Minitab® 19

python-integration

Minitab®

Contents

Python Integration Guide for Minitab Statistical Software **4**
Introduction to using Python with Minitab Statistical Software 4
Install Python 4
Install mtbpy via PIP 5
Run in an Anaconda environment 5
Verify the Python installation 5
Run Python scripts in Minitab 6
Run Python scripts from the Minitab interface 10
Python API Reference 10
Module: mtbpy 10
Example code that converts Minitab date format to Unix date format 13
Example of calling a Python script from Minitab 14

Python Integration Guide for Minitab Statistical Software

Learn to bring results from Python into Minitab Statistical Software with the mtbpy package.

Use Python with Minitab Statistical Software
Verify that your Python installation works with Minitab and learn about the different ways that you can run Python from Minitab.

Python API Reference
Learn about the mtbpy module's classes and methods for Python.

Example of calling a Python script from Minitab
An example of a Python script that uses mtbpy to send a graph and a table from Python to Minitab.

Introduction to using Python with Minitab Statistical Software

Verify that your Python installation works with Minitab and learn about the different ways that you can run Python from Minitab.
Minitab Statistical Software integrates with Python, a general-purpose programming language with applications in data science. To accomplish Python integration, Minitab, LLC. provides the mtbpy library. With this custom library, you can create tables, graphs, messages, and notes in Python and display them in Minitab. Python integration offers the flexibility of custom Python code within Minitab's easy-to-use interface, and the results can be saved, stored, and shared in Minitab Project Files. After you install the mtbpy package, you're ready to run Python code from Minitab. If you have Python code that you want to access routinely, you can customize Minitab's interface to make the analysis more accessible through custom buttons or menus.

For more information on Minitab's Python library, including Python code examples, go to Python API Reference on page 10.

For more information on Python, consult the guidance available at www.python.org.

All the files referenced in this guide are available in this .ZIP file: Datafileref: .

Requirements

- Minitab 19.2020.1 (64-bit) or higher. To check your version of Minitab, choose **Help** > **About Minitab**.
- Python 3.6.1 (64-bit) or higher.

Install Python

Typically, the default installation of Python works with Minitab. Minitab supports Python 3.6.1 or higher. Follow the directions for your operating system at Python's website.

Note For the Window's operating system, select the option **Add Python 3.x to PATH** during installation. If Python is not in your PATH, then add Python to your system environment variables. Otherwise, Minitab will not be able to find Python.

Install `mtbpy` via PIP

To use `Python` with Minitab, install the `mtbpy` package. The `mtbpy` package gives you the capability to bring data from Minitab into `Python` and to return `Python` results to Minitab. To install the latest version via `PIP`, run the corresponding command for your operating system's terminal:

Windows

```
python -m pip install mtbpy
```

MacOS

```
python3 -m pip install mtbpy
```

After `Python` and the `mtbpy` package are installed, you can use the `PYSC` command to run `Python` from Minitab.

Run in an `Anaconda` environment

To run Minitab in an `Anaconda` environment, complete the following steps:

1. Open an `Anaconda Prompt`.
2. (Optional) Activate the `Anaconda` environment you want to run Minitab in using the command: `Activate`.
3. Copy the path to Mtb.exe. You can find the path by navigating to the Minitab install location or by copying the **Target** from a shortcut to Minitab.

   Tip The default Minitab install location on Windows is "C:\Program Files\Minitab\Minitab 19\Mtb.exe".
4. Paste the path into the `Anaconda Prompt` and press **Enter**.

Verify the `Python` installation

Default folders for `Python` files for Minitab

Typically, Python integration with Minitab is easiest when all of the files that you use are in the default folder location for Minitab files. If you do not specify a file path when you run `Python` from within Minitab, Minitab looks for your `Python` files in the default folder.

Windows

The default location is the "My Documents" folder.

MacOS

The default location is the "Documents" folder.

In the Windows operating system, if your `Python` file is in the default folder and you receive the error File not found:, check the default file location in your Minitab settings. To view or change the default file location in Minitab, choose **File** > **Options** > **General** > **Default file location**.

Run a test file

Use the following file to perform the steps in this section:

File	Description
Datafileref:	A sample `Python` script that is used throughout this guide. For this section, you run the script without arguments. When the script runs successfully, the result is the message "Minitab successfully located your Python installation."

Put Datafileref: in Minitab's folder for Python scripts. The default folder location depends on your operating system.

After you save the file, run the following command in the **Command Line** pane in Minitab:

```
PYSC "test.py"
```

Important If the Minitab Command Line is not visible, choose **View** > **Command Line/History**.

