# Camoco Documentation

***Release 0.6.4***

**Rob Schaefer**

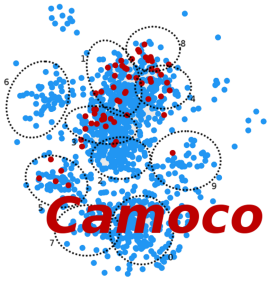
**Feb 12, 2020**

# TABLE OF CONTENTS


1 Citation 3

2 Table Of Contents 5

3 Indices and tables 33




**Camoco** is a python library for building and analyzing gene co-expression networks. Networks are built from tables of gene expression data typically derived from RNA-seq or micro-array experiments. Once networks are built, there are several tools available to validate or check the *health* of the co-expression network using annotated ontologies such as the Gene Ontology. Co-expression can then be calculated among sets of genes using several different metrics.

In addition to building and validating co-expression networks, Camoco can also be used to directly integrate data from Genome-Wide Association Studies (GWAS). Using a window based method, markers (SNPs) from GWAS are mapped to candidate genes and then analyzed for strong co-expression. Camoco identifies high priority overlap (i.e. between GWAS data and network data) by identifying genes near GWAS SNPs that have strong co-expression to genes near other GWAS SNPs. Results are compared to randomized networks to assign p-values to candidate genes. This approach is explained in detail in the publication cited below.

Camoco comes with a command line interface (CLI) for standardized analyses, but was also designed in a way where it can be extensively customized and modified within python scripts.

Camoco offers several key features for network analysis:

- **Quality control** of gene expression data before network generation
- Datasets are built once and stored in **internal databases** for repeated use
- Network **clusters** are automatically calculated
- Customizable **network plotting** methods
- On the fly GWAS **SNP-to-gene mapping**

CHAPTER

ONE

# CITATION

Camoco and its applications have been published in The Plant Cell. If you make use of Camoco in your work, please cite the following:

> Robert Schaefer, Jean-Michel Michno, Joseph Jeffers, Owen A Hoekenga, Brian P Dilkes, Ivan R. Baxter, Chad Myers. **Integrating networks and GWAS in maize.** The Plant Cell Nov 2018, tpc.00299.2018; DOI: 10.1105/tpc.18.00299

CHAPTER

TWO

# TABLE OF CONTENTS

## 2.1 Overview

Camoco handles the creation and the co-analysis/integration of several different components. We follow a *build once, use many times* philosophy as many of the data sets are computationally expensive to build. Camoco also was designed so that many of the standard analyses and procedures are available through the CLI. These procedures include building networks and other datasets, listing available datasets, or calculating the co-expression overlap between a network and GWAS data. There are also more advanced work flows that can be created by importing Camoco as a python library or by analyzing data interactively using ipython.

Once Camoco datasets are built, they can be co-analyzed together to determine if there is significant *overlap* between the datasets. This overlap vocabulary is used throughout this documentation to mean a statistical relationship between different types of data. For example, one analysis that Camoco offers is to calculate the *overlap* between GWAS data and a co-expression network. This analysis includes mapping GWAS SNPs to nearby genes, calculating the co-expression among the genes, then comparing the co-expression score to randomized sets of genes to determine the statistical significance of the overlap. Likewise, there are methods to calculate the overlap between a set of genes (potentially from a GWAS) and the clusters of a network. This *overlap* is calculated using an enrichment statistic: the hyper-geometric distribution.

In order to better understand the analysis being computed, its necessary to first define some terms to represent the types of data Camoco can co-analyze. This section will mainly describe the data-types at a high level while code and examples can be found in the tutorial section.

### 2.1.1 Camoco Data Types

There are several different types of data that Camoco build and in turn, compare. Many of these data types are computationally expensive to generate and thus, Camoco stores them in using internal databases. Several of the data types represent collections of other data types. For instance, there are Camoco objects to represent a single gene, as well as a named set of genes. There is also a distinction between what objects are persistent (stored in the internal database) and objects that are disposable.

In general, if the object is *named* it is stored in a database. Also, persistent objects, in general, generate disposable objects. This is better illustrated with an example. Camoco determines the significance of co-expression among a set of genes by comparing to the co-expression among random sets of genes. To get these random sets of genes, Camoco generates them from a reference genome object. As reference genomes include a large amount of data and are computationally expensive to generate, RefGen objects are created once and are stored in a database. From these RefGen objects, random sets of genes can be generated very quickly.

Camoco Data Types include:

**Locus** The most fundamental element that Camoco uses is a `Locus` object. A Locus is general purpose object that encodes a coordinate in the genome. It consists of a three mandatory elements: a chromosome name, a start

position, and an end position. Locus objects are also the basis of other genome coordinate objects such as SNPs and genes, which are just Loci with additional information.

**RefGen*** This object contains a reference set of Locus object, such as those that are defined in a GFF File. Typically, RefGen objects are used by other Camoco objects to generate genes either by ID or by genome location.

**Term** A `Term` object is collection of Loci objects with a common form or function. This is commonly used as a container for loci in a `Ontology` or `GWAS`.

**Ontology*** This is a object containing a set of Terms and their hierarchical relationships. This is used to store data related to things like Gene Onotologies (GO) or MapMan pathways.

**GWAS*** Similar to an Ontology, the terms in a GWAS are SNPs. A GWAS object handles SNP to gene mapping by referencing a RefGen object.

**COB*** This is the co-expression browser object. A COB is the network object in Camoco and used to perform all the network analysis components.

* Designates a persistent data type

### 2.1.2 Building data sets

Datasets can be built in two main ways, the first is from raw data and the second is from existing Camoco data sets. For example, a RefGen object is typically built from a GFF file. From a RefGen object, Loci objects can be created. Similarly Ontologies and GWAS are built from raw data files while Terms are created from Ontologies and GWAS objects.

COB objects are built from raw gene expression datasets, however, they also require that a valid RefGen object is built. During this process, a filtered RefGen object is automatically created containing only the genes in the expression data.

When a persistent Camoco object is built, a user specified name is required. This name is used in analyses where specific datasets need to be referenced. For example, in maize, there are two gene references: a full gene set as well as a filtered gene set containing only high confidence genes. Perhaps you might be interested in what a co-expression network might look like if it was built using genes in the full vs filtered gene set. First, you'd build the two RefGen objects from the raw datasets. This requires naming the dataset as they need to be referenced when the networks are built later. If we named these datasets 'ZmFull' and 'ZmFiltered', we could build two networks, one from each RefGen dataset.

Another case where naming datasets can come in handy is when there are quality control steps in the build process. Building a network requires several parameters that control quality control including:filtering missing data in genes/accession, setting minimum threshold for gene expression, and specifying a minimal level of variance in gene expression. While the parameters used to build the datasets are stored internally in the Camoco dataset and are retrievable, it might be useful to name your datasets to reflect how they were created. For instance: 'ZmStrict' and 'ZmLenient'

## 2.2 Installation

Camoco can be installed using the Python Package Index (pypi). This process is typical for python packages and uses a shell tool called `pip` to manage software dependencies. Python requires that `numpy` already be installed.:

```
$ pip install numpy
$ pip install camoco
```

Camoco does have a Python version requirement as well as a few dependencies in order to install successfully.

**Note:** Code blocks starting with $ designate shell commands.

### 2.2.1 Operating System Compatibilty

Camoco was built on and tested using Linux (Ubuntu 18.04). It is also suggested that you have a machine with at least 8Gb of usable RAM.

### 2.2.2 Python Version Requirement

Camoco requires python version 3.6+. If this differs from your system version of python, it is recommended that you install Camoco within a python virtual environment. We have had success using miniconda, which can be installed here.

Once miniconda is installed, create a virtual environment:

```
$ conda create -n camoco_env python=3.6
```

Follow the prompts on the screen. Next, activate the python virtual environment:

```
$ source activate camoco_env
```

Python should now be 3.6:

```
$ python --version
Python 3.6.8
```

Install Camoco as described above:

```
$ pip install numpy
$ pip install camoco
```

## 2.3 Tutorial Part I. - Building Objects with the CLI

This is a whirlwind tutorial to get you started with Camoco. This tutorial assumes that you've successfully completed the *installation steps*. In this tutorial we will be recreating the datasets used in this publication which examined the co-expression among genes related to the maize kernel ionome.

First, we will need to download some data.

**Note:** Data distributed here are subject to licensing terms specified by the original publications. If used, please give proper attribution to both the source article (if raw data is used) as well as the Camoco publication if Camoco was utilized.

### 2.3.1 Getting Data

Included with the source code on GitHub are test data sets as well as some raw input that can be used to create some Camoco objects. Here is a breakdown of some of the files:

#### RefGen Data

**ZmB73_5b_FGS.gff.gz** This is the raw data needed to create the RefGen object. The format of this file is a GFF, which you can read more about here.

#### Network (COB) Data

**Hirsch2014_PANGenomeFPKM.txt.gz** This is a FPKM table for 503 seedlings used to create the ZmPAN network from Schaefer et al. using data originally generated from Hirsch et al.

**Stelpflug2015_B73_Tissue_Atlas.txt.gz** This is a FPKM tables from 76 different tissues/time-points in the maize accession B73 described in Stelpflug et al.. This is the basis of the ZmSAM network in Schaefer et al..

**Schaefer2018_ROOTFPKM.tsv.gz** This dataset contains FPKM values for 46 genotypically diverse maize root samples. It is the described as the ZmRoot network in Schaefer et al..

#### Ontology Data

**go.obo.gz** This contains a list of all Gene Ontology (GO) Terms. The .obo file contains the generic GO structure and the relationships between terms. A description of the data can be found here.

