Tutorial - Cleaning GBIF data for the use in biogeography

1 Background

Big data aggregators such as the Global Biodiversity Information Facility (GBIF, www.gbif.org) have vastly increased the public availability of species occurrence records, with GBIF alone comprising more than 800 million records across all taxonomic groups. The data provided via these sources have revolutionized scientific biogeography and are highly valuable for research. However, some issues exist concerning data quality, mostly because these data are comprised from a variety of different collection methods (museum specimens, scientific surveys, citizen science, population counts for conservation purposes and genetic barcoding among others) and different sources (museums, herbaria, collections of individual researchers, citizen science, photo apps) and digitized and edited by various people and algorithms at different points in time and space.

In this tutorial we provide a pipeline on how to clean occurrence records retrieved from GBIF (or any other database) using *CoordinateCleaner* and meta data. The tutorial includes major steps we consider necessary, but by no means is complete and we explicitly encourage you to explore your data further before use. For the tutorial we will use a data set of occurrence records of a single species (lion, *Panthera leo*) downloaded from GBIF. On this example we can gauge the quality of cleaning steps, because we already have a good idea where we expect lions to occur. Of course, usually for multi-species data sets we do not have this kind of information, and that is the whole point of the automated cleaning. You can easily follow the tutorial using your own data instead. For the tutorial we will assume a global macroecological analysis with a resolution of about 100km as downstream analyses. Remember to adjust test sensitivity, if your analyses have a coarser or finer resolution.

With this tutorial you will be able to:

1. Visualize the data and identify potential problems
2. Use *CoordinateCleaner* to automatically flag problematic records
3. Use GBIF provided meta-data to improve coordinate quality, tailored to your downstream analyses
4. Use automated cleaning algorithms of *CoordinateCleaner* to identify problematic contributing datasets

2 Identifying erroneous coordinates using *CoordinateCleaner*

The `CleanCoordinates` function is a wrapper function around all record-level tests of *CoordinateCleaner*. The idea behind these tests is to use geographic gazetteers to identify records that are most likely erroneous (or very imprecise). We based the choice of tests on common problems observed in biological collection databases (see for example (Maldonado et al., 2015)), including assignment to country centroids, sea coordinate and outliers among others. You can get an overview over the individual tests using `?CleanCoordinates` or via the package wiki: https://github.com/azizka/CoordinateCleaner/wiki. This tutorial assumes occurrence data in the format as downloaded from GBIF, for other formats you might need to adapt the column names. You might need to install some of the required packages for the tutorial using `install.packages`.

2.1 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..

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

install_github("azizka/CoordinateCleaner")
```

2.2 Set up libraries and data

You might need to confirm to install the rnaturalearth package when loading `CoordinateCleaner`

```
library(tidyverse)
library(rgbif)
library(countrycode)
library(CoordinateCleaner)

#obtain data from GBIF via rgbif
dat <- occ_search(scientificName = "Panthera leo", limit = 20000,
                  return = "data", hasCoordinate = T)