If Python successfully works with Minitab, you will see the following output in Minitab:

Python Script

```
These results are from external software.
Minitab successfully located your Python installation.
```

Run Python scripts in Minitab

You can run Python scripts from Minitab in three ways:

- Run the PYSC command in the **Command Line** pane.
- Run a Minitab exec that includes the PYSC command.
- Customize the Minitab interface to run a Minitab exec that includes the PYSC command.

Command Line pane

You can run the PYSC command in the **Command Line** pane. For general information about the **Command Line** pane, go to Command Line/History pane. For general information about using session commands, go to Session Command help.

PYSC ***["filename.py"] ["Args"...]***

Runs the Python script that you specify.

The default file extension for Python scripts is .PY. If the file extension is .PY, you do not need to type the file extension.

The optional argument Args allows you to pass arguments to the Python script through sys.argv[1:].Args can be any text values separated by a space. Enclose arguments in quotation marks. The default value is None, which means that the script does not receive any arguments.

In general, you use arguments to bring data from Minitab into Python. You can enter arguments in several ways. For example, you can use arguments that are identifiers for columns, matrices, or constants:

Minitab Session Command	Value in Python	Usage
`PYSC "test.py" "C1"`	`"C1"`	Use the following function to retrieve the column: `mtbpy.mtb_instance().get_column(sys.argv[1:][0])`
`PYSC "test.py" "M1"`	`"M1"`	Use the following function to retrieve the matrix: `mtbpy.mtb_instance().get_matrix(sys.argv[1:][0])`
`PYSC "test.py" "K1"`	`"K1"`	Use the following function to retrieve the constant: `mtbpy.mtb_instance().get_constant(sys.argv[1:][0])`

You can also use arguments that are the names of columns, matrices, or constants in Minitab:

Minitab Session Command	Value in Python	Usage
`PYSC "test.py" "My Column"`	`"My Column"`	Use the following function to retrieve the column: `mtbpy.mtb_instance().get_column(sys.argv[1:][0])`
`PYSC "test.py" "My Matrix"`	`"My Matrix"`	Use the following function to retrieve the matrix: `mtbpy.mtb_instance().get_matrix(sys.argv[1:][0])`
`PYSC "test.py" "My Constant"`	`"My Constant"`	Use the following function to retrieve the constant: `mtbpy.mtb_instance().get_constant(sys.argv[1:][0])`

You can also specify arguments to pass text to use in your Python code. You can pass text directly or in a constant.

Minitab Session Command	Value in Python	Usage
`PYSC "test.py" "Text not Stored"`	`"Text not Stored"`	This case passed a value that cannot be used with an mtbpy 'get' command. However, Args are not limited to only passing columns, matrices, and constants.
`LET K1 = "Text in Constant"` `PYSC "test.py" K1`	`"Text in Constant"`	This case highlights that, although PYSC does not accept arguments that are not text values, you can pass a constant to PYSC as long as the constant is defined as a text value.

When you pass more than one argument, you can access the arguments in order from the list of arguments:

Minitab Session Command	Value in Python	Usage

`PYSC "test.py" "C1" "C2" "M1" "K3" "10"`	`"C1"`
	`"C2"`
	`"M1"`
	`"K3"`
	`"10"`
		This case is an example of passing multiple Args, where you would access them by using the following functions in Python:
		`mtbpy.mtb_instance().get_column(sys.argv[1:][0])`
		`mtbpy.mtb_instance().get_column(sys.argv[1:][1])`
		`mtbpy.mtb_instance().get_matrix(sys.argv[1:][2])`
		`mtbpy.mtb_instance().get_constant(sys.argv[1:][3])`
		`int(sys.argv[1:][4])`

Use the following file to see output from the example text for the following subcommands. Make sure that the test.py file is in Minitab's folder for Python scripts.

File	Description
Datafileref:	A sample Python script that is used throughout this guide. When you pass arguments to the script, the results include a list of the values of the arguments. When you run the script with the

File	Description
	argument "ArgToBePrintedToStdErr", the script writes the name of the argument to the stderr file. When you run the script with the argument "ArgToBePrintedToStdOut", the script writes the name of the argument to the stdout file. Use the subcommands that follow to control whether the contents of these files appear in Minitab's Output pane.

