Bacdiving

Release 1.2.7

Mahima Arunkumar

Dec 15, 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

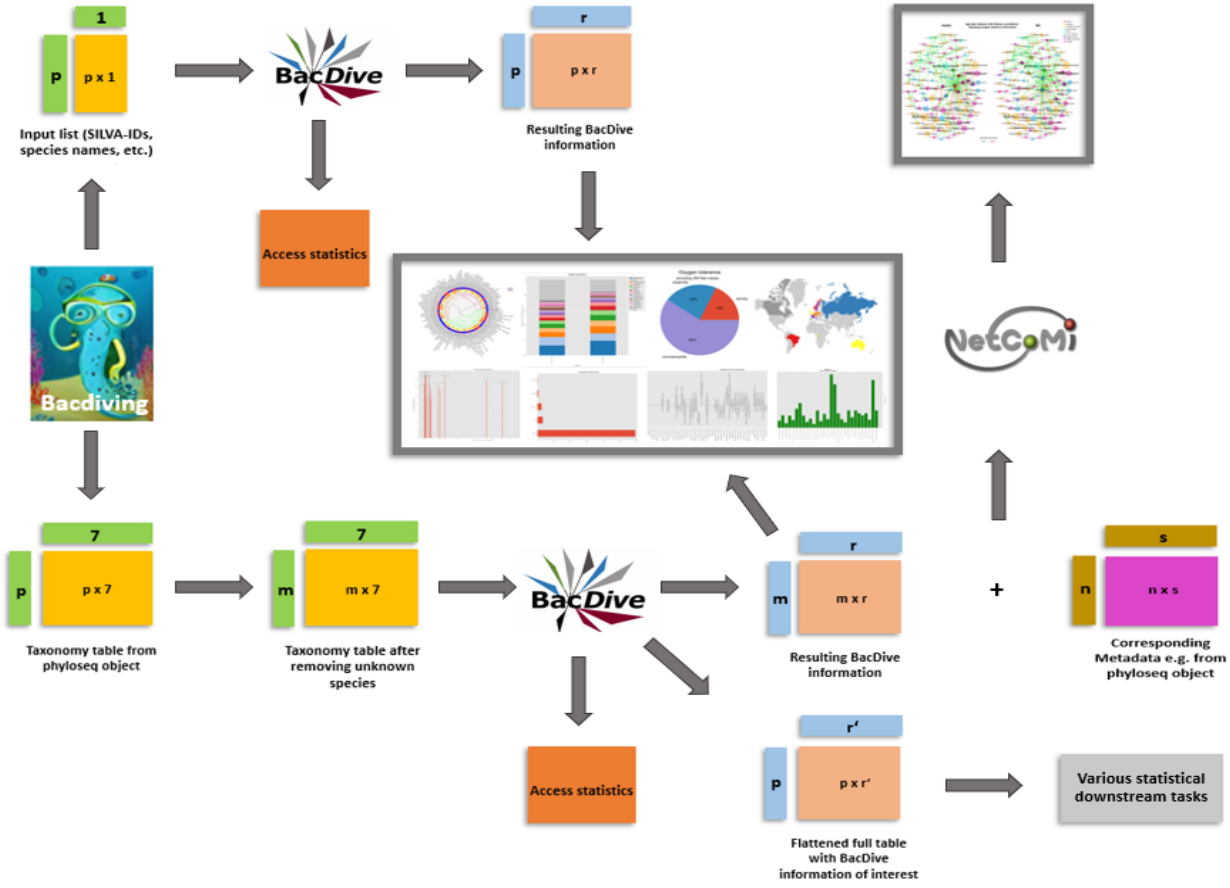
Bacdiving is a Python package which can access and retrieve information from the world's largest database for standardized bacterial phenotypic information: BacDive. Additionally, Bacdiving provides access statistics and options to visualize this information.

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.

Let us begin with the file input type (in p x 1 format). The file can contain p 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 p rows BacDive is then queried and all strain-level information is stored in a single dataframe. This resulting dataframe is of the format p x r with r being the number of BacDive columns for which we have information. This dataframe, along with the corresponding access statistics, can then be written to file. All other core functions in BacDiving rely on the resulting dataframe.

The second possible input type is a taxonomy table (in p x 7 format) which has the following 7 taxonomic ranks: kingdom, phylum, class, order, family, genus, species. 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


1
p
p x 1
Input list (SILVA-IDs, species names, etc.)
BacDive
Access statistics
r
p
p x r
Resulting BacDive information
Bacdiving
7
p
p x 7
Taxonomy table from phyloseq object
7
m
m x 7
Taxonomy table after removing unknown species
BacDive
Access statistics
r
m
m x r
Resulting BacDive information
r'
p
p x r'
Flattened full table with BacDive information of interest
+
s
n
n x s
Corresponding Metadata e.g. from phyloseq object
NetCoMi
Various statistical downstream tasks

if it 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 (m x r) will contain all strain-level information for all species of the taxonomy table. Thus, the resulting dataframe and the corresponding BacDive access statistics results can be written to file. Additionally, given a taxonomy table as input, one may wish to see Bacdive information across all strains of a given species in order to perform various other downstream statistical tasks (e.g. matrix completion, regression, etc.). Therefore, given specific BacDive features of interest, a flattened file (in p x r' format) can be outputted which contains the number of strains per species found on BacDive as well as the majority values across all strains for given BacDive features of interest.

