

JUPYTER (IPYTHON) NOTEBOOK CHEATSHEET

About Jupyter Notebooks

The Jupyter Notebook is a web application that allows you to create and share documents that contain executable code, equations, visualizations and explanatory text.

This guide walks you through the basics of using Jupyter Notebooks locally (running Python 3, Pandas, matplotlib and Pandas Treasure Data Connector) as a data analysis and visualization control panel with Treasure Data as your data backend. (To run Jupyter notebooks remotely, just omit the setup steps and surf directly to your online service in your browser.)

About Treasure Data

Treasure Data is a data collection, data storage and data analytics cloud service, which integrates easily with Jupyter Notebooks. As this guide shows, you can use Jupyter Notebooks as a flexible control panel for your data analytics running on Treasure Data.

This Guide's Audience

1. Newcomers to Jupyter: This guide shows how to get your first Jupyter notebook up and running.
2. Data scientists trying to analyze larger datasets: If you are maxing out on memory/disk on your local Jupyter notebook, integrating Treasure Data with your Jupyter notebook can help you to scale.
3. Educators: Jupyter is an excellent teaching tool for data analysis, and this guide can be used as a teaching aid.

Pre-Setup

To use Treasure Data with Jupyter Notebooks, you should sign up for Treasure Data and get your master API key from **Treasure Data Console**.

You can also get **Condas** as an environment manager, and to install dependencies.

Next, run the following at the command line, to set up your virtual python environment "analysis", and switch to it :



```
$ export MASTER_TD_API_KEY="<YOUR_TREASUREDATA_MASTER_API_KEY>"  
$ conda create -n analysis python=3  
$ source activate analysis
```

Condas Setup

Install dependencies and launch your Jupyter notebook

```
{ } # install dependencies
(analysis)$ conda install pandas
(analysis)$ conda install matplotlib
(analysis)$ conda install -c https://conda.anaconda.org/anaconda seaborn
(analysis)$ conda install ipython-notebook
(analysis)$ pip install pandas-td
#activate notebook
(analysis)$ ipython notebook
```

Cleanup and basic administration

These are the steps you'd take to clean up or manage your virtual environment.

```
{ } #deactivate notebook
(analysis)$ source deactivate

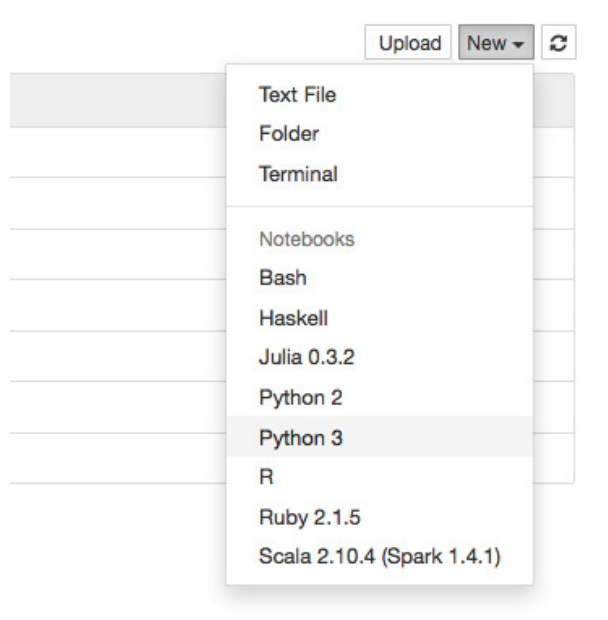
#remove environment
$ conda remove --name analysis --all

#list environments
$ conda info --envs
or
$ conda env list
```


Jupyter Notebooks

Creating a new notebook

1. Launch Jupyter Notebook by typing `$ipython notebook` at the console.
2. On the top right corner, click **New** → **Python 3** (or **Python 2**).



Executing Commands

Command execution - and embedding code - is done by entering the Python code to the current cell and hitting **Shift + Return** or hitting the play button () in the toolbar:

```
In [3]: 1+2
Out[3]: 3

In [6]: 1 < 2
Out[6]: True

In [4]: string = "Hello World"
        print(string)
Hello World
```

Embedding Text

Text is embedded by selecting **Markdown** in the toolbar dropdown list, typing text into the cell, and hitting **Shift + Return**.

Embedding Raw Text

Raw Text is embedded by selecting **Raw NBConvert** in the toolbar dropdown list, typing text into the cell, and hitting **Shift + Return**.

Embedding Links

Links are created by selecting **Markdown** in the toolbar dropdown list, typing text into the cell as shown below and hitting **Shift + Return**.

