BioPype: A crash course in bioinformatics and custom pipelines

Ethan Gniot

May 2018

Todo list

Contents

1 Foreword

1.1 Goal

This tutorial aims to improve your general understanding of bioinformatics through several methods:

- Define technical terms commonly used in bioinformatics methods and found in the literature.

- Provide a collection of various useful resources, including...

  1. Resources for finding tools, data, and background information that can help answer your research questions.

  2. Resources that explain details about bioinformatics concepts and techniques in beginner-friendly language.

- Demonstrate how Python can be used to answer your research questions by combining existing bioinformatics tools and automating repetitive or time-consuming tasks.

1.2 How to use this book

The main text of this book is written in the right-hand margin. The left-hand margin contains special markers and important notes to the reader. Most reference materials are consolidated in the appendices at the end of the book. The book can be used as a self-paced tutorial with the help of the markers described below.

——————————————-

This is a margin label. I will write things here to further explain the main text, define jargon, etc.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas eu felis sodales, interdum purus nec, interdum ex. Integer at nunc ultricies, tempus nibh eget, egestas risus. Suspendisse aliquam, lorem at dictum accumsan, dolor elit euismod velit, a gravida risus libero non felis. Donec a tortor tempor, scelerisque sapien posuere, volutpat erat. Morbi in imperdiet velit. Vivamus sagittis, massa sit amet venenatis euismod, elit eros aliquam diam, molestie faucibus lectus nisi euismod erat. Nulla id dictum mi.

! → **The arrow that points to this line is an "Attention" marker. It will indicate key pieces of information that you should pay special attention to.** *Curabitur egestas aliquam nisl, pharetra finibus mauris placerat nec. Nam in diam risus. Nulla mollis purus quis feugiat tristique. Vestibulum et sollicitudin ante, at sagittis ipsum. Nunc hendrerit ante sed massa semper eleifend.*

→ This kind of annotation will reference appendix entries that you can consult for more-detailed information about the main text.

Ut ultrices eros velit, at faucibus ante rutrum eget. Pellentesque a molestie diam. Curabitur mattis dui a risus lacinia fringilla. Phasellus porttitor elit nec neque euismod, id ultrices elit lobortis. Aliquam molestie sem. Curabitur sit amet urna faucibus, vulputate arcu sit amet, fermentum ipsum. Nulla facilisi. Nam ullamcorper eget leo id malesuada.

1.3   Source Code

The source code for this manual, the accompanying tutorial, and the BioPype package can all be found at `github.com/EthanGniot/BioPype`.

1.4   Pre-requisites

Computer requirements

- At least 4GB RAM minimum.* 16GB RAM if possible (the more RAM, the better)
  - *At 4GB RAM, one step of the BioPype analysis takes $\tilde{2}$.5 hours to finish when analyzing 20 samples. With less RAM, expect the process to take longer. Conversely, more RAM will increase the speed of the analysis.
- BioPype was developed and tested using a MacBook Pro (Retina, 13-inch, Late 2013) running macOS 10.13.4 High Sierra. BioPype should be compatible with similar macOS versions, as well as Linux, though it has not yet been tested on Linux.
- Storage requirements will vary depending on the amount of data you intend to analyze. The analysis performed in this tutorial required at least 500GB and were satisfied by using a 2TB external hard drive.

2 Microbiome Analysis

The analysis used in this tutorial will focus on analysis of the human gut microbiome, but BioPype can be used to analyze the microbiomes of other communities as well.

2.1 The Gut Microbiome

The human gut is home to trillions of microbes. The collection of physical micro-organisms that live in the gut is referred to as the **gut microbiota**. The **gut microbiome** refers to all of the genes contained within these micro-organisms, as well as their functions and interactions.

Human microbiota can play host to many different kinds of microbes, including bacteria, archaea, and fungi. For the sake of this tutorial, we are going to focus on the role of bacteria in human microbiomes.

Gut dysbiosis: deviation from "normal" gut microbiota composition.

Sometimes the biodiversity of the gut is thrown out of balance. When the composition of the microbiota is significantly altered, and bacterial populations that normally compose a large percentage of the microbiota become underrepresented (or underrepresented populations become overrepresented), the gut is in a state of *dysbiosis*. **Gut dysbiosis** is important to study because it has been implicated in many physical and neurological human health conditions, including diabetes, rheumatoid arthritis, Alzheimer's, inflammatory bowel disease (IBD), colon cancer (**???**), chronic fatigue syndrome (**?**), obesity (**??**), bacterial vaginosis (**?**), ulcerative colitis (**??**), anxiety, and depression (**??**)

2.2 Relative Abundance Analysis

Relative abundance is a measure of how common or rare a taxon is, relative to other taxa in its community, and is usually expressed as a percentage. For more information on **relative abundance** analyses, see Appendix A.4.

Operational Taxonomic Unit (OTU): A group of sequences that share significant similarity based on some marker gene. Groups can be treated as a single taxon (e.g., genus, species, etc. depending on the clustering threshold) for analysis purposes. For more info, see Appendix A.6.

BioPype lets users examine the **relative abundance** of bacterial taxa in each experimental sample. Relative abundance analyses help visualize the structure of the micro-organism community. Users can use BioPype to analyze bacterial **amplicon** sequencing data to determine which micro-organisms are present and in what proportion. The idea is to take the sequences from a gut sample and assign them to a taxon. To do that, we group (or cluster) sequences based on their similarity to define **Operational Taxonomic Units (OTUs)**. Using the number of sequences assigned to each OTU/taxon, BioPype can determine the relative abundance of each micro-organism in a sample and plot the results on an interactive bar chart.

For example, studies have found that the genera *Bacteroides* and *Lactobacillus* are both prevalent in the healthy human gut microbiome, but *Bacteroides* has a higher relative abundance than *Lactobacillus* **?**. In other words, out of all the sequences obtained from a healthy human gut sample, a larger percentage of those sequences will be assigned to the *Bacteroides* OTU than to the *Lactobacillus* OTU during a relative abundance analysis.

Relative abundance is an important measure of biodiversity in the gut. Using BioPype to perform a relative abundance analysis allows examination of gut biodiversity at 7 different taxonomic levels, which helps identify instances of gut dysbiosis in research subjects.

2.3   Metagenomics

! → *THIS FUNCTIONALITY HAS NOT BEEN COMPLETED YET, BUT WILL BE ADDED IN FUTURE UPDATES*

BioPype can also be used to predict which genes are present in microbiome samples. From these predicted genes, users can do two things.

First, they can generalize the functions encoded across all bacteria that are present in an experimental condition's microbiome (i.e., at the metagenomic scale) and visualize the relative distribution of these functions with a bar chart. Having this functional snapshot of a metagenome is useful because a functional overview of the experimental condition can be compared to a functional overview of the control condition to see if there are differences in the two groups' functional capabilities. When comparing the microbiomes of disordered vs. non-disordered samples, having a visual depiction of the differences in functional capabilities can make it easier to find the cause of the disorder by narrowing down the potential sources of the condition.

