Tutorial - Cleaning fossil data for the use in biogeography and palaeontology

1 Background

The public availability of fossils for large-scale analyses is rapidly increasing, mainly due to increased databasing efforts and data aggregators such as the paleobiology database (www.paleobiodb.org) or Neotoma (www.neotomadb.org), among others. However, data quality is an issue, in particular, for old collections or collections with uncertain taxonomy and/or bad preservation. Similar problems as known from biological collection databases (See supplementary material S2) are relevant for fossils, but in addition fossils might be dated wrongly or with very low precision. This tutorial presents a pipeline to clean fossil data from the paleobiology database (or any other) before using it in biogeographic or evolutionary analyses. We focus on identifying overly imprecisely geo-referenced and/or dated records by combining automated cleaning using *CoordinateCleaner* with cleaning based on meta-data. The proposed steps are by no means exhaustive, and keep in mind that what is "good data" depends entirely on your downstream analyses! We wrote this tutorial (and the whole *CoordinateCleaner* package) to help you to identify potential problems quicker and more reproducibly to improve data quality in large datasets. For the sake of this tutorial we will mostly remove flagged records from the dataset, however, we recommend to double check them individually.

2 Install `CoordinateCleaner`

You can install the latest stable version of CoordinateCleaner (Currently 1.0-6) from CRAN using `install.packages("CoordinateCleaner")`. Alternatively you can install the latest development version from GitHub using the devtools package. We recommend the latter, to stay up-to-date. Also, make sure to have the latset R version installed. In this tutorial, relevant R -code is shown in grey boxes, the resulting output lines are marked by ##.

```
install.packages("devtools")
library(devtools)

install_github("azizka/CoordinateCleaner")
```

3 Load required libraries

As a first step we will load the R libraries required for the tutorial. You might need to install some of them using `install.packages`.

```
library(tidyverse)
library(CoordinateCleaner)
library(countrycode)
library(paleobioDB)
```

4 Load test dataset

For this tutorial we will use a dataset of vascular plant fossils from the last 65 million years, downloaded from the paleobiology database using the plaeobioDB package. For the tutorial we'll limit the data to maximum

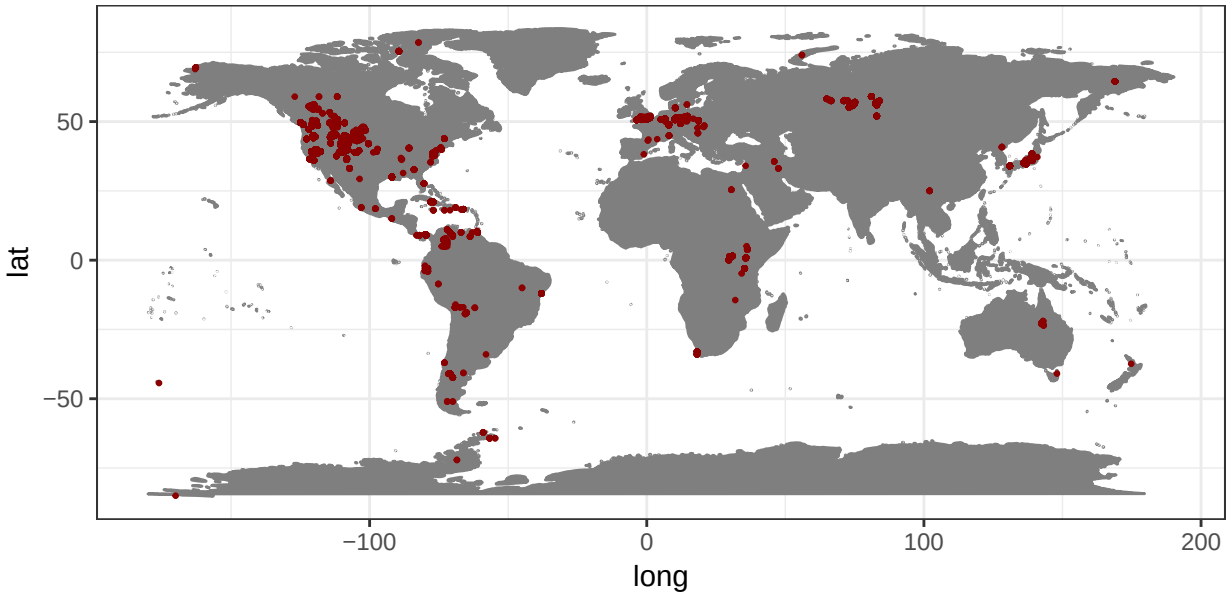
10000 records, to keep the downloading time reasonable. If you obtained your data from the web mask of the paleobiology database, or use an entirely different database, you will have to adapt the column names in the script.

```
#load data
dat <- pbdb_occurrences(base_name = "Magnoliopsida", vocab = "pbdb", limit = 10000,
                        show = c("coords", "phylo", "attr", "loc", "time", "rem"))
rownames(dat) <- NULL
```

