Bacdiving

Release 1.1.6

Mahima Arunkumar

Nov 30, 2022

CONTENTS:

Bacdiving is a Python package which can access and retrieve information from the world's largest database for standardized bacterial phenotypic information: BacDive. Additionally, Bacdiving provides access statistics and several options to visualize the retrieved this information.

Check out the *Installation* section on how to install this package as well as the *Usage* and *Tutorial* section for further information on how to use the package.

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

1.1 Contents

1.1.1 Installation

To use Bacdiving, please first install the latest version of this package using pip:

```
(.venv) $ pip install bacdiving
```

Next, you will need to register (for free) for the BacDive web services. This is needed, due to the fact that Bacdiving uses the Python API client to access specific information stored on BacDive.

1.1.2 Usage

About Bacdiving

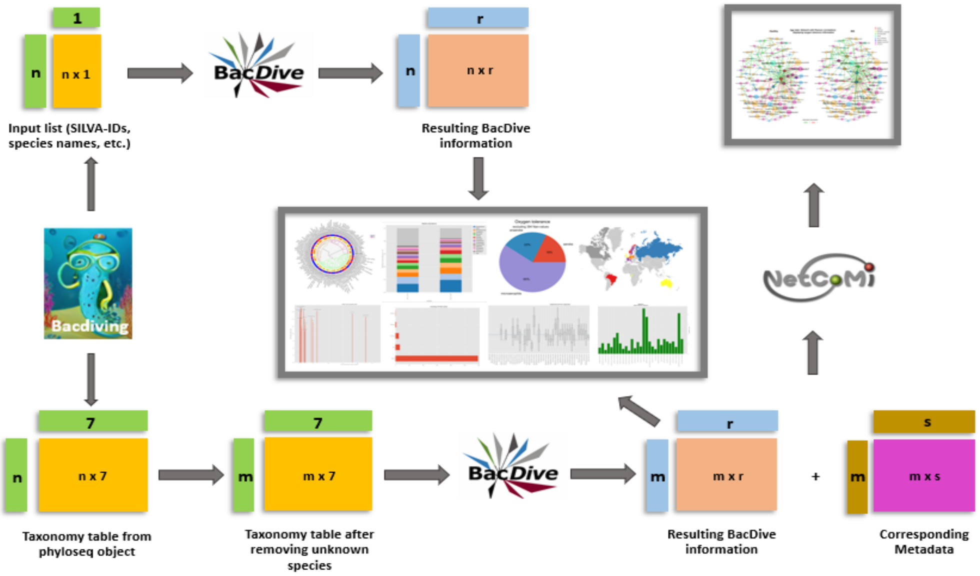
Bacdiving is a Python package which can access and retrieve information from the world's largest database for standardized bacterial phenotypic information: BacDive. Additionally, Bacdiving provides access statistics and options to visualize this information.

The following figure gives an overview of Bacdiving and how this package could be used:

As depicted in this workflow, Bacdiving can deal with two types of inputs: either a taxonomy table (e.g. resulting from a phyloseq-object) or a file input.

Starting with the taxonomy table input type (in n x 7 format) with 7 taxonomic ranks, Bacdiving first filters out all rows for which the species is unknown. This results in a "new" taxonomy table (in m x 7 format). Each one of these m species will then be checked if they can be found on BacDive or not. If information is available for a given species, then BacDive data for all of its known strains will be appended into a single dataframe. In the end, this resulting dataframe will contain all strain-level information for all species of the taxonomy table.

Regarding the second input type which is a simple file input (in n x 1 format), the file can contain n rows of either all BacDive-IDs, culture collection numbers, taxonomy information (either as full name or as list with genus name, species with optional epithet, and optional subspecies) or sequence accession numbers (either 16S sequences, SILVA-IDs or genome sequences). For each one of the n rows BacDive is then queried and all strain-level information is stored in a single dataframe, just like with the taxonomy table input. This resulting dataframe is of the format n x r (or m x r)


1
n
n x 1
Input list (SILVA-IDs, species names, etc.)
BacDive
r
n
n x r
Resulting BacDive information
NetCoMi
Bacdiving
7
n
n x 7
Taxonomy table from phyloseq object
7
m
m x 7
Taxonomy table after removing unknown species
BacDive
r
m
m x r
Resulting BacDive information
+
s
m
m x s
Corresponding Metadata

with r being the number of BacDive columns for which we have information for at least one of the input species. This dataframe can then be written to file. All other core functions in BacDiving rely on this resulting dataframe.

Depending on the research question, either BacDiving's visualization options can be used or custom visualizations can be made using the resulting dataframe. There is really no limit on how you can extend and make use of the resulting dataframe.

For instance, this resulting dataframe along with metadata (which is often stored in phyloseq-objects) could be used in tools like NetCoMi to construct various types of networks. The nodes of these networks could be colored with specific phylogenetic information from BacDive as stored in the resulting dataframe file which in turn may explain why a given network looks the way it does. In other words, coloring the nodes in a network based on phylogentic information may explain the underlying correlation between various features and conditions in a dataset.

Accessing BacDive

As soon as you have registered on BacDive, you can use your credentials to run Bacdiving's most central function `bacdiving.bacdive_call()`:

The first thing `bacdiving.bacdive_call()` does is, it will prompt you to input your login credentials prior to querying BacDive, if you did not input your credentials via the function parameters `"bacdive_id"` and `"bacdive_password"`.

After that, it generates the resulting dataframe(s) (BacdiveInformation.tsv) with all strain-level information and it can output the BacDive access statistics (if the parameter is set) as a .txt-file which gives information on the percentage of input species found on BacDive and also lists all species which could not be found on BacDive.