Text	Yields
<code>https://www.treasuredata.com</code>	https://www.treasuredata.com
<code>[click this link](https://www.treasuredata.com)</code>	click this link
<code>[click this link](https://www.treasuredata.com "Treasure Data Home")</code>	click this link (shows <i>Treasure Data Home</i> on mouseover)

Embedding Formulae & Equations as LaTeX

IPython notebook uses **MathJax** to render LaTeX inside html/markdown. To render equations as LaTeX:

1. Refer to the **MathJax Tutorial** for syntax help.
2. Place cursor in the cell where you want to type the equation.
3. Select **Markdown** in the toolbar dropdown list.
4. Type out your formula. You should wrap your formula string in '\$\$'.
5. Hit **Shift + Return**.

Formula	Yields
<code>\$\$c = \sqrt{a^2 + b^2}\$\$</code>	$c = \sqrt{a^2 + b^2}$
<code>\$\$e^{i\pi} + 1 = 0\$\$</code>	$e^{i\pi} + 1 = 0$
<code>\$\$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i\$\$</code>	$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$
<code>\$\$r'F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k x} dx'\$\$</code>	$r'F(k) = \int_{-\infty}^{\infty} f(x) e^{2\pi i k x} dx'$
<code> \$\$\begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^n \\ 1 & a_2 & a_2^2 & \cdots & a_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_m & a_m^2 & \cdots & a_m^n \end{pmatrix} \$\$ </code>	$\begin{pmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^n \\ 1 & a_2 & a_2^2 & \cdots & a_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_m & a_m^2 & \cdots & a_m^n \end{pmatrix}$

IPython help and reference commands

IPython is its own programming language environment, and a complete tutorial is out of the scope of this guide. Its kernel is supported in the Jupyter Notebook.

For a complete introduction to IPython, go here: <http://IPython.readthedocs.org/en/stable/>

Below are some commands supported in Jupyter Notebooks to get you started:

Command	Description
<code>%quickref</code>	IPython quick reference.
<code>help()</code>	Python's help utility.

Keyboard Shortcuts

The following are the most used keyboard shortcuts for a Jupyter Notebook running the Python Kernel. This list is changing all the time; **please check help->keyboard shortcuts** in your notebook for the latest.

Command	Description
Enter	enter edit mode
Command + a; Command + c; Command + v	select all; copy; paste
Command + z; Command + y	undo; redo
Command + s	save and checkpoint
Command + b; Command + a	insert cell below; insert cell above
Shift + Enter	run cell, select below
Shift + m	merge cells
Command +]; Command + [indent; dedent
Ctrl + Enter	run cell
Option + Return	run cell, insert cell below
Escape	enter command mode
Escape + d + d	delete selected cell
Escape + y	change cell to code
Escape + m	change cell to markdown
Escape + r	change cell to raw
Escape + 1	change cell to Heading 1
Escape + n	change cell to heading n
Escape + b	create cell below
Escape + a	Insert cell above

Magic Commands

Here are some commonly used magic functions.

Statement	Explanation	Example
<code>%magic</code>	Comprehensively lists and explains magic functions .	<code>%magic</code>
<code>%automagic</code>	When active, enables you to call magic functions without the '%'. returns	<code>%automagic</code>
<code>%quickref</code>	Launch IPython quick reference.	<code>%quickref</code>
<code>%time</code>	Times a single statement.	In [561]: <code>%time method = [a for a in data if b.startswith('http')]</code>
	returns	CPU times: user 772 μ s, sys: 9 μ s, total: 781 μ s Wall time: 784 μ s
<code>%pastebin</code>	Pastebins lines from your current session.	<code>%pastebin 3 18-20 ~1/1-5</code>
	returns	Lines 3 and lines 18-20 from current session, and lines 1 - 5 from the previous session to gist.github.com pastebin link.
<code>%debug</code>	Enters the interactive debugger.	<code>%debug</code>
<code>%hist</code>	Print command input (and output) history.	<code>%hist</code>
<code>%pdb</code>	Automatically enter python debugger after any exception.	<code>%pdb</code>
<code>%cpaste</code>	Opens up a special prompt for manually pasting Python code for execution.	<code>%cpaste</code>
<code>%reset</code>	Delete all variables and names defined in the current namespace.	<code>%reset</code>
<code>%run</code>	Run a python script inside a notebook.	<code>%run script.py</code>
<code>%who, %who_ls, %whos</code>	Display variables defined in the interactive namespace, with varying levels of verbosity.	<code>%who, %who_ls, %whos</code>
<code>%xdel</code>	Delete a variable in the local namespace. Clear any references to that variable.	<code>%xdel variable</code>

Timing Code (%time and %timeit)

IPython has two magic functions: `%time` and `%timeit` to automate the process of timing statement execution.