NOSERR

Specifies to not display text from the standard error (`stderr`) console output in the Output pane in Minitab. The stderr console output is where you see Python error messages when you run your code in a Python integrated development environment, although you can use Python to put other results in the stderr file. For example, by default:

```
PYSC "test.py" "ArgToBePrintedToStdErr".
```

Produces the following results that include the stderr console output:

Python Script

```
These results are from external software.
The following arguments were passed to Python:
['ArgToBePrintedToStdErr']
Python standard error
The following arguments were printed to Stderr: 'ArgToBePrintedToStdErr'
```

The following session commands exclude the stderr console output:

```
PYSC "test.py" "ArgToBePrintedToStdErr";
NOSERR.
```

The session commands produce the following results:

Python Script

```
These results are from external software.
The following arguments were passed to Python:
['ArgToBePrintedToStdErr']
```

SOUT

Specifies to display text from the standard console output (`stdout`) in the Output pane in Minitab. The `stdout` is where you would see the results of commands like `print()` in a Python integrated development environment. For example, by default:

```
PYSC "test.py" "ArgToBePrintedToStdOut".
```

Produces the following results that exclude the `stdout`:

Python Script

```
These results are from external software.
The following arguments were passed to Python:
['ArgToBePrintedToStdOut']
```

The following session commands include the `stdout`:

```
PYSC "test.py" "ArgToBePrintedToStdOut";
SOUT.
```

The session commands produce the following results:

Python Script

```
These results are from external software.
The following arguments were passed to Python:
['ArgToBePrintedToStdOut']
Python standard output
The following arguments were printed to Stdout: 'ArgToBePrintedToStdOut'
```

Minitab exec file

Use the following file to perform the steps in this section:

File	Description
Datafileref:	A sample `Python` script that is used throughout this guide. When you pass arguments to the script, the results include a list of the values of the arguments.
Datafileref:	A sample Minitab exec file that includes the session commands to run the test.py script with 2 arguments.

Execs are text files that contain Minitab session commands. You can include the `PYSC` command that runs `Python` in a Minitab exec. With exec files, you can easily run commands without re-typing them, and you can assign the exec to a custom button in Minitab. For more information about Minitab execs, go to Minitab Macros Help. To run an exec, choose **File** > **Run an Exec**.

Suppose you create the exec Datafileref: . The exec file contains the following Minitab command:

```
PYSC "test.py" "Arg1" "Arg2"
```

To run the `Python` script with the exec, use the following steps:

1. Choose **File** > **Run an Exec**.
2. Click **Select File**.
3. Select PYEXEC.MTB.
4. Click **Open**.

The script displays the values of the arguments in Minitab, and the exec produces the following results:

Python Script

```
These results are from external software.
The following arguments were passed to Python:
['Arg1', 'Arg2']
```

Stop the `PYSC` command

You can stop a `Python` script and keep Minitab open, which prevents the loss of any edits to your Minitab project since your last save. The method to stop a `Python` script depends on your operating system.

Windows

Press **Ctrl** + **Alt** + **Delete** to open the Windows Task Manager. Then, end the `Python` process.

MacOS

Press **Command** + **Option** + **Esc** to open the Force Quit window. Then, end the `Python` application.

Run `Python` scripts from the Minitab interface

If you have a Minitab exec file, you can create a custom button or menu that runs the exec. For general information on how to customize the interface in Minitab, go to Customize menus, toolbars and shortcut keys.

You can use the following steps to create a custom button that runs an exec:

1. Choose **View** > **Customize**.
2. Click the **Tools** tab.
3. On the **Tools** tab, click the **New (Insert)** button .
4. Type a name for the command, then press the **Enter** key.
5. Click the **Open** button  .
6. From the file type drop-down list, select **All Files (*.*)**.
7. Browse to and select an exec file.
8. Click **Open**.
9. Choose **View** > **Customize** again.
10. On the **Commands** tab, under **Categories**, select **Tools**.
11. While the **Customize** dialog box is open, drag the new command to where you want it to appear on the Minitab menu or toolbar.
12. Click **Close**.

In addition to customizing Minitab's interface, you can use a COM-compliant language to create custom dialog boxes and analyses. For information on how to customize Minitab through COM, go to Minitab Automation.

`Python` API Reference

Documentation for the `mtbpy` module's classes and methods for `Python`.
To accomplish `Python` integration with Minitab Statistical Software, Minitab, LLC. provides the `mtbpy` library. The following descriptions of the classes and methods from the `mtbpy` module prepare you to write `Python` code that integrates with Minitab.

For information on how to install Minitab's `Python` library and how to run `Python` from Minitab, go to Introduction to using Python with Minitab Statistical Software on page 4.

For more information on `Python`, consult the guidance available at www.python.org.

Module: `mtbpy`

Class: `mtb_instance`

The following are the methods for the `mtb_instance` class.

get_column

Retrieves a column from a Minitab worksheet to use in Python.

column_name: string

Specifies the column to retrieve. You can specify either the column number (for example, "C1") or the column name (for example, "My Column").