## names(dat) #a lot of columns

#select columns of interest
dat <- dat %>%
  dplyr::select(species, decimalLongitude, decimalLatitude, countryCode, individualCount,
         gbifID, family, taxonRank, coordinateUncertaintyInMeters, year,
         basisOfRecord, institutionCode, datasetName)
```

2.3 Visualize the data on a map

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

This map clearly indicates, that we need to prepare the data further, if we want them to represent the current day (or historic) distribution of lions.

2.4 Use *CoordinateCleaner* to automatically flag problematic records

2.4.1 Option A) Using the `CleanCoordinates` wrapper function

As a first step we will run the automatic cleaning algorithm of CoordinateCleaner. The `CleanCoordinates` function is a wrapper around a large set of automated cleaning steps to flag errors that are common to biological collections, including: sea coordinates, zero coordinates, coordinate - country mismatches, coordinates assigned to country and province centroids, coordinates within city areas, outlier coordinates and coordinates assigned to biodiversity institutions. You can switch on each test individually using logical flags, modify the sensitivity of most individual tests using the ".rad" arguments, and provide custom gazetteers using the ".ref" arguments. See `?CleanCoordinates` for help. To use the country - coordinate mismatch test we need to convert the country from ISO2 to ISO3 format.

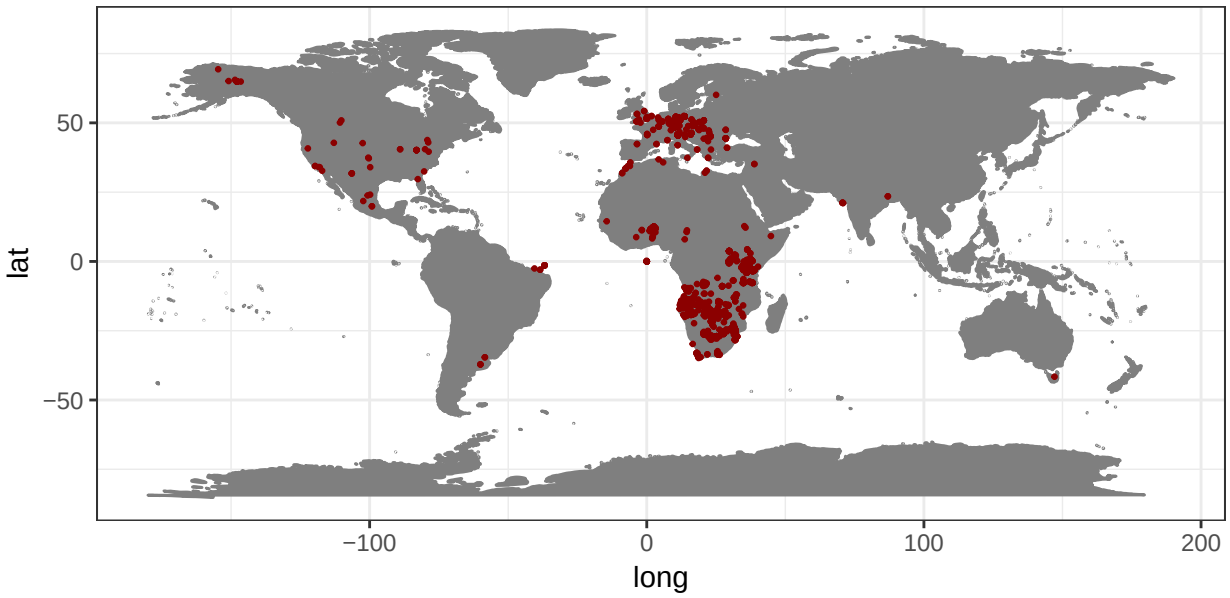


Figure 1: Occurrence records for Panthera leo obtained from GBIF.

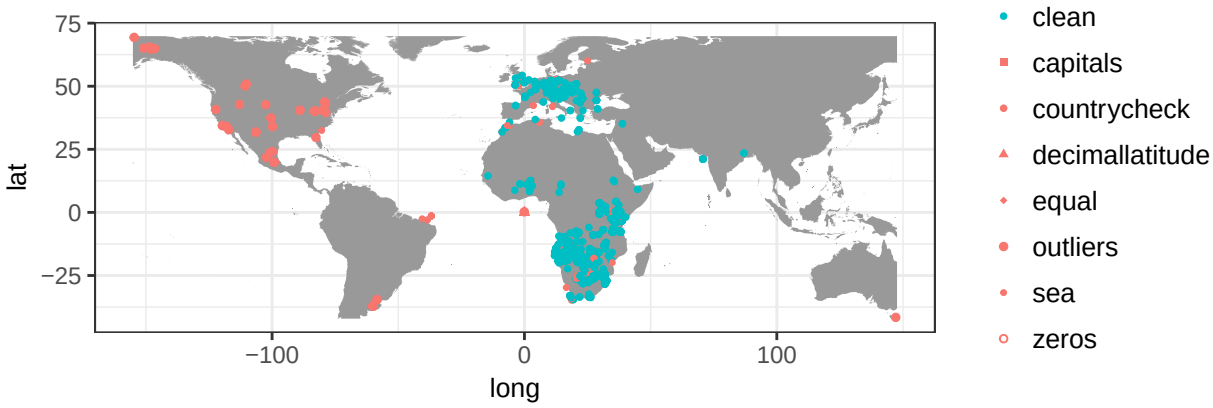


Figure 2: Records flagged by the automated cleaning.

```
#convert country code from ISO2c to ISO3c
dat$countryCode <-  countrycode(dat$countryCode, origin =  'iso2c', destination = 'iso3c')

#flag problems
dat <- data.frame(dat)
flags <- CleanCoordinates(x = dat, lon = "decimalLongitude", lat = "decimalLatitude",
                          countries = "countryCode",
                          species = "species",
                          countrycheck = T,
                          outliers = T) # most test are on by default
summary(flags)
plot(flags)
## Regions defined for each Polygons
```

```
## Testing coordinate validity
## Flagged 0 records.
## Testing equal lat/lon
## Flagged 15 records.
## Testing zero coordinates
## Flagged 14 records.
## Testing country capitals
## Flagged 5 records.
## Testing country centroids
```

```
## Flagged 12 records.
## Testing sea coordinates
## Flagged 25 records.
## Testing country identity
## Flagged 120 records.
## Testing geographic outliers
## Flagged 53 records.
## Testing GBIF headquarters
## Flagged 0 records.
## Testing biodiversity institutions
## Flagged 0 records
## Flagged 218 of 2201 records, EQ = 0.1
## decimallatitude        validity           equal           zeros
##              13               0              15              14
##        capitals       centroids             sea    countrycheck
##               5              12              25             120
##        outliers            gbif     institution         summary
##              53               0               0             218
```

The automatic test flagged 9.9% of the records. For the purpose of this tutorial we will exclude the flagged records, but in general it is recommendable to explore them further.

```
#Exclude problematic records
dat.cl <- dat[flags$summary,]

#THe flagged records
dat.fl <- dat[!flags$summary,]
```

2.4.2 Option B) Using the magrittr pipe (%>%)

Alternatively, you can run all tests implemented in *CoordinateCleaner* with a individual function and connect them using the magrittr pipe operator, which will directly result in a `data.frame` comprising only cleaned records.