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()*:

bacdiving.**bacdive_call**(*bacdive_id=''*, *bacdive_password=''*, *inputs_list=['']*, *sample_names=['']*, *print_res_df_ToFile=True*, *print_access_stats=True*, *print_flattened_file=False*, *columns_of_interest=['']*, *output_dir='./'*)

For multiple input files (either all input files or all taxonomy tables) this function reads the input, queries the BacDive database and stores resulting dataframe(s) and access statistics.

Parameters

- **bacdive_id** (*str*) – Log in credential: BacDive id.
- **bacdive_password** (*str*) – Log in credential: BacDive password.
- **inputs_list** (*list[str]*) – List which specifies (multiple) strings. Each string has the structure: "<file-path> <file-type> (<content-type>)" and is thus seperated by space(s). Content-type is, however, only required if you have input_via_file; it can have one of the following values: "search_by_id", "search_by_culture_collection", "search_by_taxonomy", "search_by_16S_seq_accession" or "search_by_genome_accession".
- **sample_names** (*list[str]*) – List of samples names.
- **print_res_df_ToFile** (*bool*) – Print the resulting dataframe with all Bacdive information to file or not.
- **print_access_stats** (*bool*) – Print the Bacdive access statistics to file or not.
- **print_flattened_file** (*bool*) – Print the flattened Bacdive information for certain columns of interest to file or not.
- **columns_of_interest** (*list[str]*) – Specify in this list which columns from Bacdive-Information.tsv you want to include in the flattened file.
- **output_dir** (*str*) – Path to where plot should be saved.

Returns

List containing the resulting dataframe(s) with all strain-level BacDive information for all those multiple inputs.

Return type
list[pandas.DataFrame]

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. Additional files (like Species_names_from_taxtable_file.csv or Flattened_Bacdive_data.tsv) may as well be outputted if your input was a taxonomy table. Note that the file Species_names_from_taxtable_file.csv lists all species from the taxonomy table, even prior to querying 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()*.

bacdiving.**get_resulting_df_values**(*resulting_df*, *plot_column=' '*, *plot_category=' '*, *species_list=[' ']*)

Access all categories of interest only for a column of interest from the resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **plot_column** (*str*) – Column of interest from resulting_df.
- **plot_category** (*str*) – Category of interest from column of interest from resulting_df.
- **species_list** (*list[str]*) – List of species of interest.

Returns
Dictionary: <species> : <values>

bacdiving.**access_list_df_objects**(*resulting_df*, *plot_column=' '*, *plot_category=' '*, *temp=0*, *pH=0*, *halophily=0*, *species_list=[' ']*)

Access all categories of interest only for the pH, temperature and halophily columns from the resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **plot_column** (*str*) – Column of interest from resulting_df.
- **plot_category** (*str*) – Category of interest from column of interest from resulting_df.
- **temp** (*int*) – Either one of temp, pH or halophily can be accessed. If temp = 1, temp will be accessed.
- **pH** (*int*) – Either one of temp, pH or halophily can be accessed. If pH = 1, pH will be accessed.
- **halophily** (*int*) – Either one of temp, pH or halophily can be accessed. If halophily = 1, halophily will be accessed.
- **species_list** (*list[str]*) – List of species of interest.

Returns
Dictionary: <species> : <values>

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):

bacdiving.**overview_treeplot**(*resulting_df*, *pallete='brg'*, *colormap1='bwr'*, *column_name1='Culture and growth conditions.culture temp.temperature'*, *column_name2='Physiology and metabolism.oxygen tolerance.oxygen tolerance'*, *label_name1='Category1'*, *label_name2='Category2'*, *colormap2='Wistia'*, *fontsize=14*, *figsize=[20, 10]*, *saveToFile=True*, *output_dir='./'*)

Makes overview tree plot showing hierarchical tree structure for all species of input as well as maximum 2 BacDive columns of interest.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **pallete** (*str*) – Color palette used.
- **colormap1** (*str*) – Color map used for first column of interest.
- **column_name1** (*str*) – First column of interest from resulting_df to plot.
- **column_name2** (*str*) – Second column of interest from resulting_df to plot.
- **label_name1** (*str*) – Legend label for first column of interest.
- **label_name2** (*str*) – Legend label for second column of interest.
- **colormap2** (*str*) – Color map for second column of interest.
- **fontsize** (*int*) – Size of font.
- **figsize** (*tuple[float, float]*) – Size of plot.
- **saveToFile** (*bool*) – Boolean to save plot as a .pdf file or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.

Returns

Overview plot

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

bacdiving.**circular_treeplot**(*resulting_df*, *width=1400*, *height=1400*, *saveToFile=True*, *output_format='pdf'*, *output_dir='./'*)

Makes tree plot showing hierarchical tree structure for all species of input.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **width** (*int*) – Width of tree plot.
- **height** (*int*) – Height of tree plot.
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_format** (*str*) – Output file type. Possible file formats include: pdf, svg and html.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.

Returns
Circular treeplot

2. Stacked bar plot to show relative abundance (of e.g. different genera) per sample:

bacdiving.**stacked_barplot_relative_abundance**(*resulting_df*, *top_x=15*, *sample_names=[' ']*, *plot_column=' '*, *title=' '*, *title_label=' '*, *saveToFile=True*, *output_dir='./'*, *figsize=[15, 10]*)

Makes stacked bar plot for any taxonomy level from resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **top_x** (*int*) – Limit for how many different color categories should be seen in the plot.
- **sample_names** (*list[str]*) – List of names for each sample.
- **plot_column** (*str*) – Taxonomy level of interest (e.g. Name and taxonomic classification.genus).
- **title** (*str*) – Title for this plot.
- **title_label** (*str*) – Title for legend (e.g. Genus).
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.
- **figsize** (*tuple[float, float]*) – Size of the resulting plot.

