Data Wrangling with Dplyr - Cute cats, cute grammar

Francesco Maria Sabatini

Timestamp: Tue Mar 24 12:25:04 2020
Version: 1.1

`dplyr` provides a set of tools for efficiently manipulating datasets in `R`. It belongs to the `tidyverse`, which is a a collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

What I personally love is that `dplyr` is **synthetic, fast and easy to read**. It allows to write code which is easy to explain, revise and mantain. Knowing that whather analysis we do, we will probably have to adjust and re-run it over and over, using `dplyr`'s grammar now is the best present we can give to our *future selves*, and will help us save a *lot* of time later.

The overarching goals of this tutorial is to fall in love with `dplyr` and to see some cute wild cats. The specific goals are:
- Understand the logic of `dplyr` and the power of piping `%>%`
- Explore the key verbs of `dplyr`'s grammar
- Simulate a `dplyr` based workflow

You will learn how to use the key verbs of `dplyr`, including `select()`, `filter()`, `mutate()`, `summarize()`, `rename()`, as well as their generalized `x_at()` and `x_all()` versions. We'll also touch on the `join` family of functions.

We will first create a simulated dataframe using data from `GBIF` and environmental predictors from some global datasets.
We start loading some packages

```
library(tidyverse)
library(downloader)
library(rgbif)

## Spatial packages
#install.packages(c("rgdal", "sp", "sf", "raster", "rnaturalearth"))
library(rgdal)
library(sp)
library(sf)
library(raster)
library(rnaturalearth)

#save temporary files
rasterOptions(tmpdir="_tmp")
```

Import species data from `GBIF`

As a toy dataset, we will compare the climatic niche of the five cutest wild cats out there. The selection follows the authoritative source www.backyardcatenclosures.com.au.

Let's get familiar with our cute cats. We first create a data.frame of names, short names and urls to retrieve pictures of our cute cats:

```
myspecies <- c("Felis manul", #Palla's cat
               "Caracal caracal", #Caracal
               "Felis margarita", #Sand cat
               "Prionailurus rubiginosus", #Rusty spotted cat
               "Leopardus wiedii" #Margay
               )
shortnames <- c("Palla's cat", "Caracal", "Sand Cat", "Rusty Spotted Cat", "Margay")
urls <- c("https://upload.wikimedia.org/wikipedia/commons/d/d6/Manoel.jpg", #pallas
"https://upload.wikimedia.org/wikipedia/commons/a/a3/Caracl_%2801%29%2C_Paris%2C_d%C3%A9cembre_2013.jpg
  #caracal
  "https://upload.wikimedia.org/wikipedia/commons/e/e9/Persian_sand_CAT.jpg",
  #Sand cat
  "https://upload.wikimedia.org/wikipedia/commons/3/3e/Rusty_spotted_cat_1.jpg",
  #Rusty spotted cat
  "https://upload.wikimedia.org/wikipedia/commons/b/bd/Margay_in_Costa_Rica.jpg"
  #Margay
)

cute <- data.frame(species=myspecies,
                   short=shortnames,
                   url=urls)
```

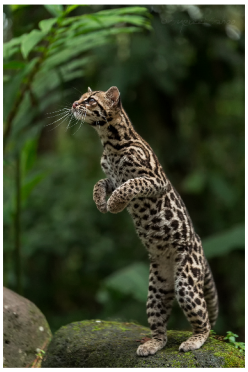
Palla's cat

Caracal

Sand cat

Rusty Spotted cat

Margay



All pictures from https://en.wikipedia.org/

Download occurrence data from GBIF

```
get.speciesKey <- function(x){name_backbone(x)$speciesKey} #get GBIF species key
key <- unlist(lapply(myspecies, get.speciesKey))
get.100occurrences <- function(x){occ_search(taxonKey=x, return="data", limit=100)}
#dat <- occ_search(taxonKey=key, return='data', limit=300)
dat <- lapply(key, get.100occurrences)
```