5 Visualize the records on a map

As a first step we will visualize the records on a map, to get a general overview.

```
#plot data to get an overview
wm <- borders("world", colour="gray50", fill="gray50")
ggplot()+ coord_fixed()+ wm +
  geom_point(data = dat, aes(x = lng, y = lat),
             colour = "darkred", size = 0.5)+
  theme_bw()
```



6 CoordinateCleaner

CoordianteCleaner includes a suite of automated tests to identify problems common to biological and palaebiological databases.

6.1 Spatial issues

We'll first check coordinate validity to check if all coordinates are numeric and part of a lat/lon coordinate reference system using `cc_val`.

```
cl <- cc_val(dat, lat = "lat", lon = "lng")
```

```
## Testing coordinate validity
## Flagged 0 records.
```

Looks good, then we will test for coordinates with equal longitude and latitude. You can use the `test` argument to specify if coordinates should be flagged if their absolute values are identical (e.g. 56,-56).

```
cl <- cc_equ(cl, lat = "lat", lon = "lng")
```

```
## Testing equal lat/lon
## Flagged 0 records.
```

For the purpose of the tutorial, we will always exclude flagged records. If you want to further explore them use the `value = "flags"` argument, valid for all functions. In that case the output value will be a vector of logical values with the same length as dat, where TRUE = valid record, FALSE = flagged record. It is generally advisable to check flagged records whenever possible, to avoid data-loss and false flags.

```
fl <- cc_equ(dat, value = "flags", lat = "lat", lon = "lng")
```

```
## Testing equal lat/lon
## Flagged 0 records.
```

```
# extract and check the flagged records
fl.rec <- dat[!fl,]
head(fl.rec)
```

```
## # A tibble: 0 x 36
## # ... with 36 variables: occurrence_no <dbl>, record_type <chr>,
## #   collection_no <dbl>, taxon_name <chr>, taxon_rank <chr>,
## #   taxon_no <int>, matched_name <chr>, matched_rank <chr>,
## #   matched_no <int>, early_interval <chr>, late_interval <chr>,
## #   early_age <dbl>, late_age <dbl>, reference_no <chr>, lng <dbl>,
## #   lat <dbl>, class <chr>, class_no <int>, phylum <chr>, phylum_no <int>,
## #   cc <chr>, state <chr>, geogscale <chr>, early_age.1 <dbl>,
## #   late_age.1 <int>, cx_int_no <int>, early_int_no <int>,
## #   late_int_no <int>, genus <chr>, genus_no <int>, family <chr>,
## #   family_no <int>, order <chr>, order_no <int>, county <chr>,
## #   reid_no <int>
```

We'll also test if the records are identical, or in close vicinity to the centroids of political units. You can modify the buffer around each centroid using the `buffer` argument and the level of testing (country centroids, province centroids, or both) using the `test argument`. In case you have a list of geographic coordinates you consider problematic, for instance a list of cities you can provide them as custom gazetteer using the `ref` argument.

```
fl <- cc_cen(cl, lat = "lat", lon = "lng", value = "flags")
## Testing country centroids
## Flagged 213 records.
fl.rec <- cl[!fl,]
unique(fl.rec$cc)
## [1] "UK" "JP" "UG" "US" "MX" "CZ" "PA" "TT" "CU"
cl <- cl[fl,]
```

Next we will test if the coordinates are within the country they are assigned to. This test is a bit more tricky, as it will also flag records, if the country name in the country column is not following ISO3 or if the records have been assigned during a different political landscape. For instance records from former Western and Eastern Germany. Here we need to convert the country annotation in column cc from ISO2 to ISO3; it is advisable to double check which records have be flagged, to avoid unnecessary data loss (see above).

```
#adapt country code to ISO3, for country test
cs.ma <- "GBR"
names(cs.ma) <- "UK"
cl$cc.iso3 <- countrycode(cl$cc, origin = "iso2c", destination = "iso3c", custom_match = cs.ma)

cl <- cc_coun(cl, lat = "lat", lon = "lng", iso3 = "cc.iso3")
```

```
## Testing country identity
## Flagged 22 records.
```