**zm_go.tsv.gz** This file contains the mapping between maize genes and GO terms.

#### GWAS Data

**ZmIonome.allLocs.csv.gz** This file contains GWAS for a study looking at nutritional quality (elemental composition) for 17 traits in a maize mapping population (NAM) commonly referred to as the ionome.

**Wallace_etal_2014_PlosGenet_GWAS.gz** GWAS results from another maize mapping population (NAM) for 41 different traits. Details can be found in Wallace et al.

To download all the data, you can use the following command:

```
$ wget \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/RefGen/ZmB73_5b_FGS.gff.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/Expr/RNASEQ/Hirsch2014_
↪PANGenomeFPKM.txt.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/Expr/RNASEQ/Stelpflug2018_
↪B73_Tissue_Atlas.txt.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/Expr/RNASEQ/Schaefer2018_
↪ROOTFPKM.tsv.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/GOnt/go.obo.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/GOnt/zm_go.tsv.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/GWAS/SchaeferPlantCell/
↪ZmIonome.allLocs.csv.gz \
 https://github.com/LinkageIO/Camoco/raw/master/tests/raw/GWAS/WallacePLoSGenet/
↪Wallace_etal_2014_PLoSGenet_GWAS_hits-150112.txt.gz
```

**Note:** Please always be cautious when pasting shell commands from the internet!

Now uncompress the files, so we can easily look inside them

```
$ gunzip ./*.gz
```

### 2.3.2 Running the CLI

The easiest way to get started with Camoco is to use the command line interface. This can be accessed using the `camoco` command from the shell:

```
$ camoco
```

You should see the following output:

```
$ camoco
usage: camoco [-h] [--debug] [--interactive] [--force] Available Commands ...


      ___           ___           ___           ___           ___           ___
     /  /\         /  /\         /__/\         /  /\         /  /\         /  /\
    /  /:/        /  /::\       |  |::\       /  /::\       /  /:/        /  /::\
   /  /:/        /  /:/\:\      |  |:|:\     /  /:/\:\     /  /:/        /  /:/\:\
  /  /:/  ___   /  /:/~/::\   __|__|:|\:\   /  /:/  \:\   /  /:/  ___   /  /:/  \:\
 /__/:/  /  /\ /__/:/ /:/\:\ /__/::::| \:\ /__/:/ \__\:\ /__/:/  /  /\ /__/:/ \__\:\
 \  \:\ /  /:/ \  \:\/:/__\/ \  \:\~~\__\/ \  \:\ /  /:/ \  \:\ /  /:/ \  \:\ /  /:/
  \  \:\  /:/   \  \::/       \  \:\        \  \:\  /:/   \  \:\  /:/   \  \:\  /:/
   \  \:\/:/     \  \:\        \  \:\        \  \:\/:/     \  \:\/:/     \  \:\/:/
    \  \::/       \  \:\        \  \:\        \  \::/       \  \::/       \  \::/
     \__\/         \__\/         \__\/         \__\/         \__\/         \__\/

Camoco (Co-analysis of Molecular Components) inter-relates and co-analyzes different
levels of genomic data. Namely it integrates genes present near and around GWAS loci
using unbiased, functional information derived from co-expression networks.

optional arguments:
  -h, --help          show this help message and exit
  --debug             Drop into ipdb when something bad happens.
  --interactive       Initiate an ipdb session right before exiting.
  --force             Overwrite output files from previous analyses.

Camoco CLI program:
  Use --help with each command for more info

  Available Commands
    help              Prints this help message
    build-gwas        build a GWAS dataset
    build-go          Build a Gene Ontology (GO)
    build-refgen      Build a Reference Genome.
    build-cob         Build a Co-expression network.
    list (ls)         List camoco datasets.
    rm                Remove camoco dataset.
    overlap           Calculate network overlap among GWAS results. See
                      --method for details.
    health            Generate network health statistics
```

(continues on next page)

(continued from previous page)

```
    snp2gene          Generate candidate genes and accompanying information
                      from GWAS SNPs
    neighbors         Generate significant gene neighbors from largest to
                      smallest Z-score

version: 0.6.1
src:/home/schae234/miniconda3/envs/camoco/lib/python3.6/site-packages/camoco/__init__.
↪py
Cache. Money. Corn.
```

### 2.3.3 Building Camoco Objects

The first Camoco object we are going to build is the RefGen object. This is needed because most of the other objects need a reference in order to properly interpret gene IDs. For example, if you look at the first few lines of the file go gene mapping, you'll see *GRMZM2G061764_P01* which corresponds to a maize gene. Without the RefGen object, Camoco has not information about what this gene is or where it's located in the genome.

```
$ head zm_go.tsv
  !Protein_id GO_id   GO_name GO_namespace
  GRMZM2G061764_P01   GO:0016020      membrane        Cellular Component
  GRMZM2G082184_P01   GO:0016020      membrane        Cellular Component
  GRMZM2G082184_P01   GO:0016021      integral to membrane    Cellular Component
  GRMZM2G082184_P02   GO:0016020      membrane        Cellular Component
  GRMZM2G036652_P01   GO:0016021      integral to membrane    Cellular Component
  GRMZM2G028036_P01   GO:0016020      membrane        Cellular Component
  AC199054.3_FGP004   GO:0005739      mitochondrion   Cellular Component
  GRMZM2G065057_P01   GO:0005622      intracellular   Cellular Component
  GRMZM2G143473_P01   GO:0005618      cell wall       Cellular Component
```

So, when building the Camoco datasets, the order will matter if some objects need information contained in other objects. Lets start with the RefGen object.

#### Building a RefGen Object

Looking at the Camoco help command above (*camoco –help*), we can see there is a command to build a RefGen object. We can find more information about the command by looking at its help message

```
$ camoco build-refgen --help
↪                                                                    !10055
usage: camoco build-refgen [-h] [--chrom-feature chromosome]
                           [--gene-feature gene] [--ID-attr ID]
                           [--attr-split =]
                           filename name description

positional arguments:
  filename              The path to the GFF file.
  name                  The name if the RefGen object to be stored in the core
                        camoco database.
  description           A short description of the RefGen for future reference

optional arguments:
  -h, --help            show this help message and exit
  --chrom-feature chromosome
```

(continues on next page)

(continued from previous page)

```
                        The name of the feature (in column 3) that designates
                        a a chromosome. default: "chromosome"
  --gene-feature gene   The name of the feature (in column 2) that designates
                        a gene. These features will be the main object that
                        the RefGen encompasses. default: "gene"
  --ID-attr ID          The key in the attribute column which designates the
                        ID or name of the feature. default: "ID"
  --attr-split =        The delimiter for keys and values in the attribute
                        column. default: "="
```

**Note:** Starting in v0.6.3 the *build* and *organism* arguments are no longer needed.

The command takes 5 required arguments (called positional argument): *filename*, *name*, *description*, *build*, and *organism*, as well as 5 optional arguments. Our build command will look something like:

```
$ camoco build-refgen \
      ZmB73_5b_FGS.gff \
      "Zm5bFGS" \
      "Maize 5b Filtered Gene Set"
```

The *filename* corresponds to the raw file we downloaded. The *name* is a short name supplied by you, that references the dataset. Correspondingly, *description* is used to supply a little more information. Then we have *build* and *organism* which are used internally to differentiate between different genome builds (gene positions change between versions) as well as genes that might have the same name but come from different species/sub-species.

As for the optional arguments, we need to look inside the *GFF* file to know if we need these.

```
$ head ZmB73_5b_FGS.gff
 9   ensembl chromosome      1       156750706       .       .       .       ID=9;
↪Name=chromosome:AGPv2:9:1:156750706:1
 9   ensembl gene    66347   68582   .       -       .       ID=GRMZM2G354611;
↪Name=GRMZM2G354611;biotype=protein_coding
 9   ensembl mRNA    66347   68582   .       -       .       ID=GRMZM2G354611_T01;
↪Parent=GRMZM2G354611;Name=GRMZM2G354611_T01;biotype=protein_coding
 9   ensembl intron  68433   68561   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=intron.1
 9   ensembl intron  67142   67886   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=intron.2
 9   ensembl intron  66671   67066   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=intron.3
 9   ensembl intron  66535   66606   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=intron.4
 9   ensembl exon    68562   68582   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=GRMZM2G354611_E02
 9   ensembl exon    67887   68432   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=GRMZM2G354611_E05
 9   ensembl exon    67067   67141   .       -       .       Parent=GRMZM2G354611_T01;
↪Name=GRMZM2G354611_E04
```

We can see that chromosomes are defined in the file using the word *chromosome* which is the default for the command, meaning we don't need to specify the argument. Some files may have the feature type coded as *chr* or *chrom* in which this option would be useful. We can also see the same case with *–gene-feature*, genes in the file are coded as *gene*, which is the default for the program. Similarly, gene IDs are specified with the *–ID-attr* option but are already coded with the default, *ID*, in the file. Lastly, attributes in the file are designated with a = sign. The GFF specification also sometimes allows spaces (' '), making this option useful.