We produce a list of 5 elements, each with up to 100 occurrences for a cat species. Let's take a look at the output:

```
dat[[1]]
```

```
## # A tibble: 64 x 127
##    key   scientificName issues datasetKey publishingOrgKey installationKey
##  * <chr> <chr>          <chr>  <chr>      <chr>            <chr>
##  1 2462~ Felis manul P~ "cuiv" 5733a11d-~ 0870a77b-587c-4~ 5f02b486-8869-~
##  2 1914~ Otocolobus ma~ "cdro~ 50c9509d-~ 28eb1a3f-1c15-4~ 997448a8-f762-~
##  3 1935~ Otocolobus ma~ "cdro~ 50c9509d-~ 28eb1a3f-1c15-4~ 997448a8-f762-~
##  4 1931~ Otocolobus ma~ "cdro~ 50c9509d-~ 28eb1a3f-1c15-4~ 997448a8-f762-~
##  5 2557~ Otocolobus ma~ "cdro~ 50c9509d-~ 28eb1a3f-1c15-4~ 997448a8-f762-~
##  6 2557~ Otocolobus ma~ "cdro~ 50c9509d-~ 28eb1a3f-1c15-4~ 997448a8-f762-~
##  7 1934~ Felis manul P~ ""     84a4404a-~ 4fd82480-ea1c-1~ 82d6992a-b20b-~
##  8 1934~ Felis manul P~ ""     84a4404a-~ 4fd82480-ea1c-1~ 82d6992a-b20b-~
##  9 2557~ Otocolobus ma~ "cdro~ 50c9509d-~ 28eb1a3f-1c15-4~ 997448a8-f762-~
## 10 6659~ Felis manul P~ ""     41fc5c40-~ 7b8aff00-a9f8-1~ 3b84ff5b-79ce-~
```

```
## # ... with 54 more rows, and 121 more variables: publishingCountry <chr>,
## #   protocol <chr>, lastCrawled <chr>, lastParsed <chr>, crawlId <int>,
## #   extensions <chr>, basisOfRecord <chr>, taxonKey <int>, kingdomKey <int>,
## #   phylumKey <int>, classKey <int>, orderKey <int>, familyKey <int>,
## #   genusKey <int>, speciesKey <int>, acceptedTaxonKey <int>,
## #   acceptedScientificName <chr>, kingdom <chr>, phylum <chr>, order <chr>,
## #   family <chr>, genus <chr>, species <chr>, genericName <chr>,
## #   specificEpithet <chr>, taxonRank <chr>, taxonomicStatus <chr>, year <int>,
## #   month <int>, day <int>, eventDate <chr>, lastInterpreted <chr>,
## #   license <chr>, identifiers <chr>, facts <chr>, relations <chr>,
## #   class <chr>, countryCode <chr>, identifier <chr>, eventID <chr>,
## #   catalogNumber <chr>, institutionCode <chr>, locality <chr>, gbifID <chr>,
## #   occurrenceID <chr>, type <chr>, identificationID <chr>, name <chr>,
## #   decimalLatitude <dbl>, decimalLongitude <dbl>, dateIdentified <chr>,
## #   coordinateUncertaintyInMeters <dbl>, stateProvince <chr>, modified <chr>,
## #   references <chr>, geodeticDatum <chr>, country <chr>, rightsHolder <chr>,
## #   http...unknown.org.nick <chr>, informationWithheld <chr>,
## #   verbatimEventDate <chr>, datasetName <chr>, collectionCode <chr>,
## #   verbatimLocality <chr>, taxonID <chr>, recordedBy <chr>,
## #   http...unknown.org.occurrenceDetails <chr>, rights <chr>, eventTime <chr>,
## #   occurrenceRemarks <chr>, sex <chr>, vernacularName <chr>, language <chr>,
## #   higherClassification <chr>, recordNumber <chr>, higherGeography <chr>,
## #   institutionID <chr>, fieldNumber <chr>, organismID <chr>,
## #   ownerInstitutionCode <chr>, startDayOfYear <chr>, datasetID <chr>,
## #   accessRights <chr>, collectionID <chr>, establishmentMeans <chr>,
## #   continent <chr>, nomenclaturalCode <chr>, county <chr>, preparations <chr>,
## #   occurrenceStatus <chr>, dynamicProperties <chr>, endDayOfYear <chr>,
## #   otherCatalogNumbers <chr>, bibliographicCitation <chr>,
## #   X.1f2c0cbe.40df.43f6.ba07.e76133e78c31. <chr>, individualCount <int>,
## #   eventRemarks <chr>, identificationVerificationStatus <chr>,
## #   locationAccordingTo <chr>, identificationRemarks <chr>, ...
```

