



# Python Data Functions in TIBCO Spotfire®

*Software Release 10.10 LTS (10.10.1)*

# Contents

---

<b>Important Information.....</b>	<b>3</b>
<b>TIBCO Documentation and Support Services.....</b>	<b>4</b>
<b>Python Packages in TIBCO Spotfire®.....</b>	<b>6</b>
Included Packages.....	6
Manage packages through roles.....	7
Developer role.....	7
Curator role.....	8
Administrator role.....	8
Creating an SPK for Python Packages.....	8
Removing a package from a deployment.....	11
SPK Versioning.....	13
<b>Tips and Tricks for Working with Python in Spotfire.....</b>	<b>15</b>
Spotfire and Python data type mapping.....	15
Tips for using a different Python interpreter.....	15
Read or Write Table and Column Metadata.....	16
Return a Graphic to Spotfire from Python.....	16
Clear the pyplot Image.....	17
Handling Extension dtypes Int32 and Int64.....	17
Integer Conversion.....	17
Using a Model Produced in A Previous Data Function.....	17

## Important Information

---

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, TIBCO Spotfire, TIBCO Spotfire Analyst, TIBCO Spotfire Automation Services, TIBCO Spotfire Server, TIBCO Spotfire Web Player, TIBCO Enterprise Runtime for R, TIBCO Enterprise Runtime for R - Server Edition, TERR, TERR Server Edition, and TIBCO Spotfire Statistics Services are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 1994-2020. TIBCO Software Inc. All Rights Reserved.

# TIBCO Documentation and Support Services

---

## How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

## Product-Specific Documentation

Documentation for TIBCO Spotfire Analyst and related products is available on the [Spotfire Analyst Product Documentation](#) page.

- TIBCO Spotfire® Analyst User's Guide
- TIBCO Spotfire® Administration Manager User's Guide
- Working with Cubes in TIBCO Spotfire®
- TIBCO Spotfire® License Agreement

Related documentation is also found on the [Spotfire Server Product Documentation](#) page.

## Release Version Support

Some release versions of TIBCO Spotfire products are designated as long-term support (LTS) versions. LTS versions are typically supported for up to 36 months from release. Defect corrections will typically be delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also [https://docs.tibco.com/pub/spotfire/general/LTS/spotfire\\_LTS\\_releases.htm](https://docs.tibco.com/pub/spotfire/general/LTS/spotfire_LTS_releases.htm).

## How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, you can request one by clicking Register on the website.

## System Requirements for Spotfire Products

For information about the system requirements for Spotfire products, visit <http://spotfi.re/sr>.

## How to join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can

submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to <https://community.tibco.com>.

For quick access to TIBCO Spotfire content, see <https://community.tibco.com/products/spotfire>.

## Python Packages in TIBCO Spotfire®

When you create an analysis using Spotfire®, you can enhance the analysis by adding your own calculations and outputs using the data function framework. You can write data function scripts using different programming languages. This guide provides help for installing Python packages to use in data functions.



This version of Spotfire includes support for Python data functions used only with Spotfire Analyst and Spotfire Desktop. Python data functions are not supported on TIBCO Spotfire® Business Author and Consumer or TIBCO Cloud™ Spotfire®.

Using Python data functions is different from the IronPython functionality available for extending Spotfire. For information about using IronPython with Spotfire, see the *TIBCO Spotfire® Analyst Help*, available from the Spotfire menu, or the article [IronPython Scripting in TIBCO Spotfire®](#) on the TIBCO® Community site.

This version of Spotfire includes a Python interpreter. For more information about using another installation of Python with your data functions, see the *TIBCO Spotfire® Help*.

You can download packages from the most popular Python repository web site PyPI, from another community repository, from a locally-managed repository, or from a package you created.

- If you are doing ad-hoc analysis and want to try a package from PyPI, you can use the Python Tools included in Spotfire for installing packages.

1. From the Spotfire menu, click **Tools > Python Tools > Package Management**.
2. Type the name of the package to install in the **Available Packages** text box, and then click **Search**.

For more information, see the *TIBCO Spotfire® User's Guide*, from the **Help** menu in Spotfire.