For accessing specific data entries in your resulting dataframe you can either run `bacdiving.get_resulting_df_values()` or `bacdiving.access_list_df_objects()`.

However, `bacdiving.access_list_df_objects()` is only designed to be used if you are interested in retrieving information for either pH, temperature or halophily (e.g. prior to making a box plot), whereas `bacdiving.get_resulting_df_values()` is more generic.

Visualizations

Bacdiving supports 8 different visualization types:

1. Circular hierarchical taxonomic tree plot (also referred to as overview tree plot since it gives information on which species have what kind of BacDive information):

A similar circular hierarchical tree plot but without BacDive information can be created as well:

2. Stacked bar plot to show relative abundance (of e.g. different genera) per sample:
3. Pie chart to plot information like oxygen tolerance:
4. World map to show all countries (not water bodies!) of origin for a given set of species:
5. Fatty acid profile plot for a fatty acid of interest:
6. Frequency plot (of e.g. most frequent sampling type):
7. Box plot to compare e.g. optimal temperature ranges for various species
8. Bar plot to compare e.g. cell length of different species

1.1.3 Tutorial

We start by importing Bacdiving:

```
from bacdiving import bacdive_caller as bc
from bacdiving import treeplots_maker as tm
from bacdiving import visualizations_maker as vm
```

Now assume we have the following taxonomy table taxtab.tsv (which we prior extracted from a phyloseq-object):

Table 1: taxtab.tsv

	Kingdom	Phylum	Class	Order	Family	Genus	Species
ASV1	Bacteria	Bacteroidota	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacteroides	vulgatus
ASV2	Bacteria	Firmicutes	Clostridia	Lachnospirales	Lachnospiraceae	Blautia	NA
ASV3	Bacteria	Bacteroidota	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacteroides	NA
ASV4	Bacteria	Bacteroidota	Bacteroidia	Bacteroidales	Bacteroidaceae	Bacteroides	uniformis
ASV5	Bacteria	Firmicutes	Clostridia	Oscillospirales	Ruminococcaceae	Faecalibacterium	NA

Note: This taxonomy table stems from Nagel et al. (2016) but you can use any taxonomy table in this .tsv format to follow along with this tutorial. For demonstration purposes this table only shows the first 5 rows of the taxonomy table.

Warning: Note how species-level information is simply not always known for all ASVs in a taxonomy table. On average, you can expect 80% - 95% of all species from a given taxonomy table to be documented in BacDive. The exact percentage is mentioned in the access statistics output file which is generated after running `bacdiving.bacdive_call()`.

To get the resulting dataframe with all strain-level BacDive information for all the species in this taxonomy table you can run:

```
# Run for single taxonomy table input (e.g. as extracted from phyloseq-object)
resulting_list_with_all_res_dfs = bc.bacdive_call(bacdive_id="<your ID>", bacdive_
↪password="<your password>", input_lists={"./taxtab.tsv" : ["taxtable_input"]}, sample_
↪names=["taxtab"], output_dir="./")
resulting_df = resulting_list_with_all_res_dfs[0]
```

Assuming you do not have a taxonomy table, but have a simple file as input instead which looks something like this:

Table 2: SILVA_ids.txt

AB681649
AB121974
EU847536
D30768
L35516
AB681086
AB052706
AF144407
AF363064
AJ430586

Note: For demonstration purposes this SILVA_ids.txt file only contains 10 SILVA-ids. Note that other input types for querying BacDive are possible as well, e.g. taxonomy (as in a list of all species names of interest), Bacdive-ids, culture collection ids or genome accession ids. However, it is important that a single file can not contain multiple input types.

Given your input file, you can run the following, depending on your input type:

```
# Run for a single input from text file for SILVA id queries
resulting_list_with_all_res_dfs = bc.bacdive_call(bacdive_id="<your ID>", bacdive_
↪password="<your password>", input_lists={"./SILVA_ids.txt" : ["input_via_file",
↪"search_by_16S_seq_accession"]}, sample_names=["silva"], output_dir="./")
resulting_df = resulting_list_with_all_res_dfs[0]

# Run for a single input from text file for taxonomy queries
resulting_list_with_all_res_dfs = bc.bacdive_call(input_lists={"./taxonomy_ids.txt" : [
↪"input_via_file", "search_by_taxonomy"]}, sample_names=["taxonomy"], output_dir="./
↪results/") # if credentials are not given via parameters, you will get prompted
resulting_df = resulting_list_with_all_res_dfs[0]

# Run for a single input from text file for BacDive id queries
resulting_list_with_all_res_dfs = bc.bacdive_call(bacdive_id="<your ID>", bacdive_
↪password="<your password>", input_lists={"./bacdive_ids.txt" : ["input_via_file",
↪"search_by_id"]}, sample_names=["bacdive"], output_dir="./")
resulting_df = resulting_list_with_all_res_dfs[0]

# Run for a single input from text file for culture collection queries
resulting_list_with_all_res_dfs = bc.bacdive_call(bacdive_id="<your ID>", bacdive_
↪password="<your password>", input_lists={"./culture_col_ids.txt" : ["input_via_file",
↪"search_by_culture_collection"]}, sample_names=["culturecol"], output_dir="./")
```

(continues on next page)

(continued from previous page)

```
resulting_df = resulting_list_with_all_res_dfs[0]

# Run for a single input from text file for genome accession queries
resulting_list_with_all_res_dfs = bc.bacdive_call(bacdive_id="<your ID>", bacdive_
↪password="<your password>", input_lists={"./genome_ids.txt" : ["input_via_file",
↪"search_by_genome_accession"]}, sample_names=["genomecol"], output_dir="./")
resulting_df = resulting_list_with_all_res_dfs[0]
```

If you have multiple inputs of possibly different input types, you can run the following command:

```
# Run for multiple inputs (of possibly different input types)
resulting_list_with_all_res_dfs = bc.bacdive_call(bacdive_id="<your ID>", bacdive_
↪password="<your password>", input_lists={"./SILVA_ids.txt" : ["input_via_file",
↪"search_by_16S_seq_accession"], "./taxonomy_ids.txt" : ["input_via_file", "search_by_
↪taxonomy"], "./taxtab1.tsv" : ["taxtable_input"], "./taxtab2.tsv" : ["taxtable_input"]}
↪,sample_names=["sample1", "sample2", "sample3", "sample4"])
resulting_df = resulting_list_with_all_res_dfs[1]  # pick your dataframe of interest␣
↪from this list
```

Now that you have the resulting dataframe at hand, you are almost ready to start visualizing the data.