First, notice that each element of the list is a `tibble`, which is the `dplyr` equivalent of a `data.frame`. It has some cool tweaks (see how is visualizes in the console) compared to a data.frame. You should be aware, though, that some functions designed for data.frames may not work with tibbles. You can always convert back and forth from data.frame to tbl with the commands `as.data.frame()` and `as.tbl()`, anyways.
Our tibble is pretty horrible, actually. There are 121 columns, and a lot of information we probably do not need.

```
unlist(lapply(dat, "ncol"))
```

```
## [1] 127 108 131 122 124
```

Also notice how each element of the `dat` list contains a different number of columns, which complicates things. Time for a good data cleaning

Clean data

Let's see how `dplyr` can help us to clean up our data into something we can work with:
1) bind all rows into a single data.frame
2) select the columns we need (species, country, coordinates)
3) filter out all observation without spatial coordinates
4) rename columns to make our life easier later

We'll do this through piping

The command pipe `%>%` allows to use the output of a previous function as the input of the following. This means we can avoid a *lot* redundancy in our code. (Do yourself a favour and memorize the RStudio shortcut: `Ctrl + Shift + m`)

Let's compare the code in `base` vs. the code in `dplyr`.

```
#base
#Since each element of the list has a different number of columns,
#I cannot simply bind them by rows, but I first need to subset them to the same format.
dat_clean <- NULL
for(i in 1:5){
  dat_clean <- rbind(dat_clean,
                     dat[[i]][,which(colnames(dat[[i]]) %in%
                                       c("species", "country",
                                         "decimalLongitude", "decimalLatitude"))])
  }
dat_clean <- dat_clean[which(!is.na(dat_clean$decimalLatitude) &
                               !is.na(dat_clean$decimalLongitude)),]
colnames(dat_clean)[2:3] <- c("Lat", "Lon")
dat_clean <- dat_clean[,c(1,4,3,2)] #reorder columns
head(dat_clean)
```

```
## # A tibble: 6 x 4
##   species     country             Lon   Lat
##   <chr>       <chr>             <dbl> <dbl>
## 1 Felis manul Mongolia           113.   46.1
## 2 Felis manul China              103.   33.8
## 3 Felis manul China               99.6  35.9
## 4 Felis manul Russian Federation  89.4  49.9
## 5 Felis manul Russian Federation  88.3  50.2
## 6 Felis manul Russian Federation 115.   50.1
```

```
#dplyr
dat_clean <- dat %>%
  bind_rows() %>%
  dplyr::select(species, country, decimalLongitude, decimalLatitude) %>%
  filter(!is.na(decimalLongitude) & !is.na(decimalLatitude)) %>%
  rename(Lon=decimalLongitude, Lat=decimalLatitude)
head(dat_clean)
```

```
## # A tibble: 6 x 4
##   species     country             Lon   Lat
##   <chr>       <chr>             <dbl> <dbl>
## 1 Felis manul Mongolia           113.   46.1
## 2 Felis manul China              103.   33.8
## 3 Felis manul China               99.6  35.9
## 4 Felis manul Russian Federation  89.4  49.9
## 5 Felis manul Russian Federation  88.3  50.2
## 6 Felis manul Russian Federation 115.   50.1
```

Shorter, cleaner, easier

Congratulations. You did your first piping. A few features worth noting:
- In the first row, we assign the output to the object `dat_clean` and select `dat` as initial input of our chain of

commands. The object `dat` will be fed into the first function (`bind_rows()`). The output from the first function will be fed as first argument in the second function (`select()`) and so on
- `bind_rows()` is a pretty powerful alternative to rbind, which has the desirable properties of binding rows only when these have the same column name (!), which avoids us having to select columns separately for each element of the `dat` list in a loop
- For the function `select()` I specified the package because there's a conflict with the `raster::select()` function. When doing spatial analysis, it's more robust to consistently specifiy the package when conflicts may exist
- `dplyr` *DOES NOT* need quoting column names (but doesn't mind if you quote either). It may be confusing at first, but it allows to work quicker, and avoids filling up your code with distracting elements.
- `filter` subsets the rows we want to keep in a data.frame, based on a (set of) condition(s). Note, again, that the columns we are applying the conditions on *are not* enclosed by quotation marks.