```
 #to avoid specifying it in each function
names(dat)[2:3] <- c("decimallongitude", "decimallatitude")

clean <- dat%>%
  cc_val()%>%
  cc_equ()%>%
  cc_cap()%>%
  cc_cen()%>%
  cc_coun(iso3 = "countryCode")%>%
  cc_gbif()%>%
  cc_inst()%>%
  cc_sea()%>%
  cc_zero()%>%
  cc_outl()%>%
  cc_dupl()
```

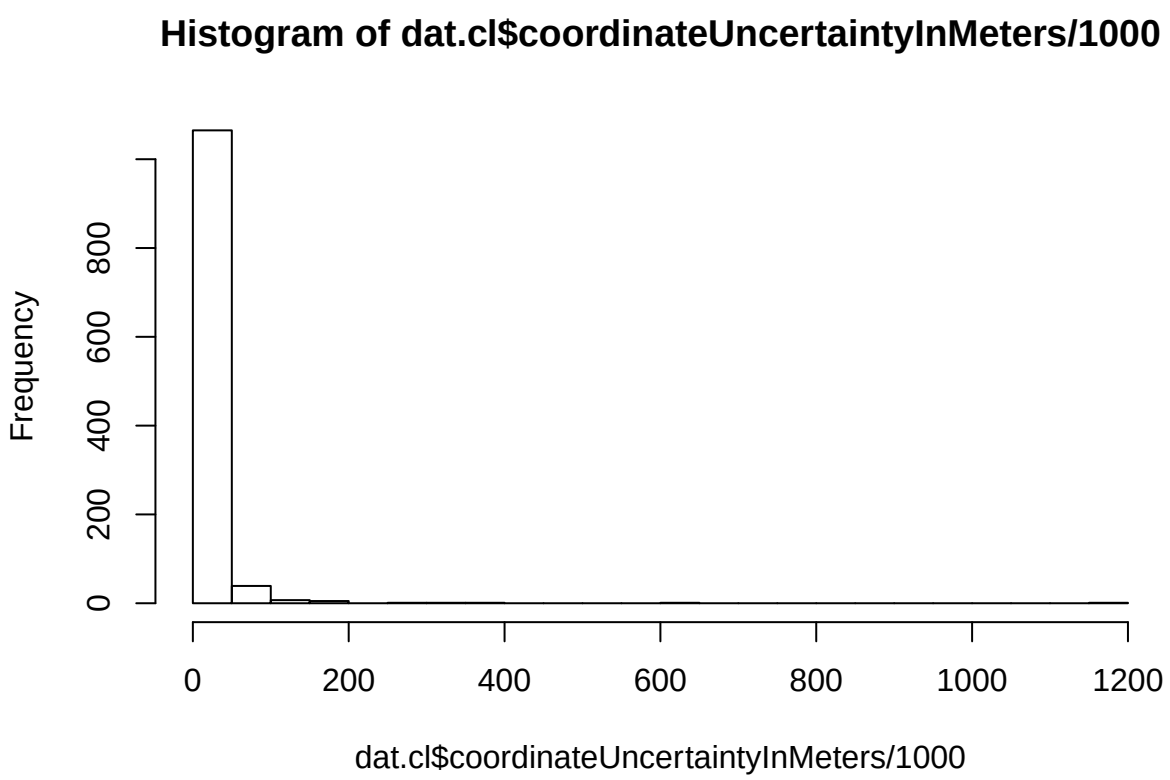


Figure 3: A histogram of the coordinate precision in the dataset..

3 Improving data quality using GBIF meta-data

That helped a lot, but unfortunately some unwanted records remain, especially within Europe (Fig. 3). This is mostly because we have used the occurrence records uncritically and ignored the meta-data. GBIF offers a whole lot of useful meta-data which we will use now to further refine quality of our dataset. First we'll remove coordinates with very low precision and from unsuitable data sources. We will remove all records with a precision below 100 km as this represent the grain size of our downstream analysis, but we recommend you to chose it based on your downstream analyses. We also exclude fossils as we are interested in recent distributions; and records from unknown sources, as we deem them not reliable enough.

```
#Remove records with low coordinate precision
hist(dat.cl$coordinateUncertaintyInMeters/1000, breaks = 20)
```

```
dat.cl <- dat.cl %>%
  filter(coordinateUncertaintyInMeters/1000 <= 100 | is.na(coordinateUncertaintyInMeters))

#Remove unsuitable data sources, especially fossils
#which are responsible for the majority of problems in this case
table(dat$basisOfRecord)
##
##     FOSSIL_SPECIMEN   HUMAN_OBSERVATION MACHINE_OBSERVATION
##                 434                1451                   3
##         OBSERVATION  PRESERVED_SPECIMEN             UNKNOWN
```

```
##                   1                 188                 124

dat.cl <- filter(dat.cl, basisOfRecord == "HUMAN_OBSERVATION" |
                         basisOfRecord == "OBSERVATION" |
                         basisOfRecord == "PRESERVED_SPECIMEN")
```

In the next step we will remove records with suspicious individual counts. GBIF includes few records of absence (individual count = 0) and suspiciously high occurrence counts, which might indicate inappropriate data or data entry problems.