- If your packages are managed by your Spotfire Server administrator, then you can create a Spotfire package (SPK) containing packages downloaded from PyPI, from another repository, or from a file location. See [Creating an SPK for Python Packages](#) on page 8 for more information.



Package versions installed using the SPK mechanism take precedence over packages installed using Python Tools. If you try to update a package that was previously installed by the Administrator, the updated package is installed, but the update is ignored.

If you share downloaded packages, you must ensure that the package versions are the same. For more information about managing packages through roles, see [Curator role](#) on page 8.

## Included Packages

Your installation of Spotfire includes a Python interpreter and several packages it needs to run under Spotfire. To run the Python interpreter, you must have the 64-bit version of Spotfire.



The packages listed in this table are required for Spotfire and the Python interpreter to work together. Removing or altering these packages can cause your Python data functions to fail.

Package name	Version	Description	More information
bitstring	3.1.6	A pure Python module that makes the creation, manipulation and analysis of binary data as simple and natural as possible.	<a href="https://pypi.org/project/bitstring/">https://pypi.org/project/bitstring/</a> <a href="https://pythonhosted.org/bitstring/">https://pythonhosted.org/bitstring/</a>

Package name	Version	Description	More information
numpy	1.17.4	Provides the following. <ul style="list-style-type: none"> <li>• An N-dimensional array object.</li> <li>• Broadcasting functions.</li> <li>• Tools for integrating C/C++ and Fortran code.</li> <li>• Linear algebra, Fourier transform, and random number capabilities.</li> </ul>	<a href="https://pypi.org/project/numpy/">https://pypi.org/project/numpy/</a> <a href="https://docs.scipy.org/doc/numpy/">https://docs.scipy.org/doc/numpy/</a>
pandas	0.25.3	Provides data structures and data analysis tools for dealing with tabular data, ordered and unordered time series data, matrix data, and other types of data sets.	<a href="https://pypi.org/project/pandas/">https://pypi.org/project/pandas/</a> <a href="https://pandas.pydata.org/pandas-docs/stable/">https://pandas.pydata.org/pandas-docs/stable/</a>
pip	19.3.1	Provides support for installing packages.	<a href="https://pip.readthedocs.io/en/stable/reference/pip_install/#description">https://pip.readthedocs.io/en/stable/reference/pip_install/#description</a>
python-dateutil	2.8.1	Provides extensions to the datetime module in Python.	<a href="https://pypi.org/project/python-dateutil/">https://pypi.org/project/python-dateutil/</a> <a href="https://dateutil.readthedocs.io/en/stable/">https://dateutil.readthedocs.io/en/stable/</a>
pytz	2019.3	Provides a platform for cross-platform time zone calculations.	<a href="https://pypi.org/project/pytz/">https://pypi.org/project/pytz/</a>
six	1.13.0	Provides utility functions for smoothing over the differences between the Python versions 2 and 3.	<a href="https://pypi.org/project/six/">https://pypi.org/project/six/</a> <a href="https://six.readthedocs.io/">https://six.readthedocs.io/</a>
setuptools	42.0.2	Provides tools for building, installing, upgrading, and uninstalling Python packages.	<a href="https://pypi.org/project/setuptools/">https://pypi.org/project/setuptools/</a> <a href="https://setuptools.readthedocs.io/en/latest/">https://setuptools.readthedocs.io/en/latest/</a>
wheel	0.33.6	The reference implementation of the Python wheel packaging standard, as defined in <a href="#">PEP 427</a> .	<a href="https://pypi.org/project/wheel/">https://pypi.org/project/wheel/</a> <a href="https://wheel.readthedocs.io/en/stable/">https://wheel.readthedocs.io/en/stable/</a>

## Manage packages through roles

Working with packages in a deployment that includes Spotfire and Spotfire Server can add some complexity to management policies.

The job of synchronizing package versions among your development computers, your testing computers, and your servers is an important package management concern for an organization. You can reduce the risk of confusion and streamline your processes by defining roles in your organization for dealing with packages. Ensure processes and rules are established to manage packages.