Returns
Stacked bar plot

3. Pie chart to plot information like oxygen tolerance:

bacdiving.**pieplot_maker**(*resulting_df*, *plot_column*, *title=' '*, *ylabel_name=' '*, *saveToFile=False*, *output_dir='./'*, *figsize=[6.4, 4.8]*)

Makes pie plot for any categorical column of interest from resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **plot_column** (*str*) – (Categorical) Column of interest from resulting_df.
- **title** (*str*) – Title for this plot.
- **ylabel_name** (*str*) – y-axis label name.
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.
- **figsize** (*tuple[float, float]*) – Size of the resulting plot.

Returns
Pie plot

4. World map to show all countries (not water bodies!) of origin for a given set of species:

bacdiving.**worldmap_maker**(*resulting_df*)

Makes world map displaying all countries where species from the input originate from.

Parameters
resulting_df (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().

Returns
World map

5. Fatty acid profile plot for a fatty acid of interest:

bacdiving.**fatty_acid_profile**(*resulting_df*, *species=' '*, *title='Fatty acid profile plot'*, *figsize=[10, 10]*, *barwidth=0.05*, *fontsize=6*, *saveToFile=True*, *output_dir='./'*)

Makes fatty acid profile plot for any one fatty acid of interest of interest from resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **species** (*str*) – Species of interest.
- **title** (*str*) – Title for this plot.
- **figsize** (*tuple[float, float]*) – Size of the resulting plot.
- **barwidth** (*float*) – Width of the bars.
- **fontsize** (*int*) – Size of the font.
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.

Returns
Fatty acid profile plot

6. Frequency plot (of e.g. most frequent sampling type):

bacdiving.**freqplot_maker**(*resulting_df*, *plot_column=' '*, *title=' '*, *ylabel_name=' '*, *saveToFile=False*, *output_dir='./'*, *figsize=[15, 10]*)

Makes freq plot for any categorical column of interest from resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **plot_column** (*str*) – Column of interest from resulting_df.
- **title** (*str*) – Title for this plot.
- **ylabel_name** (*str*) – y-axis label name.
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.
- **figsize** (*tuple[float, float]*) – Size of the resulting plot.

Returns
Frequency plot

7. Box plot to compare e.g. optimal temperature ranges for various species

bacdiving.**boxplot_maker**(*resulting_dict*, *title=' '*, *xlabel_name=' '*, *ylabel_name=' '*, *saveToFile=False*, *output_dir='./'*, *figsize=[15, 10]*)

Makes box plot given a dictionary with values of interest.

Parameters

- **resulting_dict** (*dict[Any]*) – Dictionary input with values (e.g. temperature or pH).
- **title** (*str*) – Title for this plot.
- **xlabel_name** (*str*) – x-axis label name.
- **ylabel_name** (*str*) – y-axis label name.
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.
- **figsize** (*tuple[float, float]*) – Size of the resulting plot.

Returns

Box plot

8. Bar plot to compare e.g. cell length of different species

bacdiving.**barplot_maker**(*resulting_df*, *plot_column=' '*, *title=' '*, *ylabel_name=' '*, *xlabel_name=' '*, *color='green'*, *species_list=[]*, *saveToFile=False*, *output_dir='./'*, *figsize=[15, 10]*)

Makes bar plot for any continuous column of interest from resulting dataframe.

Parameters

- **resulting_df** (*pandas.DataFrame*) – Resulting dataframe as outputted by bacdive_call().
- **plot_column** (*str*) – (Categorical) Column of interest from resulting_df.
- **title** (*str*) – Title for this plot.
- **ylabel_name** (*str*) – y-axis label name.
- **xlabel_name** (*str*) – x-axis label name.
- **color** (*str*) – Color of bars.
- **species_list** (*list[str]*) – List of species of interest.
- **saveToFile** (*bool*) – Boolean to save plot or not.
- **output_dir** (*str*) – Path to where plot should be saved if saveToFile is set to True.
- **figsize** (*tuple[float, float]*) – Size of the resulting plot.

Returns

Bar plot

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>", inputs_list=["./taxtab.tsv taxtable_input"], sample_names=[
↪"taxtab"], print_res_df_ToFile = True, print_access_stats = True, print_flattened_
↪file=True, columns_of_interest=["Physiology and metabolism.oxygen tolerance.oxygen␣
↪tolerance", "Culture and growth conditions.culture temp.temperature", "Isolation,␣
↪sampling and environmental information.isolation.origin.country","Morphology.cell␣
↪morphology.motility"], output_dir="./")
resulting_df = resulting_list_with_all_res_dfs[0]
```

Warning: If you set `"print_flattened_file"` to True, then make sure your `"columns_of_interest"` contain meaningful BacDive features. For example, “General.BacDive-ID” or “General.description” would not be a smart choice as the majority value over all strains per species will be written to the flattened output file. Also,

columns like "Culture and growth conditions.culture pH" may not be a good choice as this is typically a column with mixed types (neither fully categorical nor numerical). If you are interested in obtaining information from such columns with mixed types, use rather other functions like *bacdiving.get_resulting_df_values()*. Thus, you should use columns like the ones above instead, which will print the majority value (which is not nan) for a given BacDive feature across all strains per species.

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>", inputs_list=["./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(inputs_list=["./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>", inputs_list=["./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>", inputs_list=["./culture_col_ids.txt input_via_file search_
↪by_culture_collection"], sample_names=["culturecol"], output_dir="./")
resulting_df = resulting_list_with_all_res_dfs[0]
```