Warning: If you try to plot information for a column which is not present in the resulting dataframe or if your parameters are set incorrectly or do not match the resulting dataframe, you may get an error. So, make sure to get to know your resulting dataframe (and especially its columns) beforehand.

Let’s first take a look at the resulting dataframe:

```
#Dataframe exploration
print(resulting_df.head()) # prints head of resulting dataframe
print(len(resulting_df.index)) #print number of resulting_df rows
print(resulting_df.keys())  #print resulting_df column names
print(resulting_df.iloc[0:5, 2:4]) #print all specific column information via column␣
↪index
print(resulting_df["Physiology and metabolism.oxygen tolerance.oxygen tolerance"].
↪unique()) #print unique values in a given column
print(resulting_df.loc[resulting_df["Name and taxonomic classification.species" ] ==
↪"Bacteroides uniformis"])  # print all strains and all columns for Bacteroides␣
↪uniformis from resulting_df
print(resulting_df.loc[resulting_df["Name and taxonomic classification.species"] ==
↪"Helicobacter pylori"]["Culture and growth conditions.culture temp"])  # print all␣
↪strains for column "Culture and growth conditions.culture temp" for Helicobacter␣
↪pylori from resulting_df
print(resulting_df.loc[(resulting_df["Name and taxonomic classification.species"] ==
↪"Helicobacter pylori") & (resulting_df["Isolation, sampling and environmental␣
↪information.isolation.country"] == "Germany")]) #Subset resulting_df to certain␣
↪parameters
print(len(resulting_df.loc[resulting_df["Name and taxonomic classification.species"] ==
↪'Zhihengliuella alba'].index)) #find out how many strains are present for a given␣
↪species
print(get_resulting_df_values(resulting_df, "Culture and growth conditions.culture pH",
↪"pH", species_list=["Helicobacter pylori", "Bacteroides clarus", "Actinomyces␣
```

(continues on next page)

(continued from previous page)

```
↪odontolyticus", "Bacteroides salyersiae", "Zhihengliuella alba"])) #Given a list of␣
↪species of interest, access elements in resulting_df which are nested
```

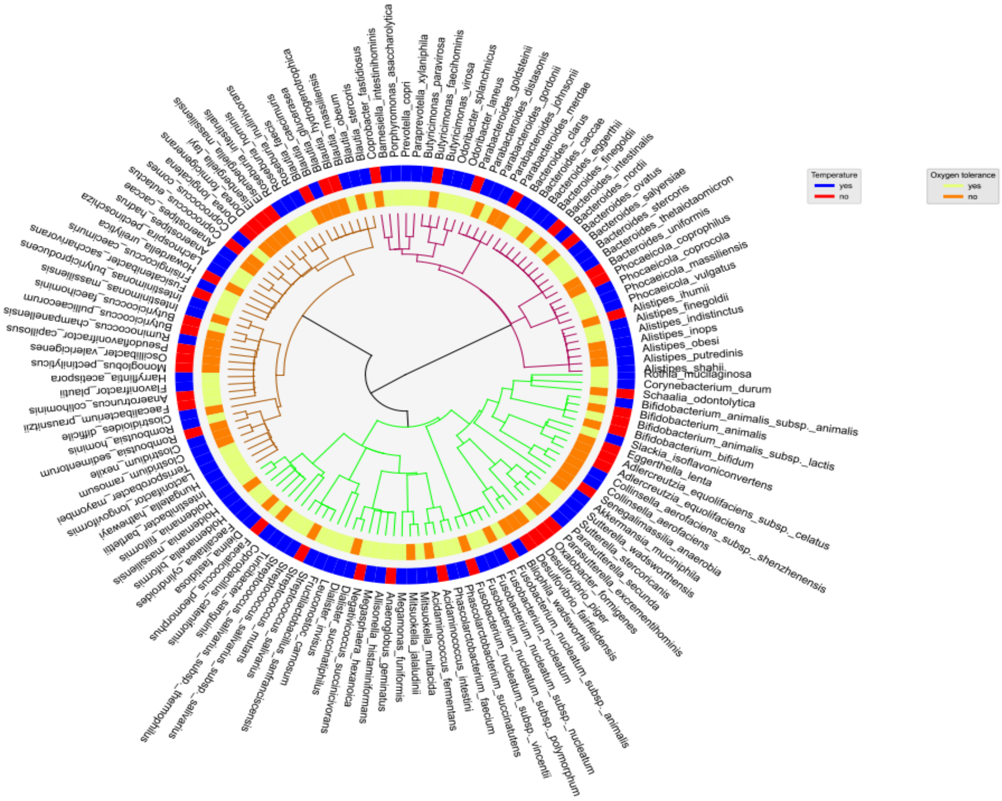
Great, now we know the basic structure of our resulting dataframe and what kind of BacDive information we have, so it is finally time to start plotting!

Note: There are many possibilites for which columns from the resulting dataframe can be plotted for each plotting function. This tutorial shall only demonstrate a few examples.