Next we will test if any of the records bear the coordinates of a hosting biodiversity institution or the GBIF headquarters, using the `institutions` database of *CoordinateCleaner*. As for the country centroid test you can change the buffer around the institutions with the `buffer` argument.

```
cl <- cc_inst(cl, lat = "lat", lon = "lng")
```

```
## Testing biodiversity institutions
## Flagged 0 records
```

```
cl <- cc_gbif(cl, lat = "lat", lon = "lng")
```

```
## Testing GBIF headquarters
## Flagged 0 records.
```

Finally, we will test for plain zero coordinates (e.g. 0/0).

```
cl <- cc_zero(cl, lat = "lat", lon = "lng")
```

```
## Testing zero coordinates
## Flagged 2 records.
```

6.2 Temporal issues

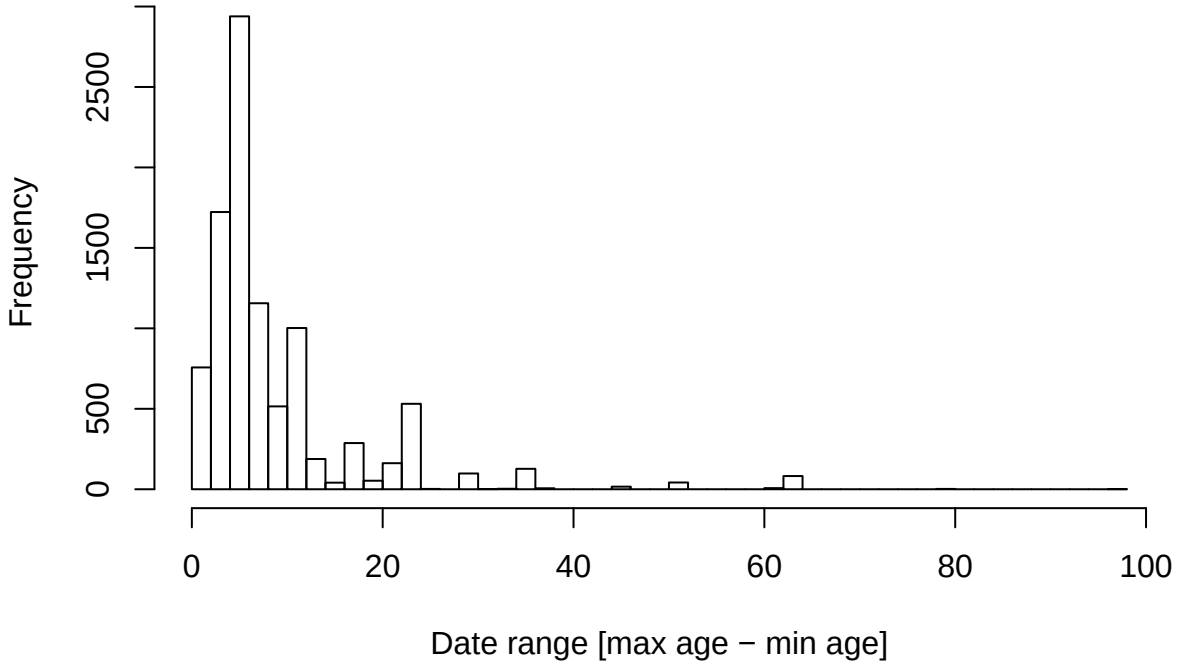
The spatial cleaning above is mostly identical with steps from recent geographic records. Additionally *CoordinateCleaner* includes three functions to test the temporal dimension of fossils. Fossil ages are usually defined with a maximum and a minimum range, based on geological strata. First we will exclude records without dating information (NA) and then test for records with equal minimum and maximum range. Unless your data includes absolutely dated fossils, this will most likely be an data entry error.

```
cl <- cl[!is.na(cl$late_age),]
cl <- cl[!is.na(cl$early_age),]
cl <- tc_equal(cl, min.age = "late_age", max.age = "early_age")
```

```
## Testing age validity
## Flagged 0 records.
```

Next we will look at the age range (= max age - min age) of each record. The age range is the dating precision and can vary considerably, depending on the data available for dating. For many analyses, for instance in PyRate, very imprecisely dated records are not suitable. Lets first have a look at the age ranges in our test dataset.

```
rang <- cl$early_age - cl$late_age
hist(rang, breaks = 40, xlab = "Date range [max age - min age]", main = "")
```