For example, certain G protein-coupled receptors (GPCRs) in the gut are associated with some of the diseases associated with gut dysbiosis, such as inflammatory bowel disease (IBD) (**?**). Research has also shown that bacteria interact with their environment via the release of small molecules (**?**). From these premises, one could hypothesize that IBD may be caused by the over-production of certain effector molecules from the gut microbiome. This hypothesis may be tested by using BioPype to compare the functional capabilities of the IBD microbiome with that of a healthy control microbiome. If the results show, for example, that the IBD gut metagenome has a greater proportion of genes associated with lipid production than the control metagenome, it indicates potential support for the user's hypothesis. They can now refine their investigation by narrowing the scope of their search to only analyze genes associated with lipid synthesis.

Second, users can use BioPype to predict the functions of *individual* genes. This is useful because it allows users to identify a specific gene of interest for their research question, which they can then design a validation experiment for.

Continuing with our above example hypothesis about the gut microbiome's role in IBD, the user can now turn their focus to the functions of specific lipid-synthesis genes that are being overrepresented in IBD patients' microbiomes. Upon examining the functions of individual genes, they find that N-acyl amide synthase genes are enriched in the IBD microbiome. Wet-lab validation experiments will confirm that the lipids encoded by these genes are able to interact with GPCRs that regulate GI tract physiology (**?**). This suggests that these lipids, and their associated genes in the microbiome, have the potential to cause IBD during times of dysbiosis by activating GPCRs in the gut (**?**), thus supporting the user's original hypothesis. Additionally, one can use these findings to focus future research questions about the role

of the human microbiome in health disorders by searching for correlations between N-acyl amide synthase genes and other conditions. For instance, since the products of these genes interact with GPCRs to potentially regulate IBD, and IBD is associated with anxiety and depression, investigation may reveal that the interaction between these genes and GPCRs potentially regulate anxiety and depression as well.

2.4 Python

To make use of this tutorial and the BioPype package, you will need to be familiar with the basics of the Python coding language. The free DataCamp course *Intro to Python for Data Science* (`https://www.datacamp.com/courses/intro-to-python-for-data-science?tap_a=5644-dce66f&tap_s=116411-750171`) is a quick and easy introduction to the basics of the language. It covers:

- Variables and variable assignment
- Basic data types (strings, integers, floats, lists, booleans,) and converting between types
- Calculations with variables
- Functions and arguments
- The `help()` function.
- Methods
- Packages and importing packages
- The basics of the NumPy package (The material covered in the NumPy chapter of the DataCamp tutorial is not required for using BioPype, but if you are new to Python, it's a great way to practice/apply the concepts they teach you in Chapters 1-3.)

The DataCamp course should take ~4 hours to complete (less, if you forego NumPy chapter), and will teach you everything you'll need to know about Python in order to make use of BioPype.

If you decide that you'd like to start creating your own tools with Python, there's a bit more you'll need to know. Resources for learning how to create your own functions, classes, packages, and more can be found in Appendix B.

2.5 BioPype Tutorial Dataset

The dataset we will analyze in this tutorial comes from the *Longitudinal Multi'omics of the Human Microbiome in Inflammatory Bowel Disease* study (unpublished) (see `https://trace.ncbi.nlm.nih.gov/Traces/sra/?study=SRP115494`).

From the abstract: "The main Inflammatory Bowel Disease (IBD) Multi'omics Database (IBDMDB) study includes multi'omics measurements from over 100 subjects, sampled biweekly over up to a year in both adult and pediatric patients with IBD (Crohn's disease and ulcerative colitis), along with non-IBD controls. Data types include fecal metagenomes, metatranscriptomes,

metabolomes, and proteomes, as well as host genetics, intestinal biopsy transcriptomes, epigenetics, and 16S amplicon profiles. Subjects' medical histories and demographics are collected at baseline and medication, diet, and disease activity profiled longitudinally."

In our analysis, we will be downloading sequencing data from their NCBI Sequence Read Archive page that corresponds to patients with ulcerative colitis and Crohn's disease (both conditions fall under the umbrella condition of "inflammatory bowel disease". We will examine whether there are compositional differences between the gut microbiota of younger patients with ulcerative colitis or Crohn's disease compared with their older cohort. While we expect both groups to exhibit gut dysbiosis and clear deviation from the composition of a healthy human gut microbiome, what we're hoping to study is whether or not increased age exacerbates the severity of the dysbiosis. Increased age is associated with gut dysbiosis, even in otherwise healthy individuals, so we predict that older IBD patients will exhibit greater degrees of dysbiosis than their younger counterparts.

3 Software and Set-up

3.1 BioPype Dependencies Information

The following tables contain version information and other resources for the packages that BioPype needs in order to function.

> The command line install for trim_galore is failing for some reason. Get to the bottom of this and update the manual.

Table 3.1: **Python packages that BioPype requires in order to function.**

Package name	Version number	Command-line install
Anaconda	5.1	
fastqc	0.11.5	`conda install -c bioconda fastqc`
multiqc	1.5	`conda install -c bioconda multiqc`
pandas	0.22.0	`conda install pandas`
parallel-fastq-dump	0.6.3	`conda install -c bioconda parallel-fastq-dump`
qiime2	2018.4	(see `https://docs.qiime2.org/2018.4/install/native/` if the install instructions below fail)
sra-tools	2.8.2	`conda install -c bioconda sra-tools`
trim_galore	0.4.5	`conda install -c bioconda trim_galore`

Table 3.2: **Links to the documentation pages for BioPype's dependencies.**

Software name	Documentation link
Anaconda	`https://docs.anaconda.com/anaconda/`
	`https://conda.io/docs/user-guide/getting-started.html`
fastqc	`https://www.bioinformatics.babraham.ac.uk/projects/fastqc/`
multiqc	`http://multiqc.info/`
pandas	`https://pandas.pydata.org/pandas-docs/stable/install.html`
parallel-fastq-dump	`https://github.com/rvalieris/parallel-fastq-dump`
qiime2	`https://docs.qiime2.org/2018.4/`
sra-tools	`https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=toolkit_doc`
trim_galore	`https://github.com/FelixKrueger/TrimGalore`

3.2 Set-up and Install Dependencies

Before we write any code, there are several steps that must be completed to prep your machine for the tasks we will be performing in this tutorial. Without these prerequisites, the code you write during this tutorial will not work correctly, and you won't be able to make use of BioPype:

1. Install and open Anaconda

2. Create a new virtual environment

3. Install packages

3.2.1   Install Anaconda or Miniconda

As you can see in Table 3.1, there are several packages that BioPype needs in order to function. Manually downloading and installing packages is an incredible pain, because the quirks of one package frequently conflict with quirks of another. To download the dependencies we need, we will use what's called a **package manager**. Essentially, package managers make it easy to install lots of packages because they can (usually) handle all of the minute details of the process, adjusting the versions and dependencies of each package to resolve any conflicts that would normally wreck the installation.

The package manager that we will be using is called *conda*. There are two ways to obtain conda: the Anaconda distribution, and the Miniconda distribution.

Anaconda is a set of about a hundred packages (many of which are useful scientific computing tools) that also comes bundled with a version of Python. One of these packages is the **conda** package (yes, conda is both a package and a package manager). One of the benefits of Anaconda is that it comes with a graphical interface for downloading and managing new packages called the Anaconda Navigator (see Figures 3.3 and 3.4). Conda can also be used to manage packages from a command-line interface (on a Mac, this is the Terminal application). However, the fact that Anaconda comes with over 100 packages pre-installed can sometimes cause problems when installing new packages. For that reason, you may choose to download Miniconda instead.

