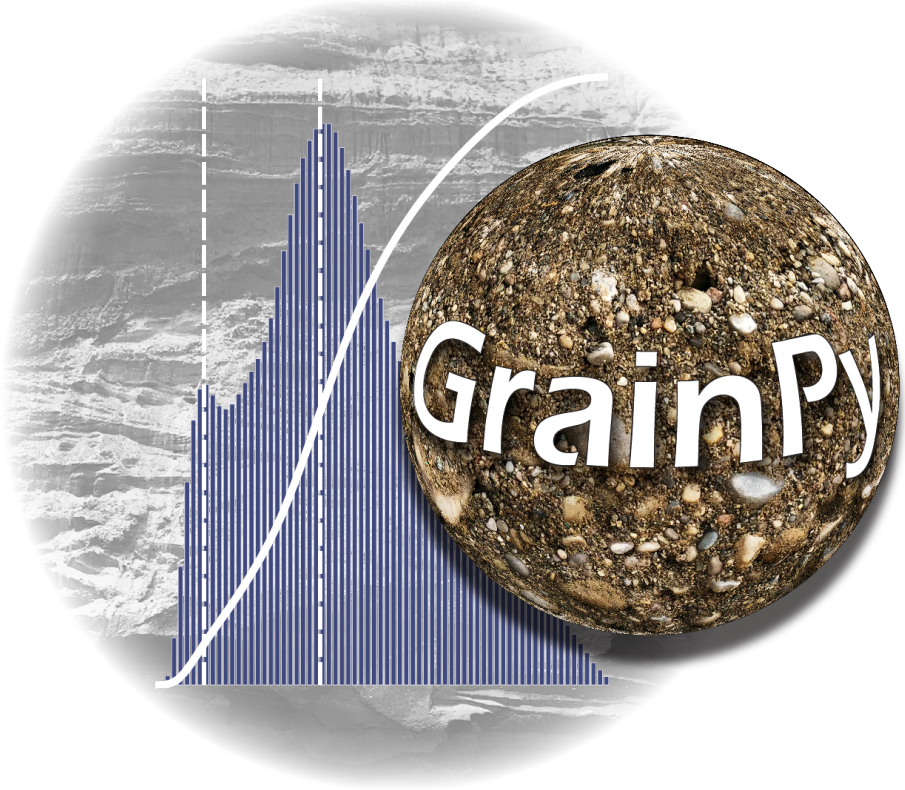
# GrainPy

*Release v0.1.0*

**Matt Massey**

**May 01, 2022**

# GETTING STARTED




GrainPy is a Python package for compiling, analyzing, visualizing, and interpreting grain size distribution data. The idea for GrainPy started with an abundance of grain size distribution data that wasn't being fully utilized. Initially, GrainPy was developed to quickly compile analyses of multiple samples, analyze results with geologic statistics and figures, and produce publication-quality grain size distribution plots. GrainPy will continue to add functionality, develop new methodologies of grain size distribution analysis and interpretation, and provide a user-friendly experience for sediment analyses.

CHAPTER

ONE

# INSTALLATION

## 1.1 Create a Virtual Environment

It is common practice to set up and work within virtual environments, as it helps avoid potential dependency conflicts, and represents good practice. It is also strongly recommended to do that here with GrainPy. GrainPy was built and continues to be developed using Conda and Anaconda environment, although it is not required and we present several command line procedures below for creating and then activating your new virtual environment.

**Conda**

```
# create
conda create -n CoolNewGrainPyVirtualEnvironment

# activate
conda activate CoolNewGrainPyVirtualEnvironment
```

**Mac OSX Terminal**

```
# create
python3 -m venv CoolNewGrainPyVirtualEnvironment

# activate
source CoolNewGrainPyVirtualEnvironment/bin/activate
```

**Windows Command Prompt**

```
# create
py -m venv CoolNewGrainPyVirtualEnvironment

# activate
.\CoolNewGrainPyVirtualEnvironment\Scripts\activate
```

## 1.2 Install the GrainPy Package

Once your virtual environment is created and activated, installing GrainPy is similar across each of the three platforms discussed above. Each released version of GrainPy is also indexed on PyPi, and it is recommended to install this package using pip.

```
python3 -m pip install grainpy
# Note for Windows: "python3" should be substituted with "py"
```

Congratulations! the most current stable release of GrainPy is now installed and ready to use!