(continues on next page)

(continued from previous page)

```
# 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>", inputs_list=["./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>", inputs_list=["./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"], print_flattened_file=True, columns_of_interest=["Physiology and␣
↪metabolism.oxygen tolerance.oxygen tolerance", "Culture and growth conditions.culture␣
↪temp.temperature"])
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!

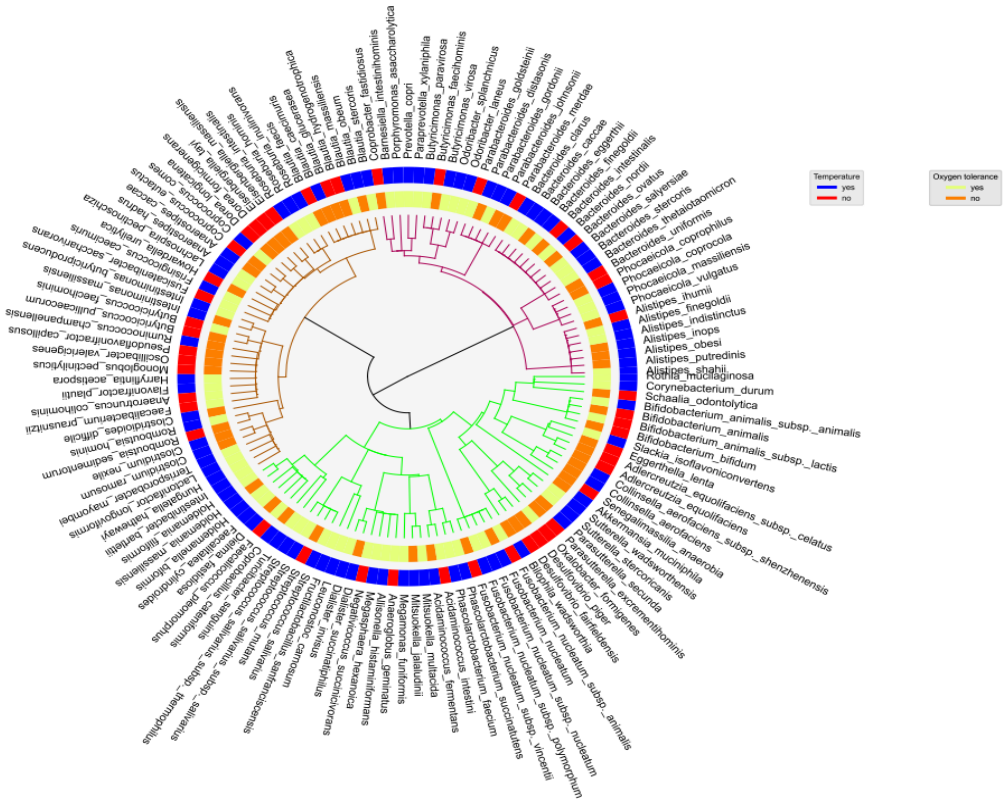
Note: There are many possibilites for which columns from the resulting dataframe can be plotted for each plotting function. This tutorial shall only demonstrate a few examples.

Warning: Since we are dealing with strain-level information per species, please be aware when investigating your plotting results that BacDive information does not neccessarily exist for all strains.

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="./")
```

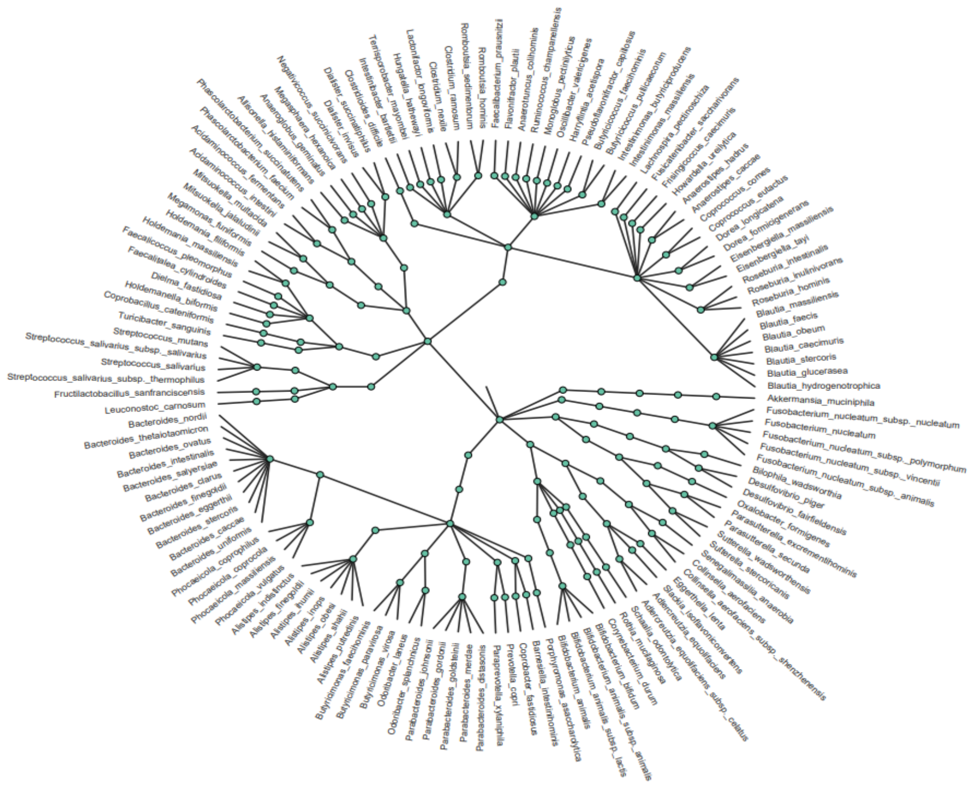
This results in the following plot:



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="./")
```

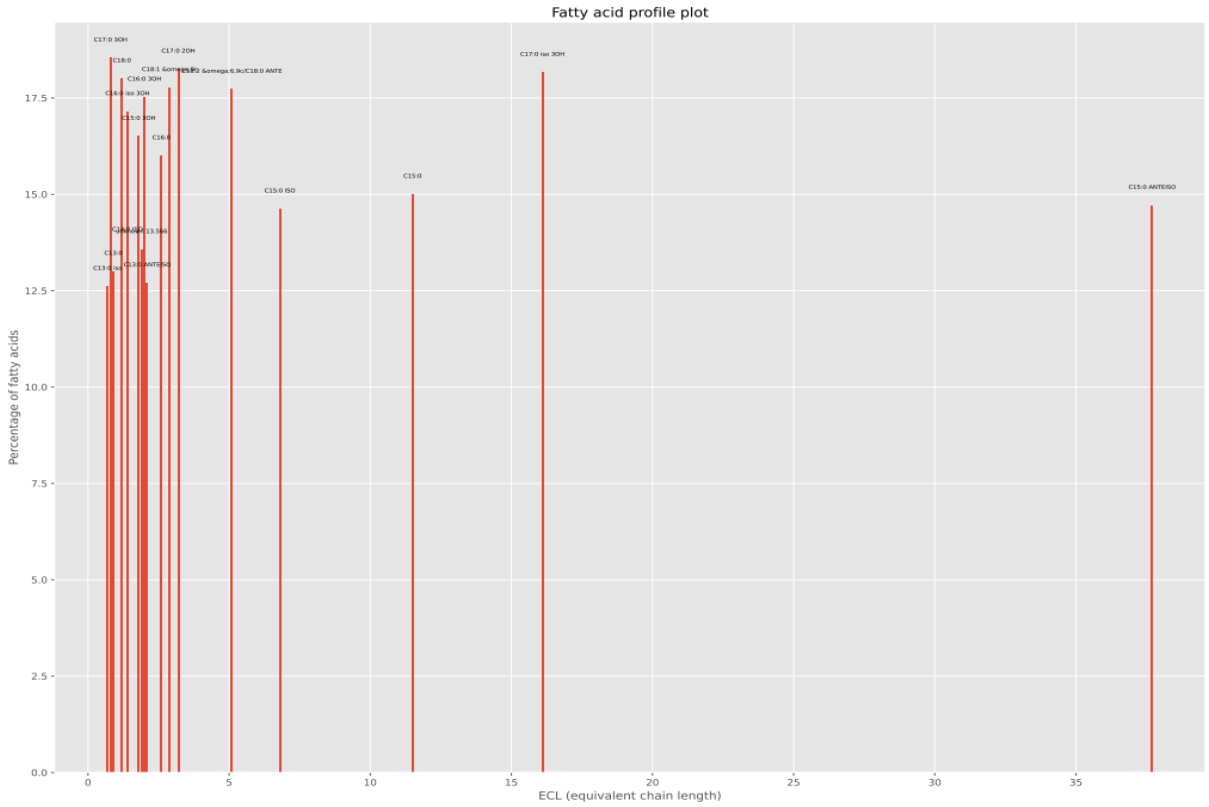
This results in the following plot:

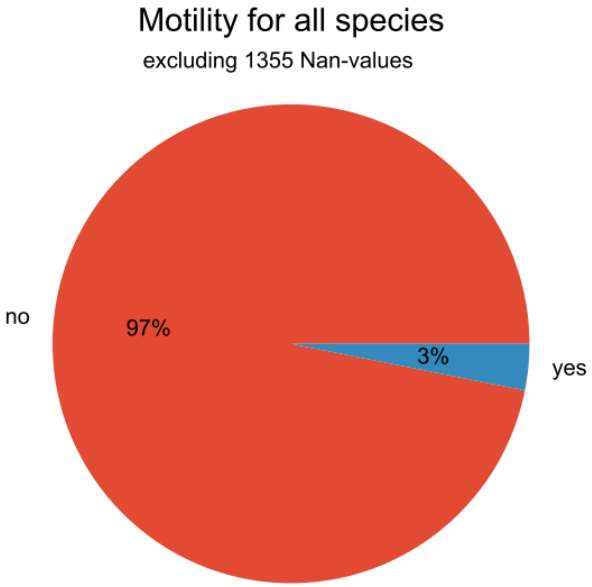
If we are interesting in knowing from which countries the species in our dataset originate from we can create a world map:

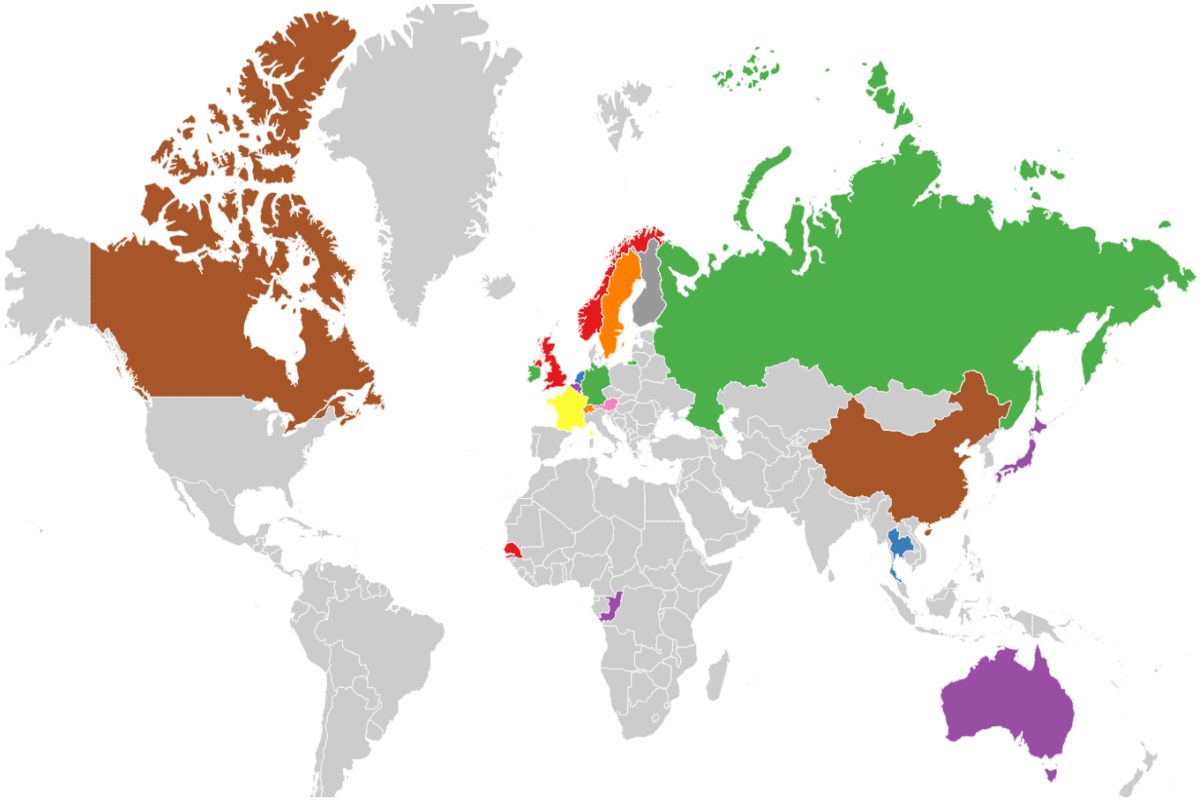
```
#World map
vm.worldmap_maker(resulting_df)
```

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:


Fatty acid profile plot
Percentage of fatty acids
ECL (equivalent chain length)
C17:0 3OH
C18:0
C17:0 2OH
C16:0 3OH
C15:0 3OH
C15:0 ISO
C15:0
C17:0 iso 3OH
C15:0 ANTEISO
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15
20
25
30
35


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