Miniconda is a smaller alternative to Anaconda that is *just* conda and its dependencies (as opposed to Anaconda, which is conda and a bunch of other packages). This means that Miniconda doesn't come with a graphical interface like the Anaconda Navigator, so conda must be operated from the command-line. However, it also means that Miniconda requires less storage on your computer (400 MB vs 3 GB) and also avoids some of the difficulties that come from Anaconda's pre-installed packages.

We will cover two methods by which you can download the dependencies for BioPype using conda, one method uses the command-line with a Miniconda installation (faster, command-line only), and the other uses the Anaconda Navigator with an Anaconda installation (slower, standard graphical interface).

! →
- For the record, the command-line method should theoretically work just as well with an Anaconda installation as it does with a Miniconda installation, but I experienced unknown technical difficulties when trying to download all the dependencies with an Anaconda installation. Due to the unique file-state of individual machines, your mileage may vary.

3.2.2 Installing dependencies with Miniconda

1. Go to `https://conda.io/docs/user-guide/install/index.html#regular-installation`, select your operating system, then follow the instructions to install Miniconda.

2. Once the installation is finished, go to the BioPype Github repository at `https://github.com/EthanGniot/BioPype`. It should bring you to a page that looks similar to Figure 3.1. (There may be different files listed depending on the current version of BioPype)
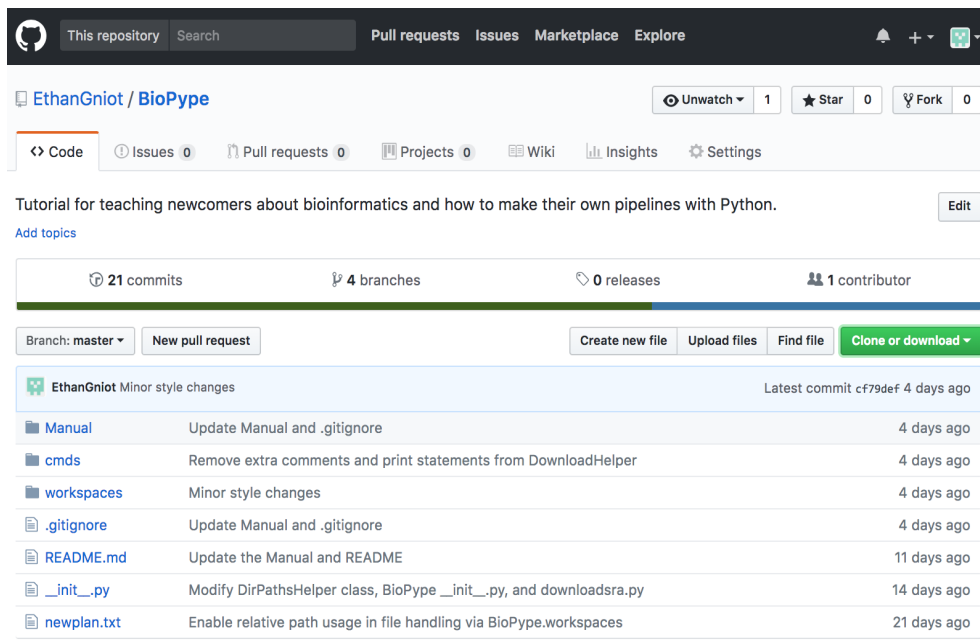

Figure 3.1: The BioPype Github webpage.

3. Click on the "biopype-dependencies.txt" file to view its contents. Next, right-click on the "Raw" button situated to the top-right of the text field and select "Save Link As...". Save the file to your Home folder (for most users, this should be the folder that shares your username).

4. Open your computer's command-line application (on a Mac, this is the Terminal application). In the Terminal prompt, type the following and then press "Enter" (Note: the "$" symbol represents the terminal prompt where you type. Do not actually type the $):

   $ pwd

   The next line in the Terminal should then display a file path that represents your "current working directory". The current working directory is essentially the "folder" that your Terminal is currently looking at. It can see all the files and folders in that directory and access them by name if you ask it to. For most users, this file path

→ For more information on navigating your files via the command-line, see Appendix A.7.

11

should lead to your home directory (e.g., /Users/your-username). If the file path does not lead to your home directory, please read Appendix A.7 to learn how to change your current working directory to your home directory.

5. Enter the following command in the Terminal prompt and then press "Enter":

```
$ ls
```

The Terminal should now display a list of all the files and folders in your current working directory. If you can't see the "biopype-dependencies.txt" file in the list, double-check that you downloaded it to your home directory. If you can see the file in the list, proceed to the next step.

6. To install all of the dependencies needed to run BioPype, enter the following into the command prompt:

```
$ conda create −−name biopype −−file biopype−dependencies.txt
```

The Terminal will take a while to think after you press "Enter" as it prepares to download all of the necessary packages. The different parts of this command accomplished several things:

(a) `conda`: this tells the Terminal that we want to execute one of conda's functions.

(b) `create`: the conda function we want to use is the "create" function, which creates a new conda **virtual environment**.

→ For information on **virtual environments**, what they are, and why they're useful, see Appendix A.8.

(c) `--name`: this tells the Terminal that we are about to tell it what to name the new virtual environment.

(d) `biopype`: this is the name of the new virtual environment.

(e) `--file`: this tells the Terminal that we are about to tell it the name of a file containing dependencies that we want it to use when creating the new environment.

(f) `biopype-dependencies.txt`: this is the name of the file containing the dependencies.

After the terminal finishes thinking you will see the normal empty command prompt again. You should now have all of BioPype's dependencies installed in the "biopype" virtual environment.

3.2.3   Installing dependencies with Anaconda

This method is longer than the Miniconda method, so it will be broken up into three sections: 1) Installing and opening Anaconda, 2) creating a new virtual environment, and 3) installing dependencies.

Installing and Opening Anaconda

1. Go to the download page for the Anaconda distribution at `https://www.anaconda.com/download`.

2. Select your preferred operating system from the Windows, macOS, or Linux tabs, then select the Download option for the **Python 3.6 version** (Figure 3.2) and follow the installation instructions.
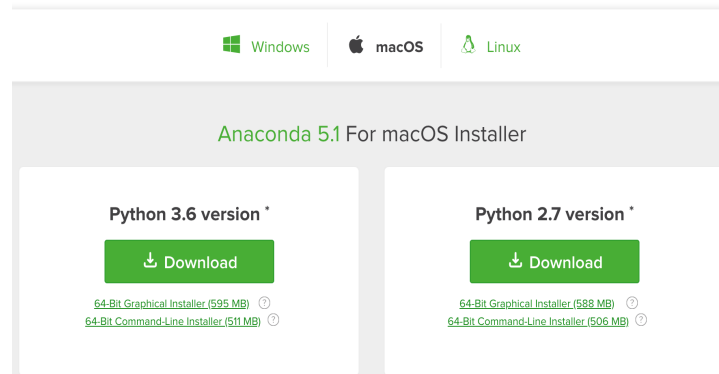

Figure 3.2: The Anaconda download options provided on the Anaconda distribution website at `https://www.anaconda.com/download`.