Some individual records are dated with a precision of more than 60 milion years! *CoordinateCleaner* offers two ways to flag records based on their age range (1) based on absolute age, e.g. age range > 35 million years or (2) based on age range outlier detection in the entire dataset (e.g. if few records are much less precisely dated than the rest of all records) and (3) based on age range outlier detection on taxon level (e.g. all *Quercus* records that are much less precisely dated than the other *Quercus* records. The second and third approach can be combined and offer some more flexibility over the absolute age limit, but need some consideration on the desired sensitivity. Here, we will run all three variants for illustration, if you use your own data you should decide which one is more suitable depending on your downstream analyses. In the case of (2) and (3) you can tweak the test sensitivity using the `mltpl` argument.

```
# Outlier dataset
cl <- tc_range(cl, taxon = "", min.age = "late_age", max.age = "early_age")
```

```
## Testing temporal range outliers on dataset level
## Flagged 134 records.
```

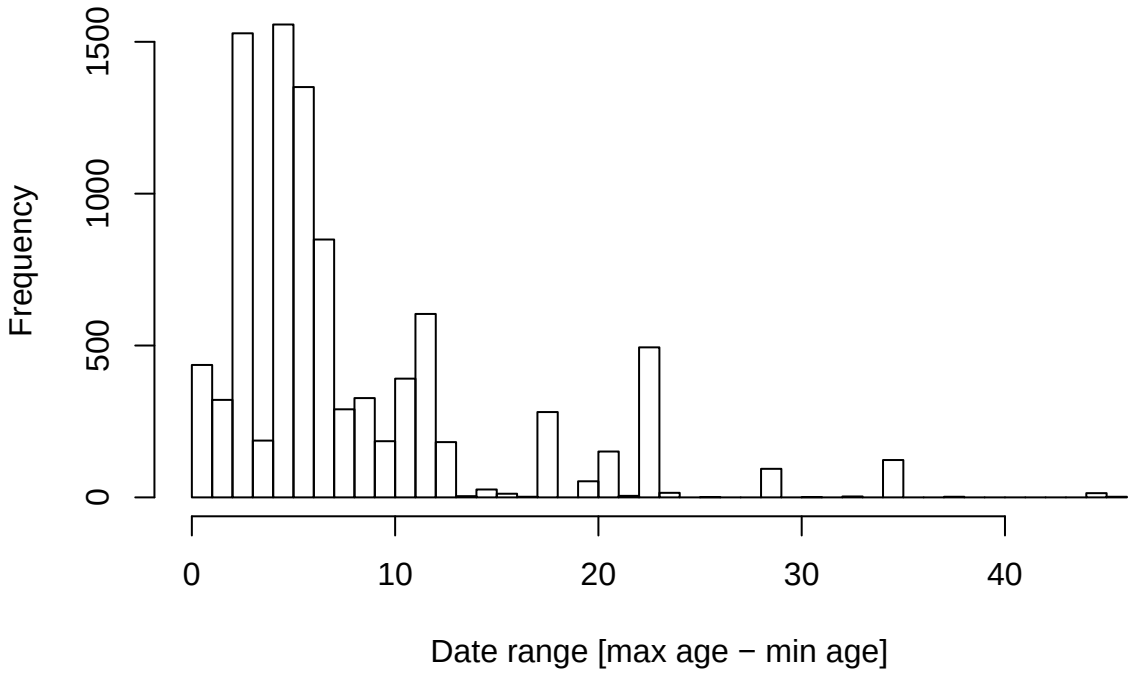
```
# Outlier per taxon
cl <- tc_range(cl, taxon = "taxon_name", min.age = "late_age", max.age = "early_age")
```

```
## Testing temporal range outliers on taxon level
## Flagged 115 records.
```

```
# Absolute age limit
cl <- tc_range(cl, taxon = "taxon_name", min.age = "late_age",
               max.age = "early_age", method = "time", max.range = 35)
```

```
## Testing temporal range outliers on taxon level
## Flagged 1 records.
```

```
rang <- cl$early_age - cl$late_age
hist(rang, breaks = 40, xlab = "Date range [max age - min age]", main = "")
```