Imagine we wanted to select more columns from each element of the `dat` list, or maybe simply reorder the column order. It can be tedious to explicitly specify all the column names to select. `dplyr` is the best friend of lazy people, and provides many quick alternatives for selecting multiple columns.

```
dat2 <- dat %>%
  bind_rows() %>%
  dplyr::select(kingdom:species, year:day, country, decimalLongitude,
                decimalLatitude, everything())
```

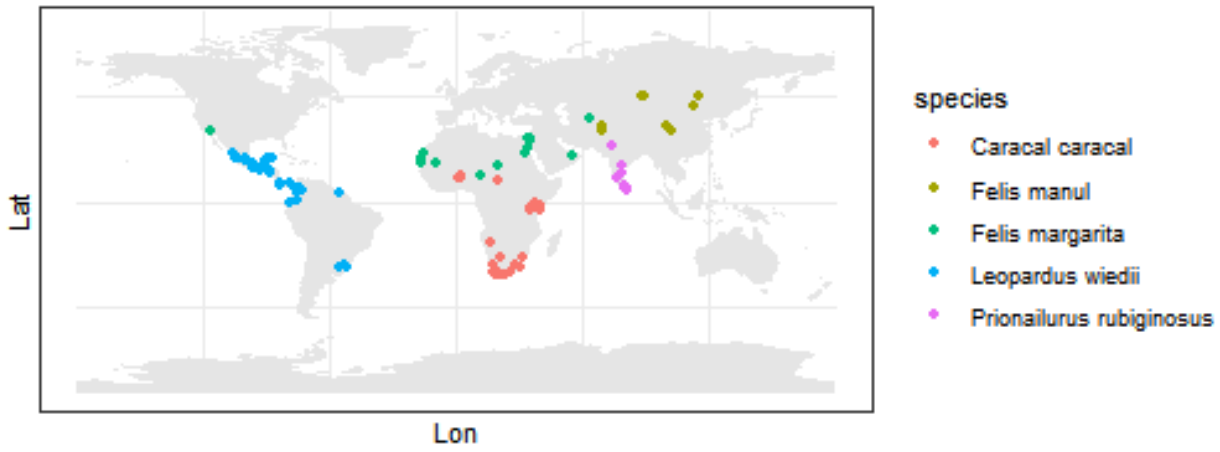
In the example, we reorder the elements from `dat`. Look how we used the `:` colon sign to indicate *from* `kingdom` *to* `species`. See also how we used the function `everything()` to select all the remaining columns (which is extremely useful in the context of reordering columns columns).

Create a map of the data

We are now ready to plot these points on a map. The map of the worls here derives from the package `rnaturalearth`. Plotting is done using `ggplot` another important member of the `tidyverse`. Spatial data are handled using the `sf` library. All very cool stuff that we don't discuss here.

```
countries <- ne_countries(returnclass = "sf") %>%
  st_geometry()

## basic graph of the world
ggplot() +
  geom_sf(data = countries, fill = "grey90", col = NA, lwd = 0.3) +
  geom_point(data=dat_clean, aes(x=Lon, y=Lat, col=species)) +
  theme_bw()
```



There's a point for the Sand cat which is clearly out of the species native range. We delete it using `filter`.

```
dat_clean <- dat_clean %>%
  filter(!(species=="Felis margarita" & country=="United States of America"))
```

Extract predictors from global datasets

Imagine we are interesting in calculating some metrics related to the environmental niche of the five cute cat species. To do spatial operations, we need to transform our dataset into a `SpatialPointDataFrame()`. This is a basic class of objects from the `sp` package, but we don't discuss it here.

```
dat.shp <- SpatialPointsDataFrame(coords=dat_clean %>%
                                    dplyr::select(Lon, Lat),
                                  data=dat_clean %>%
                                    dplyr::select(species, country),
                                  proj4string = CRS("+init=epsg:4326")) #Lat long in WGS 84
```

See how we nested short pipelines of commands *within* the arguments of the `SpatialPointsDataFrame` function.

We can then download Bioclimatic variables from CHELSA
We focus on two only:

Bio1 = Annual Mean Temperature
Bio12 = Annual Precipitation

Download raster files with `downloader`:

```
dir.create("Ancillary_data")
dir.create("Ancillary_data/CHELSA")
chelsa.url <- "https://envidatrepo.wsl.ch/uploads/chelsa/chelsa_V1/bioclim/integer/"
download(url = pasteo(chelsa.url, "CHELSA_bio10_01.tif"),
        "Ancillary_Data/CHELSA/CHELSA_bio10_01.tif", mode = "wb")
download(url = paste0(chelsa.url, "CHELSA_bio10_12.tif",
        "Ancillary_Data/CHELSA/CHELSA_bio10_12.tif", mode = "wb")
```