3. After installation is complete, open the application named "Anaconda-Navigator" (the icon looks like ⟳). After a brief start-up period, you should see the following window (Figure 3.3):
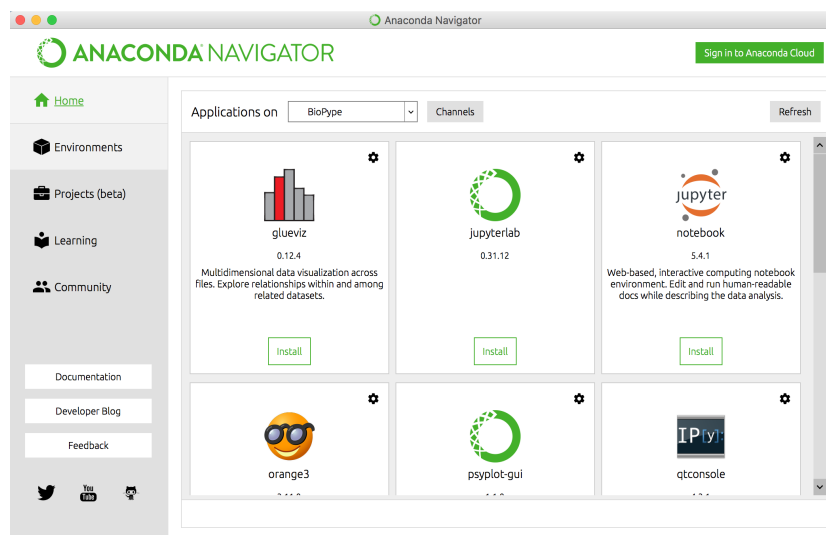

Figure 3.3: The window displayed to the user upon opening Anaconda-Navigator.

Create a New Virtual Environment

Link to resource for further reading on virtual environments

Make sure the computer has an internet connection while completing this section, otherwise Anaconda will not let you create a virtual

1. On the left side of the Anaconda-Navigator window, click on the tab labeled **Environments**. (Figure 3.4)
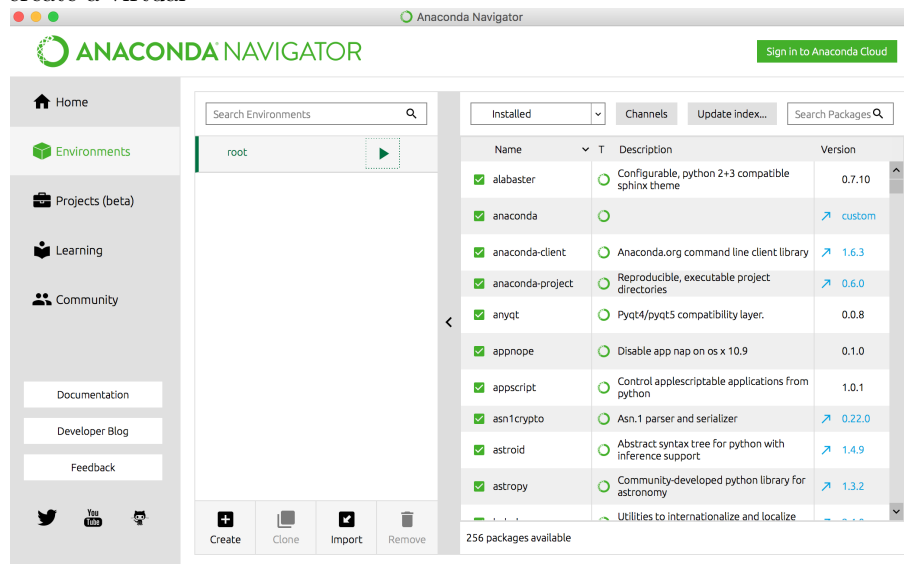

Figure 3.4: The Environments window of the Anaconda-Navigator.

2. Click the **Create** button on the bottom of the center panel. A new window titled "Create new environment" will appear. (Figure 3.5)

3. Enter a **Name** for the environment. You may choose any name you want, but for the sake of this tutorial we will name the new environment "biopype".

4. Select the box labeled **Python** next to the **Packages** heading.

5. Choose a version of Python from the adjacent drop-down menu (Python 3.6 is the most current version at the time of this writing, but the packages we use require Python 3.5 so we chose **3.5**. If you are following the tutorial analysis in this manual, choose version 3.5).

6. Click the **Create** button within the "Create new environment window".

Install packages

1. Change Anaconda's current environment from the **root** environment by selecting the **biopype** tab in the middle panel of the Environments window.

2. Click on the drop-down menu in the right-hand panel that says "Installed" and change it to "All".

3. In the "Search Packages" box we can search for the packages that BioPype needs in order to function. For instance, if BioPype depended on the "biopython" package, we could enter "biopython" into the box.
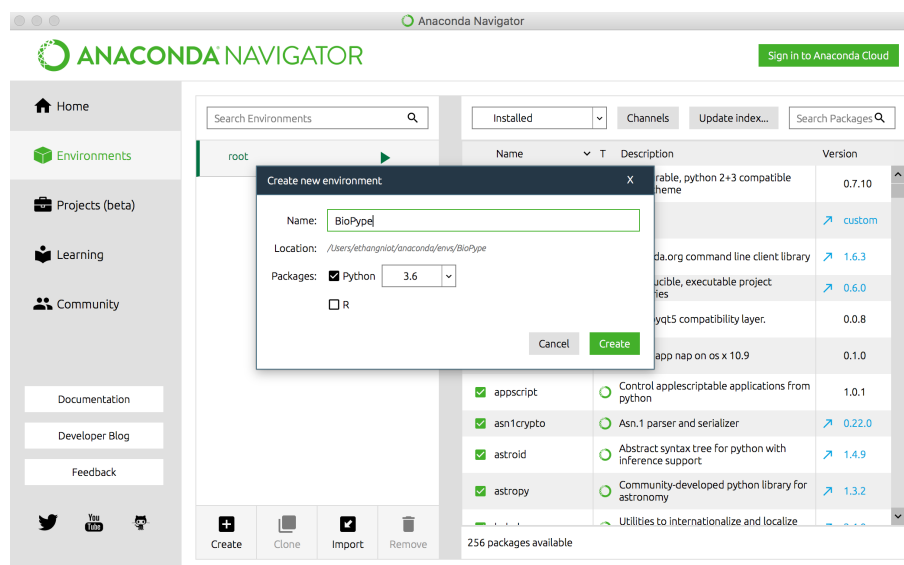
Figure 3.5: The "Create new environment" window.

The search should then return a package named "biopython". We would then select the checkbox to the left of the name. (Figure 3.6)

- A pair of green and red boxes (reading "Apply" and "Clear", respectively) will appear in the bottom-right of the window once a package is selected. Do not click these just yet.

4. Use the search bar to find and select the packages listed in Table 3.1 (except qiime2. That will need to be added to the environment by obtaining the .yml file from the qiime2 installation link and using the `conda env install --name <env-name> --file qiime2-2018.4-py35-osx-conda.yml` command). Once all packages have been selected, click the green "Apply" button in the bottom right corner of the window, then select "Apply" again within the "Install Packages" window that appears. (Figure 3.7) Anaconda will now install the selected packages.

- Note: it may take a while for the windows to fully load when selecting "Apply" and "Install Packages", due to so many being selected at a time.

3.2.4   Downloading BioPype

With the dependencies installed, we now need to download BioPype.