```
#Individual count
table(dat.cl$individualCount)
```

```
##
##   0   1   2   3   5   6  15
##   4 103  29   3   1   1   1
```

```
dat.cl <- dat.cl%>%
  filter(individualCount > 0 | is.na(individualCount))%>%
  filter(individualCount < 99 | is.na(individualCount)) # high counts are not a problem
```

We might also want to exclude very old records, as they are more likely to be unreliable. For instance, records from before the second world war are often very imprecise, especially if they were geo-referenced based on political entities. Additionally old records might be likely from areas where species went extinct (for example due to land-use change).

```
#Age of records
table(dat.cl$year)
```

```
##
## 1896 1906 1911 1912 1920 1923 1927 1928 1929 1930 1931 1936 1941 1948 1949
##    1    1    7    7    2    1   17    8    8    4    2    2    4   26    1
## 1951 1958 1959 1964 1966 1967 1968 1969 1970 1972 1974 1980 1981 1982 1983
##    1    2    3    1    7    5    4    7    2    1    1    6    1    2   11
## 1984 1985 1989 1990 1991 1992 1994 1995 1996 1997 1998 1999 2000 2001 2002
##    5    2    7    1    1    3    2    2    7    5    9   76    2   10    5
## 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
##   23    9    5   21   19   27   19   42   50   58   56  146  147  128  241
## 2018
##   16
```

```
dat.cl <- dat.cl%>%
  filter(year > 1945) # remove records from before second world war
```

On top of the geographic cleaning, we also want to make sure to only include species level records and records from the right taxon. The latter is not a problem in this case, as we only have one species, but it can be helpful for large datasets. Taxonomic problems such as spelling mistakes in the names or synonyms can be a severe problem. We'll not treat taxonomic cleaning here, but if you need to, check out the taxize R package (https://ropensci.org/tutorials/taxize_tutorial.html) or the taxonomic name resolution service (plants only, http://tnrs.iplantcollaborative.org/).

```
table(dat.cl$family) #that looks good
##
## Felidae
##    1225
dat.cl <- dat.cl%>%
  filter(family == 'Felidae')
```

```
table(dat.cl$taxonRank) # this is also good
##
##    SPECIES SUBSPECIES
##        634        591
```

We excluded almost 50% of the initial data points with the data cleaning, and the general picture has improved considerably. We confined the records mostly to what can be considered current day distribution of the species of interest (Fig. 5).

We have, however, also lost quite a number of records. In general, there is no "one-size-fits-it-all" for data quality of geographic species occurrence records. Of course highest coordinate precision is desirable, but what is acceptable will strongly depend on the downstream analyses. For species distribution modelling, usually high precision is necessary e.g. 1-10 km, but for other analyses such as biogeographic reconstructions using tectonic plates, a record might be considered good enough quality, as long as it is on the right continent. As another example for conservation purposes it might be sufficient to know that a species is present within a certain country.

4 Improving data quality using external information

Figure 5 shows the success of automated cleaning. However, three records within Europe remain. A short inspection of the data suggests that these are a dubious human observation and five specimens, potentially assigned to their specimen location, or fossils with misclassified meta-data. One option to automatically flag these records is to rerun the outlier test on the cleaned data. However, this would most likely also flag the isolated Indian population (which is a true presence) as problematic. Another alternative is to use additional knowledge or the study outline for additional cleaning, using species defined ranges (for animals for example the IUCN range maps, http://www.iucnredlist.org/technical-documents/spatial-data or for plants the botanical countries of the World Checklist of selected plant families, http://wcsp.science.kew.org/home.do). A third alternative is to exclude records outside a certain study extent. In our example the latter is the easiest solution because we know that lions do not occur in high latitudes any more.