We can thus run the above command as is. We will see this following output:

```
$ camoco build-refgen ZmB73_5b_FGS.gff "Zm5bFGS" "Maize 5b Filtered Gene Set"
[LOG] Wed Nov 14 11:10:58 2018 - Building Indices
[LOG] Wed Nov 14 11:10:58 2018 - Building Indices
[LOG] Wed Nov 14 11:10:58 2018 - Found a chromosome: 9
[LOG] Wed Nov 14 11:10:59 2018 - Found a chromosome: 1
[LOG] Wed Nov 14 11:10:59 2018 - Found a chromosome: 4
[LOG] Wed Nov 14 11:11:00 2018 - Found a chromosome: 5
[LOG] Wed Nov 14 11:11:00 2018 - Found a chromosome: 2
[LOG] Wed Nov 14 11:11:01 2018 - Found a chromosome: 3
[LOG] Wed Nov 14 11:11:01 2018 - Found a chromosome: 6
[LOG] Wed Nov 14 11:11:01 2018 - Found a chromosome: 8
[LOG] Wed Nov 14 11:11:02 2018 - Found a chromosome: 7
[LOG] Wed Nov 14 11:11:02 2018 - Found a chromosome: 10
[LOG] Wed Nov 14 11:11:02 2018 - Found a chromosome: UNKNOWN
[LOG] Wed Nov 14 11:11:02 2018 - Found a chromosome: Pt
[LOG] Wed Nov 14 11:11:02 2018 - Found a chromosome: Mt
[LOG] Wed Nov 14 11:11:03 2018 - Adding 39656 Genes info to database
[LOG] Wed Nov 14 11:11:04 2018 - Adding Gene attr info to database
[LOG] Wed Nov 14 11:11:05 2018 - Building Indices
Build successful!
```

Your output will however have a current date and time. We can now build some of the other objects that rely on the RefGen object being present.

### Building a COB object (co-expression network)

A COB is a co-expression object, or more specifically a co-expression browser (pun intended). We can get an idea of what data we need to build the network by running the *–help* command.

```
$ camoco build-cob -h
  usage: camoco build-cob [-h] [--rawtype RAWTYPE] [--sep SEP]
                          [--index-col INDEX_COL] [--max-val MAX_VAL]
                          [--skip-quantile] [--skip-quality-control]
                          [--skip-normalization] [--min-expr MIN_EXPR]
                          [--max-gene-missing-data MAX_GENE_MISSING_DATA]
                          [--max-accession-missing-data MAX_ACCESSION_MISSING_DATA]
                          [--min-single-sample-expr MIN_SINGLE_SAMPLE_EXPR]
                          [--allow-non-membership]
                          [--zscore_cutoff default: 3.0] [--dry-run]
                          filename name description refgen

  positional arguments:
    filename              Path to CSV or TSV
    name                  Name of the network.
    description           Short description of network
    refgen                Name of a pre-built RefGen. See build-refgen command

  optional arguments:
    -h, --help            show this help message and exit
    --rawtype RAWTYPE     Passing in a rawtype can help determine default
                          parameters for standard data types such as MICROARRAY
                          and RNASEQ. Specifies the fundamental datatype used to
                          measure expression. During importation of the raw
                          expression data, this value is used to make decisions
                          in converting data to log-space. Options: (one of:
```

(continues on next page)

(continued from previous page)

```
                        'RNASEQ' or 'MICROARRAY')
  --sep SEP             Field separators in the CSV or TSV, default=\t
  --index-col INDEX_COL
                        If not None, this column will be set as the gene index
                        column. Useful if there is a column name in the text
                        file for gene names.
  --max-val MAX_VAL     This value is used to determine if any columns of the
                        dataset have already been normalized. If any
                        'normailzed' values in an Accession column is larger
                        than max_val, an exception is thown. max_val is
                        determined by Expr.raw_type (default 100 for
                        MicroArray and 1100 for RNASeq) but a max_val can be
                        passed in to override these defaults.
  --skip-quantile       Flag specifies whether or not to perform quantile
                        normalization on expression data.
  --skip-quality-control
                        A Flag indicating to skip quality control procedure on
                        expression data. Default: False
  --skip-normalization  Flag indicating that expression normalization should be
                        skipped. Default: False
  --min-expr MIN_EXPR   Expression values (e.g. FPKM) under this threshold
                        will be set to NaN and not used during correlation
                        calculations. default: 0.01
  --max-gene-missing-data MAX_GENE_MISSING_DATA
                        Maximum percentage missing data a gene can have. Genes
                        not meeting this criteria are removed from
                        dataset.default: 0.2
  --max-accession-missing-data MAX_ACCESSION_MISSING_DATA
                        maximum percentage missing data an accession
                        (experiment) canhave before it is removed.Default: 0.3
  --min-single-sample-expr MIN_SINGLE_SAMPLE_EXPR
                        Genes that do not have a single accession having an
                        expression value above this threshold are removed from
                        analysis. These are likely presence/absence and will
                        not have a strong coexpression pattern.
  --allow-non-membership
                        Flag indicating that feature (genes, metabolites, etc)
                        should not be filtered out of the expression matrix
                        because they are not present in the reference genome.
                        This is useful for features, such as metabolites, that
                        cannot be anchored to a RefGen. If true, features will
                        be added to the RefGen with unknown coordinates
  --zscore_cutoff (default: 3.0)
                        The zscore threshold used for edges in the network.
                        Edges with z-scores under this value will not be used
                        for thresholded calculations such as locality. Un-
                        thresholded calculations, such as density will not be
                        affected by this cutoff.
  --dry-run             Dry run will only process the first 5000 genes
```

This command has lots of options, many of which are for specific cases and are not of major concern here. There are 4 required arguments to the command: *filename*, *name*, *description*, and *refgen*. Many of these are familiar from the *build-refgen* command, the only new one is the *refgen* argument. This argument is the *name* of the RefGen that we build above: *Zm5bFGS*. You can see that it would be very easy to swap out the *refgen* name to build networks from different reference genomes. Lets start with the first expression dataset: *Schaefer2018_ROOTFPKM.tsv*.

Lets look at the fist few lines of the file.

```
$ head
A5554   B57     B73     B76     B97     CML103  CML108  CML157Q CML158Q CML228 ␣
↪CML277 [...truncated]
AC147602.5_FG004        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0  ␣
↪ [...truncated]
AC148152.3_FG001        12.6150891209   32.1081261372   0.956572257358  50.0753099485␣
↪ [...truncated]
AC148152.3_FG005        41.6177222435   85.5673756791   2098.45885272   22.0156281789␣
↪ [...truncated]
AC148152.3_FG006        9.25242061055   12.4374416494   0.0     10.4829916573   0.
↪5443 [...truncated]
AC148152.3_FG008        0.0     0.0     0.0     0.0     1.52487782359   0.
↪570642689653 [...truncated]
AC148167.6_FG001        84.0997521233   66.9761434839   113.56493263    102.270836629␣
↪ [...truncated]
AC149475.2_FG002        221.494880288   118.795515591   178.468248745   62.5394927926␣
↪ [...truncated]
AC149475.2_FG003        96.6201144036   114.022532276   88.1003049027   48.3058643672␣
↪ [...truncated]
[...truncated]

** output was truncated **
```

Camoco expects the file to by default be tab-delimited. The first line contains the names of all the experimental accessions/experiments. Each remaining line first contains the ID of the gene corresponding to the RefGen object created above (specified in its *–ID-attr* field). The remaining values are the expression values (e.g. FPKM) for each of the accessions. There are a few optional arguments that will help with slightly different file formats (see *–index-col* and *–sep*) but in general Camoco expects a expression matrix where each row is a gene and each column is an accession.

The build command will look something like:

```
$ camoco build-cob Schaefer2018_ROOTFPKM.tsv ZmRoot "Maize Root Network" Zm5bFGS
```

We specified the raw data file, we called out network *ZmRoot* giving it a short description "Maize Root Network". Finally, we specified the *name* of the RefGen object that contains all the gene information for our data. Camoco will attempt to build the network using the data provided. The output will look something like:

```
[LOG] Wed Nov 14 12:07:22 2018 - Building Indices
[LOG] Wed Nov 14 12:07:23 2018 - Loading Expr table
[LOG] Wed Nov 14 12:07:24 2018 - Building Expr Index
[LOG] Wed Nov 14 12:07:24 2018 - Loading RefGen
[LOG] Wed Nov 14 12:07:24 2018 - RefGen for ZmRoot not set!
[LOG] Wed Nov 14 12:07:24 2018 - Loading Coex table
[LOG] Wed Nov 14 12:07:24 2018 - ZmRoot is empty
[LOG] Wed Nov 14 12:07:24 2018 - Loading Global Degree
[LOG] Wed Nov 14 12:07:24 2018 - ZmRoot is empty
[LOG] Wed Nov 14 12:07:24 2018 - Loading Clusters
[LOG] Wed Nov 14 12:07:24 2018 - Clusters not loaded for: ZmRoot ()
[LOG] Wed Nov 14 12:07:24 2018 - Resetting raw expression data
[LOG] Wed Nov 14 12:07:24 2018 - Resetting expression data
[LOG] Wed Nov 14 12:07:24 2018 - Extracting raw expression values
[LOG] Wed Nov 14 12:07:24 2018 - Importing Raw Expression Values
[LOG] Wed Nov 14 12:07:24 2018 - Trans. Log: raw->RawRNASEQ
[LOG] Wed Nov 14 12:07:24 2018 - Resetting expression data
[LOG] Wed Nov 14 12:07:24 2018 - Extracting raw expression values
[LOG] Wed Nov 14 12:07:24 2018 - Performing Quality Control on genes
[LOG] Wed Nov 14 12:07:24 2018 - ------------Quality Control
```

(continues on next page)

(continued from previous page)

```
[LOG] Wed Nov 14 12:07:25 2018 - Raw Starting set: 39655 genes 46 accessions

[...]
```