1. Go back to the BioPype Github repository at `https://github.com/EthanGniot/BioPype`. (See Figure 3.1).

2. Click on the green "Clone or download" button (see Figure 3.1)

- Note: The Github page that the URL from step 1 brings you to contains the code for the most recent "master" update to the BioPype
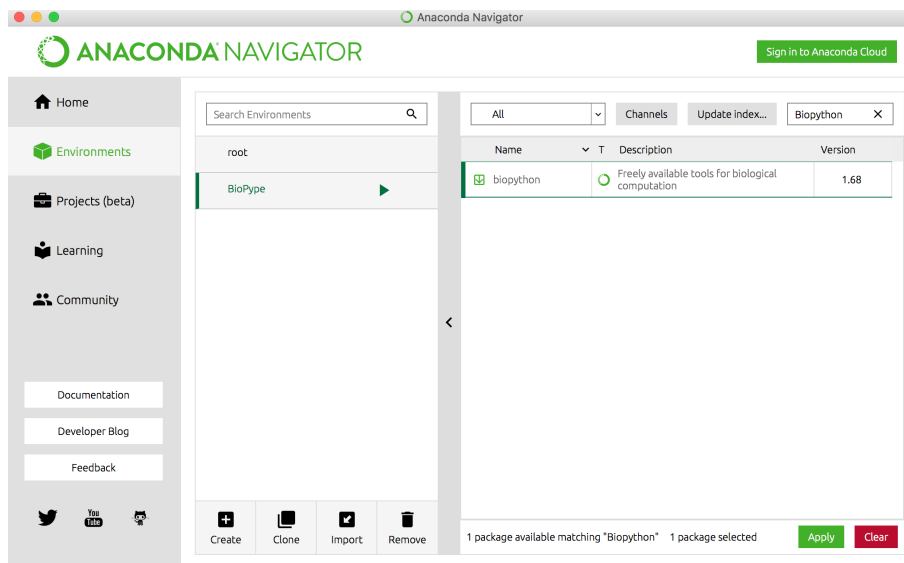
15

Figure 3.6: Searching for a package. When a package is selected, the checkbox next to the package's name will be green.
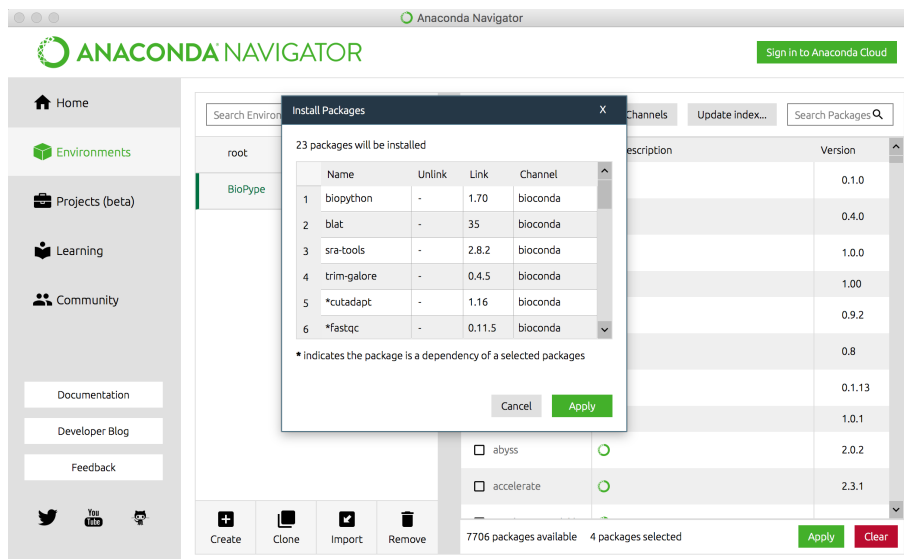

Figure 3.7: The window displaying the packages and dependencies that will be installed.

repository. The "master" updates are usually the most recent *stable* versions of BioPype. However, there may also be other versions of BioPype that are newer, but may contain bugs due to being under active development. To access these versions, click on the "branches" link (in Figure 3.1, it is the link that says "4 branches" next to a branching symbol). If there are any branches (i.e., "versions") of BioPype that are more-current than the "master" branch, you will be able to see them on this page. This page also has links to the branches, where you can download that version of the BioPype code.

3. Click on the "Download ZIP" button (the blue button in Figure 3.8).
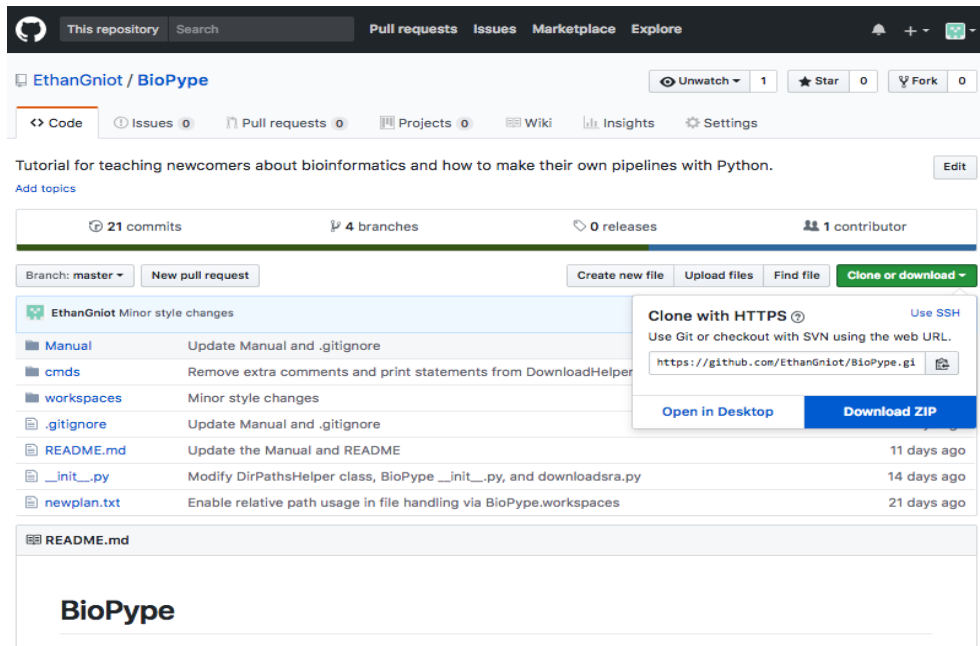

Figure 3.8: The button to download the BioPype code repository as a .zip file.

4. Navigate to your Downloads folder and unzip the BioPype .zip file.

5. Copy the BioPype folder, then navigate to the following file path and paste the folder into the "site-packages" folder (note: the parts of the file path that are in parentheses are not the literal names of directories. Substitute the appropriate directory into those spots based on the name of your home folder and whether you chose Anaconda or Miniconda):
(home-folder)/(Anaconda-or-Miniconda)/envs/biopype/lib/python3.5/site-packages

3.2.5   The BioPype Project Directory and the SRA Toolkit Workspace Location

To perform a microbiome analysis with BioPype, we will need a location where we can save our data and files.

The **SRA database** is a place where researchers can upload the sequencing data they use in their experiments for public access. Along with the raw sequence data, they also provide information about the experimental methods they used in the study that the sequencing data come from.

1. Create a directory somewhere in your file system and name it "biopype_project". (Remember that microbiome sequencing files are very large, so make sure to create the project folder somewhere with a lot of free storage space.)