```
#exclude based on study area
dat.fin <- filter(dat.cl, decimalLatitude < 40)
```

5 Identifying problematic data sets

Some types of potentially problematic coordinates can cause bias, but are not identifiable on record-level if the relevant meta-data are missing. This is especially the case if the erroneous records have been combined with precise GPS-based point occurrences into datasets of mixed precision. Two important cases are: (A) coordinate conversion errors based on the misinterpretation of the degree sign as decimal delimiter and (B) data derived from rasterized data collection designs (e.g. presence in a 50x50 km grid cell). *CoordinateCleaner* implements two algorithms to identify these problems on a dataset level.

5.1 Identify dataset with ddmm to dd.dd conversion error

We will first run the test for erroneous data conversion due to the misinterpretation of the degree sign as decimal delimiter. We will use the `dc_ddmm` function, alternatively, you can use the `CleanCoordinatesDS` wrapper. See supplementary material S1 for a detailed description of the algorithm and implementation of the test. You can control the output of the function via the `value` argument.

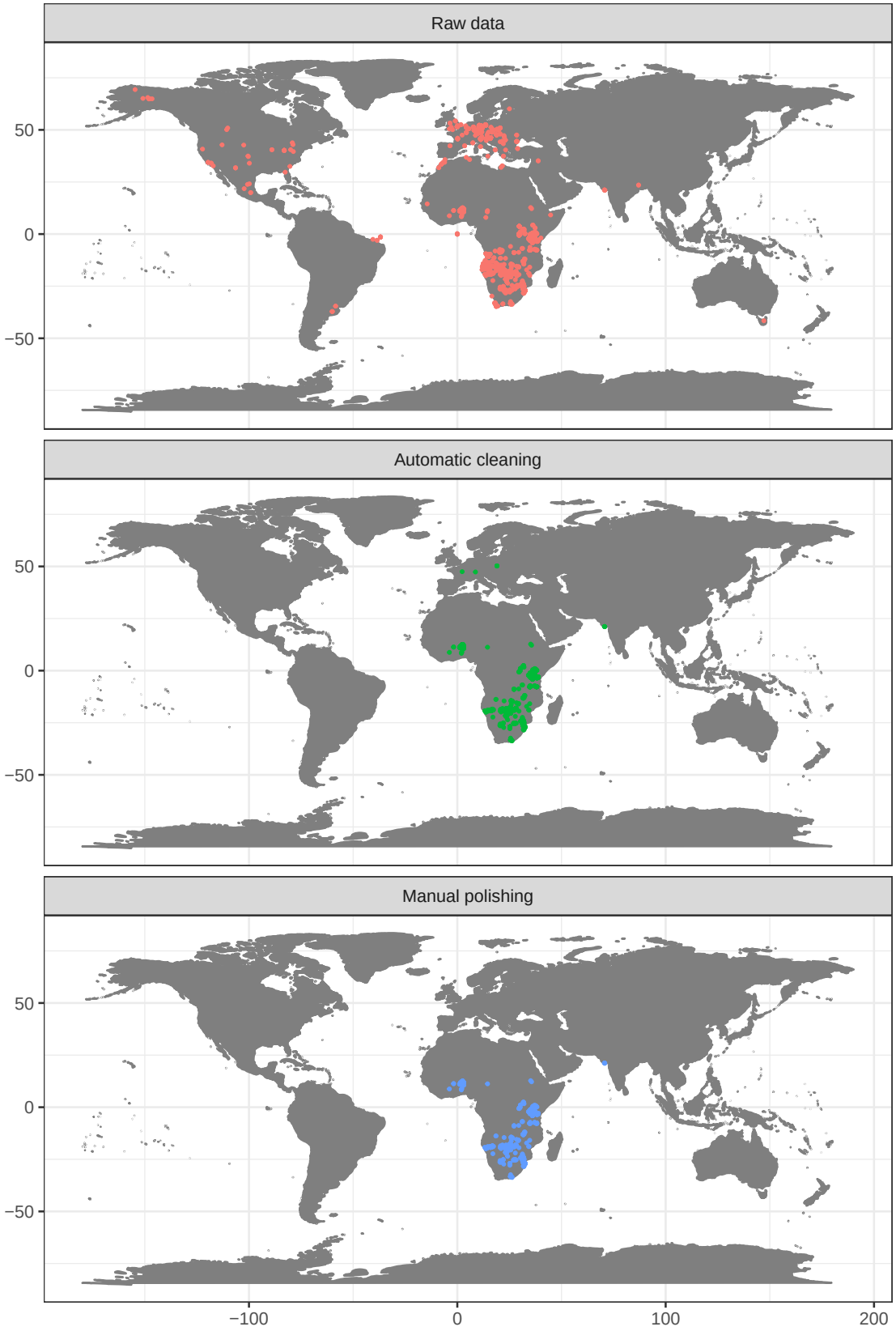
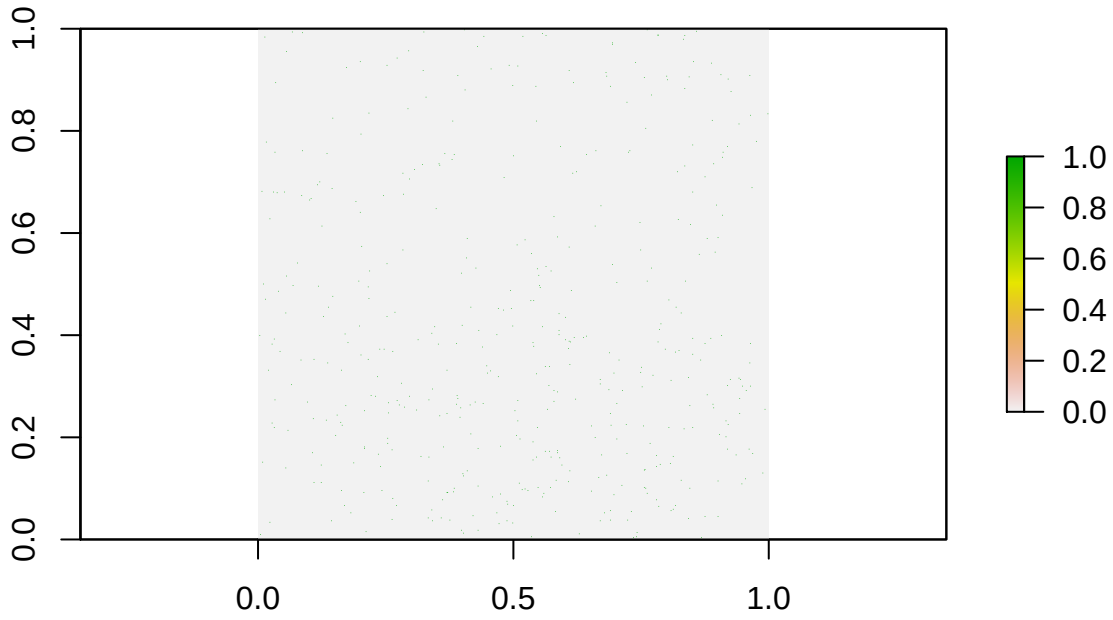


Figure 4: The dataset of occurrence of lions after different cleaning phases.

