```
summarize(arrest_tab, n_distinct(district))
```

```
## # A tibble: 1 x 1
##   `n_distinct(district)`
##                    <int>
## 1                     10
```

We may also refer to these summarization operation as **aggregation** since we are computing *aggregates* of attribute values.

Operations that group entities

Summarization (aggregation) goes hand in hand with data grouping, where summaries are computed *conditioned* on other attributes.

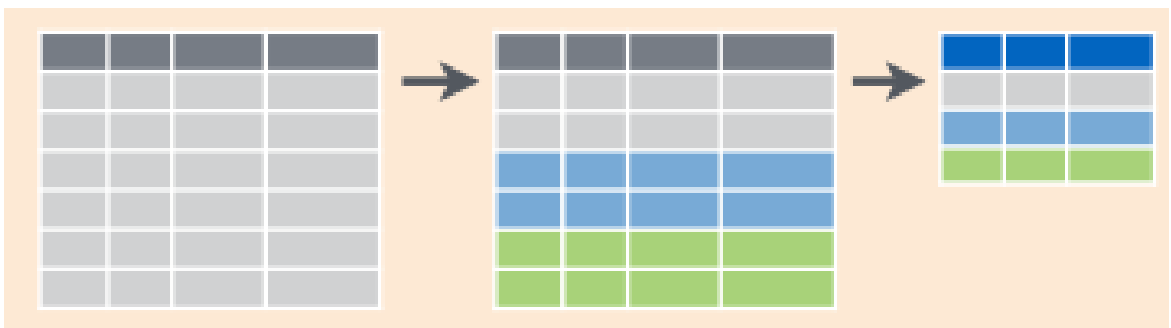
The notion of *conditioning* is fundamental to data analysis and we will see it very frequently through the course.

It is the basis of statistical analysis and Machine Learning models and it is essential in understanding the design of effective visualizations.

Operations that group entities

The goal is to group entities with the same value of one or more attributes.

The `group_by` function in essence annotates the rows of a data frame as belonging to a specific group based on the value of some chosen attributes.



Operations that group entities

Group entities by the value of the `district` attribute.

```
group_by(arrest_tab, district)
```

```
## # A tibble: 104,528 x 15
## # Groups:   district [10]
##    arrest   age race  sex   arrestDate arrestTime arrestLocation
##     <dbl> <dbl> <chr> <chr> <chr>      <time>     <chr>
##  1 1.11e7    23 B     M     01/01/2011 00'00"     <NA>
##  2 1.11e7    37 B     M     01/01/2011 01'00"     2000 Wilkens …
##  3 1.11e7    46 B     M     01/01/2011 01'00"     2800 Mayfield…
##  4 1.11e7    50 B     M     01/01/2011 04'00"     2100 Ashburto…
```

Operations that group entities

Subsequent operations are then performed **for each group independently**.

For example, when `summarize` is applied to a grouped data frame, summaries are computed for each group of entities, rather than the whole set of entities.

Operations that group entities

Calculate minimum, maximum and average age for each district:

```
arrest_tab %>%
  group_by(district) %>%
  summarize(min_age=min(age), max_age=max(age), mean_age=mean(age))
```

```
## # A tibble: 10 x 4
##    district     min_age max_age mean_age
##    <chr>          <dbl>   <dbl>    <dbl>
##  1 CENTRAL            0      86     33.0
##  2 EASTERN            0      85     34.1
##  3 NORTHEASTERN       0      78     30.4
```

Operations that group entities

`group_by/summarize` defines new entities.

The entities in our original dataset are arrests. The entities for the result of the last example are the districts.

This is a general property of group_by and summarize: it defines a data set where entities are defined by distinct values of the attributes we use for grouping.

Operations that group entities

Another example: average age for subjects 21 years or older grouped by district and sex:

```
arrest_tab %>%
  filter(age >= 21) %>%
  group_by(district, sex) %>%
  summarize(mean_age=mean(age))
```

```
## # A tibble: 20 x 3
## # Groups:   district [10]
##    district     sex   mean_age
##    <chr>        <chr>    <dbl>
```

Operations that group entities

Exercise: Write a data operation pipeline that

1) filters records to the southern district and ages between 18 and 25 2) computes mean arrest age for each sex

Vectors

We briefly saw previously operators to create vectors in R. For instance, we can use `seq` to create a vector that consists of a sequence of integers:

```
multiples_of_three <- seq(3, 30, by=3)

multiples_of_three
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30
```

Vectors

Let's how this is represented in R (the `str` is very handy to do this type of digging around):

```
str(multiples_of_three)
```

```
##  num [1:10] 3 6 9 12 15 18 21 24 27 30
```

Vectors

Like many other languages we use square brackets `[]` to index vectors:

```
multiples_of_three[1]
```

```
## [1] 3
```

Vectors

We can use ranges as before

```
multiples_of_three[1:4]
```

```
## [1]  3  6  9 12
```

Vectors

We can use vectors of non-negative integers for indexing:

```
multiples_of_three[c(1,3,5)]
```

```
## [1]  3  9 15
```

Vectors

Or even logical vectors:

```
multiples_of_three[c(TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE)]
```

```
## [1]  3  9 15 21 27
```

Vectors

In R, most operations are designed to work with vectors directly (we call that *vectorized*).

For example, if I want to add two vectors together I would write: (look no `for` loop!):

```
multiples_of_three + multiples_of_three
```

```
##  [1]  6 12 18 24 30 36 42 48 54 60
```

This also works for other arithmetic and logical operations (e.g., `-`, `*`, `/`, `&`, `|`).

Vectors

In data analysis the *vector* is probably the most fundamental data type (other than basic numbers, strings, etc.).

Why? Consider getting data about one attribute, say height, for a group of people. What do you get? An vector of numbers, all in the same unit (say feet, inches or centimeters).

How about their name? Then you get a vector of strings.

Abstractly, we think of vectors as arrays of values, all of the same *class* or datatype.

Attributes as vectors

Each column, corresponding to an attribute, is a vector. We use the `pull` function to extract a vector from a data frame.

We can then operate index them, or operate on them as vectors

```
age_vec <- arrest_tab %>% pull(age)

age_vec[1:10]
```

```
##  [1] 23 37 46 50 33 41 29 20 24 53
```

Attributes as vectors

Or,

```
12 * age_vec[1:10]
```

```
##  [1] 276 444 552 600 396 492 348 240 288 636
```

Attributes as vectors

The $ operator serves the same function.

```
age_vec <- arrest_tab$age

age_vec[1:10]
```

```
##  [1] 23 37 46 50 33 41 29 20 24 53
```

Attributes as vectors

The `pull` function however, can be used as part of a pipeline (using operator %>%):

```
arrest_tab %>%
  pull(age) %>%
  mean()
```

```
## [1] 33.19639
```

Attributes as vectors

Functions

How to abstract pipelines? Factor into reusable functions that we can apply in other analyses. E.g., a function that executes the age by district/sex summarization we created before:

```
summarize_district <- function(df) {
  df %>%
    filter(age >= 21) %>%
    group_by(district, sex) %>%
    summarize(mean_age=mean(age))
}
```

Attributes as vectors

You can include multiple expressions in the function definition (inside brackets `{}`). Notice there is no `return` statement in this function. When a function is called, it returns the value of the last expression in the function definition. In this example, it would be the data frame we get from applying the pipeline of operations.

Attributes as vectors

You can find more information about vectors, functions and other programming matters we might run into in class in Chapters 17-21 of R for Data Science

Attributes as vectors

Exercise Abstract the pipeline you wrote in the previous unit into a function that works for arbitrary districts. The function should take arguments `df` and `district`.