```
#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="./")
```

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=
↪"./")
```

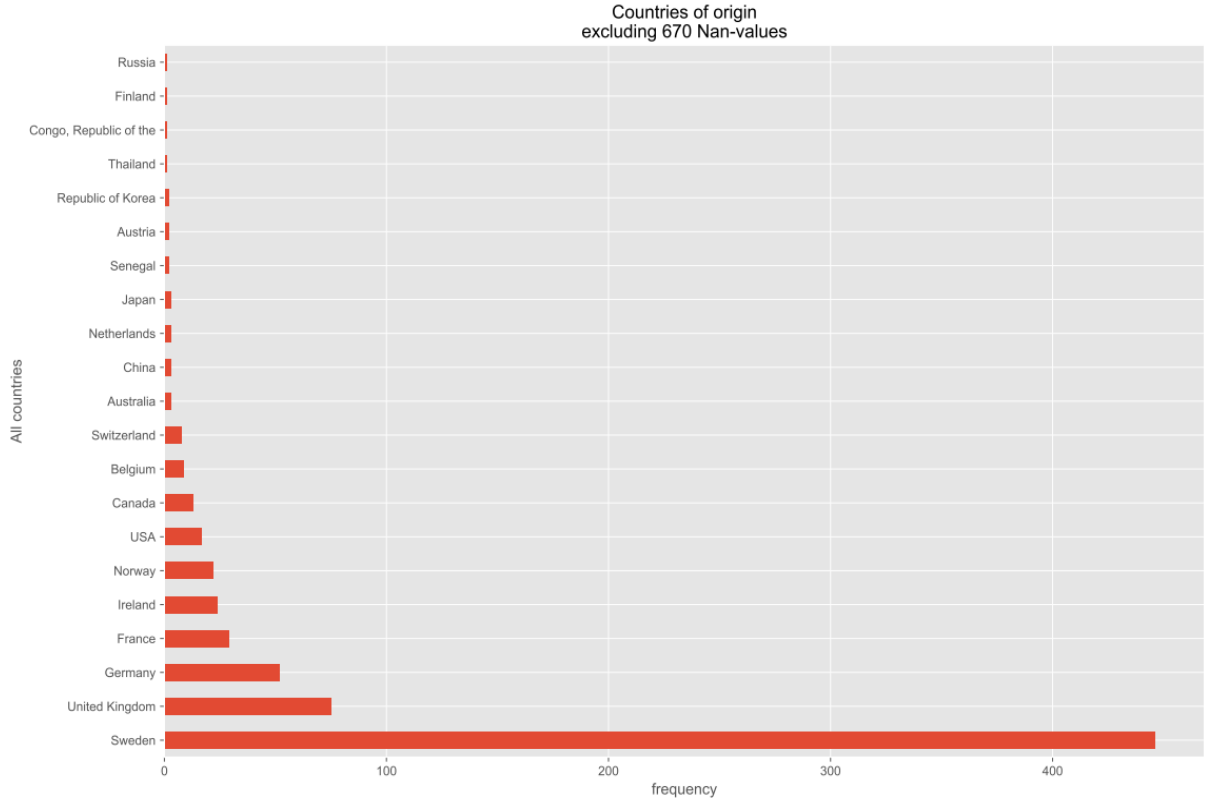
This results in the following plot:

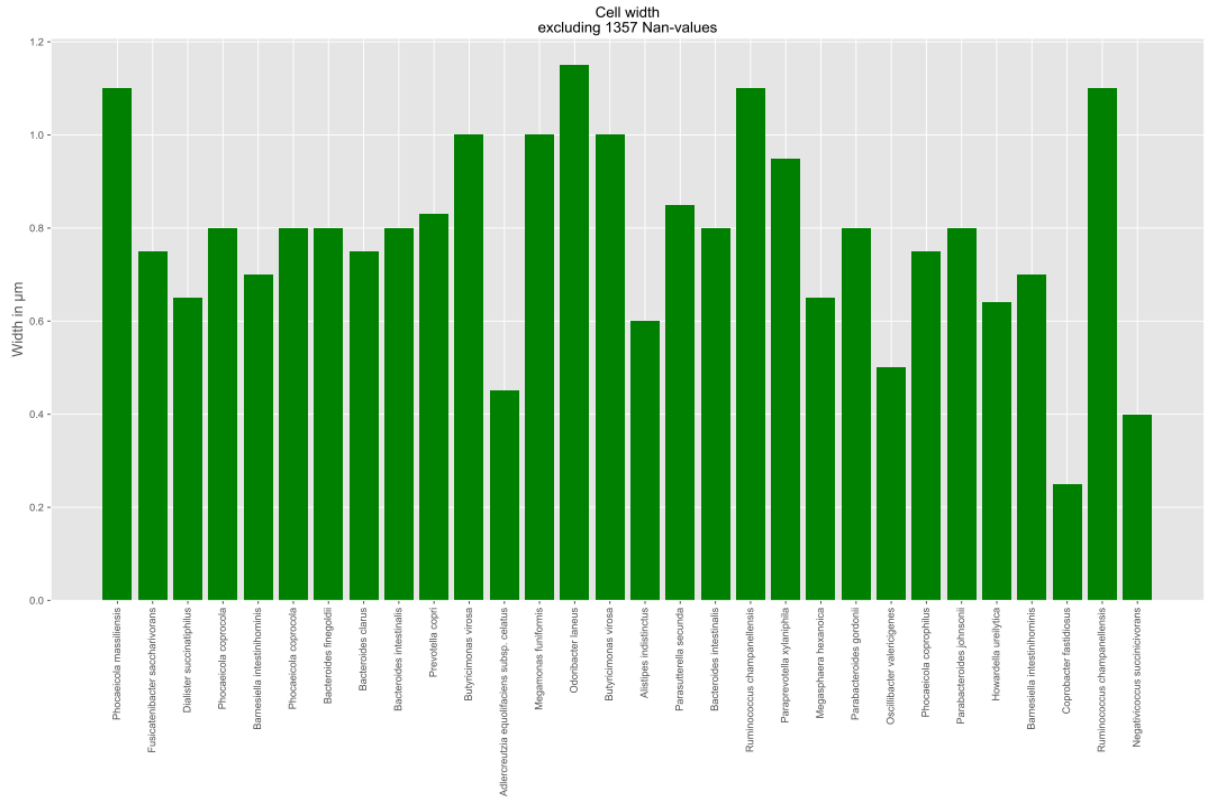
Let's make a boxplot which shows the optimal pH range for all the species in our dataset:

```
#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)
```

(continues on next page)