```
out.ddmm <- dc_ddmm(dat.cl, lon = "decimalLongitude", lat = "decimalLatitude",
                    ds = "species", diagnostic = T, diff = 1,
                    value = "dataset")
```

```
## Testing datasets for dd.mm to dd.dd conversion errors
```




```
## Flagged 0 records
```

This looks good. The test indicates a slightly higher fraction of records with decimals below .60 than expected at random, but this is within the expected range and thus the test indicates no bias, which is confirmed by the diagnostic plot. In the case of a strong bias, the green points would be clustered in the bottom left quarter of the plot.

5.2 Test for rasterized sampling

As a second step we will use the `dc_round` function to identify datasets with a significant proportion of coordinates that have been collected in large scale lattice designs. These records might have a low precision and might therefore be problematic for some analyses. For instance presence derived from a 1 degree grid of a national atlas might be to coarse for small scale species distribution models.

```
par(mfrow = c(2,2))
out.round <- dc_round(dat.fin, lon = "decimalLongitude",
                      lat = "decimalLatitude",
                      ds = "species",
                      value = "dataset",
                      T1 = 7,
```

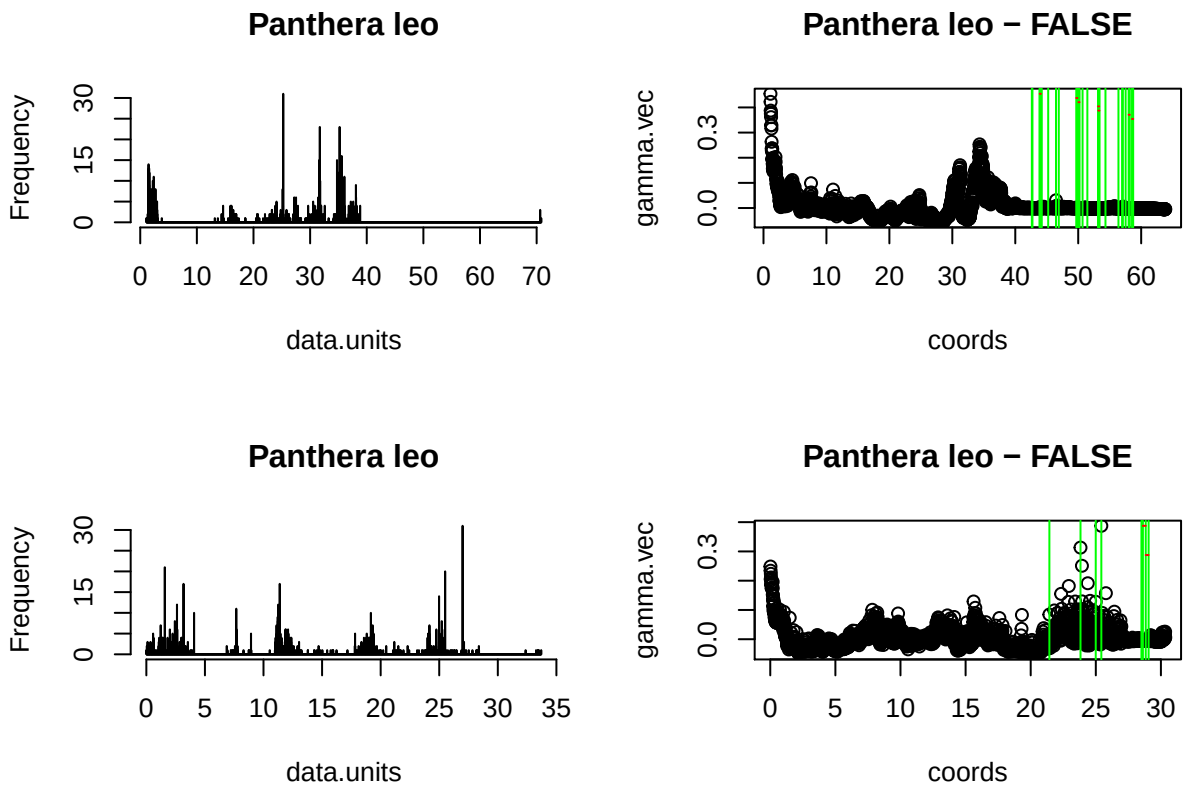


Figure 5: Diagnostic plots testing for rasterized sampling or excessive rounding. The left panel shows histograms of the record distribution, the right panel shows the autoorrelation plots. The upper panel shows longitude, the lower panel shows latitude. The logical flag in the heading of the right panel indicates the binary flag.

```
                    graphs = T)
```

These results look good. The dataset does not show rasterized collection schemes (see Supplementary material S1 for examples of biased datasets). The test has detected and flagged some small scale and low intensity periodicity in the longitude coordinates, however, the entire dataset is only flagged if both longitude and latitude show a pattern (as expected from rasterized sampling). You can modify the test sensitivity using various arguments. See `?dc_round` for more information.

The lion dataset is relatively small and consistent, at least in the way that it only comprises on species. For larger scale analyses you might need to deal with larger datasets, composed from a larger variety of sources.

6 Testing larger databses comprised from multiple datasets

6.1 Load data

First we will download a larger example database that includes multiple contributing datasets with a sufficient number of records each. Here will use records from all African mammals as example. The download takes some time, alternatively, you can download the data via www.gbif.org. Based on the assumptions of the

dataset-level tests (See supplementary material S1), we will exclude very small datasets with less than 100 individual records or less than 50 unique locations or spanning less than 2 degrees from the analyses.