Load CHELSA rasters, and intersect `dat.shp` with each of them

```
bio01.raster <- raster("Ancillary_Data/CHELSA/CHELSA_bio10_01.tif")
bio12.raster <- raster("Ancillary_Data/CHELSA/CHELSA_bio10_12.tif")

bio01 <- raster::extract(bio01.raster, dat.shp) # °C * 10
bio12 <- raster::extract(bio12.raster, dat.shp) # mm
```

Bind columns and transform temperature data (they are multiplied by 10 so that they can be stored as integers).

```
envdata <- dat_clean %>%
  mutate(Temp=bio01) %>%
  mutate(P=bio12) %>%
  mutate(Temp=Temp/10)
```

The function `mutate()` is a foundation of `dplyr`. It allows creating and/or modifying a column, and can be used to do operations between columns.

Now things get interesting. To calculate the climatic niche we may want to calculate some statistics for each predictor for species (min, max, mean). `dplyr` comes to our help with the function `group_by()` (and its opposite `ungroup()`). Once you group a data.frame by a factor (or character!), all operations are applied on a group by group basis. Easy, quick, efficient. We can then use the function `summarize()`.

```
niche <- envdata %>%
  group_by(species) %>%
  summarize(n=n(),
            min.T=min(Temp, na.rm=T),
            mean.T=mean(Temp, na.rm=T),
            max.T=max(Temp, na.rm=T),
            min.P=min(P, na.rm=T),
            mean.P=mean(P, na.rm=T),
            max.P=max(P, na.rm=T))
```

Checkout how we used the function `n()` to count the number of observations for each group!.
!! *watch out* !! Once you group a `tibble` it stays grouped! Don't forget to ungroup afterwards!

How do our cute cats rank in terms of their resistance to maximum temperature? All we have to do is to `arrange()` our rows in a descending order.

```
niche %>%
  arrange(desc(max.T))
```

```
## # A tibble: 5 x 8
##   species                    n min.T mean.T max.T min.P mean.P max.P
##   <chr>                  <int> <dbl>  <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 Felis margarita           27  15.9  23.9   30.4     1   79.4   298
## 2 Caracal caracal          100  11.8  22.4   29.2   190  710.   1371
## 3 Leopardus wiedii          70   9.2  21.8   27.6   391 2308.   4086
## 4 Prionailurus rubiginosus  12  19.9  25.4   27.5   428 1467.   2998
## 5 Felis manul                8  -6     2.85  18.5   133  309.    612
```

To sort in ascending order, it's enough to do `arrange(niche, max.T)`.

You, fine ecologists, may be unhappy of the unequal sample size in our data. How would the results change if we were using only one randomly selected observations for each species in each country?

Easy-peasy!

All we need to do is to group our data by both species AND country, and then we can take advantage of a bunch of useful commands which subset our dataset by selecting only a set of rows. Check `slice()`, `sample_n()`, `sample_frac()`.

```
envdata %>%
  group_by(species, country) %>%
  sample_n(1, replace=F) %>%
  head()
```

```
## # A tibble: 6 x 6
## # Groups:   species, country [6]
##   species          country        Lon    Lat  Temp     P
##   <chr>            <chr>        <dbl>  <dbl> <dbl> <dbl>
## 1 Caracal caracal Benin         1.92   11.4   28.7   923
## 2 Caracal caracal Burkina Faso  1.28   11.6   28.8   907
## 3 Caracal caracal Chad         19.8    10.8   26.9   874
## 4 Caracal caracal Eswatini     31.2   -26.4   16.9  1151
## 5 Caracal caracal Kenya        40.1    -1.73  27.1   501
## 6 Caracal caracal Namibia      15.9   -19.4   22.6   386
```

`sample_n()` randomly samples n rows for each group. `slice(1:n)` can be used to subset the dataset by row numbers

Note that many `base` functions useful for checking the shape of your dataset work also inside pipes. Try for instance to add the commands `nrow()`, `ncol()`, `dim()`, `class()`, `str()`, `head()`, `summary()` at the end of your pipe .

Often, you have the need to delete duplicated rows in your dataset. Let's say we want to count the number of countries we have data for, separately for each cute cat. Our best friend is the command `distinct()`

```
ncountries <- envdata %>%
  group_by(species) %>%
  distinct(country) %>%
  summarize(n=n())
```

!! *watch out* !! By using `distinct` you effectively use (and retain) only the information in the column(s) you are using `distinct` on. If you want to retain the information in the other columns, try `distinct(country, .keep_all=T)`.