Dispersed within the information about the stage of the build is information describing the input data. For example, Camoco is reporting that it found *39655* genes and *46* accessions. If this is not what was expected, the input file was probably not formatted correctly and the build will fail. Lets look at the next few lines of the output:

```
[...]
[LOG] Wed Nov 14 17:31:31 2018 - ------------Quality Control
[LOG] Wed Nov 14 17:31:31 2018 - Raw Starting set: 39655 genes 46 accessions
[LOG] Wed Nov 14 17:31:31 2018 - Found out 0 genes not in Reference Genome: Zea mays -
↪ 5b - Zm5bFGS
[LOG] Wed Nov 14 17:31:31 2018 - Filtering expression values lower than 0.01
[LOG] Wed Nov 14 17:31:34 2018 - Found 15132 genes with > 0.2 missing data
[LOG] Wed Nov 14 17:31:40 2018 - Found 12934 genes which do not have one sample above␣
↪5
[LOG] Wed Nov 14 17:31:43 2018 - Found 0 accessions with > 0.3 missing data
[LOG] Wed Nov 14 17:31:43 2018 - Genes passing QC:
    has_id                 39655
    pass_membership        39655
    pass_missing_data      24523
    pass_min_expression    26721
    PASS_ALL               22909
    dtype: int64
[LOG] Wed Nov 14 17:31:43 2018 - Accessions passing QC:
    has_id               46
    pass_missing_data    46
    PASS_ALL             46
dtype: int64
[LOG] Wed Nov 14 17:31:46 2018 - Genes passing QC by chromosome:
            has_id  pass_membership  pass_missing_data  pass_min_expression  PASS_ALL
    chrom
    1          6056             6056               3782                 4088      3528
    10         2727             2727               1673                 1833      1555
    2          4766             4766               2894                 3180      2686
    3          4197             4197               2662                 2882      2488
    4          4197             4197               2496                 2733      2333
    5          4503             4503               2951                 3145      2766
    6          3293             3293               2046                 2227      1894
    7          3147             3147               1968                 2128      1837
    8          3531             3531               2245                 2493      2109
    9          3006             3006               1806                 2012      1713
    Mt          123              123                  0                    0         0
    Pt           57               57                  0                    0         0
    UNKNOWN      52               52                  0                    0         0
[LOG] Wed Nov 14 17:31:46 2018 - Kept: 22909 genes 46 accessions
[...]
```

Camoco is now performing quality control on the input data. The first thing it does is filter out any genes that are not present in the RefGen object. In this case 0 genes were not in the RefGen. Next, Camoco sets gene expression values lower than 0.1 to NaN. Values of NaN will be ignored when calculating co-expression and this is to control for missing genes versus lowly expressed genes. Next, Camoco filters out genes with over 20% missing data as well as genes that don't have at least one accession with an FPKM above 5. This removes genes that have very little variance in their gene expression profiles and are not informative given the accession provided (i.e low variance). Finally, Camoco finds that 0 accessions have over 30% missing data so it keeps all the samples in the experiment. Next is a breakdown of

the QC data showing how many genes passed each filter criteria. Of the 39,655 raw genes, only 22,909 genes pass all (PASS_ALL) the QC steps. This output allows you to check that nothing unexpected is happening during QC. Similar output is produced for accessions and finally there is a gene QC breakdown by chromosome which will again show any unexpected results.

Unexpected results will help you determine the source of input errors and potential biases. For example, if all of the genes are filtered out due to not being in the reference genome, you might have provided an incompatible RefGen object. Similarly, if all your genes were filtered out because no samples have values above 5, perhaps the data was pre-normalized or not FPKM.

The values for all of these QC steps are adjustable using options in the command line. For example to change the threshold for setting values to NaN to 0, you'd adjust the *–min-expr* flag. Similarly, *–max-gene-missing-data*, *–max-accession-missing-data*, *–min-single-sample-expr* and *–allow-non-membership* will allow you to customize the quality control steps. Finally, there is an option to skip QC all together.

The next bit of output shows the steps taken to normalize the data:

```
[...]
[LOG] Wed Nov 14 17:31:46 2018 - ------------ Normalizing
[LOG] Wed Nov 14 17:31:46 2018 - Trans. Log: raw->quality_control->arcsinh
[LOG] Wed Nov 14 17:31:46 2018 - Performing Quantile Gene Normalization
[LOG] Wed Nov 14 17:31:46 2018 - ------------ Quantile
[LOG] Wed Nov 14 17:31:46 2018 - Ranking data
[LOG] Wed Nov 14 17:31:47 2018 - Sorting ranked data
[LOG] Wed Nov 14 17:31:48 2018 - Calculating averages
[LOG] Wed Nov 14 17:31:50 2018 - Range of normalized values:0.045112951180571875..11.
↪715845314898932 (n = 22909)
[LOG] Wed Nov 14 17:31:50 2018 - Asserting that no Genes are nan...
[LOG] Wed Nov 14 17:31:50 2018 - Applying non-floating normalization
[LOG] Wed Nov 14 17:32:03 2018 - Updating values
[LOG] Wed Nov 14 17:32:03 2018 - Trans. Log: raw->quality_control->arcsinh->quantile
[LOG] Wed Nov 14 17:32:03 2018 - Filtering refgen: Zm5bFGS
[LOG] Wed Nov 14 17:32:04 2018 - Building Indices
[LOG] Wed Nov 14 17:32:05 2018 - Building Indices
[LOG] Wed Nov 14 17:32:05 2018 - Adding 22909 Genes info to database
[LOG] Wed Nov 14 17:32:06 2018 - Adding Gene attr info to database
[LOG] Wed Nov 14 17:32:07 2018 - Building Indices

[...]
```

The first step is the normalize the raw gene expression values. By default, Camoco expects the data to be from an RNA-Seq experiment and performs an inverse hyperbolic sine transformation to the expression data due to the dynamic range of the values (as suggested here. If *–rawtype* is changed to *MICROARRAY*, Camoco will use a log2 transformation. After transforming the values Camoco performs a quantile normalization on each of the Accessions (columns) as suggested here. Log messages also indicate that data are being added to the internal Camoco databases.

Next, Camoco is ready to calculate gene co-expression:

```
[...]
[LOG] Wed Nov 14 17:32:07 2018 - Calculating Coexpression
[LOG] Wed Nov 14 17:34:03 2018 - Applying Fisher Transform
[LOG] Wed Nov 14 17:34:09 2018 - Calculating Mean and STD
[LOG] Wed Nov 14 17:34:14 2018 - Finding adjusted scores
[LOG] Wed Nov 14 17:34:15 2018 - Build the dataframe and set the significance␣
↪threshold
[LOG] Wed Nov 14 17:34:16 2018 - Calculating Gene Distance
Calculating for 22909 genes
[LOG] Wed Nov 14 17:34:41 2018 - Done
```

(continues on next page)

(continued from previous page)

```
[LOG] Wed Nov 14 17:34:41 2018 - Building Degree
[LOG] Wed Nov 14 17:34:41 2018 - Calculating Gene degree
[LOG] Wed Nov 14 17:34:58 2018 - Calculating hierarchical clustering using single
[LOG] Wed Nov 14 17:35:10 2018 - Finding the leaves
[LOG] Wed Nov 14 17:35:10 2018 - Getting genes
[LOG] Wed Nov 14 17:35:10 2018 - Pulling edges
[LOG] Wed Nov 14 17:35:16 2018 - Creating Index
[LOG] Wed Nov 14 17:35:17 2018 - Making matrix symmetric
[LOG] Wed Nov 14 17:35:17 2018 - Creating matrix
[LOG] Wed Nov 14 17:35:56 2018 - Building cluster dataframe
[LOG] Wed Nov 14 17:35:56 2018 - Creating Cluster Ontology
[LOG] Wed Nov 14 17:35:57 2018 - Adding 7203 terms to the database.
[LOG] Wed Nov 14 17:35:58 2018 - Building the indices.
[LOG] Wed Nov 14 17:35:58 2018 - Your gene ontology is built.
[LOG] Wed Nov 14 17:35:58 2018 - Finished finding clusters
Build successful!

[...]
```

Here, Camoco calculated the Pearson correlation coefficient for all pairwise combinations of genes that passed QC. In this case there are 22,909 genes which means there are 262,399,686 interactions that must be calculated which takes about 2 minutes of compute time. Camoco stores all unthresholded interactions as well as which interactions have a z-score of 3 or above for thresholded analyses. This threshold can be changed at anytime, even after the network is built.

Similar to co-expression, Camoco calculates pairwise gene distance for all pairs of genes.

After that, several clustering functions are run on the network data and `Ontology` objects are build and stored for these groups of genes. As network clusters are just sets of genes, they are represented using the same Camoco data objects as Gene Ontologies or GWAS data are. See the **overview_** section for more details on this.

Finally, a summary of the network is printed:

```
[...]
[LOG] Wed Nov 14 17:35:58 2018 - Extracting raw expression values

            COB Dataset: ZmRoot
                Desc: Maize Root Network
                RawType: RNASEQ
                TransformationLog: raw->quality_control->arcsinh->quantile
                Num Genes: 22,909(58% of total)
                Num Accessions: 46

            Network Stats
            -------------
            Unthresholded Interactions: 262,399,686
            Thresholded (Z >= 3): 996,621

            Raw
            ------------------
            Num Raw Genes: 39,655
            Num Raw Accessions: 46

            QC Parameters
            ------------------
            min expr level: 0.01
                - expression below this is set to NaN
```

(continues on next page)

(continued from previous page)

```
            max gene missing data: 0.2
                - genes missing more than this percent are removed
            max accession missing data: 0.3
                - Accession missing more than this percent are removed
            min single sample expr: 5
                - genes must have this amount of expression in
                  at least one accession.

            Clusters
            ------------------
            Num clusters (size >= 10): 115
```

This data is stored internally and accessible anytime. It is printed here for your convenience.

### Building Ontology Datasets

Currently, the only Ontology that is supported from the command line is GO.

As you've likely gotten the hang of the procedure to look at the help message for building Camoco commands, building an Ontology object containing GO data should be straight forward.