Return value

Returns the column of data from the active worksheet as a Python list. The list can contain either text or numeric values.

Example

```
from mtbpy import mtbpy

column1 = mtbpy.mtb_instance().get_column("C1")
column2 = mtbpy.mtb_instance().get_column("My Column")
```

get_constant

Retrieves a constant from a Minitab worksheet to use in Python.

constant_name: string

Specifies the constant to retrieve. You can specify either the constant number (for example, "K1") or the constant name (for example, "My Constant").

Return value

Returns a constant that can be either a text or numeric value.

Example

```
from mtbpy import mtbpy

constant1 = mtbpy.mtb_instance().get_constant("K1")
constant2 = mtbpy.mtb_instance().get_constant("My Constant")
```

get_matrix

Retrieves a matrix from a Minitab worksheet to use in Python.

matrix_name: string

Specifies the matrix to retrieve. You can specify either the matrix number (for example, "M1") or the matrix name (for example, "My Matrix").

Return value

Returns the columns of data from the matrix as a Python list of lists.

Example

```
from mtbpy import mtbpy

matrix1 = mtbpy.mtb_instance().get_matrix("M1")
matrix2 = mtbpy.mtb_instance().get_matrix("My Matrix")
```

add_message

Appends a message to the Minitab Output pane.

message: string

Specifies the message to display.

Return value

None

Example

```
from mtbpy import mtbpy

mtbpy.mtb_instance().add_message("This is a message.")
```

set_note

Sets a note at the top of the Minitab Output pane.

message: string

Specifies the text to display.

Return value

None

Example

```
from mtbpy import mtbpy

mtbpy.mtb_instance().set_note("The output contains one note.")
```

add_image

Appends an image to the Minitab Output pane when you have a supported image file.

path: string

Specifies the path to the image.

Return value

None

Example

```
from mtbpy import mtbpy
import numpy as np
import matplotlib.pyplot as plt

N_points = 1000
n_bins = 50
x = np.random.randn(N_points)
y = .4 * x + np.random.randn(N_points) + 5
fig, axs = plt.subplots(1, 2, sharey=True, tight_layout=True)
axs[0].hist(x, bins=n_bins)
axs[1].hist(y, bins=n_bins)
fig.savefig("histogram.png")
mtbpy.mtb_instance().add_image("histogram.png")
```

`add_image_bytes`

Appends an image to the Minitab Output pane when you have a bytes object.

data: bytes

Specifies the bytes of data for an image. For example, you can enter a bytes array as the parameter.

Return value

`None`

Example

```
from mtbpy import mtbpy

image_data =
b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x00\t\x00\x00\x00\t\x08\x02\x00\x00\x00o\xf3\x9
mtbpy.mtb_instance().add_image_bytes(image_data)
```

`add_table`

Appends a table to the Minitab Output pane.

columns: list of lists

Specifies the columns of data for the table as a `list` of `lists`.

headers: list (Optional)

Specifies the column headers for the table. The default `headers` is an empty `list`.

title: string (Optional)

Specifies the title for the table. The default `title` is `""`.

footnote: string (Optional)

Specifies the footnote under the table. The default `footnote` is `""`.

Return value

`None`

Example

```
from mtbpy import mtbpy

mytitle = "My table title"
myheaders = ["Header for column 1", "Header for column 2"]
mycolumns = [[1,1,1],[2,2,2]]
myfootnote = "My footnote for the table."
mtbpy.mtb_instance().add_table(columns=mycolumns, headers=myheaders, title=mytitle,
 footnote=myfootnote)
```

Example code that converts Minitab date format to Unix date format

By default, Minitab uses a different `datetime` format than `Python`. To convert from the Minitab `datetime` format to the Unix `datetime` format, use the following code:

```
from datetime import datetime, timedelta

def minitab_to_unix_datetime(pOrdinal, pEpoch0=datetime(1899, 12, 30)):
    return(pEpoch0 + timedelta(days=pOrdinal))
```

Example of calling a Python script from Minitab

A marketing firm is hired to create a campaign for a new coffee blend. As part of their research, the team collects data from online reviews that include the word "coffee". To visualize the most popular words used in the reviews, the team wants to create a word cloud and add it to a Minitab project.

The example Python script opens a .TXT file that contains the coffee reviews. The script counts the words and creates a word cloud and table of frequencies. Then the script sends the word cloud image and the table to the Minitab Output pane.

All the files referenced in this guide are available in this .ZIP file: Datafileref: .