   One of the packages BioPype uses is the SRA Toolkit (the package name in your file system will be "sra-tools"). This package lets us download .sra files from the NCBI **Sequence Read Archive (SRA) database**. In order to use the package, it requires that a directory called the "Workspace Location" be configured first.

2. Create a folder inside the `biopype_project` directory called "data".

3. In the Terminal, navigate to the "bin" subdirectory of the sra-tools package using the `cd <path-to-subdir>` Terminal command.

   - The bin subdirectory will be *very* deep in your file system. The following file path is where the subdirectory is located on my machine:
     /Users/ethangniot/miniconda3/envs/biopype_project/share/ncbi/sra-tools/mac/clang/x86_64/rel/bin

4. In your web browser, go to `https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=toolkit_doc&f=std#s-4` and follow the configuration instructions to change the Workspace Location to the "data" folder you created in the `biopype_project` folder.

5. In the Terminal, navigate back to the biopype_project folder.

6. Activate the "biopype" virtual environment by typing the following into the command prompt:

   ```
   $ source activate biopype
   ```

7. Activate the Python interpreter by typing the following into the command prompt:

   ```
   $ python3
   ```

8. Import BioPype into the current Python environment:

   ```
   >>> import BioPype
   ```

9. Upon importing BioPype, you should see a prompt appear in the Terminal window telling you what the current working directory for BioPype is, as well as the current Workspace Location for the SRA Toolkit. To change the location of where BioPype thinks these directories are, enter the following command:

   ```
   >>>BioPype.path_helper.setup('biopype')
   ```

10. The Terminal will now ask you to "input a path to an existing folder to define BioPype's working directory". Input the file path to the "biopype_project" folder. (e.g., /Users/username/biopype_project)

11. Once the previous command is finished running, execute the following:

    ```
    >>>BioPype.path_helper.setup('sra')
    ```

A prompt should appear that asks you to "input the path for the SRA Toolkit's configured Workspace Location." Input the path to the "biopype_project/data" directory that you configured as the Workspace Location in step 4 of this section.

The BioPype package should now be fully configured and ready to use for microbiome analyses!

4 The Dataset

> ⓘ **Recap**
>
> In the previous chapter, we set up our machine so that it has all of the software BioPype needs in order to function.
>
> In this chapter, we will use BioPype to **download** experimental data, and then prepare them for analysis via a process called **"quality control"**. We will also discuss how the BioPype functions used in this workflow were created.

As described in Chapter **??**, we want to investigate if there are any differences in the gut microbiomes of younger patients with IBD compared to older patients with IBD. Using the methods described in Chapter **??**, we found a study in the SRA database with sequencing data that are useful to us. The webpage for the SRA Study is reproduced in Figure 4.1 and can be accessed at `https://trace.ncbi.nlm.nih.gov/Traces/sra/?study=SRP115494` .

Figure 4.1: The "SRA Study" webpage.

The dataset is reproduced in Figure 4.2 and can be found by clicking on "Runs" under "Related SRA data" on the right side of the SRA Study webpage, or by going to `https://trace.ncbi.nlm.nih.gov/Traces/study/?acc=SRP115494#` .
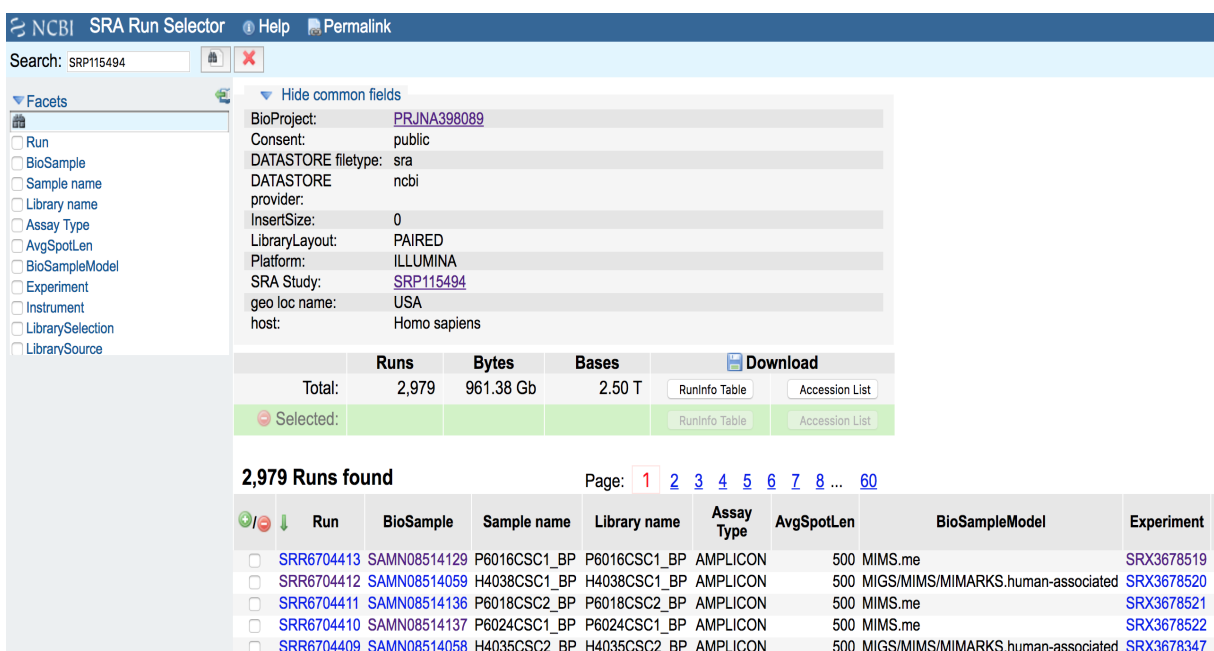

Figure 4.2: The sequencing data collected by the *Longitudinal Multi'omics of the Human Microbiome in Inflammatory Bowel Disease* study. (Note: the table's columns extend beyond the right side of the page because there are 36 metadata categories.)

To learn more about how to use these pages to find information about the study and its methods, please refer to Chapter **??**.

4.1   Download and Quality Control Dataset

Note: The dataset has a metadata category called "BioSample." This is not to be confused with the generic term "samples." In this case, one sample is equivalent to one run (i.e., one row) of the data table.

1. First, we must decide what criteria we will use to choose our experimental samples.
   Remember: our research question investigates the effects of age on the gut microbiomes of inflammatory bowel disease patients. Looking at the study abstract from Figure 4.1, we can see that the researchers investigated two types of IBD: Crohn's disease and ulcerative colitis. So one of the metadata categories from the dataset (i.e., the columns from the dataset in Figure 4.2) that we will use to select our samples is the "host_disease" category. We will also select our samples based on the "host_age" metadata. Finally, our relative abundance analysis will require amplicon sequencing data, so the third metadata category we will sort by is the "Assay_Type" column.

2. Now we want to select our samples based on these categories.

Download the RunInfo Table file found on the study's sample page and put it in the project folder you set up in Chapter 3.1. The RunInfo table is just the browser's sample table in a .txt file format.

3. Open the command line interface for your machine's operating system. On a mac, this is the Terminal application.

4. Navigate to your project's folder and then confirm that you are in the right location by using the the following commands at the command prompt:

```
$ cd <path_to_your_project>
$ pwd
```

5. Activate the Python interpreter by typing the following into the command prompt:

```
$ python3
```

6. Import BioPype to the current Python environment:

>>> **import** BioPype

! →
  • If not done already, stop and follow the instructions in Chapter 3 to configure the BioPype working directory and the SRA Toolkit Workspace Location.