Finally we will test for outliers in space-time, that is records that are either very distant in space or in time from all other records (1) in the dataset (2) per taxon. The test is again based on quantile outlier detection and can be modified using various arguments. Here it is important to carefully consider the desired test sensitivity. See `?tc_outl` for help.

```
# Outlier dataset
cl <- tc_outl(cl, taxon = "", lat = "lat", lon = "lng",
              min.age = "late_age", max.age = "early_age")
```

```
## Testing spatio-temporal outliers on dataset level
## Flagged 8 records.
```

```
# Outlier taxon
cl <- tc_outl(cl, taxon = "taxon_name", lat = "lat", lon = "lng",
              min.age = "late_age", max.age = "early_age")
```

```
## Testing spatio-temporal outliers on taxon level
## Flagged 58 records.
```

Done! To check how many records have been flagged in total, you can compare the two datasets.

```
nrow(dat) - nrow(cl)
```

```
## [1] 575
```

All test combined have removed about 575 records (5.8%). If you want to identify all flagged records, to

double check or correct them, take a look at the `CleanCoordinatesFOS` wrapper function below.

So far so good, we have significantly refined the data for our needs. In section 6 we will have a look at the meta-data for further refinement, but before, note that there are two different ways to run *CoordinateCleaner*. You can connect all functions directly in a row using the magrittr pipe (%>%) operator.

```
cl <- dat%>%
  cc_val(lat = "lat", lon = "lng")%>%
  cc_equ(lat = "lat", lon = "lng")%>%
  cc_cen(lat = "lat", lon = "lng")%>%
  cc_coun(lat = "lat", lon = "lng", iso3 = "cc")%>%
  cc_gbif(lat = "lat", lon = "lng")%>%
  cc_inst(lat = "lat", lon = "lng")%>%
  cc_zero(lat = "lat", lon = "lng")%>%
  tc_equal(min.age = "late_age", max.age = "early_age")%>%
  tc_range(taxon = "taxon_name",
           min.age = "late_age", max.age = "early_age")%>%
  tc_outl(taxon = "taxon_name",
          lat = "lat", lon = "lng",
          min.age = "late_age", max.age = "early_age")
```

Alternatively you can use `CleanCoordinatesFOS`, a wrapper around all quality tests provided by *CoordinateCleaner* relevant for fossil data. See `?CleanCoordiantesFOS` for help.

```
#adapt country code to ISO3, for country test
cs.ma <- "GBR"
names(cs.ma) <- "UK"
dat$cc <- countrycode(dat$cc, origin = "iso2c", destination = "iso3c", custom_match = cs.ma)

#run automated testing
flags <- CleanCoordinatesFOS(x = dat,
                             taxon = "taxon_name",
                             min.age = "late_age", max.age = "early_age",
                             value = "spatialvalid")

head(flags)
cl <- dat[flags$summary,] #the cleaned records
fl.rec <- dat[!flags$summary,] # the flagged records for verification
```

7 Improving data quality using meta-data

Usually, at least some type of meta-data are provided with fossil occurrences, as is the case in the paleobiology database. We'll now explore these and see if we can identify further problems.

7.1 Basic taxonomy

First we'll take a short look at taxonomy. Fossil taxonomy is very complex and composite databases often have taxonomic issues that are extremely difficult to resolve. Here we will only do some very basic checks to test if: 1. all taxa in our dataset are plants, 2. they are at least identified to genus level.

```
#1. This looks OK
table(cl$phylum)
```

```
##
```

```
## Spermatophyta
##          9425
```

```
#2. Taxonomic level of identification
table(cl$taxon_rank)
```

```
##
##     class    family     genus     order   species  subclass subfamily
##       727       413      1473        66      6700        16         2
```

The required taxonomic level of course depends on the downstream analyses, but here we will exclude everything other than genus or species, which is a reasonable approach for most PyRate analyses.

```
cl <- cl %>%
  filter(taxon_rank %in% c("species", "genus"))
```

7.2 Spatial coordinates

The Paleobiology database includes some information on the basis of the geographic data for many records.

```
table(cl$geogscale)
```