Given:

```
{ } data = ['http://www.google.com', 'http://www.treasure-
data.com', 'ftp://myuploads.net', 'John Hammink'] * 1000
```

Statement	Explanation	Example Input	Example output
<code>%time</code>	Times a single statement.	In [561]: <code>%time method = [a for a in data if a.startswith('http')]</code>	CPU times: user 772 μ s, sys: 9 μ s, total: 781 μ s Wall time: 784 μ s
<code>%timeit</code>	Runs a statement multiple times to get an average runtime.	<code>%timeit [a for a in data if a.startswith('http')]</code>	1000 loops, best of 3: 713 μ s per loop

Profiling (%prun and %run -p)

Statement	Explanation
<code>%prun my_function()</code>	Runs a function (or code) within the python profiler.
<code>%run -p my_script.py</code>	Runs a script under the profiler.

Matplotlib

Matplotlib is the default library for plotting in both IPython and Jupyter Notebooks. Here's how you would plot a graph from data you have stored in Treasure Data.

```
{ } %matplotlib inline

import os
import pandas as pd
import pandas_td as td

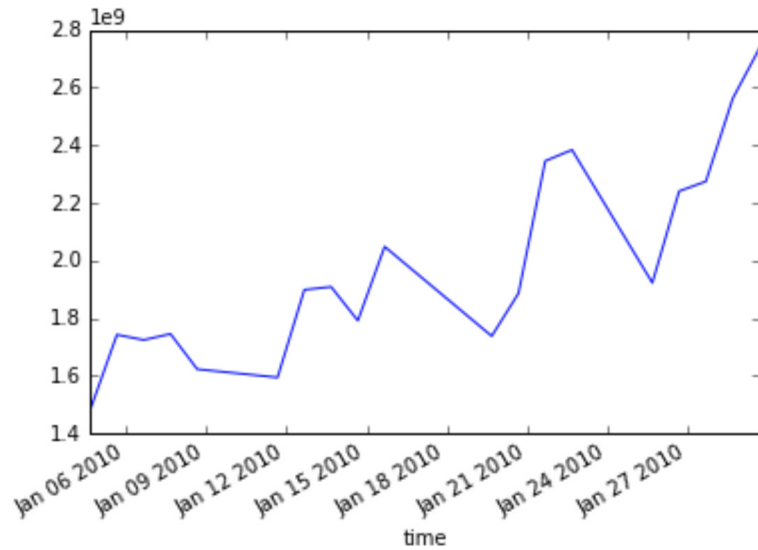
#Initialize our connection to Treasure Data
con = td.connect(apikey=os.environ['MASTER_TD_API_KEY'],
endpoint='https://api.treasuredata.com')

#Query engine for Presto
engine=con.query_engine(database='sample_datasets',
type='presto')

#Get all rows from 2010-01-01 to 2010-02-01
df = td.read_ttable('nasdaq', engine, limit=None,
time_range=('2010-01-01', '2010-02-01'),
index_col='time', parse_dates={'time': 's'})

#Plot the sum of volumes, grouped by time (index level = 0)
df.groupby(level=0).volume.sum().plot()
```


This returns



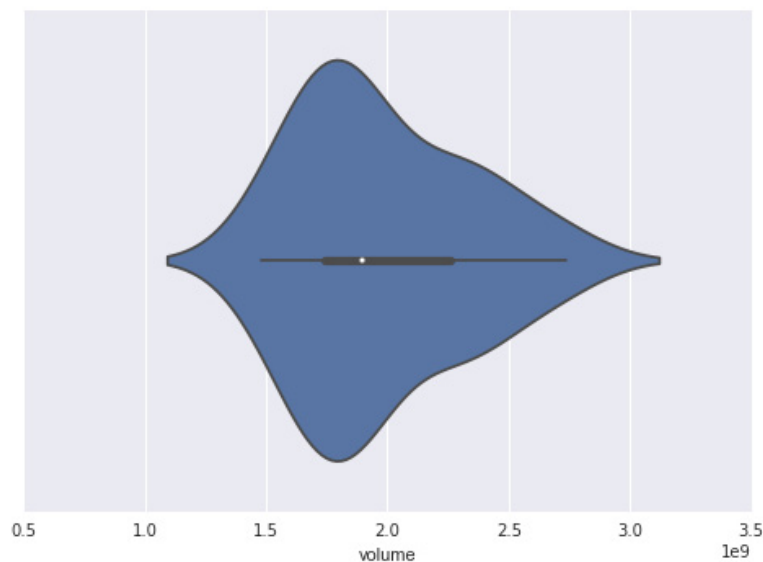
Seaborn

Based on Matplotlib, Seaborn has a strong focus on visualizing statistical results such as univariate and bivariate linear regression, data matrices, time series and more. Seaborn also offers better aesthetics by default with built-in themes and color palettes.