Transforming all your scripts to `dplyr` grammar may be challenging at first. Sometimes you want to extract one column from a dataframe and use it as a vector. You can obviously run your pipe, and then subset the output using the `$` sign, as usual. A real `dplyr` feticist would never do it, though, but rather use `pull()`. Let's say we want to extract the string of countries we have data for (altogehter, this time, not separately by species)

```
envdata %>%
  distinct(country) %>%
  pull(country)
```

```
##  [1] "Mongolia"                  "China"
##  [3] "Russian Federation"        "Afghanistan"
##  [5] "South Africa"              "Chad"
```

```
##  [7] "Benin"                          "Kenya"
##  [9] "Namibia"                        "Tanzania, United Republic of"
## [11] "Niger"                          "Burkina Faso"
## [13] "Eswatini"                       "Morocco"
## [15] "Mauritania"                     "Uzbekistan"
## [17] "Jordan"                         "Israel"
## [19] "Egypt"                          "Oman"
## [21] "India"                          "Sri Lanka"
## [23] "Brazil"                         "Mexico"
## [25] "Costa Rica"                     "Colombia"
## [27] "Ecuador"                        "Honduras"
## [29] "Panama"                         "Suriname"
```

Cool. This is it. Now you know the basics and you can start navigating on your own.

Your turn (1)

Ready to answer some cute-cat related questions based on our data?

Q1) Which cute cat has the widest range in term of precipitation?
Q2) Which countries host the highest number of different cat species?
Q3) Which species occur in these countries? (pick up one)
Q3b) BONUS POINT - Can you extract the species for all countries from Q2 within the same pipeline?

OBVIOUSLY - you can only answer these questions using the `dplyr` functions showed so far in *ONE* single pipeline. Data in, data out. You find the solutions at the end of the document.

Slighlty more advanced stuff

The power of `dplyr` is that you can really make your data management workflow more general, by writing your code more programmatically. We are only touching this argument here, but I want to introduce you to two cool families of functions.

Joins

Often, we need to join information based on two or more data.frames, based on a common field. `dplyr` has the whole family of join functions which we can use: `left_join()`, `right_join()`, `full_join()`, `inner_join()`. Here I just show one simple application.
Let's say we want to standardize the Temperature values of each individual observation, by the maximum Temperature we calculated for each cute cat species. Basically, we need to join our `envdata` object with our `niche` object, using `species` as key.

```
envdata.st <- envdata %>%
  left_join(niche %>%
              dplyr::select(species, max.T),
            by="species") %>%
  mutate(Temp.st=Temp/max.T)
head(envdata.st)
```

```
## # A tibble: 6 x 8
##   species     country             Lon   Lat  Temp     P max.T Temp.st
##   <chr>       <chr>             <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1 Felis manul Mongolia          113.   46.1   1.6   218  18.5  0.0865
## 2 Felis manul China             103.   33.8   2     612  18.5  0.108
## 3 Felis manul China              99.6  35.9  -4.1   423  18.5 -0.222
```

```
## 4 Felis manul Russian Federation  89.4  49.9  -6     138  18.5 -0.324
## 5 Felis manul Russian Federation  88.3  50.2  -2.2   133  18.5 -0.119
## 6 Felis manul Russian Federation 115.   50.1   0.9   298  18.5  0.0486
```

Two things here. First the `by` argument from `left_join` do want you to quote the key column(s). Second, notice how we nested a pipeline within a pipeline!

Not everybody knows, you can join an object with itself using the point `.` sign. In `dplyr` grammar the point `.` sign symbolizes 'itself' and its use is widespread. In the next example we first calculate the maximum temperature, species by species, and then join these summarized values to the `envdata` object itself.

```
envdata.st <- envdata %>%
  left_join(x=.,
            y={.} %>%
              group_by(species) %>%
              summarize(max.T=max(Temp, na.rm=T)),
            by="species") %>%
  mutate(Temp.st=Temp/max.T)
head(envdata.st)
```

```
## # A tibble: 6 x 8
##   species     country              Lon   Lat  Temp     P max.T Temp.st
##   <chr>       <chr>              <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
## 1 Felis manul Mongolia           113.   46.1   1.6   218  18.5  0.0865
## 2 Felis manul China              103.   33.8   2     612  18.5  0.108
## 3 Felis manul China               99.6  35.9  -4.1   423  18.5 -0.222
## 4 Felis manul Russian Federation  89.4  49.9  -6     138  18.5 -0.324
## 5 Felis manul Russian Federation  88.3  50.2  -2.2   133  18.5 -0.119
## 6 Felis manul Russian Federation 115.   50.1   0.9   298  18.5  0.0486
```