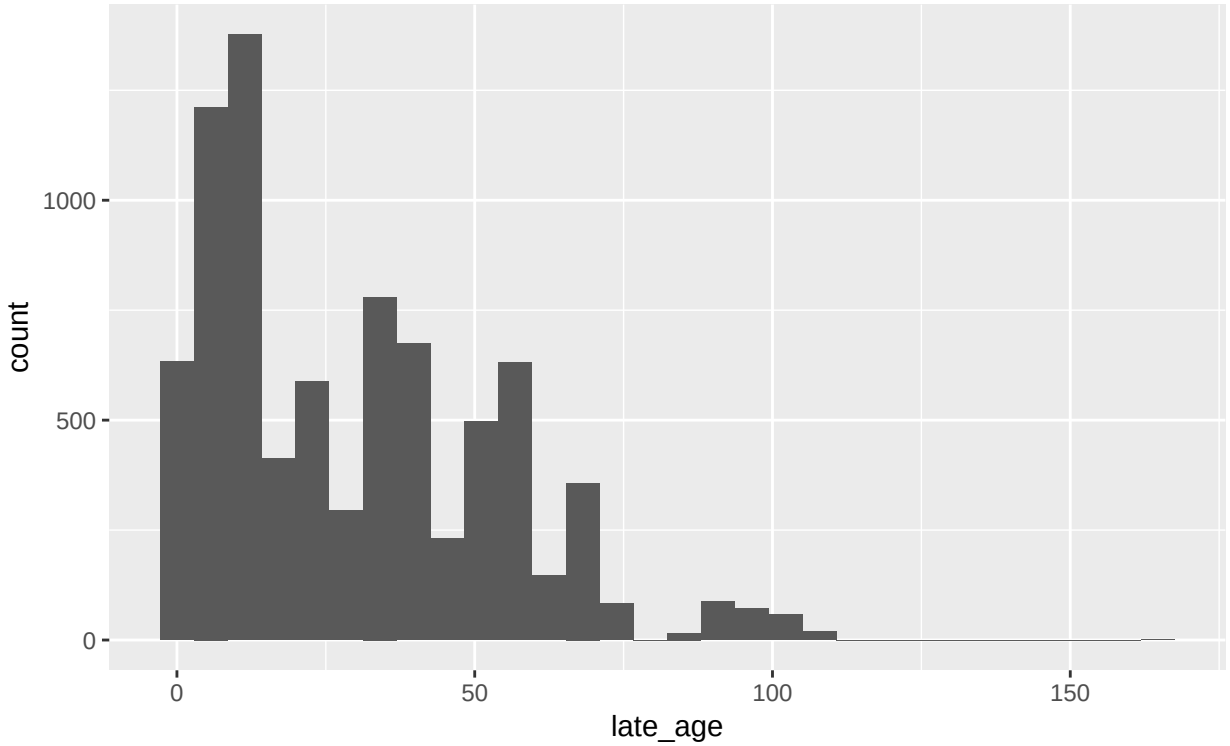
```
##
##            basin       local area          outcrop small collection
##               32              289             3416             3544
```

As expected most records are only roughly geo-referenced, but the precision is still relatively high for many records.