7. Import `BioPype.cmds.runtable` to gain access to the `RunTable` class:

>>> **import** BioPype.cmds.runtable as rt

8. Import the rest of the Python packages that we will need for the analysis:

>>>**import** os
>>>**import** glob
>>>**import** pandas as pd
>>>**from** BioPype.cmds.downloadsra **import** DownloadHelper

Using "as" in an import statement lets you reference the full name of the imported package by some shorter name. It is generally considered better practice to **not** import packages "as" some other name, because it makes the code less explicit (which goes against the purpose of Python), but we do it in this manual for the sake of space.

9. Create a `RunTable` object called "my_table" out of the RunInfo Table .txt file:

>>> my_table = rt.RunTable('runinfotable_filename.txt')

  • With the `my_table` object created, you can remind yourself what the metadata categories are by using the following command:

```
>>> my_table.column_names
['Assay_Type', 'AvgSpotLen', 'BioSample', 'BioSampleModel',
'Experiment', 'Instrument', 'LibrarySelection', 'LibrarySource',
'Library_Name', 'LoadDate', 'MBases', 'MBytes', 'Organism',
'ReleaseDate', 'Run', 'SRA_Sample', 'Sample_Name',
'collection_date', 'env_biome', 'env_feature', 'env_material',
'host_age', 'host_disease', 'host_sex', 'host_subject_id',
'lat_lon', 'BioProject', 'Consent', 'DATASTORE_filetype',
'DATASTORE_provider', 'InsertSize', 'LibraryLayout', 'Platform',
'SRA_Study', 'geo_loc_name', 'host']
```

10. We want to first group the samples based on whether they contain amplicon sequencing data instead of whole genome sequencing data...

```
>>>amp_samples = my_table.filter_data("Assay_Type", '==', 'AMPLICON')
```

11. We then want to group the samples based on the patients' condition...

```
>>>uc_samples=amp_samples.filter_data("host_disease", '==', 'UC')
>>>cd_samples=amp_samples.filter_data("host_disease", '==', 'CD')
```

12. ... then we want to group them based on age:

```
>>>uc_young = uc_samples.filter_data("host_age", '<=', 21)
>>>uc_old = uc_samples.filter_data("host_age", '>=', 40)
>>>cd_young = cd_samples.filter_data("host_age", '<=', 21)
>>>cd_old = cd_samples.filter_data("host_age", '>=', 40)
```

! →
- Note that the third argument passed to the `filter_data()` method is **not** a string like the previous two arguments. It is simply an integer (no quotation marks around it).

13. Now that we have groups of samples selected, we want to get a list of SRA accession numbers for each group. The accession numbers are what we will use to specify the files we want to download from the SRA database.

```
>>>uc_young_nums = uc_young.get_accession_numbers()
>>>uc_old_nums = uc_old.get_accession_numbers()
>>>cd_young_nums = cd_young.get_accession_numbers()
>>>cd_old_nums = cd_old.get_accession_numbers()
```

14. To handle the process of downloading the .sra files from the SRA database, we instantiate 'DownloadHelper' objects:

```
>>>uc_young_obj = DownloadHelper(uc_young_nums, 5, threads=4)
>>>uc_old_obj = DownloadHelper(uc_old_nums, 5, threads=4)
>>>cd_young_obj = DownloadHelper(cd_young_nums, 5, threads=4)
>>>cd_old_obj = DownloadHelper(cd_old_nums, 5, threads=4)
```

Three arguments are passed to the objects when they're created:

(a) A list of accession numbers

(b) The number of samples we want to randomly select from the list of accession numbers. To account for hardware limitations that would severely bottleneck the analysis at file-handling steps like these, we will randomly select a subset of only 5 accession numbers

→ To learn more about what threads are, and how many you should use, see Appendix A.11.

(c) The number of threads we want our computer to use while handling the download and conversion of .sra files to .fastq files.

- The more threads your computer can divide this process between, the better (it's more complicated than that, but for now let's go with it). The question is: how many threads is your computer capable of using? For that, you'll have to do some Googling. You can find the resources I used to figure out how many threads my MacBook Pro can run (4) in Appendix A.12. To simplify: my computer has 1 CPU, the CPU has 2 cores, and each core can run 2 threads. 1 x 2 x 2 = 4 threads.

→ To learn more about .sra file format, see Appendix **??**.

→ To learn more about .fastq format, see Appendix A.10.

The .sra files will not be downloaded to the correct location if the SRA Toolkit Workspace Location has not been properly configured (see Chapter 3 for details)

15. We can now download the SRA files and convert them to FASTQ format:

```
>>>uc_young_obj.download_paired_end_data('uc_young', 'uc_young')
>>>uc_old_obj.download_paired_end_data('uc_old', 'uc_old')
>>>cd_young_obj.download_paired_end_data('cd_young', 'cd_young')
>>>cd_old_obj.download_paired_end_data('cd_old', 'cd_old')
```

Here, we are using the `.download_paired_end_data()` class method, which takes two arguments:

(a) The name of the folder which will contain the downloaded .sra files for an experimental group, and that will be saved inside the "data.grouped_sra" directory in the SRA Workspace (e.g., we want the .sra files for the "Crohn's-Disease-Younger" experimental group to be saved to the "biopype_project/data/grouped_sra/cd_young" directory).

(b) The name of the folder which will contain the downloaded .fastq files for an experimental group, and that will be saved inside the "data.grouped_fastq" directory in the SRA Workspace (e.g., we want the .fastq files for the "Crohn's-Disease-Younger" experimental group to be saved to the "biopype_project/data/grouped_fastq/cd_young" directory).

After this last step, we will have downloaded all of our experimental samples and converted them from .sra files to .fastq files! Next, we will perform quality control on the dataset, then perform the relative abundance analysis.

Appendix A Web Resources and Explanations

A.1 QIIME Illumina Overview Tutorial

`http://nbviewer.jupyter.org/github/biocore/qiime/blob/1.9.1/examples/ipynb/illumina_overview_tutorial.ipynb`

A.2 QIIME 2 Moving Pictures Tutorial

`https://docs.qiime2.org/2018.2/tutorials/moving-pictures/`

A.3 Microbiota vs Microbiome

"What is the gut microbiota? What is the human microbiome?" from medicalnewstoday.com. `https://www.medicalnewstoday.com/articles/307998.php`

A.4 Relative- vs Differential-Abundance Analyses

(For more in-depth discussion of differential abundance analyses, see (?).)

Differential abundance analysis: seeing which samples are most similar/dissimilar based on the raw sequence counts for each sample (E.g., 5 sequences associated with the *Lactobacillus* OTU were found in Sample A, and 10 sequences were found in Sample B). Differential abundance analysis answers the following questions:

- Is Sample A more similar to Sample B, or to Sample C in terms of its microbiome composition? (has an associated p-value)

- Does sequence variant/OTU/taxon/bacterium "X" appear more/less often in Subject A than in Subject B? (raw count)

Relative abundance analysis: seeing which samples are most similar/dissimilar based on the proportion of sequence counts for each sample (E.g., out of all the sequences obtained from Sample A, %50 were associated with the *Lactobacillus* OTU. Out of all the sequences obtained from Sample B, %100 were associated with the *Lactobacillus* OTU. Relative abundance analysis answers the following question:

- Does Sample A have higher/lower percentage of Bacteria X in its microbiome than Sample B?