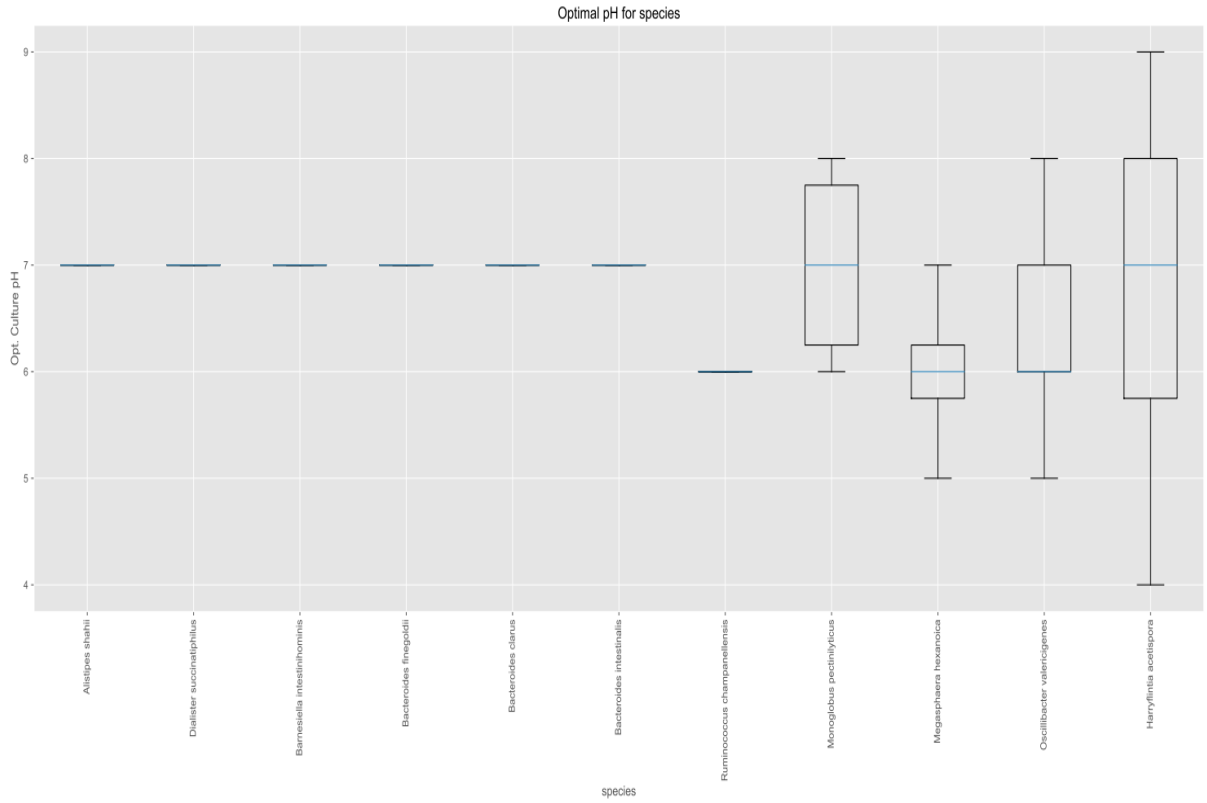

Countries of origin
excluding 670 Nan-values
All countries
Russia
Finland
Congo, Republic of the
Thailand
Republic of Korea
Austria
Senegal
Japan
Netherlands
China
Australia
Switzerland
Belgium
Canada
USA
Norway
Ireland
France
Germany
United Kingdom
Sweden
0
100
200
300
400
frequency


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 goldsnii
Oscillibacter valericigenes
Phocaeicola coprophilus
Parabacteroides johnsonii
Howardella ureilytica
Barnesiella intestinihominis
Coprobacter fastidiosus
Ruminococcus champanellensis
Negativicoccus succinicivorans

(continued from previous page)

```
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:

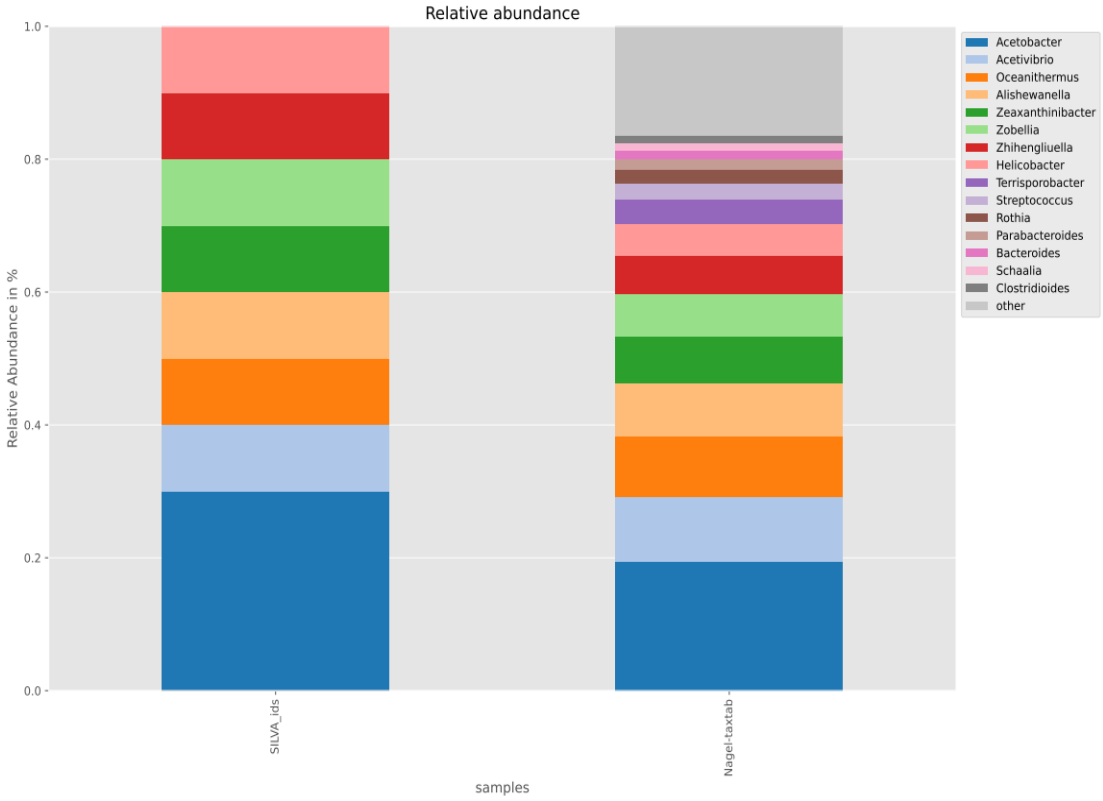
```
# Run for multiple inputs
resulting_list_with_all_res_dfs = bc.bacdive_call(input_lists={"./SILVA_ids.txt" : [
↪"input_via_file", "search_by_16S_seq_accession"], "./taxtab.tsv" : ["taxtable_input"]},
↪sample_names=["SILVA_ids", "Nagel_taxtab"])
#Relative abundance plot
vm.stacked_barplot_relative_abundance(resulting_list_with_all_res_dfs, 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: A Qiime2 plugin named q2-bacdiving has also been created which implements the function *bacdiving.bacdive_call()* and can be found here.

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 like NetCoMi! Here you can find out how to use BacDive data to enhance your networks with phylogenetic information.

INDEX

B