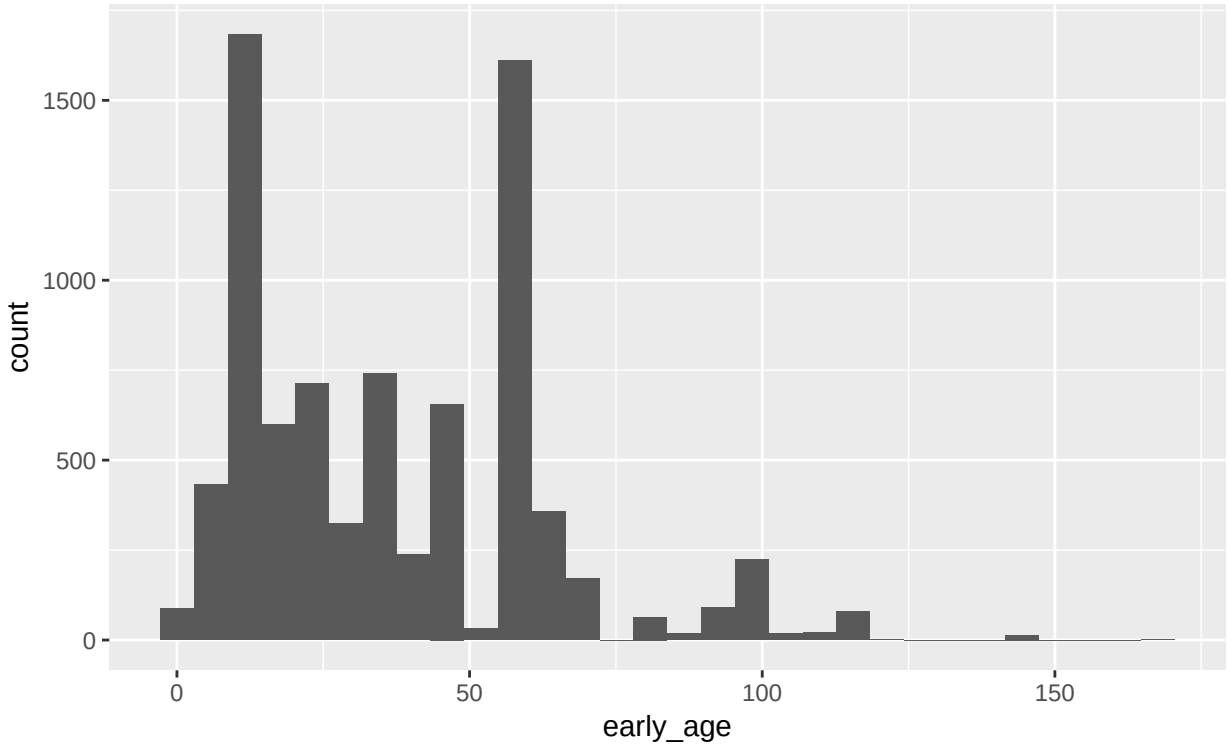
7.3 Time

We have checked for potentially problematic records in time and space above, but it is definitively advisable to check again.

```
#minimum ages
tail(table(cl$late_age))
##
##  93.9  99.6 100.5 105.3   109 164.7
##    72    17    42     6    14     1

ggplot(cl)+
  geom_histogram(aes(x = late_age))
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#maximum ages
tail(table(cl$early_age))
##
##    109 112.03    113 122.46    145  167.7
##      3     18     78      2     14      1

ggplot(cl)+
  geom_histogram(aes(x = early_age))
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The minimum and maximum ages look unproblematic, but there are still some records with very large temporal uncertainties, and at least one case where the minimum and maximum age seem reversed. This might be informative in some cases, but for most analysis this might be problematic, so here we exclude all records with temporal uncertainty above NA million years, which will retain 95% of the data. This is an arbitrary choice, and you'll have to choose a more suitable value based on your planned analyses.

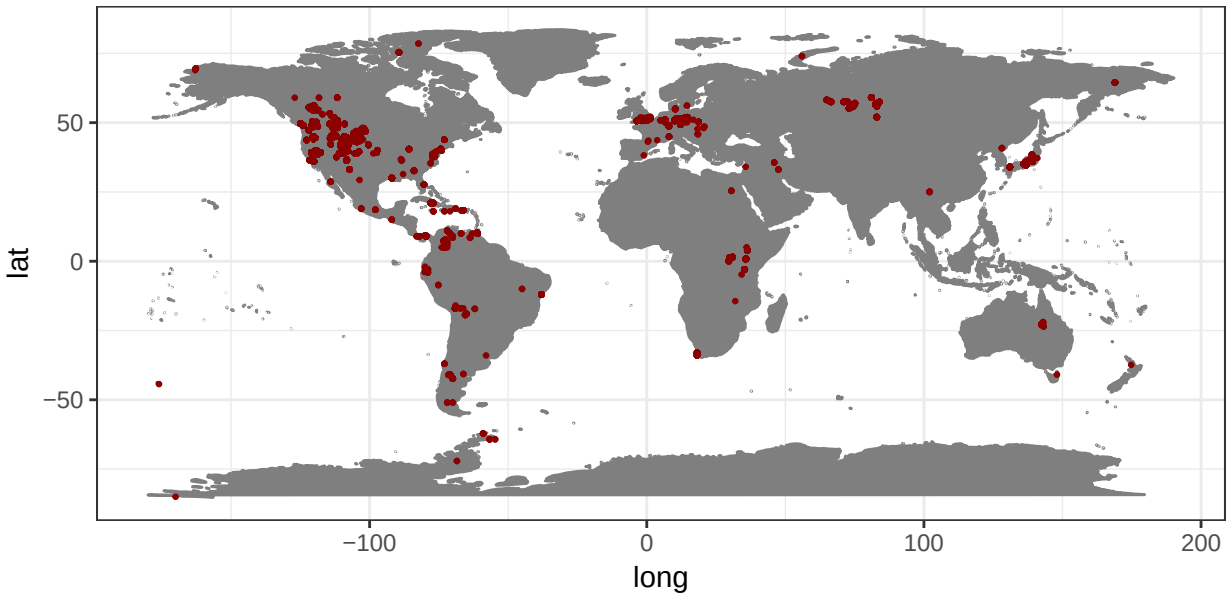
7.4 Additional meta-data

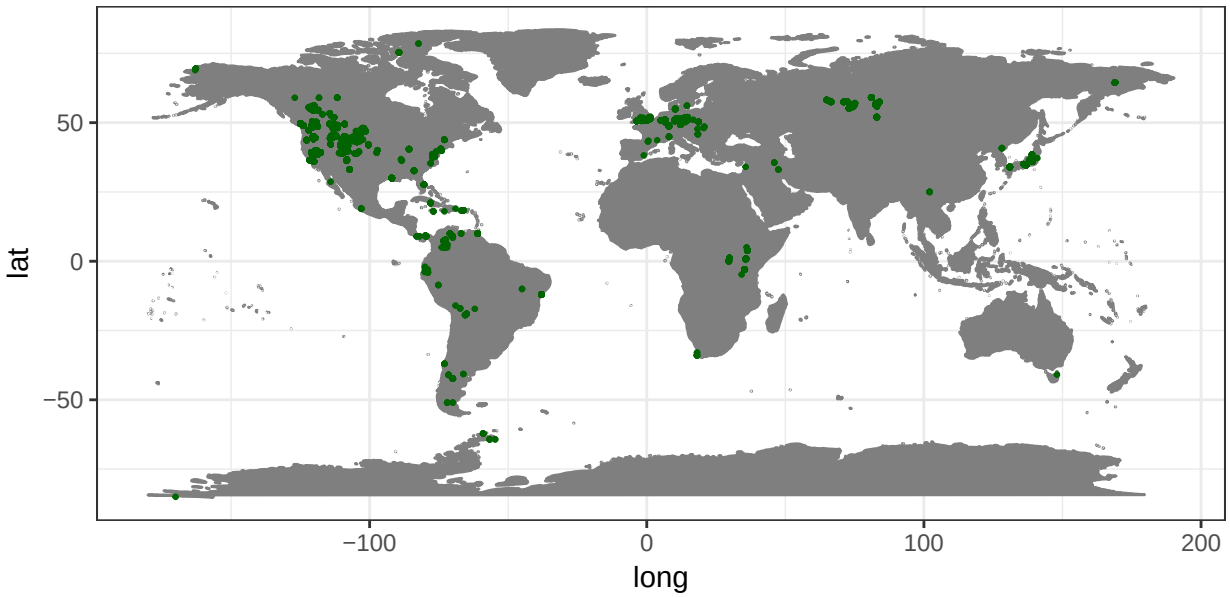
If you download data using the paleobiology database web portal, there are some additional fields you can check to generally gauge data precision, namely basis and precision of the geo-reference, the year of publications (old publications are more likely to be imprecise), preservation quality (bad preservation increases taxonomic uncertainty) and plant organ (also taxonomic uncertainty) and collection type. You might need to adapt the column names, and not all columns will be relevant for all taxa (e.g. "plant organ").