Not convinced by the power? Let's see how we would do the same in `base`

```
#calculate max for each species
Tmax.sp <- tapply(envdata$Temp, envdata$species, "max", na.rm=T)
Tmax.sp <- data.frame(species=names(Tmax.sp), max.T=Tmax.sp)
index1 <- match(envdata$species, Tmax.sp$species)
envdata2 <- data.frame(envdata, max.T=Tmax.sp$max.T[index1])
envdata2$Temp.st <- envdata2$Temp/envdata2$max.T
head(envdata2)
```

```
##       species            country       Lon      Lat Temp   P max.T     Temp.st
## 1 Felis manul           Mongolia 112.89788 46.14518  1.6 218  18.5  0.08648649
## 2 Felis manul              China 102.89451 33.83966  2.0 612  18.5  0.10810811
## 3 Felis manul              China  99.60770 35.90814 -4.1 423  18.5 -0.22162162
## 4 Felis manul Russian Federation  89.43221 49.85573 -6.0 138  18.5 -0.32432432
## 5 Felis manul Russian Federation  88.28557 50.16552 -2.2 133  18.5 -0.11891892
## 6 Felis manul Russian Federation 115.43511 50.12729  0.9 298  18.5  0.04864865
```

Three versions for the same task. Which one is easier to read?

`_at` & `_all` family

All basic functions in `dplyr` have an `x_at()` and an `x_all()` version. This helps **enormously** to script in a more general and programmatical fashion.
Remember when we summarized temperature and precipitation to get min, mean and max? Actually, our code had some redundancy that a `dplyr` would never accept, since we needed to specify the same functions for each of the variables we wanted to calculate those functions on. Imagine we have to calculate summaries for all the 18 bioclim variables in Chelsa. Should we repeat the functions we want 18 times?

NO!!
We can simply use the function `summarize_at()`. The grammar is slighlty more complicated, but discloses a world of opportunities.

```
niche <- envdata %>%
  group_by(species) %>%
  summarize_at(.vars=vars(Temp, P),
               .funs=list(min=~min(., na.rm=T),
                          mean=~mean(., na.rm=T),
                          max=~max(., na.rm=T)))
niche
```

```
## # A tibble: 5 x 7
##   species                 Temp_min P_min Temp_mean P_mean Temp_max P_max
##   <chr>                      <dbl> <dbl>     <dbl>  <dbl>    <dbl> <dbl>
## 1 Caracal caracal             11.8   190      22.4   710.      29.2  1371
## 2 Felis manul                 -6     133       2.85  309.      18.5   612
## 3 Felis margarita             15.9     1      23.9    79.4     30.4   298
## 4 Leopardus wiedii             9.2   391      21.8  2308.      27.6  4086
## 5 Prionailurus rubiginosus    19.9   428      25.4  1467.      27.5  2998
```

Awesome! Not only we summarized over both variables, but we also got columns having the correct naming. HOW COOL IS THAT?
Spend a minute looking at the syntax of `summarize_at()`. Not exactly simple, with those point `.` and tilde `~` signs. Definitely worth learning, though. Basically, these functions want two basic arguments.
- `.vars` specifies the variables we want to use. Note how we are telling `dplyr` that all elements inside the `var()` function are actually columns.
- `.funs` specifies which functions we want to apply to the selected variables. These functions are listed in a `list`, whose elements are then specified as `[name_suffix]=~function(.)`. In this case the `~` sign is telling `dplyr` this is a formula and the `.` sign symbolizes any of the variables listed in `.vars`.