### 1.2.1 Developmental Version

In some cases, the user may want to install the current working version of GrainPy, which can be done directly from Github.

```
python3 -m pip install git+https://github.com/masseygeo/GrainPy@main
# Note for Windows: "python3" should be substituted with "py"
```

Congratulations! the most current, developmental version of GrainPy is now installed and ready to use! If you experience bugs, please create a new issue on Github with the appropriate label

CHAPTER

TWO

# GRAINPY FUNDAMENTALS

## 2.1 Input

Grain size distribution data can be obtained from a variety of equipment, including sieves, hydrometers, and laser diffraction particle size analyzers, however, there are a few basic requirements for the input data structure in order to utilize GrainPy:

1. Grain size distribution data must be an Excel file (.xlsx or .xls).
2. Each file contains data for one sample only.
3. Bins used in the user's methodology are located in a column immediately preceding the data collected.
4. Bin sizes are in microns.
5. Bins are arranged from smallest to largest.
6. Bins represent the lower limit of each grain size interval, and the last row contains the maximum *UPPER* limit (therefore there is one more row of bins than data.

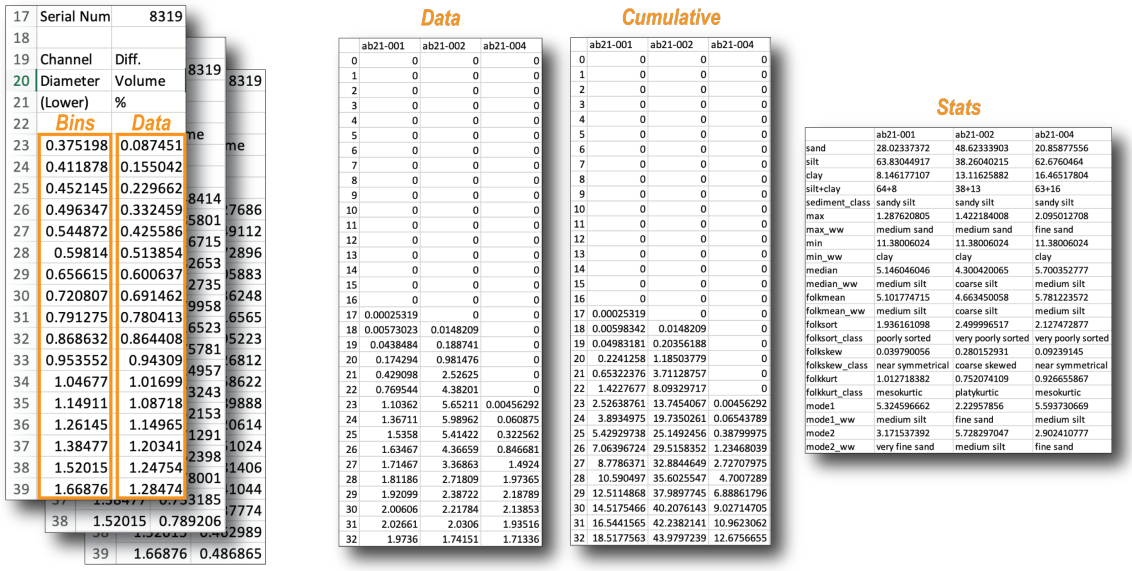
## 2.2 Compilation

Given the correct input format, GrainPy can easily collect and organize single or multiple files, and calculate cumulative percentages and a variety of sample statistics.

```
# list or tuple of complete paths to files
files = ['path to file 1', ..., 'path to file n']

# compile data, cumulative proportions, and statistics from file(s)
var = GrainSizeDist(files)
var.data()
var.datacp()
var.datast()
```

The figures above show an example of three Excel files on the left, with bins in one column and data in the following column. After these files have been input into GrainPy, the user can see the compiled data, cumulative percentages, and data statistics as shown on the right.