```
table(cl$latlng_basis)
table(cl$latlng_precision)


table(cl$ref_pubyr)
table(cl$collection_type)
table(cl$preservation_quality)
table(cl$plant_organ)

cl <- filter(cl, preservation_quality != "very poor")
```

8 Conclusions

Through the various cleaning steps outline above, we have identified some potential major caveats and hopefully increased the quality of the dataset. We have excluded a significant fraction of all records (-80.73 %). Data quality is a delicate issue, especially for fossils from compound data bases and the usefulness of individual records will depend on your downstream analyses. We hope that you find this tutorial useful in exploring data downloaded from the Paleobiology database and to explore the quality of any fossil dataset.





9 Writing the result to disk in PyRate format

If you want to use fossil data to estimate speciation and extinction rates, their correlation with environmental factors or to do biogeography, PyRate (https://github.com/dsilvestro/PyRate) might be useful for you. You can use the `WritePyrateOut` function of *CoordinateCleaner* to write ready-to-use PyRate input files from your now cleaned dataset to disk. To to so, you additionally need an assessment of which species are extinct and which are extant today, which you can provide via the `status` argument of `WritePyRate`. If you want to specify a path were to save the file, use the `path` argument instead of `fname`.

```
# replace  blanks in taxon names
cl$taxon_name <- gsub("[[:blank:]]{1,}","_", cl$taxon_name)

#simulated current status, soley for demonstration purposes, replace with your own data
mock.status <- data.frame(taxon_name = unique(cl$taxon_name),
                          status = sample(c("extinct", "extant"),
                                          size = length(unique(cl$taxon_name)),
                                          replace = TRUE))

#add current status to fossils
cl2 <- inner_join(cl, mock.status, by = "taxon_name")

#Write PyRate input to disk
WritePyRate(cl, fname = "paleobioDB_angiosperms", status = cl2$status,
            taxon = "taxon_name", min.age = "late_age", max.age = "early_age")
```