### Developer role

The developer is a Python programmer or statistician who develops packages or writes data functions using Python.

The package developer accomplishes the following tasks using the Spotfire tools.

- Develops and tests Python packages or data functions using the local Python interpreter available from Spotfire Analyst.
- Reviews and recommends packages to be included on the Spotfire Server for data functions to use.

## Curator role

The curator maintains the standards and lists of officially-sanctioned packages. The curator keeps all of the package versions synchronized. The curator might be the same person who fills the developer role.

The approval process for adding a packaging is up to your organization, and might vary from minimal to extensive, depending on your usual practices. Designate a developer familiar with Python packages and package versioning to be the package curator. The package curator works with package developers and server administrators to perform the following management tasks.

- Maintains the list of tested and sanctioned package versions (the gold standard), which would be the set of packages available for general use under Spotfire applications.
  - Creates a Spotfire SPK containing the Python packages, and then gives it to the administrator who manages Spotfire distributions on Spotfire Server. Packages uploaded to Spotfire Server are distributed to other Python users who write data functions for using Python in Spotfire Analyst.
  - Ensures that the SPK containing the "gold standard" package versions are placed on the Spotfire Server, to be distributed to Spotfire analysts who develop data functions, using the same packages, or who want to see the shared analyses running the data functions.
- Python package versions shared among team members must be kept synchronized.
  - You can install multiple SPKs containing Python packages on the Spotfire Server, as long as each SPK has a unique name and ID.
  - Uploading a new SPK overwrites any older version of that same SPK that was previously deployed.



See [SPK Versioning](#) on page 13 for more information.

## Administrator role

The Spotfire administrator manages packages on the Spotfire Server.

The responsibilities for the administrator role include the following.

- Deploys the SPK containing Python packages that are distributed to Spotfire Analyst users.
- Assigns licenses for access to the Data Functions feature in Spotfire Analyst.
- Uploads, maintains, and removes packages. (Might assign server permissions to the curator for this task.)

## Creating an SPK for Python Packages

---

Your installation of Spotfire Analyst includes a Python interpreter and a set of packages to enable using Python in Spotfire. One of these packages, `spotfire.zip`, provides tools for building SPKs to share Python packages with other data function authors in your organization.

Spotfire Analyst relies on `pip`, the Python command-line application for Python package installation. Spotfire Analyst uses a `requirements.txt` file to specify the packages to include in your SPK.



By default, the file `requirements.txt` searches the PyPI package site for the specified package and version.

- To include a package from a different repository or in a local file path, in the `requirements.txt` file, use the option `-i` or `--index-url`, followed by the location URL.

```
#example
#
mylib -i http://my.domain.org/lib/1.0.0/mylib/
```

- To include a `.whl` package, in the `requirements.txt` file, provide the relative path to the package from the current working directory.

```
#simple-example
#
./my_path/my_package.whl
packaging==1.0.0
```

For more information about creating a `requirements.txt` file for your package list, see its documentation at the following location.

- [https://pip.readthedocs.io/en/stable/user\\_guide/#requirements-files](https://pip.readthedocs.io/en/stable/user_guide/#requirements-files)
- [https://pip.readthedocs.io/en/stable/reference/pip\\_install/#requirements-file-format](https://pip.readthedocs.io/en/stable/reference/pip_install/#requirements-file-format)

Perform this task from a command prompt on the Windows computer where Spotfire Analyst is installed.



Your installation of Spotfire Analyst relies on the Python packages included in the installation. Removing any of these packages causes your Spotfire Analyst installation to not work with the included Python interpreter. See [Included Packages](#) on page 6 for more information.

### Prerequisites

- You must have the appropriate Spotfire license for authoring data functions.
- You must have created the file `requirements.txt` containing the list of packages to include in your SPK.

The following example specifies these packages and specified versions from PyPI.

```
#
##### example-requirements.txt #####
#
scipy == 1.3.3
matplotlib == 3.1.2
statsmodels == 0.10.2
```

### Procedure

1. At the command prompt, set the Python path (`PYTHONPATH`) to the directory where the Python package `spotfire.zip` is installed.

```
set PYTHONPATH=%SPOTFIRE_HOME%\Modules\Core_<core-build-version>\python\spotfire.zip
```



For example purposes, we refer to the directory where Spotfire Analyst is installed as `%SPOTFIRE_HOME%`.