| | | |
|---|---|---|
| 17 | Serial Num | 8319 |
| 18 | | |
| 19 | Channel | Diff. |
| 20 | Diameter | Volume |
| 21 | (Lower) | % |
| 22 | Bins | Data |
| 23 | 0.375198 | 0.087451 |
| 24 | 0.411878 | 0.155042 |
| 25 | 0.452145 | 0.229662 |
| 26 | 0.496347 | 0.332459 |
| 27 | 0.544872 | 0.425586 |
| 28 | 0.59814 | 0.513854 |
| 29 | 0.656615 | 0.600637 |
| 30 | 0.720807 | 0.691462 |
| 31 | 0.791275 | 0.780413 |
| 32 | 0.868632 | 0.864408 |
| 33 | 0.953552 | 0.94309 |
| 34 | 1.04677 | 1.01699 |
| 35 | 1.14911 | 1.08718 |
| 36 | 1.26145 | 1.14965 |
| 37 | 1.38477 | 1.20341 |
| 38 | 1.52015 | 1.24754 |
| 39 | 1.66876 | 1.28474 |

Data

| | ab21-001 | ab21-002 | ab21-004 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 |
| 17 | 0.00025319 | 0 | 0 |
| 18 | 0.00573023 | 0.0148209 | 0 |
| 19 | 0.0438484 | 0.188741 | 0 |
| 20 | 0.174294 | 0.981476 | 0 |
| 21 | 0.429098 | 2.52625 | 0 |
| 22 | 0.769544 | 4.38201 | 0 |
| 23 | 1.10362 | 5.65211 | 0.00456292 |
| 24 | 1.36711 | 5.98962 | 0.060875 |
| 25 | 1.5358 | 5.41422 | 0.322562 |
| 26 | 1.63467 | 4.36659 | 0.846681 |
| 27 | 1.71467 | 3.36863 | 1.4924 |
| 28 | 1.81186 | 2.71809 | 1.97365 |
| 29 | 1.92099 | 2.38722 | 2.18789 |
| 30 | 2.00606 | 2.21784 | 2.13853 |
| 31 | 2.02661 | 2.0306 | 1.93516 |
| 32 | 1.9736 | 1.74151 | 1.71336 |

Cumulative

| | ab21-001 | ab21-002 | ab21-004 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 |
| 17 | 0.00025319 | 0 | 0 |
| 18 | 0.00598342 | 0.0148209 | 0 |
| 19 | 0.04983181 | 0.20356188 | 0 |
| 20 | 0.2241258 | 1.18503779 | 0 |
| 21 | 0.65322376 | 3.71128757 | 0 |
| 22 | 1.4227677 | 8.09329717 | 0 |
| 23 | 2.52638761 | 13.7454067 | 0.00456292 |
| 24 | 3.8934975 | 19.7350261 | 0.06543789 |
| 25 | 5.42929738 | 25.1492456 | 0.38799975 |
| 26 | 7.06396724 | 29.5158352 | 1.23468039 |
| 27 | 8.7786371 | 32.8844649 | 2.72707975 |
| 28 | 10.590497 | 35.6025547 | 4.7007289 |
| 29 | 12.5114868 | 37.9897745 | 6.88861796 |
| 30 | 14.5175466 | 40.2076143 | 9.02714705 |
| 31 | 16.5441565 | 42.2382141 | 10.9623062 |
| 32 | 18.5177563 | 43.9797239 | 12.6756655 |