Use the following files to perform the steps in this section:

File	Description
Datafileref:	A Python script that takes data from comments in the file coffee_reviews.txt and displays the most frequent words in a word cloud and table.
Datafileref:	A data file that contains a column of customer reviews of coffee.
Datafileref:	An image file that the Python package wordcloud uses to create the shape of the word cloud.

The Python script in the below example requires the following Python modules:

mtbpy
: The Python module that integrates Minitab and Python. In the example, functions from this module send Python results to Minitab.

numpy
: A Python module that has various applications for scientific computing. In this example, functions from this module create a data array.

wordcloud
: A Python module that creates word clouds.

You can install the modules with pip. For example, the following command works for a typical installation of Python:

```
pip install mtbpy numpy wordcloud
```

1. Save the Python script file, Datafileref: , to your Minitab file location.

   Tip To change the Minitab file location in the Microsoft Windows version of Minitab, choose **File** > **Options** > **General** > **Default file location**.

2. Save the .PNG file, Datafileref: , to your Minitab file location.
3. Save the .TXT file, Datafileref: , to your Minitab file location.
4. In the Minitab **Command Line** pane, enter `PYSC "example.py"`.
5. Click **Run**.

Example.py

```
from wordcloud import WordCloud, STOPWORDS
import numpy as np
from PIL import Image
from mtbpy import mtbpy

# Set the maximum number of words for the word cloud.
nWords = 150

# Open and read the dataset into a variable.
```

```
dataset = open("coffee_reviews.txt", "r", encoding="utf8").read()

# Change all the characters in the dataset to lowercase.
dataset = dataset.lower()

# Open and read the mask image into a numpy array.
maskArray = np.array(Image.open("cloud.png"))

# Specify the properties of the word cloud.
cloud = WordCloud(background_color = "white", max_words = nWords, mask = maskArray,
stopwords = set(STOPWORDS))

# Generate the word cloud.
cloud.generate(dataset)

# Save the word cloud to a png file.
cloud.to_file("word_cloud.png")

# Send the png file to the Minitab Output pane.
mtbpy.mtb_instance().add_image("word_cloud.png")

# Initialize the arrays to store words and their frequencies in a table.
words = []
freqs = []

# Begin a loop to count the words in the word cloud frequency dictionary.
for word in cloud.words_:
    # Append the word to the list of words for the table.
    words.append(word)
    # Append the frequency of the word to the list of frequencies for the table.
    freqs.append(int(cloud.words_[word]*4716))

# Send the table to the Minitab Output pane.
mtbpy.mtb_instance().add_table(columns=[words, freqs], headers=["Word", "Frequency"],
 title="Word Cloud Data", footnote="{0} words are in this table".format(nWords))
```

Results

Python Script

These results are from external software.

Word Cloud Data

Word	Frequency
coffee	4716
cup	1439
flavor	1374
taste	1149
one	1048
good	1043
love	884
make	858
tea	833
great	798
use	701
drink	641
cup coffee	566
tried	563
product	558
amazon	548
will	521
morning	506
now	486

really	478
price	466
much	441
best	431
strong	428
time	428
found	426
buy	409
little	408
pod	401
well	396
even	388
brand	386
keurig	364
find	356
blend	349
brew	344
try	343
enjoy	341
drinking	339
starbuck	339
work	331
day	331
sugar	319
nice	318
perfect	316
always	314
delicious	311
way	306
order	306
want	291
better	288
say	276
used	274
go	271
two	268
favorite	266
made	264
pack	259
box	259
think	258
without	254
first	254
smooth	253
bag	253
come	253
flavored coffee	251
excellent	249
chocolate	249
bought	248
know	246
decaf	244
need	244
water	238
wonderful	234
still	233
got	233
lot	228
never	224
espresso	223

rich	214
love coffee	213
store	211
coffee maker	211
bitter	208
add	208
ordered	203
keep	201
sweet	201
give	198
right	198
using	196
milk	188
home	186
bold	186
thing	186
green mountain	184
year	183
vanilla	183
many	181
bean	179
machine	178
caffeine	178
bit	176
prefer	176
put	176
coffee drinker	173
fresh	171
may	169
smell	166
take	164
buying	164
husband	164
people	161
enough	159
us	159
actually	158
new	158
cookie	154
dark roast	154
package	153
hazelnut	151
purchased	151
purchase	151
review	151
strong coffee	146
dark	144
far	144
see	144
calorie	144
back	143
small	143
organic	141
feel	139
highly recommend	139
another	138
best coffee	138
every	136
thought	136
seem	136

almost	136
sure	134
aroma	134
trying	134
packet	134
french roast	134
expensive	133
less	133
especially	133
latte	133
thank	133

150 rows are in this table