```
#get taxon key
key <- name_lookup(query='mammalia')

#download data, this can take some time
dat <- occ_search(scientificName = "mammalia", limit = 200000, continent = "africa",
                  return = "data", hasCoordinate = T)

#select relevant columns
dat <- select(dat, decimalLongitude, decimalLatitude, datasetKey)%>%
  as.data.frame()

#identify datasets with < 300 records and
excl1 <- table(dat$datasetKey)
excl1 <-excl1[excl1>100]

#identify datasets with  less than 10 unique locations
excl2 <- dat[!duplicated(dat[,c("decimalLongitude", "decimalLatitude", "datasetKey")]),]
excl2 <- table(excl2$datasetKey)
excl2 <- excl2[excl2 > 50] #at more than 50 different locations


# exclude smal datasets
dat <- dat%>%
  filter(datasetKey %in% names(excl1))%>%
  filter(datasetKey %in% names(excl2))

dat <- select(dat, decimalLongitude, decimalLatitude, datasetKey)%>%
  as.data.frame()
```

6.2 Identify dataset with ddmm to dd.dd conversion error

We will first run the `dc_ddmm` test again, alternatively, you can use the `CleanCoordinatesDS` wrapper. You can control the output of the function via the `value` argument.

```
# output with a summary for each dataset
out.ddmm <- dc_ddmm(dat, lon = "decimalLongitude", lat = "decimalLatitude",
                    ds = "datasetKey", diff = 1,
                    value = "dataset")
```

```
## Testing datasets for dd.mm to dd.dd conversion errors
## Flagged 634 records
```

```
head(out.ddmm)
```

```
##                                    binomial.pvalue perc.difference pass
## 026af01f-d227-476f-9e81-d5273eca6a2b          0.4011           0.017    1
## 09baff40-d4a3-11dc-8719-b8a03c50a862          0.3744           0.018    1
## 0daed095-478a-4af6-abf5-18acb790fbb2          0.0000           0.542    1
## 19cb2741-a430-4139-8680-e11eba996978          0.0000           0.375    1
## 1d04e739-98a9-4e16-9970-8f8f3bf9e9e3          0.6526          -0.044    1
## 22a66350-7947-4a49-84a3-39c7c1b0881f          0.0680           0.219    1
```

The result suggest that 1 dataset (5%) including 634 records (0.32 %) might include a significant portion of

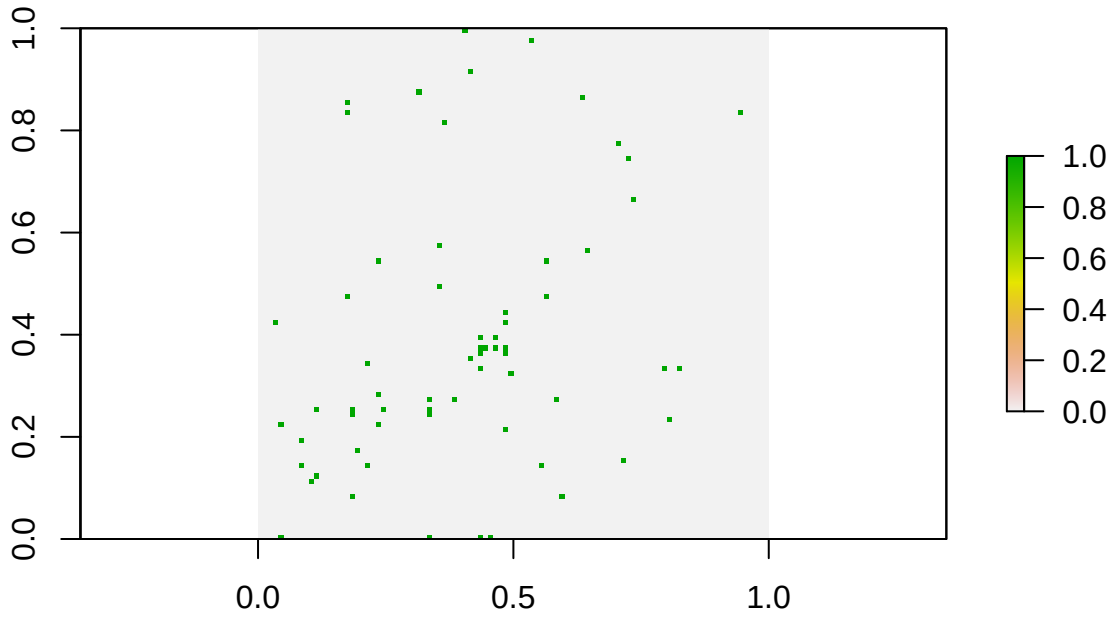
records with potentially erroneous coordinates; 5 datasets where to small to be tested

We will have a closer look at the flagged dataset and its analyses matrix by retuning the test on the dataset individually. This is a relatively small dataset (< 10000 records) thus we can decrease the size of the analyses matrix to make the plot more easily interpretable.