Examine what is required to build the dataset from the CLI help message

```
$ camoco build-go --help
usage: camoco build-go [-h] [--go-col GO_COL] [--id-col ID_COL]
                       filename obo_filename name description refgen

positional arguments:
  filename         Gene-term map file
  obo_filename     GO .obo filename
  name             GO dataset name
  description      short dataset description
  refgen           Camoco reference Genome name.

optional arguments:
  -h, --help       show this help message and exit
  --go-col GO_COL  The GO Term ID column in Gene-term map file (default:1)
  --id-col ID_COL  The gene ID column in Gene-term map file (default:0)
```

We can thus build the command using similar information from previous commands. We must first specify the gene-to-term mapping file: *zm_go.tsv* then the obo file: *go.obo* before our familiar *name*, *description*, and *refgen* arguments.

```
$ camoco build-go \
  zm_go.tsv \
  go.obo \
  "ZmGO" \
  "Maize GO" \
  Zm5bFGS

[LOG] Wed Nov 14 18:24:34 2018 - Building Indices
[LOG] Wed Nov 14 18:24:34 2018 - Importing OBO: go.obo
[LOG] Wed Nov 14 18:24:36 2018 - Building the indices.
[LOG] Wed Nov 14 18:24:36 2018 - Importing Gene Map: zm_go.tsv
[LOG] Wed Nov 14 18:24:36 2018 - Adding GO-gene assignments
[LOG] Wed Nov 14 18:24:59 2018 - The following terms were referenced in the obo file␣
↪but were not found in the gene-term mapping:
```

(continues on next page)

(continued from previous page)

```
GO:0000119
GO:0004091
GO:0004811

[... truncated]
```

Camoco imports the data and creates the internal databases from the two files and the RefGen object.

### Building a GWAS Object

Finally we build a GWAS object. Fist lets check out the help output from the command.

```
$ camoco build-gwas --help
usage: camoco build-gwas [-h] [--trait-col Trait] [--chrom-col CHR]
                         [--pos-col POS] [--sep     ]
                         [--skip-traits [SKIP_TRAITS [SKIP_TRAITS ...]]]
                         filename name description refgen

positional arguments:
  filename              Filename of csv/tsv.
  name                  Name of GWAS dataset.
  description           Short description of dataset.
  refgen                Name of camoco RefGen dataset.

optional arguments:
  -h, --help            show this help message and exit
  --trait-col Trait     The name of the column in the table containing the
                        trait name.
  --chrom-col CHR       The name of the column containing the SNP chromosome
                        (note: must correspond with chromosome names if
                        RefGen)
  --pos-col POS         The name of the column containing the SNP position
                        (note: must correspond with the positions in RefGen)
  --sep                    Field Separator in CSV/TSV (default:\t)
  --skip-traits [SKIP_TRAITS [SKIP_TRAITS ...]]
                        Skip these traits. Can provide as many traits as you
                        like
```

The input to this command is very similar to previous commands. We build out our command as before:

```
$ camoco build-gwas ZmIonome.allLocs.csv ZmIonome "Maize Ionome GWAS" Zm5bFGS
[LOG] Wed Nov 14 18:37:25 2018 - A baaaad thing happened.

Only 1 column found, check --sep, see --help
```

The command failed! Luckily it suggest a problem. Lets look at the input file.

```
$ head ZmIonome.allLocs.csv
↪1
chr,pos,loc,el,cM,SNP,avgEffectSize,avgFval,avgPval,numIterations
1,1825,allLocs,Al27,-5.01370141392623,G/A_hm2,-0.0795269870874929,62.296735941693704,
↪8.77087952759627e-10,12
1,1637264,allLocs,Al27,-0.999833108356853,C/T_hm2,0.10869453140417899,40.
↪6930710376346,3.25756837266622e-09,6
1,2087475,allLocs,B11,-0.0856134701200706,G/C_hm2,0.0252463715299129,30.1737737870891,
↪1.8383025530902398e-07,13
```

(continues on next page)

(continued from previous page)

```
1,2426922,allLocs,B11,0.226708254550976,G/A_hm2,0.0196295712086249,30.5393640036167,1.
↪8205640561226603e-07,5
1,2430001,allLocs,B11,0.229541199148001,win2k-,0.0316936215735605,31.8587890663695,9.
↪822757122522949e-08,7
1,2768707,allLocs,As75,0.541180475686618,G/A_hm2,-0.000244643364229126,30.
↪735044551037802,3.4788513574393703e-07,7
1,2785753,allLocs,Ca43,0.556864332704606,T/C_hm2,0.8888294627733091,29.4795951468653,
↪3.30905067590346e-07,5
1,2827438,allLocs,As75,0.595218291392557,G/A_hm2,-0.0347906096896323,54.
↪23656575072479,5.71794333538972e-08,6
1,3458001,allLocs,As75,2.5270027050441097,win2k+,-0.000287726493696968,39.
↪708883927648294,1.8251979778166e-07,7
```

We can see that the fields are separated by commas. Also the column designating the trait is *el* and the columns designating SNP chromosome and position are *pos* and *chr*, which differ from the defaults. Let's try again with more optional arguments.

```
$ camoco build-gwas ZmIonome.allLocs.csv ZmIonome \
  "Maize Ionome GWAS" Zm5bFGS --sep ',' --trait-col el \
  --chrom-col chr --pos-col pos

Loading Zm5bFGS
[LOG] Wed Nov 14 18:43:33 2018 - Building Indices
[LOG] Wed Nov 14 18:43:33 2018 - Building Indices
[LOG] Wed Nov 14 18:43:33 2018 - Importing Term: Al27, Desc: , 176 Loci
[LOG] Wed Nov 14 18:43:33 2018 - Importing Term: As75, Desc: , 182 Loci
[LOG] Wed Nov 14 18:43:33 2018 - Importing Term: B11, Desc: , 108 Loci
[LOG] Wed Nov 14 18:43:33 2018 - Importing Term: Ca43, Desc: , 105 Loci
[LOG] Wed Nov 14 18:43:33 2018 - Importing Term: Cd111, Desc: , 630 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: Co59, Desc: , 1347 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: Cu65, Desc: , 165 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: Fe57, Desc: , 171 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: K39, Desc: , 130 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: Mg25, Desc: , 153 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: Mn55, Desc: , 168 Loci
[LOG] Wed Nov 14 18:43:34 2018 - Importing Term: Mo98, Desc: , 154 Loci
[LOG] Wed Nov 14 18:43:35 2018 - Importing Term: Ni60, Desc: , 99 Loci
[LOG] Wed Nov 14 18:43:35 2018 - Importing Term: P31, Desc: , 123 Loci
[LOG] Wed Nov 14 18:43:35 2018 - Importing Term: Rb85, Desc: , 135 Loci
[LOG] Wed Nov 14 18:43:35 2018 - Importing Term: Se82, Desc: , 162 Loci
[LOG] Wed Nov 14 18:43:35 2018 - Importing Term: Sr88, Desc: , 113 Loci
[LOG] Wed Nov 14 18:43:35 2018 - Importing Term: Zn66, Desc: , 149 Loci
Build Successful:
Ontology:ZmIonome - desc: Maize Ionome GWAS - contains 18 terms for Reference Genome:␣
↪Zea mays - 5b - Zm5bFGS
```

It worked! Once the right arguments were passed in, the dataset was created.

### 2.3.4 Listing and deleting Camoco datasets

By now we've built a few datasets. Camoco comes with some commands to manage the datasets that we've build so far.

List datasets:

```
$ camoco ls
                    Name                  Description           Date Added
Type
Camoco             Camoco                  Camoco base  2018-11-14 23:21:29
Expr               ZmRoot          Maize Root Network  2018-11-14 23:31:30
GOnt                 ZmGO                    Maize GO  2018-11-15 00:24:34
GWAS             ZmIonome           Maize Ionome GWAS  2018-11-15 00:43:18
Ontology       ZmRootMCL         ZmRoot MCL Clusters  2018-11-14 23:35:57
RefGen           Zm5bFGS  Maize 5b Filtered Gene Set  2018-11-14 23:30:35
RefGen    FilteredZmRoot             Filtered Refgen  2018-11-14 23:32:04
```

You'll see the datasets we've built so far, but also there are other datasets (e.g. *FilteredZmRoot* and *ZmRootMCL*) that were built behind the scenes during the COB build process.

---

**Note:** Due to legacy reasons, camoco lists COB objects as "Expr". This is due to that internally, a COB object is composed of two smaller objects that work together.

---

Delete a dataset:

```
$ camoco rm RefGen Zm5bFGS
[LOG] Wed Nov 14 18:52:36 2018 - Deleting Zm5bFGS
[LOG] Wed Nov 14 18:52:36 2018 - Removing /home/rob/.camoco/databases/RefGen.Zm5bFGS.
↪db
Done
```

This removed our RefGen object from the available Camoco datasets.

**Warning:** Deleting datasets that other objects depend on (such as the example above) can cause analyses to fail. Re-build required datasets before proceeding with analyses.

Lets get that RefGen object re-built

```
$ camoco build-refgen \
  ZmB73_5b_FGS.gff \
  "Zm5bFGS" \
  "Maize 5b Filtered Gene Set" \
  5b \
  "Zea mays"
```

### 2.3.5 Exercises

Included in the downloaded data from above are raw files for three more Camoco objects. Build the following datasets:

- ZmPAN COB Object from *Hirsch2014_PANGenomeFPKM.txt*
- ZmSAM COB Object from *Stelpflug2018_B73_Tissue_Atlas.txt*
- ZmWallace GWAS Object from *Wallace_etal_2014_PLoSGenet_GWAS_hits-150112.txt*

*solutions can be found at the bottom of this page

### 2.3.6 Calculating co-expression with the CLI

Once all the necessary Camoco objects are built, co-expression can be calculated among genes represented by the Camoco datasets. Through the CLI, this is performed using the *overlap* command.