The `PYTHON_PATH` environment variable is set to the directory where the `spotfire.zip` package is installed.

2. From the command line, type the following command.

```
"%SPOTFIRE_HOME/Modules/Python Interpreter_<version#>/python/python.exe" -m spotfire.spk
packages [--name "<package-name>"] [--analyst] <name.spk> <path-to>requirements.txt
```



Remember that the path to the Python interpreter has spaces in it, so you must quote the path string.

Option	Description
spotfire.spk	The package containing the Python code to download and bundle the needed packages specified in <code>requirements.txt</code> .  <div style="display: flex; align-items: center;"> <div> <p>From the command prompt, you can view help for the package <code>spotfire.zip</code> by typing the following command:</p> <pre>"&lt;path-to-python-interpreter&gt;\python\python.exe" -m spotfire.spk packages --help</pre> </div> </div>
packages	The subcommand that creates the package bundle.
--analyst	The flag to create the SPK to be shared with other Spotfire Analyst users connected to the Spotfire Server where the SPKs are deployed.
--name	A string containing the name of the SPK to contain the packages.  If this parameter is not specified, then the package name defaults to whatever was last used in the brand located in the <code>requirements.txt</code> file, or if no brand exists, in the current "Python Packages". Additionally, the package ID defaults to the current value from the brand in the <code>requirements.txt</code> file, or a random UUID if none exists.
--analyst	This option specifies that the Spotfire Server should distribute the packages in the SPK to other Spotfire Analyst clients connected to the Spotfire Server.  (If you do not use this option, then the packages are placed on the server for Spotfire Business Author and Consumer users who connect using the Spotfire Web Player.)
name.spk	The name of the SPK file that is created by this task.
requirements.txt	The full path to the file <code>requirements.txt</code> .

The following example creates an SPK named `my_pkgs.spk`, containing the packages specified in `requirements.txt`.

```
"%Spotfire_Home%\Modules\Python Interpreter_3.7.5.0\python\python" -m spotfire.spk
packages --name "my_pkgs.spk --analyst c:\files\requirements.txt
```

The packages and all of their dependencies are written to the SPK named `my-pkgs.spk` in the current working directory where the command was run, and the version information is recorded in the file `requirements.txt`. For example:

```
#
##### example-requirements.txt #####
#
scipy == 1.3.3
matplotlib == 3.1.2
statsmodels == 0.10.2

## spotfire.spk: {"BuiltBy":"3.7.5 (tags/v3.7.5:e09359112e, Nov 30 2019,
## spotfire.spk: 20:34:20) [MSC v.1916 64 bit (AMD64)]","BuiltAt":"Thu
## spotfire.spk: Dec 12 15:41:32 2019","BuiltFile":"my-data.spk","BuiltN
## spotfire.spk: ame":"Python Packages","BuiltId":"b8b9e4ec-324d-4da1-bd
## spotfire.spk: 4b-6a3509c7877d","BuiltVersion":"1.0.0.0","BuiltPackage
```

```
## spotfire.spk: s":{"cyclor":"0.10.0","kiwisolver":"1.1.0","matplotlib"
## spotfire.spk: : "3.1.2", "numpy": "1.17.4", "pandas": "0.25.3", "patsy": "0.
## spotfire.spk: 5.1", "pyparsing": "2.4.5", "python-dateutil": "2.8.1", "pyt
## spotfire.spk: z": "2019.3", "scipy": "1.3.3", "setuptools": "42.0.2", "six"
## spotfire.spk: : "1.13.0", "statsmodels": "0.10.2"}}
```

3. Locate the SPK you created.

This file is saved in the directory from which you ran the command.

4. Give the SPK to the Spotfire Server administrator to deploy.

### Result

After the SPK is placed into the deployment area, Spotfire Analyst users connected to that deployment area are prompted to update their Spotfire Analyst installations. The packages are included in the update.