Stats

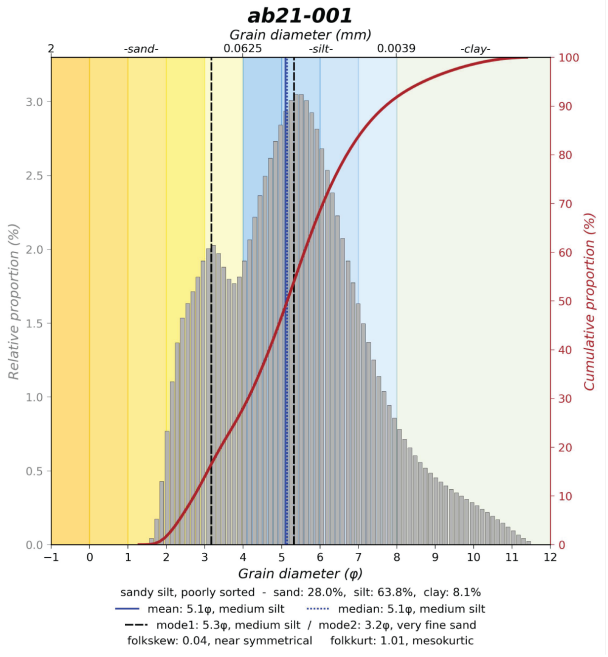
| | ab21-001 | ab21-002 | ab21-004 |
|---|---|---|---|
| sand | 28.02337372 | 48.62333903 | 20.85877556 |
| silt | 63.83044917 | 38.26040215 | 62.6760464 |
| clay | 8.146177107 | 13.11625882 | 16.46517804 |
| silt+clay | 64+8 | 38+13 | 63+16 |
| sediment_class | sandy silt | sandy silt | sandy silt |
| max | 1.287620805 | 1.422184008 | 2.095012708 |
| max_ww | medium sand | medium sand | fine sand |
| min | 11.38006024 | 11.38006024 | 11.38006024 |
| min_ww | clay | clay | clay |
| median | 5.146046046 | 4.300420065 | 5.700352777 |
| median_ww | medium silt | coarse silt | medium silt |
| folkmean | 5.101774715 | 4.663450058 | 5.781223572 |
| folkmean_ww | medium silt | coarse silt | medium silt |
| folksort | 1.936161098 | 2.499996517 | 2.127472877 |
| folksort_class | poorly sorted | very poorly sorted | very poorly sorted |
| folkskew | 0.039790056 | 0.280152931 | 0.09239145 |
| folkskew_class | near symmetrical | coarse skewed | near symmetrical |
| folkkurt | 1.012718382 | 0.752074109 | 0.926655867 |
| folkkurt_class | mesokurtic | platykurtic | mesokurtic |
| mode1 | 5.324596662 | 2.22957856 | 5.593730669 |
| mode1_ww | medium silt | fine sand | medium silt |
| mode2 | 3.171537392 | 5.728297047 | 2.902410777 |
| mode2_ww | very fine sand | medium silt | fine sand |

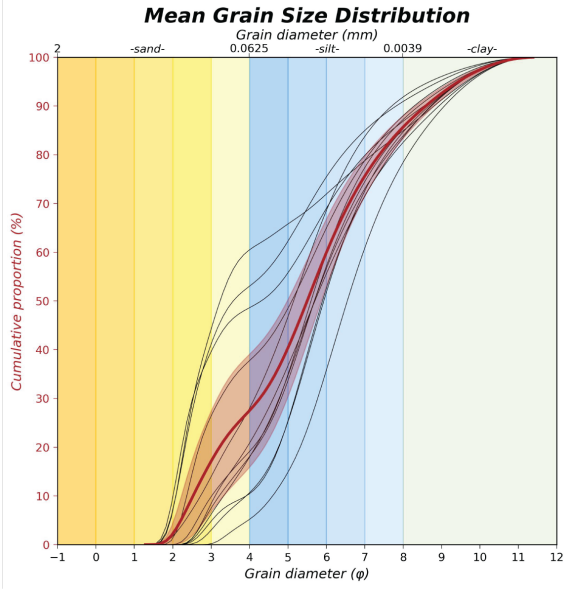
## 2.3 Visualization

Data can then be visualized and interpreted with publication-quality grain size distribution plots.

```
# plots of individual files
var.gsd_single()

# plot of all files
var.gsd_multi()
```




The two figures above show a GrainPy plot for a single sample (left) and multiple samples (right). The single sample plot shows a histogram of relative proportions within each bin, a cumulative proportion curve, mean, median, and

mode(s) of the sample (vertical lines), and selected statistics below the plot. The multiple sample plot shows cumulative proportion curves for all samples (black lines), the mean cumulative proportion curve (dark red lines), and a 95% confidence interval of the mean (translucent red polygon). GrainPy has multiple options for plots, which are discussed in detail in the Plotting section.

CHAPTER

# THREE

# THE 'GRAINSIZE' MODULE

The **grainsize** module contains the *GrainSizeDist* class, which is the fundamental GrainPy object. The *GrainSizeDist* object contains a variety of attributes and methods to aid the user with data compilation and visualization.