Lets look at the help message for the *overlap* command:

```
$ camoco overlap --help
usage: camoco overlap [-h] [--genes [GENES [GENES ...]]] [--gwas GWAS]
                      [--go GO] [--terms [TERMS [TERMS ...]]]
                      [--skip-terms [SKIP_TERMS [SKIP_TERMS ...]]]
                      [--min-term-size 2] [--max-term-size None]
                      [--snp2gene effective] [--strongest-attr pval]
                      [--strongest-higher] [--candidate-window-size 1]
                      [--candidate-flank-limit 0] [--num-bootstraps auto]
                      [--out OUT] [--dry-run]
                      cob {density,locality}

positional arguments:
  cob                   The camoco network to use.
  {density,locality}    The metric used to evaulate overlap between Loci and
                        the Network

optional arguments:
  -h, --help            show this help message and exit
  --genes [GENES [GENES ...]]
                        Provide a [comma, space, semicolon] separated list of
                        input genes.
  --gwas GWAS           Calculate overlap on a Camoco GWAS dataset.
  --go GO               Caluclate overlap among GO terms
  --terms [TERMS [TERMS ...]]
                        The term within the ontology to use. If all, terms in
                        gwas will be iteratively analyzed. (default: all)
  --skip-terms [SKIP_TERMS [SKIP_TERMS ...]]
                        Terms specified here will be skipped.
  --min-term-size 2     The minimum number of gene in a term to calculate
                        overlap.
  --max-term-size None  The maximum number of genes in a term to calculate
                        overlap.
  --snp2gene effective  The SNP to gene mapping to use. If *effective*, SNPs
                        with overlappingn windows will be collapsed into
                        genomic intervals resulting in all genes within the
                        intervals to be used. If *strongest* is specified,
                        SNPs with lower values designated from --strongest-
                        attr (e.g. pvals) will be dropped when SNPs have
                        overlapping windows. Value must be in
```

(continues on next page)

(continued from previous page)

```
                        ["effective","strongest"]. (default: effective)
  --strongest-attr pval
                        The locus attr used to determine which locus is
                        thestrongest locus. (defualt=pval).
  --strongest-higher    Flag indicating the value in --strongest-attr
                        isstronger if higher. Default behavior is to
                        treatlower values as stronger (i.e. p-vals)
  --candidate-window-size 1
                        The window size, in bp, for mapping effective SNPs to
                        genes. (default: 1)
  --candidate-flank-limit 0
                        The number of flanking genes included in SNP to gene
                        mapping. on each side of the locus. (default: 1)
  --num-bootstraps auto
                        The number of bootstraps to perform in order to
                        estimate a null distribution. If auto the algorithm
                        will bootstrap *until* the term is not significant at
                        n=1000 *or* 1000 bootstraps have been performed.
                        (default: auto)
  --out OUT             OutputFile Name (default: Standard Out)
  --dry-run             Do not perform the expensive computations
```

The first two required arguments are a COB and one of two co-expression scores. These scores are defined in depth in the Camoco manuscript, but briefly here, *density* is the *mean, unthresholded co-expression among input genes* while *locality* calculates the *mean proportion of thresholded co-exression interactions between input genes compared to the number of total interactions they have*. In essence, density measures how co-expressed a set of genes are while locality performs a correction for genes that have many interactions (i.e. genes that have lots of interactions are down-weighted).

After our first two positional arguments, we have a whole list of options. Lets start with a basic case. Lets calculate the co-expression using both scores on a set of random input genes.

### Calculating co-expression of a gene set

**22 Random Genes:** GRMZM2G338161,GRMZM2G067943,GRMZM5G859099,GRMZM2G127050,GRMZM2G122498,GRMZM2G392798,GRMZM2G096585,GRMZM2G012280,GRMZM5G844080,GRMZM2G160351,GRMZM2G395535,GRMZM2G176576,GRMZM2G151873,GRMZM2G479596,GRMZM2G058910,GRMZM2G164649,GRMZM2G127101,GRMZM2G043396,GRMZM2G132780,AC189750.4_FG004,GRMZM2G108090,AC194970.5_FG009

```
camoco overlap \
  ZmRoot \
  density \
  --genes GRMZM2G338161,GRMZM2G067943,GRMZM5G859099,GRMZM2G127050,GRMZM2G122498,
↪GRMZM2G392798,GRMZM2G096585,GRMZM2G012280,GRMZM5G844080,GRMZM2G160351,GRMZM2G395535,
↪GRMZM2G176576,GRMZM2G151873,GRMZM2G479596,GRMZM2G058910,GRMZM2G164649,GRMZM2G127101,
↪GRMZM2G043396,GRMZM2G132780,AC189750.4_FG004,GRMZM2G108090,AC194970.5_FG009

[LOG] Thu Nov 15 08:20:46 2018 - Loading Expr table
[LOG] Thu Nov 15 08:20:46 2018 - Building Expr Index
[LOG] Thu Nov 15 08:20:46 2018 - Loading RefGen
[LOG] Thu Nov 15 08:20:46 2018 - Building Indices
[LOG] Thu Nov 15 08:20:46 2018 - Loading Coex table
[LOG] Thu Nov 15 08:20:48 2018 - Loading Global Degree
[LOG] Thu Nov 15 08:20:48 2018 - Loading Clusters
```

(continues on next page)

(continued from previous page)

```
[LOG] Thu Nov 15 08:20:48 2018 - Building Indices
[LOG] Thu Nov 15 08:20:48 2018 -  ---------- Calculating overlap for 0 of 1 Terms
[LOG] Thu Nov 15 08:20:48 2018 - Generating SNP-to-gene mapping
[LOG] Thu Nov 15 08:20:48 2018 - Calculating Overlap for CustomTerm of GeneList in␣
↪ZmRoot with window:1 and flank:0 (22 Loci)
[LOG] Thu Nov 15 08:20:48 2018 - Generating bootstraps
[LOG] Thu Nov 15 08:20:50 2018 - Iteration: 50 -- current pval: 0.58 0.58% complete
[LOG] Thu Nov 15 08:20:52 2018 - Iteration: 100 -- current pval: 0.59 1.18% complete
[LOG] Thu Nov 15 08:20:52 2018 - Calculating Z-Scores
[LOG] Thu Nov 15 08:20:52 2018 - Calculating FDR
[LOG] Thu Nov 15 08:20:52 2018 - Overlap Score (density): -0.4020806831780932 (p<0.59)
```

Lets break down the output. We specified the *ZmRoot* network here, so Camoco fetched the COB from the database. While it took several minutes to build the network, but only 2 seconds to load from the database. Nice! Additionally, Camoco loaded all the necessary information to perform this calculation. Notice how we did not specify a RefGen. Camoco used the one that was assigned to it when it was built.

Next, Camoco performed SNP-to-Gene mapping. In this case, it was a no-op meaning that the mapping window size was 1 so only the input genes were included. Next, Camoco calculates the density between the input genes as well as to randomized *bootstraps* which are gene sets of the same size as our input. The p-value stabilizes after around 100 bootstraps then the score is reported. This input set of genes has a density of -0.40 and 60% of the randomized bootstraps had a density that was greater (i.e. more extreme) than the input set. Based on this, the co-expression among these random 22 genes seems to indeed be random.

Lets look at the locality.

```
camoco overlap \
  ZmRoot \
  locality \
  --genes GRMZM2G338161,GRMZM2G067943,GRMZM5G859099,GRMZM2G127050,GRMZM2G122498,
↪GRMZM2G392798,GRMZM2G096585,GRMZM2G012280,GRMZM5G844080,GRMZM2G160351,GRMZM2G395535,
↪GRMZM2G176576,GRMZM2G151873,GRMZM2G479596,GRMZM2G058910,GRMZM2G164649,GRMZM2G127101,
↪GRMZM2G043396,GRMZM2G132780,AC189750.4_FG004,GRMZM2G108090,AC194970.5_FG009

[LOG] Thu Nov 15 08:29:42 2018 - Loading Expr table
[LOG] Thu Nov 15 08:29:42 2018 - Building Expr Index
[LOG] Thu Nov 15 08:29:42 2018 - Loading RefGen
[LOG] Thu Nov 15 08:29:42 2018 - Building Indices
[LOG] Thu Nov 15 08:29:42 2018 - Loading Coex table
[LOG] Thu Nov 15 08:29:43 2018 - Loading Global Degree
[LOG] Thu Nov 15 08:29:43 2018 - Loading Clusters
[LOG] Thu Nov 15 08:29:43 2018 - Building Indices
[LOG] Thu Nov 15 08:29:43 2018 -  ---------- Calculating overlap for 0 of 1 Terms
[LOG] Thu Nov 15 08:29:43 2018 - Generating SNP-to-gene mapping
[LOG] Thu Nov 15 08:29:43 2018 - Calculating Overlap for CustomTerm of GeneList in␣
↪ZmRoot with window:1 and flank:0 (22 Loci)
[LOG] Thu Nov 15 08:29:43 2018 - Generating bootstraps
[LOG] Thu Nov 15 08:29:47 2018 - Iteration: 50 -- current pval: 0.98 0.98% complete
[LOG] Thu Nov 15 08:29:50 2018 - Iteration: 100 -- current pval: 0.97 1.94% complete
[LOG] Thu Nov 15 08:29:50 2018 - Calculating Z-Scores
[LOG] Thu Nov 15 08:29:50 2018 - Calculating FDR
[LOG] Thu Nov 15 08:29:51 2018 - Overlap Score (locality): -0.05498542488574266 (p<0.
↪97)
```

About the same results. Random co-expression for random genes. No real surprise here. How does this compare to something that exhibits strong co-expression? Lets perform the same two commands, but this time we will look at a non-random set of genes.