## Removing a package from a deployment

In some cases, you might need to remove a Python package from a deployment. You must recreate and redeploy the SPK.

### Prerequisites

- You must have the appropriate Spotfire license for authoring data functions.
- You must create a new `requirements.txt` file containing the list of packages to keep in the SPK.

The following example shows three packages from the example for creating the original SPK.

```
#
##### example-requirements.txt #####
#
scipy == 1.3.3
matplotlib == 3.1.2
statsmodels == 0.10.2
```



For information about how changing a `requirements.txt` affects how the SPK version changes, see [SPK Versioning](#) on page 13.

### Procedure

1. To remove one of the packages, recreate the `requirements.txt` file, removing the package name to delete.

```
#
##### example-requirements.txt #####
#
scipy == 1.3.3
statsmodels == 0.10.2
```

2. At the command prompt, set the Python path (`PYTHONPATH`) to the directory where the Python package `spotfire.zip` is installed.

```
set PYTHONPATH=%SPOTFIRE_HOME%\Modules\Core_<core-build-version>\python\spotfire.zip
```



For example purposes, we refer to the directory where Spotfire Analyst is installed as `%SPOTFIRE_HOME%`.


The path to Python is set to the directory where the `spotfire.zip` package is installed.

- From the command line, type the following command.

```
"<path-to-python-interpreter>/python/python.exe" -m spotfire.spk packages --analyst <name.spk> <path-to>requirements.txt
```



Remember that the path to the Python interpreter has spaces in it, so you must quote the path string.

Option	Description
spotfire.spk	The package containing the Python code to download and bundle the needed packages specified in <code>requirements.txt</code> .   From the command prompt, you can view help for the package <code>spotfire.zip</code> by typing the following command: <pre>"&lt;path-to-python-interpreter&gt;\python\python.exe" -m spotfire.spk packages --help</pre>
packages	The subcommand that creates the package bundle.
--analyst	The flag to create the SPK to be shared with other Spotfire Analyst users connected to the Spotfire Server where the SPK is deployed.
<name.spk>	The name of the SPK to contain the packages.
requirements.txt	The full path to the <code>requirements.txt</code> .

The following example creates an SPK named `my_pkgs.spk`, containing the packages specified in `requirements.txt` (which no longer contains the package to delete).

```
"%Spotfire_Home%\Modules\Python Interpreter_3.7.5.0\python\python" -m spotfire.spk packages --analyst my-pkgs.spk c:\files\requirements.txt
```

The packages and all of their dependencies are written to the SPK named `my-pkgs.spk`, and the version information is recorded in the file `requirements.txt`. For example:

```
#
##### example-requirements.txt #####
#
scipy == 1.3.3
statsmodels == 0.10.2## spotfire.spk: {"BuiltBy":"3.7.5 (tags/v3.7.5:e09359112e, Nov 30 2019,
## spotfire.spk: 20:34:20) [MSC v.1916 64 bit (AMD64)]", "BuiltAt": "Fri
## spotfire.spk: Jan 10 15:19:35 2020", "BuiltFile": "my_pkgs.spk", "BuiltN
## spotfire.spk: ame": "Python Packages", "BuiltId": "b8b9e4ec-324d-4dal-bd
## spotfire.spk: 4b-6a3509c7877d", "BuiltVersion": "1.0.0.0", "BuiltPackage
## spotfire.spk: s": {"numpy": "1.18.1", "pandas": "0.25.3", "patsy": "0.5.1",
## spotfire.spk: "python-dateutil": "2.8.1", "pytz": "2019.3", "scipy": "1.3.
## spotfire.spk: 3", "six": "1.13.0", "statsmodels": "0.10.2"}}
```

- Locate the SPK you created.

This file is saved in the directory from which you ran the command.

- Give the SPK to the Spotfire Server administrator to deploy.

## Result

After the SPK is placed into the deployment area, Spotfire Analyst users connected to that deployment area are prompted to update their Spotfire Analyst installations. Only the specified packages are included in the update, and the package removed from the `requirements.txt` file is deleted from users' computers.