```
dat.sub <- dat%>%
  filter(datasetKey %in% rownames(out.ddmm[!out.ddmm$pass,]))

dc_ddmm(dat.sub, lon = "decimalLongitude", lat = "decimalLatitude",
                     ds = "datasetKey", diff = 1,
                     value = "dataset", diagnostic = T, mat.size = 100)
```

```
## Testing datasets for dd.mm to dd.dd conversion errors
```




```
## Flagged 634 records
```

```
##                                     binomial.pvalue perc.difference pass
## 96ca66b4-f762-11e1-a439-00145eb45e9a               0           1.065    0
```

There is indeed a clustering of records with decimals below 0.6! This might indicate conversion bias. To learn more we will have a look at the coordinates in the dataset.

```
table(dat.sub$decimalLongitude)
```

```
##
##   -152.55   -118.32   -106.43    -73.73    -69.17  -68.9414    -68.83
##         1         1         1        20         2        31         2
##     -68.4       -68  -67.8008    -66.17     -62.7  -62.5394    -59.65
```

```
##         3         1         5         5        15         4        15
##    -53.73    -53.63     -49.8     -8.23     -5.45 -5.433333 -5.333333
##         1         1         2         7         1        68         1
## -4.583333 -4.333333 -4.083333     -4.05 10.333333 10.433333     10.45
##        28        29        66         4         1        24        43
## 10.466667 10.483333      11.2     11.25 12.483333     16.23      20.5
##        65        86         4         2         3         1         2
## 21.083333 21.116667 25.566667 27.033333      28.1 30.216667 31.116667
##         1         2        20         1         1         4        12
## 31.183333 31.233333     31.35 31.366667 31.383333 31.566667 31.583333
##         3         3         2         6         8         2         2
## 33.183333 33.416667 48.416667 53.183333     54.35  103.7125  156.5983
##         2         4         1         5         5         1         1
##  157.0483
##         3
```

It seems the problem here is rather a high number of rounded coordinates, inflating the number of decimals below 0.6 , and not a conversion error. We'll get some more information from GBIF.

```
uid <- dat.sub%>%
  dplyr::select(datasetKey)%>%
  unlist()%>%
  unique()

lapply(uid, function(k) rgbif::datasets(uuid = k, type = "data")$data$description)
```

```
## [[1]]
## [1] "Founded in 1869, the AMNH mammal collections are among the oldest in the museum. Today, the Depa
```

The flagged dataset is a global mammal dataset form the collection of the American Museum of Natural History. The results are not fully decisive. The contribution of the dataset to our total dataset is rather small, so we could decide to drop the relevant records. In this case however, based on the analyses matrix, the coordinate distribution and the source, we suggest to keep the dataset for further analyses.

6.3 Identify datasets with rasterization bias

As a second step we will use the `dc_round` function to identify datasets with a significant proportion of coordinates that have undergone decimal rounding or that have been collected in rasterized designs. We can control the output value and exclude flagged datasets as above.

```
out.round <- dc_round(dat, lon = "decimalLongitude",
                      lat = "decimalLatitude",
                      ds = "datasetKey",
                      value = "dataset",
                      T1 = 9,
                      graphs = FALSE)

out.round$dataset <- rownames(out.round)
```

The result suggest that 13 data sets (50%) including 122195 records (62%) show a conspicuous pattern of periodicity. Thus more than half the records are flagged by the test. A look a the diagnostic plots (you can see them by rerunning the test with `graphs = TRUE`, not shown here) shows that indeed most of the datasets do not seem to contain exclusively GPS coordinates but rather either concatenated, rounded, or rasterized coordinates (the isolated peaks in the example plot). This is expected, since we downloaded all occurrences without restrictions and thus will also include for instance old records with only rough georeferencing. Remember also, that imprecise, rounded coordinates might not be a problem, depending on

the resolution of the downstream analyses. Since we downloaded all mammals for Africa, we are probably interested in a large scale analyses, an will thus not need a coordinate precision higher than 100 km. To account for this we rerun the `dc_round` test with different settings, only flagging datasets with periodicities at more than 1 degree.

```
out.round <- dc_round(dat, lon = "decimalLongitude",
                      lat = "decimalLatitude",
                      ds = "datasetKey",
                      value = "dataset",
                      T1 = 9,
                      reg.dist.min = 1,
                      graphs = FALSE)

out.round$dataset <- rownames(out.round)
out.round$summary
```

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

So at this scale none of the datasets shows periodicity. If you need a higher resolution you can easily exclude erroneous records using `value == "clean"`, and we can proceed to the downstream analyses. Note however, that a passed test does not mean that there are no rounded coordinates in the data (we know there are), it just indicates that there are now returning patterns, which would be indicative of a rasterized sampling or rounding of many coordinates over a certain geographic range.

Maldonado, C., Molina, C. I., Zizka, A., Persson, C., Taylor, C. M., Albán, J., Chilquillo, E., et al. (2015). Estimating species diveristy and distribution in the era of Big Data: To what extent can we trust public databases. *Glob Ecol Biogeogr*, *24*(8), 973–984.