## 3.1 'path' Attribute

The *path* attribute is the only reqquirement for creating a *GrainSizeDist* object, and consists of either a list or tuple of path(s) for the file(s) containing the grain size distribution data. This parameter can be input manually, or interactively using the selectdata function from the util module.

```
# selectdata function opens a user dialog window to select files interactively
files = selectdata()

# create a new instance of the GrainSizeDist class with the selected files
var = GrainSizeDist(files)
```

## 3.2 'lith' & 'area' Attributes

The *lith* and *area* attributes are optional. Their intent is to provide a means to differentiate GrainSizeDist objects by lithology and/or location. Currently, these are only used for grain size distribution plot titles.

```
# using GrainSizeDist instance from above, assign attributes after instantiation
var.lith = 'alluvium'
var.area = 'Lebanon Junction'

# alternatively, assign attributes at time of instantiation
var = GrainSizeDist(files, lith='alluvium', area='Lebanon Junction')
```

## 3.3 'bins' Method

The *bins* method returns a dataframe of the bin intervals in phi units, microns, and millimaters.

```
# using the GrainSizeDist instance from above
var.bins()
```

## 3.4 'data' Method

The *data* method returns a dataframe of the grain size distribution data compiled for all file(s) selected and input for a *GrainSizeDist* object.

```
# using the GrainSizeDist instance from above
var.data()
```

## 3.5 'datacp' Method

The *datacp* method returns a dataframe of the cumulative proportions compiled for all file(s) selected and input for a *GrainSizeDist* object.

```
# using the GrainSizeDist instance from above
var.datacp()
```

## 3.6 'datast' Method

The *datast* method returns a dataframe of statistics calculated from the data and cumulative proportions for all file(s) selected and input for a *GrainSizeDist* object.

```
# using the GrainSizeDist instance from above
var.datast()
```

## 3.7 'gsd_single' Method

The *gsd_single* method saves two image files of grain size distribution plots (.pdf and .jpg) in the directory where the sample files are located. The default for this method is to plot all samples, however, the user has the option to manually select sample(s) by either:

1. manually inputting a list of sample names using the *files* parameter
2. defining indices of a 'slice' from the *path* attribute using the *i* and *j* parameters

```
# grain size distribution plots of all samples
var.gsd_single()

# grain size distribution plots of two samples, 'file 1' and 'file 7'
var.gsd_single(files=['file 1', 'file 7'])
```

(continues on next page)

(continued from previous page)

```
# grain size distribution plots of first three samples included in a *GrainSizeDist*␣
↪object
var.gsd_single(i=0, j=3)
```

## 3.8 'gsd_multi' Method

The *gsd_multi* method saves two image files of grain size distribution plots (.pdf and .jpg) of multiple samples in the directory where the sample files are located. There are three plot options (1-3) and two additional options (4-5) to customize each plot according to user specifications:

1. Cumulative proportion curves of all files, with mean and 95% confidence interval. This is the default with parameters *bplt* = False, *cplt* = True, *stplt* = True, and *ci* = True.
2. Mean cumulative proportion curve with 95% confidence interval, and histogram of mean grain size distributions using the *bplt* parameter.
3. Histogram of mean grain size distribution per bin with 95% confidence interval, and grain size distributions per bin for each sample (as curves) using the *bplt* and *cplt* parameters.
4. Plotting mean, median, and modes, and selected statistics of mean in plot legend using the *stplt* parameter.
5. Plotting the 95% confidence interval using the *ci* parameter.

```
# option 1, default
var.gsd_multi()

# options 2 and 4
var.gsd_multi(bplt=True, stplt=False)

# options 3 and 5
var.gsd_multi(bplt=True, cplt=False, ci=False)
```

## 3.9 'samplenames' Method

The *samplenames* method assumes that the input file names represent the sample names and are unique. Both of these assumptions are not strictly required, but are used in other methods of the GrainSizeDist class, as well as in plot titles. Calling the *samplenames* method returns a list of the assumed sample names.

```
# using the GrainSizeDist instance from above
var.samplenames()
```

CHAPTER

FOUR

# THE 'CLASSIFY' MODULE

The **classify** module contains conversion functions, including conversion of numerical statistics into qualitative classifcation names. These functions are called within the *GrainSizeDist* class, but can also be used individually by the user.

## 4.1 The 'folk_sed' Function

The *folk_sed* function converts sand, silt, and clay proportions into the appropriate sediment classification scheme of Folk (1954).

```
folk_sed(20, 20, 60)
folk_sed(2.5, 90.2, 7.3)
```

- Folk, R.L., 1954, The Distinction between Grain Size and Mineral Composition in Sedimentary-Rock Nomenclature: Journal of Geology, Volume 62, Number 4, DOI: https://doi.org/10.1086/626171.