## SPK Versioning

To share packages among data function authors in your organization, you can create the file `<your-filename>.spk` containing the packages to distribute to others. You might need to change or update the packages or package versions that you distribute, which requires changing the version of the SPK containing the packages.

You can create or change a Spotfire SPK using the steps described in [Creating an SPK for Python Packages](#) on page 8. The package `spotfire.spk` creates a new SPK using the versioning rule details for the following tasks.



- Python package versions shared among team members must be kept synchronized.
- You can install multiple SPKs containing Python packages on the Spotfire Server, as long as each SPK has a unique name and ID.
- Uploading a new SPK overwrites any older version of that same SPK that was previously deployed.

### Versioning rules

Task	Procedure	Version result	Version example	Comment
Generating a new <code>requirements.txt</code> .	Pass the new <code>requirements.txt</code> to the Python script.	The version is always set to 1.0.0.0 by default.	1.0.0.0	The script overwrites the old SPK, and the list contains only the packages you provide in <code>requirements.txt</code> .
Recreating a new <code>requirements.txt</code> using the same version. (That is, you do not need to increment or keep the older <code>requirements.txt</code> .)	Regenerate the SPK, passing the new <code>requirements.txt</code> to the Python script.	The version is always set to 1.0.0.0 by default.	1.0.0.0	Spotfire Server does not register the package as a new one, so it does not distribute the package to the users.
Adding package names to an existing <code>requirements.txt</code> .	Edit the <code>requirements.txt</code> , and then pass it to the Python script.	The version is incremented to a minor version number.	1.1.0.0	The script overwrites the old SPK, and the list contains only the packages you provide in <code>requirements.txt</code> .  Spotfire Server registers the SPK as changed and distributes it to the users.
Removing package names from an existing <code>requirements.txt</code> .	Edit the <code>requirements.txt</code> , and then pass it to the Python script.	The version is incremented to a major version number.	2.0.0.0	The script overwrites the old SPK, and the list contains only the packages you provide in <code>requirements.txt</code> .  Spotfire Server registers the SPK as changed and distributes it to the users.

Task	Procedure	Version result	Version example	Comment
Assigning a specific version number to a <code>requirements.txt</code> .	Run the Python script and pass in the <code>requirements.txt</code> in the command, along with the <code>version</code> , setting it to the version you want.	The version number is set to the value provided in the argument	1.2.3.4	The version argument must be passed as a string containing four components (for example, " <code>--version = 1.2.3.4</code> " or " <code>-v 1.2.3.4</code> ").

# Tips and Tricks for Working with Python in Spotfire

You can expand the functionality of Spotfire using Python data functions, but innate differences in these two tools can cause unexpected behavior.

This section contains some tricks for working with graphic types, models, and data types between Python and Spotfire.

## Spotfire and Python data type mapping

To create Python data functions in Spotfire, you need to know how the data types in each application map to each other. This table provides that mapping, including data type mapping to Pandas column dtype, and for mapping columns and tables.

Spotfire	Python	Pandas column dtype
Integer	int	Int32
LongInteger	int	Int64
Real	float	float64
SingleReal	float	float32
Currency	decimal.Decimal	object
Date	datetime.date	object
DateTime	datetime.datetime	object
Time	datetime.time	object
TimeSpan	datetime.timedelta	object
Boolean	bool	object
String	str	object
Binary	bytes	object

- Spotfire columns map in Python to Pandas `Series` type.
- Spotfire tables map in Python to Pandas `DataFrame` type.

## Tips for using a different Python interpreter

If you have another Python interpreter installed on the computer where Spotfire is installed, you can specify using that interpreter with Spotfire.

From the **Tools > Options > Data Functions** dialog box, you can select **Use specific local python.exe**, and then provide an alternative path to your Python interpreter. If you use a different interpreter, make sure that you consider the following.

- The Python interpreter must be version 3.5 or later.
- Any packages you install are installed in the `site-library` directory for that interpreter.

- You must install the packages required by Spotfire to work with Python. See [Included Packages](#) on page 6 for the list.

## Read or Write Table and Column Metadata

---

When you read or write a Pandas data frame, table and column metadata is provided to help you ensure that the content type is set correctly.