If it's not yet enough, let's look at another application of the `_at` family.
Let's say we are doing an analysis on the summarized data, and decide to log transform all the summarized variables referring to Precipitation in `niche`.

```
niche.log <- niche %>%
  mutate_at(.vars=vars(starts_with("P_")),
            .funs=list(~log(.)))
niche.log
```

```
## # A tibble: 5 x 7
##   species                 Temp_min P_min Temp_mean P_mean Temp_max P_max
##   <chr>                      <dbl> <dbl>     <dbl>  <dbl>    <dbl> <dbl>
## 1 Caracal caracal             11.8  5.25      22.4   6.57      29.2  7.22
## 2 Felis manul                 -6    4.89       2.85  5.73      18.5  6.42
## 3 Felis margarita             15.9  0         23.9   4.37      30.4  5.70
## 4 Leopardus wiedii             9.2  5.97      21.8   7.74      27.6  8.32
## 5 Prionailurus rubiginosus    19.9  6.06      25.4   7.29      27.5  8.01
```

Notice that, not defining new names in the `.funs` argument, we are now overwriting the variables that we selected in the `.vars` argument. The function `starts_with()` (and its companions `ends_with()`, `contains()`), are cool helper functions specifically thought for selecting columns (which means they can also be used when using the `select()` function).

Your turn (2)

Q4) Attach to the `envdata` object both the short names and picture urls from the object `cute`. Find the cat with the second highest number of occurrences, use the corresponding url to draw its picture, and the short name for the label (see code below for plotting)
Q5) How can you use the function `rename_all` to add the prefix "cute" to all column names in `envdata`?

Q4-hint - Check what happens when doing `left_join` first and then `summarize`, or the other way around. Is the result the same?
Q5-hint - Take a look at the `summarize_at` example above. If you rename *all* fields, will you need to specify the argument `.vars`?

```
ggdraw() + draw_image("myurl") + draw_figure_label("mylabel")
```

Enough for today. We only scratched the surface here, but I'm sure you saw the huge potential for writing quicker and cleaner code compared to `base R`. Now you know the power of `dplyr`. Will you be able to harness it?

Solutions

```
#Question 1
niche %>%
  mutate(range.P=P_max-P_min) %>%
  arrange(desc(range.P)) %>%
  slice(1) %>%
  pull(species)
```

```
## [1] "Leopardus wiedii"
```

```
#Question 2
topcountries <- dat_clean %>%
  group_by(country) %>%
  distinct(species) %>%
  summarize(n=n()) %>%
  arrange(desc(n)) %>%
  filter(n>1) %>%
  pull(country)
topcountries
```

```
## [1] "Chad"  "Niger"
```

```
#Question3
dat_clean %>%
  filter(country == topcountries[1]) %>%
  distinct(species) %>%
  pull(species)
```

```
## [1] "Caracal caracal" "Felis margarita"
```

```
#Questio3b
dat_clean %>%
  filter(country %in% topcountries) %>%
  distinct(country, species) %>%
  arrange(country)
```

```
## # A tibble: 4 x 2
##   country species
##   <chr>   <chr>
## 1 Chad    Caracal caracal
## 2 Chad    Felis margarita
## 3 Niger   Caracal caracal
## 4 Niger   Felis margarita
```

```
#Question 4
second.most.common <- envdata %>%
  group_by(species) %>%
  summarize(n=n()) %>%
  left_join(cute, by="species") %>%
  mutate(url=as.character(url)) %>%
  arrange(desc(n)) %>%
  ungroup() %>%
  slice(2)
```

```
## Warning: Column `species` joining character vector and factor, coercing into
## character vector
```

```
ggdraw() +
  draw_image(second.most.common %>% pull(url), scale=0.5) +
  draw_figure_label(second.most.common %>% pull(short))
```


```
#Question 5
envdata %>%
  rename_all(.funs=~paste("cute", ., sep="_"))
```

```
## # A tibble: 217 x 6
##    cute_species    cute_country       cute_Lon cute_Lat cute_Temp cute_P
##    <chr>           <chr>                 <dbl>    <dbl>     <dbl>  <dbl>
##  1 Felis manul     Mongolia              113.      46.1       1.6    218
##  2 Felis manul     China                 103.      33.8       2      612
##  3 Felis manul     China                  99.6     35.9      -4.1    423
##  4 Felis manul     Russian Federation     89.4     49.9      -6      138
##  5 Felis manul     Russian Federation     88.3     50.2      -2.2    133
##  6 Felis manul     Russian Federation    115.      50.1       0.9    298
##  7 Felis manul     Afghanistan            69.2     34.5      12.1    357
##  8 Felis manul     Afghanistan            68.9     36.7      18.5    295
##  9 Caracal caracal South Africa           25.8    -33.4      19.4    460
## 10 Caracal caracal Chad                   19.8     10.8      26.9    874
## # ... with 207 more rows
```