### Calculating co-expression on a GO term

**GO:0006270** Name: DNA replication initiation, Desc: "The process in which DNA-dependent DNA replication is started; this involves the separation of a stretch of the DNA double helix, the recruitment of DNA polymerases and the initiation of polymerase action." [ISBN:071673706X, ISBN:0815316194], 22 Loci GRMZM2G066101,GRMZM2G021069,GRMZM2G126453,GRMZM2G075584, GRMZM2G162445,GRMZM2G327032,GRMZM2G092778,GRMZM2G062333, GRMZM2G065205,GRMZM2G075978,GRMZM2G082131,GRMZM2G450055, GRMZM2G163658,GRMZM2G130239,GRMZM2G160664,GRMZM2G139894, GRMZM2G104373,GRMZM2G158136,GRMZM2G126860,GRMZM2G157621, GRMZM2G100639,GRMZM2G112074

This information was pulled from the GO maize gene assignments. Its a set of 22 genes that are annotated to be involved with the initialization of DNA replication.

Now, we could copy and paste the gene IDs for this GO term into the CLI, but Camoco already knows information about this GO term because we built the GO database. We can change our command options to tell Camoco where to get this information.

```
camoco overlap \
  ZmRoot \
  density \
  --go ZmGO --term GO:0006270
[LOG] Thu Nov 15 08:37:17 2018 - Loading Expr table
[LOG] Thu Nov 15 08:37:17 2018 - Building Expr Index
[LOG] Thu Nov 15 08:37:17 2018 - Loading RefGen
[LOG] Thu Nov 15 08:37:17 2018 - Building Indices
[LOG] Thu Nov 15 08:37:17 2018 - Loading Coex table
[LOG] Thu Nov 15 08:37:18 2018 - Loading Global Degree
[LOG] Thu Nov 15 08:37:18 2018 - Loading Clusters
[LOG] Thu Nov 15 08:37:18 2018 - Building Indices
[LOG] Thu Nov 15 08:37:18 2018 - Building Indices
[LOG] Thu Nov 15 08:37:18 2018 -  ---------- Calculating overlap for 0 of 1 Terms
[LOG] Thu Nov 15 08:37:18 2018 - Generating SNP-to-gene mapping
[LOG] Thu Nov 15 08:37:18 2018 - Calculating Overlap for GO:0006270 of ZmGO in ZmRoot␣
↪with window:1 and flank:0 (22 Loci)
[LOG] Thu Nov 15 08:37:19 2018 - Generating bootstraps
[LOG] Thu Nov 15 08:37:21 2018 - Iteration: 50 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:23 2018 - Iteration: 100 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:25 2018 - Iteration: 150 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:27 2018 - Iteration: 200 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:29 2018 - Iteration: 250 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:31 2018 - Iteration: 300 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:33 2018 - Iteration: 350 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:35 2018 - Iteration: 400 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:37 2018 - Iteration: 450 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:39 2018 - Iteration: 500 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:41 2018 - Iteration: 550 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:44 2018 - Iteration: 600 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:46 2018 - Iteration: 650 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:48 2018 - Iteration: 700 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:50 2018 - Iteration: 750 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:52 2018 - Iteration: 800 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:54 2018 - Iteration: 850 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:57 2018 - Iteration: 900 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:37:59 2018 - Iteration: 950 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:38:01 2018 - Iteration: 1000 -- current pval: 0.0 0.0% complete
[LOG] Thu Nov 15 08:38:02 2018 - Calculating Z-Scores
```

(continues on next page)

(continued from previous page)

```
[LOG] Thu Nov 15 08:38:02 2018 - Calculating FDR
[LOG] Thu Nov 15 08:38:15 2018 - Overlap Score (density): 19.553358790010673 (p<0.0)
```

We can just specify the name of the GO Ontology object we build, *ZmGO*, as well as the name of the term, *GO:0006270*. Again, Camoco will fetch the information and start computing. We can indeed see that with the default gene mapping, we pull out the 22 genes we expected. Camoco calculates a density of 19.5 for this gene set, much higher than our random set of genes above (-0.40). This time, Camoco performs 1000 randomized bootstraps and we find that **none** of the random sets of genes has a density that can beat our GO term.

Checking locality is as easy as swapping out the command

```
camoco overlap \
  ZmRoot \
  locality \
  --go ZmGO \
  --term GO:0006270
[.. truncated ...]
[LOG] Thu Nov 15 08:48:25 2018 - Overlap Score (locality): 0.1513447281613 (p<0.004)
```

Here the p-value is still significant, but we see that 4 of the 1000 randomized bootstraps showed a locality score greater than 0.15.

### Calculating co-expression among all GO terms

Modifying the above commands just slightly will instruct Camoco to iterate over all the terms in an Ontology.

---

**Note:** Calculating co-expression on a full Ontology can result in a **lot** of computation!

---

If an Ontology is specified with no term, Camoco calculates the co-expression on all the terms in the Ontology that meet the filtering criteria in the command options.

```
$ camoco overlap \
  ZmRoot \
  locality \
  --go ZmGO
```

This command will calculate the co-expression of *all* terms in the Ontology! We can refine our calculation in several different ways. The first, is we can specify a list of terms instead of a single one.

```
$ camoco overlap \
  ZmRoot \
  density \
  --go ZmGO \
  --terms GO:0006270 GO:0004812 GO:0006481
```

This will calculate the density of three GO terms. While parsing out the results of one or three terms is doable, scrolling back and extracting the information from hundreds or even thousands of terms is unmanageable. Using the *–out* flag, Camoco will print an expanded results table to an output file.

```
$ camoco overlap \
  ZmRoot \
  locality \
  --go ZmGO \
  --out ZmGO_overlap_results
```

The same information will be printed to the screen to allow you to follow along and track progress, but in addition, an output file is created not only with the Term co-expression results, but also a breakdown of each gene's contribution.

```
$ head ZmGO_overlap_results.overlap.tsv
gene    local   global  score   fitted  zscore  fdr     num_real        num_random
↪  bs_mean bs_std [..]
GRMZM2G075978   9       242     6.092622797537619       2.907377202462381       43.
↪2248805848083       [..]
GRMZM2G327032   10      449     4.605734033447897       5.394265966552103       32.
↪65972351674033      [..]
GRMZM2G062333   10      675     1.8905801170987306      8.10941988290127        13.
↪367071448197386     [..]
GRMZM2G162445   10      776     0.6771706235090598      9.32282937649094        4.
↪745134019335283      [..]
GRMZM2G450055   9       741     0.09765906188171769     8.902340938118282       0.
↪6273877432024378     [..]
GRMZM2G163658   0       2       -0.024027910764151908   0.024027910764151908    -0.
↪23726468419577254   [..]
[..truncated..]
```

Similarly, you can control the command using *–min-term-size* and *–max-term-size*. These options allow you to filter out the Terms so that overlap is only calculated on terms that are the right size. In the case we'd want to limit out analysis to terms with less than 20 and more than 10 genes we'd run:

```
$ camoco overlap \
  ZmRoot \
  locality \
  --go ZmGO \
  --min-term-size 10 \
  --max-term-size 20
```

### Calculating co-expression on GWAS traits

GWAS traits are similar to Ontology terms except that SNPs are mapped to genes. The loci that are stored in the GWAS terms are slightly different in that they encode SNPs and not genes. To map them to genes a simple window based method is used with additional flanking genes added. For example, if a 50 kb window and 1 flanking gene is specified, a 50kb up and 50kb down (100kb total) window is calculated around the SNP and an additional 1 flanking gene outside the window is used. The flanking gene allows for nearest genes to be utilized if the window does not cover any genes. These options are specified in the *overlap* command.

```
$ camoco overlap \
  ZmRoot \
  density \
  --gwas ZmIonome \
  --term Al27 \
  --candidate-window-size 50000 \
  --candidate-flank-limit 1
```

Again, to avoid calculating the co-expression for **all** GWAS terms (i.e. traits) we specify a *–term* with the option *Al27* to calculate co-expression for the Aluminum GWAS. The output looks like:

```
[LOG] Thu Nov 15 23:07:14 2018 - Loading Expr table
[LOG] Thu Nov 15 23:07:14 2018 - Building Expr Index
[LOG] Thu Nov 15 23:07:14 2018 - Loading RefGen
[LOG] Thu Nov 15 23:07:14 2018 - Building Indices
[LOG] Thu Nov 15 23:07:14 2018 - Loading Coex table
```

(continues on next page)

(continued from previous page)

```
[LOG] Thu Nov 15 23:07:15 2018 - Loading Global Degree
[LOG] Thu Nov 15 23:07:15 2018 - Loading Clusters
[LOG] Thu Nov 15 23:07:15 2018 - Building Indices
[LOG] Thu Nov 15 23:07:15 2018 - Building Indices
[LOG] Thu Nov 15 23:07:15 2018 -  ---------- Calculating overlap for 0 of 1 Terms
[LOG] Thu Nov 15 23:07:15 2018 - Generating SNP-to-gene mapping
[LOG] Thu Nov 15 23:07:15 2018 - Al27: Found 176 SNPs -> 149 effective SNPs with␣
↪window size 50000 bp
[LOG] Thu Nov 15 23:07:15 2018 - Calculating Overlap for Al27 of ZmIonome in ZmRoot␣
↪with window:50000 and flank:1 (149 Loci)
[LOG] Thu Nov 15 23:07:16 2018 - Generating bootstraps
[LOG] Thu Nov 15 23:07:35 2018 - Iteration: 50 -- current pval: 0.82 0.82% complete
[LOG] Thu Nov 15 23:07:54 2018 - Iteration: 100 -- current pval: 0.8 1.6% complete
[LOG] Thu Nov 15 23:07:54 2018 - Calculating Z-Scores
[LOG] Thu Nov 15 23:07:54 2018 - Calculating FDR
[LOG] Thu Nov 15 23:07:54 2018 - Overlap Score (density): -1.1740572539338927 (p<0.8)
```

In this case the p-value indicates there is not a significant amount of co-expression among the genes near the GWAS SNPs.