Two attributes are provided that make the metadata accessible:

- `spotfire_table_metadata`
- `spotfire_column_metadata`



Table metadata applies only to tables, and column metadata applies only to columns.

Table and column names are prefixed to the metadata names. The following examples illustrate the naming.

- If your input parameter is 'Table', and if your table name is MyTable, then the table attribute is `MyTable.spotfire_table_metadata`.
- If your input parameter is 'Table', and if a column name in MyTable is Cost, then that column metadata is `MyTable['Cost'].spotfire_column_metadata`.
- If your input parameter is 'Column', and if a column name is Cost, then that column metadata is `Cost.spotfire_column_metadata`.



Due to limitations in Pandas, metadata does not survive a Pandas operation that constructs a new data frame or series. If you try to access the metadata on a newly-created object, an `AttributeError` is raised.

Setting the `spotfire_table_metadata` on a Pandas `DataFrame` object usually triggers a warning from Pandas of the format "UserWarning: Pandas doesn't allow columns to be created via a new attribute name - see <https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access>", which is reported through Spotfire as a warning when it runs the data function. You can silence the warning using the following construction.

```
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    x.spotfire_table_metadata = {...}
```



Setting `spotfire_column_metadata` on `Series` objects does not trigger this warning.

You can copy metadata using the function `spotfire.copy_metadata(source, destination)`. This function is located in the top-level "spotfire" module, and can be imported into your script using the command `import spotfire`. This function copies all of the metadata from the first argument to the second argument, provided that both arguments are the same type (that is, both must be either `DataFrames`, or both must be `Series`). If the types differ, a `TypeError` is raised.

## Return a Graphic to Spotfire from Python

---

Spotfire uses an image binary to display graphic images in a document property.

When you return a graphic from Python to be used in Spotfire, Spotfire automatically converts these image types into an image binary that it can use as a document property. This technique works for the following graphical formats.

- `matplotlib` figure objects.
- `PIL` image objects.



- seaborn Grid objects.

These formats cover the majority of the graphical content available in Python packages.

## Clear the pyplot Image

---

A plot created in a data function using matplotlib's `pyplot` function can persist between data functions unless the plot is explicitly cleared.

If the buffer retains the plot, you can explicitly clear it by including the command `plt.clf()` in your data function.

## Handling Extension dtypes Int32 and Int64

---

Third-party packages might not support the extension dtypes `Int32` and `Int64`.

The Python implementation in Spotfire uses the extension dtypes `Int32` and `Int64`. These dtypes support missing values in data coming from Spotfire.

However, some packages, including pandas-profiling, do not support these extension dtypes. If you use a package with these dtypes, you could encounter errors. For third-party packages to work as expected, try casting the columns that cause problems to `int32` or `int64`.

For example, for a simple table with one column of 64-bit integers, you can recast it as follows for your Python package.

```
MyTable = MyTable2.astype('int64')
```

For more information, see <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.astype.html>.

## Integer Conversion

---

When a data function sends a single integer between Python and Spotfire, Spotfire can automatically promote the integer to a longInteger, causing the data function to fail if the output is set to an integer type property.

Because Python has only one integer type for single integers, and because Spotfire converts certain integers to long integers, you must pay attention to single integers passed between Python and Spotfire.

You can avoid this error by specifying ints as longIntegers in your data functions.

For more information about how to map data types, column types, and tables between Spotfire and Python, see [Spotfire and Python data type mapping](#) on page 15.

## Using a Model Produced in A Previous Data Function

---

If you produce an object using a Python data function, and you want to bring it into another data function for further handling, the object might need special handling.

If you create an object, such as a machine-learning model, and you want to use that object in a second data function, you can use the pickle package to store the object in a document property. The following example demonstrates using the pickle package to manage this process.

```
#From data function number 1
import pickle
binary = pickle.dumps(myObject)
```

```
#To data function number 2
import pickle
myObject = pickle.loads(binary)
```

The binary data can be stored in a Spotfire document property in the first data function, and then loaded from that document property to the second data function. This technique should work generically for almost any object that you might want to retain between data function calls.