In order to first gain some overview of our data, let us start with Bacdiving's overview treeplot. Assume we want to know for which species BacDive information on temperature and oxygen tolerance is known or not. We can do this by running the following command:

```
#Overview treeplot
tm.overview_treeplot(resulting_df, label_name1="Temperature", label_name2="Oxygen␣
↪tolerance", saveToFile=True, output_dir="./")
```

This results in the following plot:

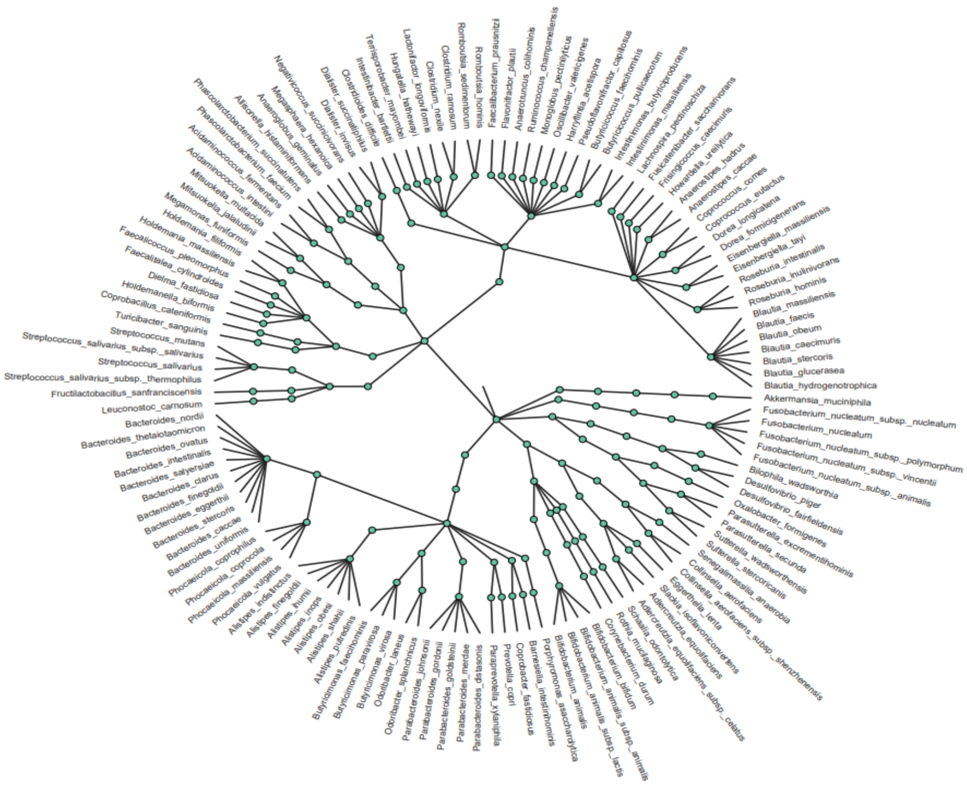


If you do not want the BacDive information to be shown and just prefer the hierarchical taxonomy tree plot, then run:

```
#Circular treeplot
tm.circular_treeplot(resulting_df, output_dir="./")
```

This results in the following plot:

Let's say we are interested in generating a fatty acid profile plot for "Achromobacter denitrificans":



```
#Fatty acid profile plot
vm.fatty_acid_profile(resulting_df, species = "Achromobacter denitrificans", ␣
↪figsize=[20, 15], saveToFile=True, output_dir="./")
```

This results in the following plot:

We can also make pie plots to look at the motility of our species:

```
#Pie plot
vm.pieplot_maker(resulting_df,"Morphology.cell morphology.motility", title="Motility for␣
↪all species", saveToFile = True, output_dir="./")
```

This results in the following plot:

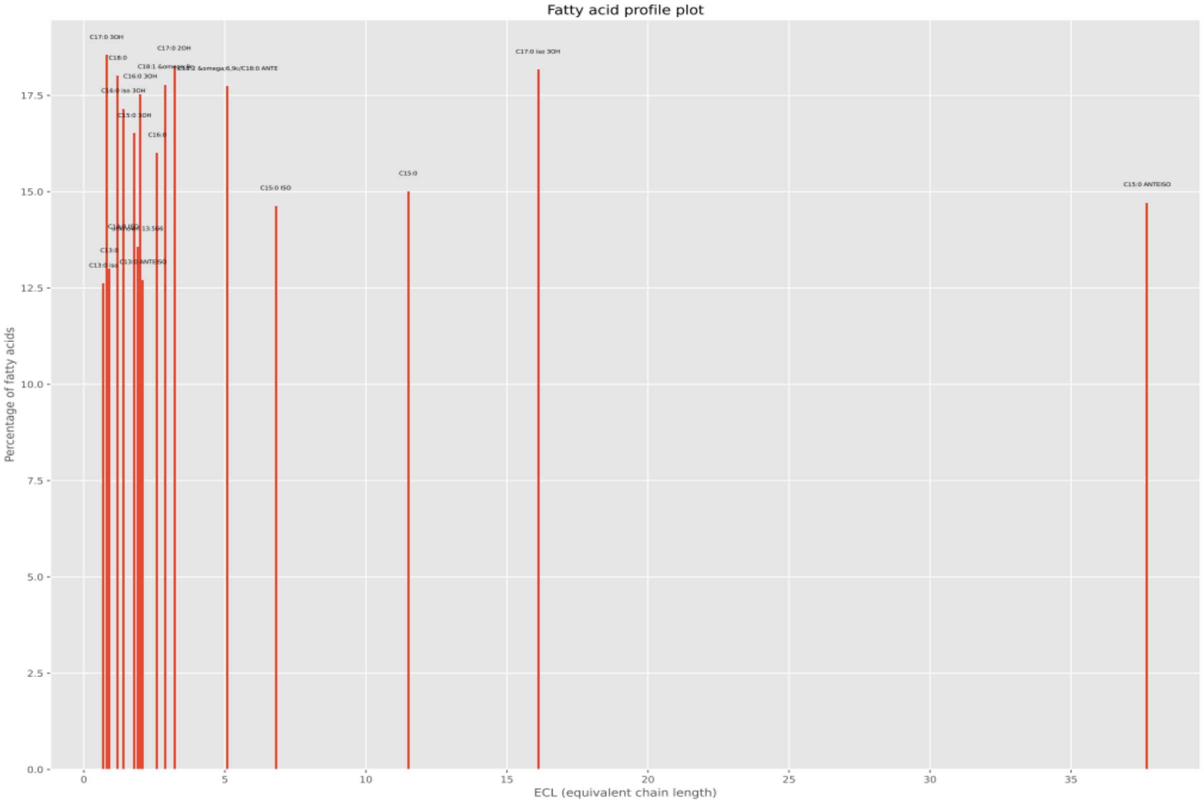
If we are interesting in knowing from which countries the species in our dataset originate from we can create a world map:

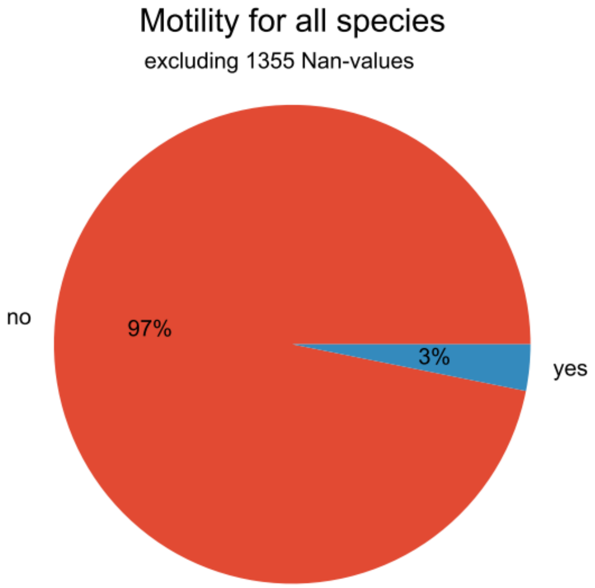
```
#World map
vm.worldmap_maker(resulting_df)
```

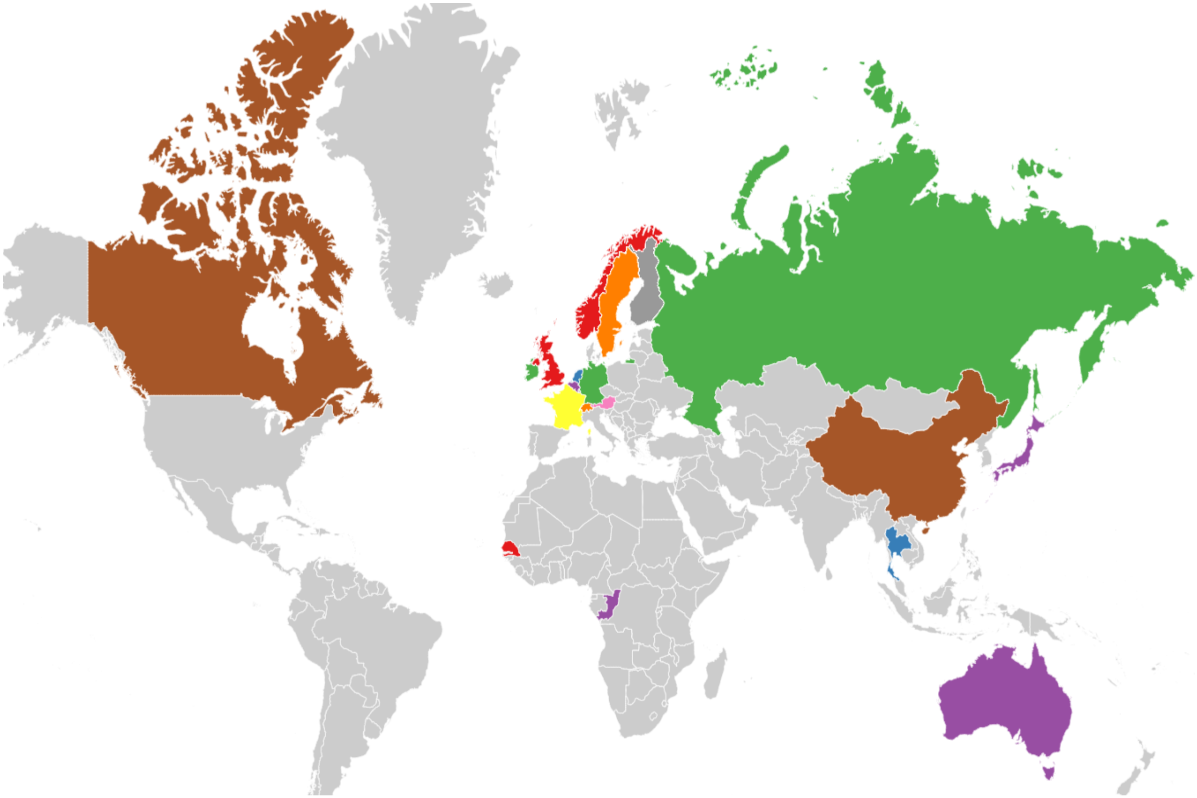
This results in the following plot:

Going from this world map if we want to know which country is the most frequent, we can run:

```
#Frequency plot
vm.freqplot_maker(resulting_df, "Isolation, sampling and environmental information.
↪isolation.country", title="Countries of origin", ylabel_name = "All countries",␣
↪saveToFile=True, output_dir="./")
```


Fatty acid profile plot
C17:0 3OH
C18:0
C17:0 2OH
C18:1 &omega;9c
C18:2 &omega;6,9c/C18:0 ANTE
C16:0 3OH
C16:0 iso 3OH
C15:0 3OH
C16:0
C14:0 ISO
unknown 13.566
C13:0
C13:0 iso
C13:0 ANTEISO
C15:0 ISO
C15:0
C17:0 iso 3OH
C15:0 ANTEISO
Percentage of fatty acids
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
20
25
30
35
ECL (equivalent chain length)


Motility for all species
excluding 1355 Nan-values
no
97%
3%
yes



This results in the following plot:

Next, we want to make a bar plot to visualize the differences in cell width across various species:

```
#Species list for ALL species in resulting_df, not for a subset
species_list = resulting_df["Name and taxonomic classification.species"].tolist()

#Barplot
vm.barplot_maker(resulting_df, "Morphology.cell morphology.cell width", "Cell width",
↪"Width in µm", figsize=[20,10], species_list=species_list, saveToFile=True, output_dir=
↪"./")
```

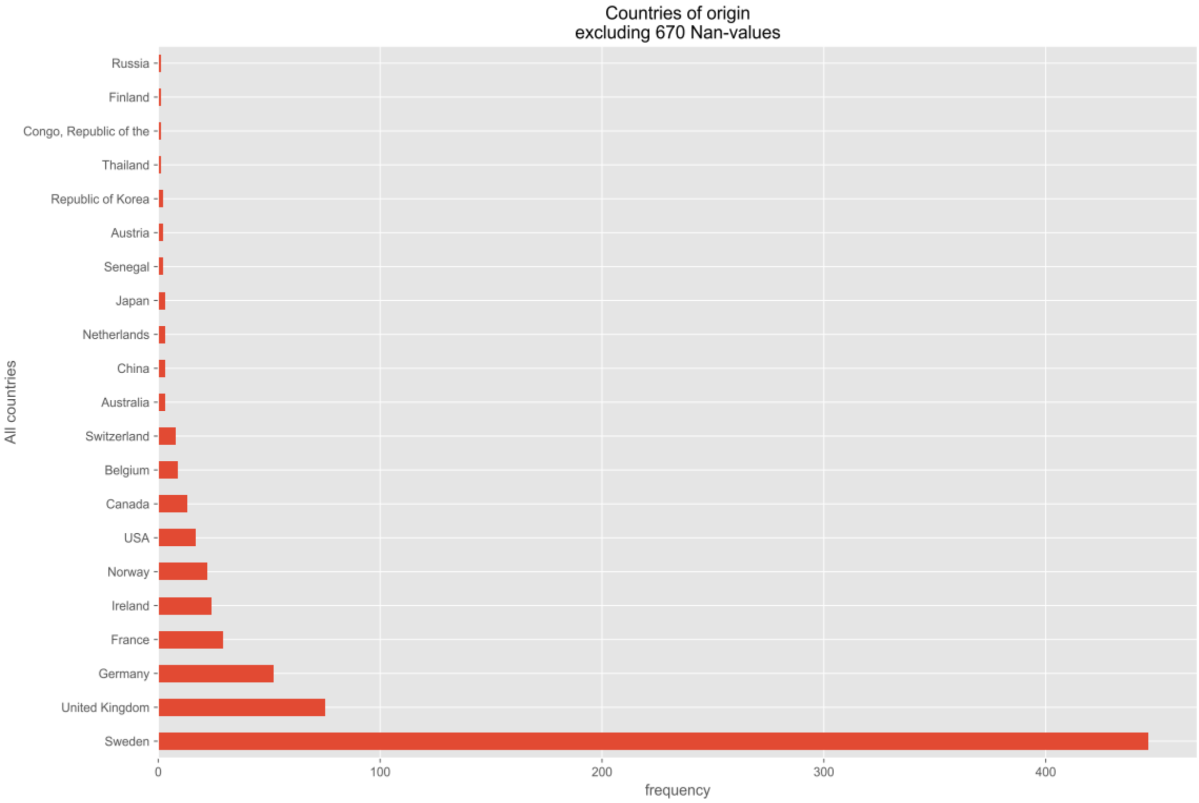
This results in the following plot:

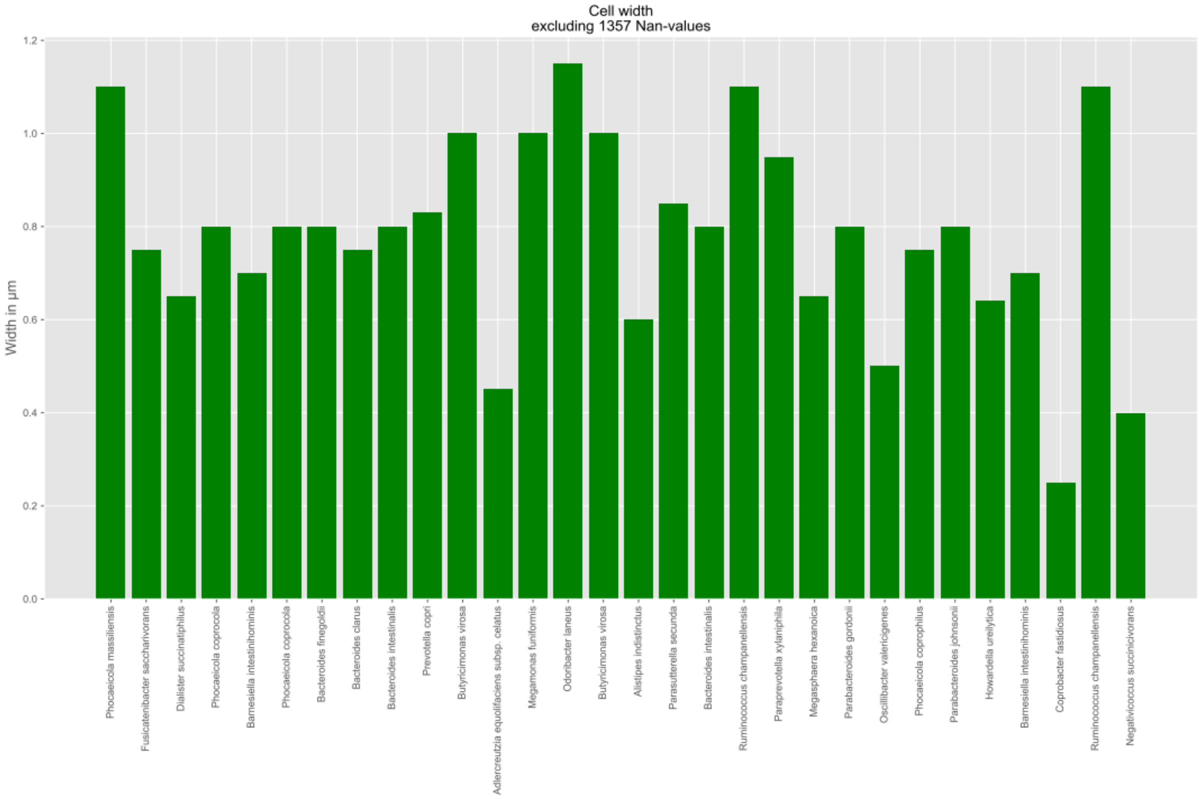
Let's make a boxplot which shows the optimal pH range for all the species in our dataset:

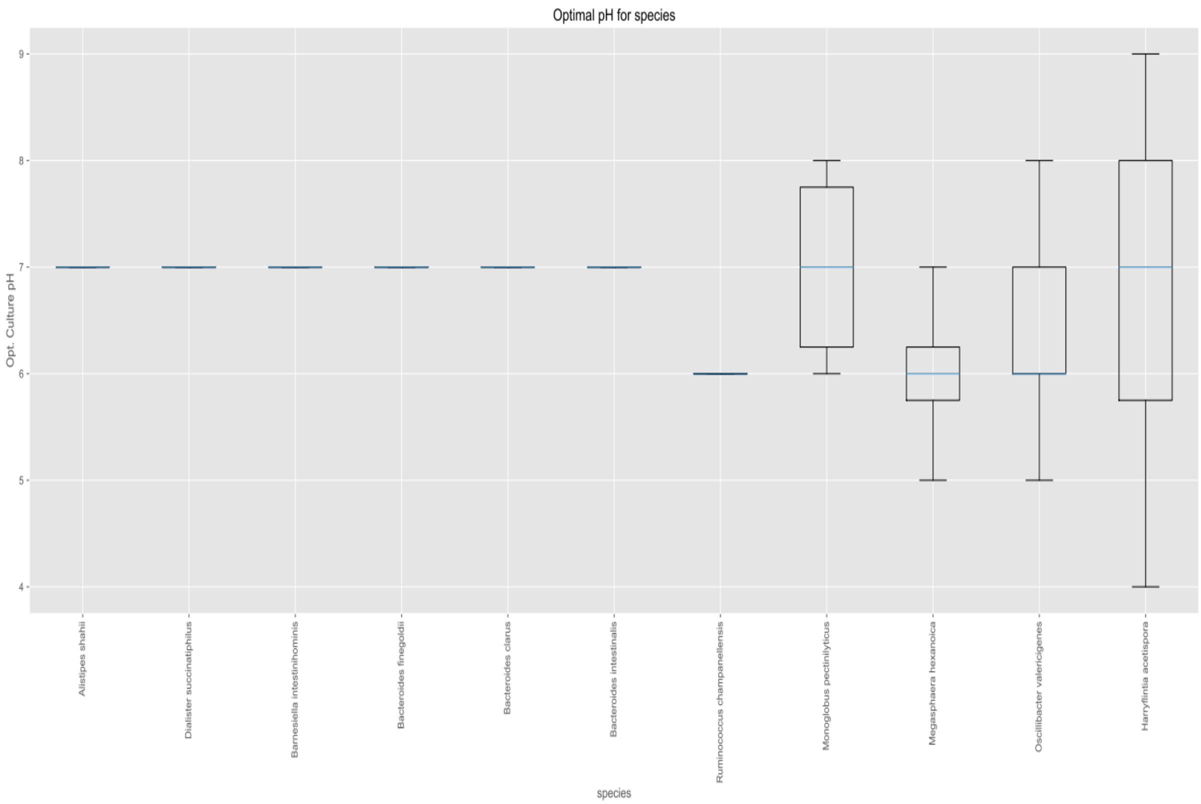
```
#Species list for ALL species in resulting_df, not for a subset
species_list = resulting_df["Name and taxonomic classification.species"].tolist()

#Boxplot
value_dict = vm.access_list_df_objects(resulting_df, "Culture and growth conditions.
↪culture pH", "pH", pH= 1, species_list=species_list)
vm.boxplot_maker(value_dict, title= "Optimal pH for species", xlabel_name= "species",␣
↪ylabel_name="Opt. Culture pH",figsize=[20, 10], saveToFile=True, output_dir="./")
```

This results in the following plot:

Lastly, we can compare the relative abundances of e.g. the genera for our SILVA-ids.txt and our taxonomy table input in a stacked bar plot:


Countries of origin
excluding 670 Nan-values
All countries
Russia
Finland
Congo, Republic of the
Thailand
Republic of Korea
Austria
Senegal
Japan
Netherlands
China
Australia
Switzerland
Belgium
Canada
USA
Norway
Ireland
France
Germany
United Kingdom
Sweden
0
100
200
300
400
frequency


Cell width
excluding 1357 Nan-values
Width in µm
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Phocaeicola massiliensis
Fusicatenibacter saccharivorans
Dialister succinatiphilus
Phocaeicola coprocola
Barnesiella intestinihominis
Phocaeicola coprocola
Bacteroides finegoldii
Bacteroides clarus
Bacteroides intestinalis
Prevotella copri
Butyricimonas virosa
Adlercreutzia equolifaciens subsp. celatus
Megamonas funiformis
Odoribacter laneus
Butyricimonas virosa
Alistipes indistinctus
Parasutterella secunda
Bacteroides intestinalis
Ruminococcus champanellensis
Paraprevotella xylaniphila
Megasphaera hexanoica
Parabacteroides gordonii
Oscillibacter valericigenes
Phocaeicola coprophilus
Parabacteroides johnsonii
Howardella ureilytica
Barnesiella intestinihominis
Coprobacter fastidiosus
Ruminococcus champanellensis
Negativicoccus succinicivorans

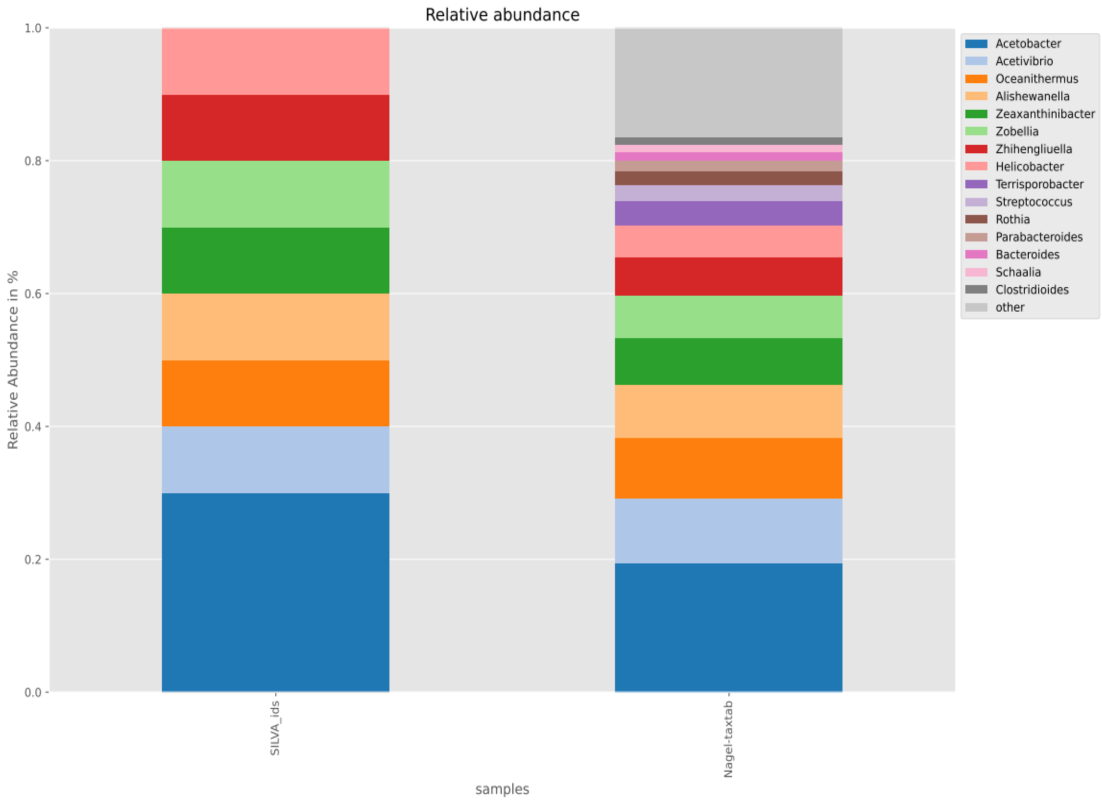


```
# Run for multiple inputs
resulting_list_with_all_res_dfs = bc.bacdive_call(input_lists={"./SILVA_ids.txt" : [
↪"input_via_file", "search_by_16S_seq_accession"], "./taxtab.tsv" : ["taxtable_input"]},
↪sample_names=["SILVA_ids", "Nagel_taxtab"])
#Relative abundance plot
vm.stacked_barplot_relative_abundance(resulting_list_with_all_res_dfs, sample_names=[
↪"SILVA_ids", "Nagel_taxtab"], plot_column="Name and taxonomic classification.genus",␣
↪title="Relative abundance", saveToFile = True, output_dir="./")
```

This results in the following plot:

In effect, this plot shows us the genera composition for all those species (for which BacDive information is available) in the resulting dataframe.

Note: This concludes this tutorial for Bacdiving but feel free to use the resulting dataframe to either generate your own custom visualizations or to use it as an input for other tools!


Relative abundance
Relative Abundance in %
1.0
0.8
0.6
0.4
0.2
0.0
SILVA_ids
Nagel-taxtab
samples
Acetobacter
Acetivibrio
Oceanithermus
Alishewanella
Zeaxanthinibacter
Zobellia
Zhihengliuella
Helicobacter
Terrisporobacter
Streptococcus
Rothia
Parabacteroides
Bacteroides
Schaalia
Clostridioides
other

1.1.4 API