### 2.3.7 Conclusions

This was a whirlwind walk-through on some of the computations that can be done with Camoco. Please direct any inquiries/comments/suggestions to out GitHub repository.

### 2.3.8 Exercise Solutions

#### ZmSAM Network

To build the ZmSAM network:

```
$ camoco build-cob \
  Stelpflug2018_B73_Tissue_Atlas.txt \
  ZmSAM \
  "Tissue Devel Atlas" \
  Zm5bFGS \
  --max-val 250
```

In this case *–max-val* needs to be specified since some of the expression data has a low maximum value. It was hand checked that it was valid data.

#### ZmPAN Network

To build the ZmPAN network:

```
$ camoco build-cob \
  Hirsch2014_PANGenomeFPKM.txt \
  ZmPAN \
  "Maize PAN Genome (Hirsch et al.)" \
  Zm5bFGS \
  --sep=','
```

In this case, the data was separated by commas, so a *–sep* option was needed.

### ZmWallace GWAS Dataset

To build the ZmWallace GWAS dataset specify the filename along with the column names. The head of the file looks like:

```
$ zcat Wallace_etal_2014_PLoSGenet_GWAS_hits-150112.txt.gz | head
trait   chr     pos     allele  rmip    source
100 Kernel weight       1       3364007 A/G     1       Hapmap1
100 Kernel weight       1       22247033        A/G     3       Hapmap1
100 Kernel weight       1       22987420        C/T     1       Hapmap2
100 Kernel weight       1       23056483        C/G     1       Hapmap2
100 Kernel weight       1       23066099        A/G     1       Hapmap2
100 Kernel weight       1       23097979        T/C     60      Hapmap2
100 Kernel weight       1       23099013        C/A     1       Hapmap2
100 Kernel weight       1       23401419        G/A     1       Hapmap2
100 Kernel weight       1       23478001        win2k-  1       CNV_edgeR

$ zcat Wallace_etal_2014_PLoSGenet_GWAS_hits-150112.txt.gz| wc -l
38421
```

The file is compressed, so the *zcat* command is needed to pipe the first few lines into the *head* command. The trait is in the *trait* column, the chromosome and position are in columns *chr* and *pos* respectively. Using the *wc -l* command, we can see that there are 38,420 SNPS (and 1 header) in the file. In Wallace et al., they report that there were ~4,800 significant SNPs in their dataset. Upon closer inspection of the publication, we see that they only considered SNPs with a RMIP of 5 or above, however in this file, SNPs with RMIPs of 1 or above are listed.

---

**Note:** RMIP is a method for setting a significance threshold. See more here

---

We need to do some data wrangling to filter out the SNPs with an RMIP below 5. It's up to the researcher to determine the significance threshold cutoff used for the overlap analysis. Camoco can tolerate a certain level of noise, however it is assumed that the majority of SNPs used in the analysis are near a causal gene. It's recommended that the input SNP list only contains significant SNPs (or an FDR the researcher is comfortable with).

The *build-gwas* help command has the following flag that allows for filtering to be done using a pandas data frame query.

```
$ camoco build-gwas --help

[ ... Truncated ... ]

--query QUERY         Before importing the traits from the dataframe, filter
                      the columns via DataFrame.query() e.g. --query "pval <
                      0.05" would execute df.query("pval < 0.05") assuming
                      the pval column exists. Useful for filtering data
                      before importing the traits. See
                      https://pandas.pydata.org/pandas-
                      docs/stable/generated/pandas.DataFrame.query.html for
                      more info.
```

We can build the dataset using the following camoco command:

```
$ camoco build-gwas Wallace_etal_2014_PLoSGenet_GWAS_hits-150112.txt.gz \
   "ZmWallaceRMIP5" \
   "Wallace Maize data with RMIP 5" \
   Zm5bFGS \
   --trait-col trait \
```

(continues on next page)

(continued from previous page)

```
   --chrom-col chr \
   --pos-col pos \
   --query "rmip >= 5"

[LOG] Fri Jun 28 15:08:46 2019 - Building Indices
[LOG] Fri Jun 28 15:08:46 2019 - Building Indices
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: 100 Kernel weight, Desc: , 109 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Anthesis-silking interval, Desc: ,␣
↪128 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Average internode length (above ear),
↪ Desc: , 166 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Average internode length (below ear),
↪ Desc: , 231 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Average internode length (whole␣
↪plant), Desc: , 196 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Boxcox-transformed leaf angle, Desc:␣
↪, 159 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Chlorophyll A, Desc: , 34 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Chlorophyll B, Desc: , 58 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Cob diameter, Desc: , 153 Loci
[LOG] Fri Jun 28 15:08:46 2019 - Importing Term: Days to anthesis, Desc: , 189 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Days to silk, Desc: , 178 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Ear height, Desc: , 200 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Ear row number, Desc: , 164 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Fructose, Desc: , 20 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Fumarate, Desc: , 5 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Glucose, Desc: , 42 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Glutamate, Desc: , 29 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Height above ear, Desc: , 162 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Height per day (until flowering),␣
↪Desc: , 181 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Leaf length, Desc: , 171 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Leaf width, Desc: , 186 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Malate, Desc: , 43 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Nitrate, Desc: , 54 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Nodes above ear, Desc: , 138 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Nodes per plant, Desc: , 188 Loci
[LOG] Fri Jun 28 15:08:47 2019 - Importing Term: Nodes to ear, Desc: , 170 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Northern Leaf Blight, Desc: , 135␣
↪Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: PCA of metabolites: PC1, Desc: , 36␣
↪Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: PCA of metabolites: PC2, Desc: , 54␣
↪Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Photoperiod Growing-degree days to␣
↪silk, Desc: , 80 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Photoperiod growing-degree days to␣
↪anthesis, Desc: , 95 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Plant height, Desc: , 201 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Protein, Desc: , 27 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Ratio of ear height to total height,␣
↪Desc: , 184 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Southern leaf blight, Desc: , 160␣
↪Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Stalk strength, Desc: , 87 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Starch, Desc: , 72 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Sucrose, Desc: , 27 Loci
```

(continues on next page)

(continued from previous page)

```
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Tassel branch number, Desc: , 213␣
↪Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Tassel length, Desc: , 194 Loci
[LOG] Fri Jun 28 15:08:48 2019 - Importing Term: Total amino acids, Desc: , 92 Loci
Build Successful:
Ontology:ZmWallaceRMIP5 - desc: Wallace Maize data with RMIP 5 - contains 41 terms␣
↪for Reference Genome: zea mays - 5b - Zm5bFGS
```

We can see that the ZmWallace dataset contains 41 GWAS "Terms" with between 5 and 231 SNPs. In this case, an additional call to the *query* flag filtered out SNPs in the input file based on a criteria defined by the user.

## 2.4 Strengths and Limitations of Camoco

Camoco excels at building and integrating different genomic data types due to its internal architecture and design. This *build once, use many times* philosophy allows you to effectively explore your datasets and to design experiments that utilize and compare different input parameters and options.

Camoco also straddles the divide between being high performance and high utility. Being written in Python, Camoco is very flexible in how it can be used. Critical parts of the code leverage optimized numpy/scipy/pandas code and several crucial parts of Camoco are written in Cython to optimize performance.

The CLI provides basic functionality, but using Camoco interactively through IPython or by importing camoco as a module allows for more sophisticated and specialized scripts to be written. Scripts are typically straight forward to write as Camoco shares a python API similar to that of numpy/scipy/pandas.

Camoco was built and designed with a modern Python workflow in mind. These Due to some design choices and dependencies, Camoco has some limitations that make it difficult to run on some systems / platforms.

***Older python releases (<3.6)*** Several newer features released in newer versions of Python are heavily used internally within Camoco. Requiring Python versions 3.6+ may conflict the system installed version of Python. Its recommended that Camoco is installed using a python virtual environment. We personally have had success with miniconda which can be installed here. See the installation documentation for more details.

***Network file systems*** Camoco heavily utilizes SQLite for data persistence. As SQLite is file based and does not support concurrent writing. Some network based file systems that are commonly used within academic departments and super computer institutes (e.g. NFS) do not play well with SQLite and can corrupt the underlying database. By default, Camoco stores its databases in *~/.camoco*. If you are running Camoco from a computer that utilizes a network file system, it is recommended that you create a symbolic link from *~/.camoco* to a local (i.e. non-network) directory that is stored somewhere else on the machine.

***Parallelizing Camoco*** Being written in Python means that Camoco is limited by the GIL. Long story short, Camoco is inherently single-threaded. This issue is slowly being addressed by converting some code to utilize the newer python async functionality. Until that is fully supported, the easiest way to parallelize your Camoco workflow is by running many instances of Camoco from the command line. As Camoco leverages databases (which are not as limited by parallelization as python), running many instances of Camoco in parallel is feasible. For more a short tutorial showcasing this, please read this blog post.

CHAPTER

THREE

# INDICES AND TABLES

- genindex
- search