A.5 Amplicon vs WGS data

`http://galaxyproject.github.io/training-material/topics/metagenomics/tutorials/general-tutorial/tutorial.html#amplicon-data`

A.6 Operational Taxonomic Units (OTUs)

`https://www.drive5.com/usearch/manual/otu_definition.html`

A.7 Navigating Files and Folders Using the Command Line

`https://www.macworld.com/article/2042378/master-the-command-line-navigating-files-and-folders.html`

A.8 Virtual Environments

`https://realpython.com/python-virtual-environments-a-primer/`

A.9 Explanation of @staticmethod Decorator vs @classmethod Decorator in Python

The Basics: Static methods make code easier to read and let you use a class' methods without needing to have an object of that class first. This is useful when it makes logical sense to place a function within a class (because the function is related to the other tasks that the class handles), but the function doesn't *need* to operate on an object/the data of that class. Normal methods are called by typing: `my_object.method`. Static methods are called by typing: `MyClass().method` or `MyClass.method`.

Basic explanation with slight background on class methods:
`https://julien.danjou.info/guide-python-static-class-abstract-methods/`

More technical explanations:
`https://stackoverflow.com/questions/12179271/meaning-of-classmethod-and-staticmethod-for-beginner`
This Stack Overflow question has some basic, easy to understand explanations mixed in with more in-depth explanations. The top-voted answer isn't the only one that is useful; each of the answers presents their explanation with a different degree of simplicity. Make sure to check several answers if the top ones don't seem helpful.

A.10 FASTQ Format

`https://galaxyproject.org/tutorials/ngs/`

This link contains:

1. An explanation of the FASTQ file format

2. An explanation of PHRED quality scores with an accompanying figure (Fig 4)

3. The following quote: "Fastq format is not strictly defined and its variations will always cause headache for you. See `https://www.ncbi.nlm.nih.gov/books/NBK242622/` for more information."

   - From the NCBI link: "Text formats, such as FASTQ, are supported, but are not the preferred submission medium. Poorly defined specifications and high variability within these formats tend to lead to a higher frequency of failed or problematic submissions."

A.11 How Many Threads Should You Use?

`https://www.jstorimer.com/blogs/workingwithcode/7970125-how-many-threads-is-too-many`

A.12 How Many Threads can my Computer Run?

`https://superuser.com/questions/1101311/how-many-cores-does-my-mac-have`

A.13 How to edit bash_profile

`https://stackoverflow.com/questions/30461201/how-do-i-edit-path-bash-profile-on-osx/30462883`

A.14 What "export PATH" commands mean

`https://askubuntu.com/questions/720678/what-does-export-path-somethingpath-mean`

A.15 Parallel-fastq-dump

`https://github.com/rvalieris/parallel-fastq-dump`

A.16 Thorough explanation of NCBI's (poorly-documented) fastq-dump command

`https://edwards.sdsu.edu/research/fastq-dump/`

A.17 What is a "feature" in qiime2?

`https://forum.qiime2.org/t/what-is-a-feature-exactly/2201`

A.18 Metadata in qiime2

`https://docs.qiime2.org/2018.4/tutorials/metadata/`

A.19 Multiqc modules

`http://multiqc.info/docs/#multiqc-modules`

A.20 Paired end sequencing vs single end sequencing

`https://www.illumina.com/science/technology/next-generation-sequencing/paired-end-vs-single-read-sequencing.html`

A.21 .glob formatting

(uses python 2 for the examples but other than the print statement, everything should be the same for python3)
`https://pymotw.com/2/glob/`

A.22 Qiime2 workflow from Center for Host-Microbial Interactions

`https://chmi-sops.github.io/mydoc_qiime2.html`

A.23 Importing data in qiime2

`https://docs.qiime2.org/2018.4/tutorials/importing/`

A.24 Why you shouldn't blindly trust public data for accuracy.

Always look into the metadata to understand your results
`https://sequencing.qcfail.com/articles/data-can-be-corrupted-upon-extraction-from-sra-files/`

A.25   Jargon: Base calling

Base Calling: "The deduction of nucleotide sequences from the images acquired during sequencing is commonly referred to as base calling." Definition comes from `https://galaxyproject.org/tutorials/ngs/` when it cites `http://chagall.med.cornell.edu/RNASEQcourse/Intro2RNAseq.pdf`

A.26   Illumina Next Generation Sequencing

`http://chagall.med.cornell.edu/RNASEQcourse/Intro2RNAseq.pdf`

- Figures explaining Illumina NGS
- Overview of the SRA database
- Introduction to RNA-seq
    - Library prep methods
    - Sequencing (illumina)
    - Experimental design
        * Avoiding bias
        * Capturing variability
    - Raw Data (Sequencing Reads)
        * Quality control
- Read Alignment
    - Reference Genomes and annotations
    - Storing aligned reads: SAM/BAM file format
    - Quality control of aligned reads
- Read Quantification
- Normalizing and Transforming Read Counts
- Differential Gene Expression Analysis

Appendix B Python Resources

B.1 Creating Python Functions

`https://www.tutorialspoint.com/python3/python_functions.htm`
`https://www.datacamp.com/courses/python-data-science-toolbox-part-1` A resources for learning how to create your own Python functions. The DataCamp course is a paid course, but the first chapter, which covers how to write your own functions, is available for free.

B.2 Video Tutorial: Installing and Using Anaconda

`https://www.youtube.com/watch?v=YJC6ldI3hWk`

B.3 Python Dictionaries

`https://docs.python.org/3/tutorial/datastructures.html#dictionaries`

B.4 Pandas.DataFrame.mask()

`https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.mask.html#pandas.DataFrame.mask`

B.5 How to search all (pandas) data frame rows for values outside a defined range of numbers (StackOverflow)

`https://stackoverflow.com/questions/41618224/how-to-search-all-data-frame-rows-for-values-outside-a-defined-range-of-numbers`

B.6 Assert Variable Type in Python

`https://stackoverflow.com/questions/2589522/proper-way-to-assert-type-of-variable-in-python`

B.7 Parallel-fastq-dump

`https://github.com/rvalieris/parallel-fastq-dump`

B.8 Get full path to a Python file

To get the full path to the directory a Python file is contained in, write this in that file: `https://stackoverflow.com/questions/5137497/find-current-directory-and-files-directory`

B.9 Importing a Class in the _ _init_ _.py file

`http://mikegrouchy.com/blog/2012/05/be-pythonic-__init__py.html`

B.10 Git stashing and applying

`https://git-scm.com/book/en/v1/Git-Tools-Stashing`
`https://stackoverflow.com/questions/19003009/git-how-to-recover-stashed-uncommitted-changes`

B.11 Importing packages and the need to repeatedly reference subpackages

`https://stackoverflow.com/questions/12229580/python-importing-a-sub-package-or-sub-module`

B.12 If _ _name_ _ == "_ _main_ _":

`https://stackoverflow.com/questions/419163/what-does-if-name-main-do`

B.13 Underscores in python

`https://shahriar.svbtle.com/underscores-in-python`

B.14 Python Modules vs Packages

`https://docs.python.org/3/reference/import.html`