## 4.2 The 'folk_sort', 'folk_skew', & 'folk_kurt' Functions

The *folk_sort*, *folk_skew*, and *folk_kurt* functions classify sediment sorting, distribution skewness and distribution kurtosis into the appropriate classification schemes of Folk and Ward (1957).

```
folk_sort(1.35)
folk_skew(0.12)
folk_kurt(0.96)
```

- Folk, R.L., and Ward, W.C., 1957, Brazos River bar: a study in the significance of grain size parameters: Journal of Sedimentary Petrology, Volume 27, Number 1, DOI: https://doi.org/10.1306/74D70646-2B21-11D7-8648000102C1865D.

## 4.3 The 'wentworth_gs' Function

The *wentworth_gs* function converts grain sizes, in phi units, into the appropriate grain size classification scheme of Wentworth (1922).

```
wentworth_gs(-1)
wentworth_gs(3.6)
```


- Wentworth, C.K., 1922, A Scale of Grade and Class Terms for Clastic Sediments: Journal of Geology, Volume 30, Number 5, DOI: https://doi.org/10.1086/622910.

CHAPTER

# FIVE

# THE 'UTIL' MODULE

The **util** module provides functionalities called within the *GrainSizeDist* class, but may also be called individually to the user.

## 5.1 The 'datacheck' Function

The most common problem with input data is bin sizes are not consistent in all files. For example, one bin may be 0.375198, but some files may have been rounded inadvertently to 0.37520). The *datacheck* function checks the smallest expected bin value input as a function parameter by the user, displays the results of the datacheck, and offers the option to automatically fix the problematic files by changing them, or let's the user manually examing the file(s) themselves. We warn the user that the auto-fix solution is permanent and changes the original files!

```
# call function using "files" variable created above
# min_bin and bin_rows have default values, but may be changed accordingly by user
datacheck(files)
```

## 5.2 The 'df_ex' & 'gems_ex' Functions

The *df_ex* and *gems_ex* functions afford the user the option to export *GrainSizeDist* object data as tables (.csv or .xlsx).

The *df_ex* function requires a **dataframe** parameter, then saves that dataframe according to the chosen location/name from the interactive user-dialog window. Dataframes include returns of the *bins*, *data*, *datacp*, or *datast* methods, or any other type of non-GrainPy dataframe.

The *gems_ex* function requires a **GrainSizeDist** object parameter, then saves the data and selected statistics in a transposed format to a location/name from the interactive user-dialog window.

```
# df_ex function
# first export cumulative proportion dataframe
df_ex(gsd.datacp())

# then statistics dataframe from a GrainSizeDist object named 'gsd'
df_ex(gsd.datast())

# gems_ex function with a GrainSizeDist object named 'gsd'
gems_ex(gsd)
```

## 5.3 The 'selectdata' Function

The *selectdata* function provides an interactive user dialog window to manually select file(s). This provides a user-friendly option to select single or multiple Excel files (.xlsx or .xls), save a list of paths to the selected files, then can be used as a parameter for creating a GrainSizeDist object.

```
# selectdata function, call function then select file(s) using dialog window
files = selectdata()
files
```

CHAPTER

SIX

# STATISTICS

. . . under construction

CHAPTER

SEVEN

# ROADMAP

GrainPy will continue to add modern, user-friendly tools for the interpretation and presentation of sediment analyses. Below is a list of benchmarks that have been achieved or in the queue for development.

Initial build

Package release v0.1.0

Compatibility with multiple data sources

Additional statistics, including engineering and USCS

Plotting/comparison of multiple instances

Sediment mixing models for multi-modal samples

Integration with GIS

CHAPTER

EIGHT

# CONTRIBUTIONS

It is very much hoped that users of GrainPy will suggest improvements and/or new functionalities by one of two methods:

1. Simply create a new issue with the appropriate 'label' on Github.

**...OR...**

2. Personally contributing to the coding using a fork...

```
# create a branch
git checkout -b feature/AmazingFeature

# develop the feature code and commit your changes
git commit -m 'Added this new feature that does amazing things'

# push to the branch
git push origin feature/amazingfeature

# finally, `create a pull request on the GrainPy Github repo page
```

CHAPTER

NINE

# INDICES AND TABLES

- genindex
- modindex
- search