You can read more about Seaborn here: <http://stanford.edu/~mwaskom/software/seaborn/>

```
{ }
```

```
#Create a violin plot
import seaborn as sns
sns.violinplot(df.groupby(level=0).volume.sum())
```



Pandas Treasure Data connector

Pandas-td is an open source tool that allows you to connect to Treasure Data on the backend, and use Pandas from within Jupyter Notebooks. Pandas-td is available, as open source, here: <https://github.com/treasure-data/pandas-td>

Connecting to Treasure Data



```
#Initialize our connection to Treasure Data
con = td.connect(apikey=os.environ['MASTER_TD_API_KEY'], endpoint='https://api.treasuredata.com')
```

Listing tables



```
#list all tables from Treasure Data database 'sample_datasets'
con.tables('sample_datasets')
```

Reading a table



```
#set the engine to Presto
engine=con.query_engine(database='sample_datasets',
type='presto')

#Get 3 lines, converting time to DateTimeIndex
td.read_td_table('nasdaq', engine, limit=3,
                  index_col='time', parse_dates={'time':
's'})
```

Sending queries



```
td.read_td_query('''
select count(*) from nasdaq where symbol='AAPL'
''', engine)

td.read_td_query('''
select time, close from nasdaq where symbol = 'AAPL'
''', engine, index_col='time', parse_dates={'time':
's'})
```

Sampling data from a table



```
#1-percent sample, with 100k rows
td.read_td_table('nasdaq', engine, limit=100000,
                  sample=0.01,
                  index_col='time', parse_dates={'time': 's'})
```

Dataframes

Dataframe is a core data structure for Pandas, a popular Python data analysis framework. Treasure Data's Pandas connector maps Dataframes to Treasure Data's tables and vice versa.

Getting Data into a dataframe

```
{ } #Get all rows from 2010-01-01 to 2010-02-01
df = td.read_td_table('nasdaq', engine, limit=None,
                      time_range=('2010-01-01', '2010-02-01'),
                      index_col='time', parse_dates={'time': 's'})
```

Grouping Data in a dataframe

```
{ } #Plot the sum of volumes, grouped by time (index level
=0)
df.groupby(level=0).volume.sum().plot()
```

Getting number of rows in a dataframe

```
{ } len(df)
```

Sampling data from a large set into a dataframe

```
{ } #1-percent sample, with 100k rows
df= td.read_td_table('nasdaq', engine, limit=100000,
                    sample=0.01,
                    index_col='time', parse_dates={'time': 's'})
```

System Commands

To run any command at the system shell, simply prefix it with `!`, e.g.:



```
!ping www.google.com
!ls -l
...
```

Statement	Explanation
<code>!cmd</code>	Run <code>cmd</code> in the system shell.
<code>b?</code>	Displays the variable's type with a description.
<code>b??</code>	Displays the source code of a function, <code>b</code> .
<code>output = !cmd args</code>	Run <code>cmd</code> and store stdout in the variable <code>output</code> .
<code>%alias alias_name command</code>	Create an alias for a given command.
<code>%bookmark</code>	Bookmark a directory from within IPython.
<code>%cd directory</code>	Change the working directory.
<code>%pwd</code>	Get the current working directory.
<code>%env</code>	Get the system environment variables as a dict.

To work with the current directory stack:

Statement	Explanation
<code>%pushd directory</code>	Place the current directory on the stack and change to the specified target directory.
<code>%popd</code>	Change to the directory popped from the stack.
<code>%dirs</code>	List the current directory stack.

Command history

Statement	Explanation	Example
<code>%hist</code>	Print command input (and output) history.	%hist
<code>%dhist</code>	Get the visited directory history.	%dhist
	returns	Directory history (kept in <code>_dh</code>) Directory history (kept in <code>_dh</code>) 0: /Users/you/Documents/notebooks/ 1: /Users/you/Documents
<code>_</code>	Returns variable from the previous line.	
<code>_33</code>	Returns output variable from line 33.	
<code>_i33</code>	Returns input variable from line 33.	
<code>%pastebin</code>	Pastebins lines from your current session.	<code>%pastebin 3 18-20 ~1/1-5</code>
	returns	Lines 3 and lines 18-20 from current session, and lines 1 - 5 from the previous session to gist.github.com pastebin link.
<code>%logstart</code>	Initiates logging for a session.	%logstart

IPython Debugger

The following magic commands support debugging.

Statement	Explanation	Example
<code>%pdb</code>	Automatically enter python debugger after any exception.	<code>%pdb</code>
<code>%run -t</code>	Run and time execution of a script.	<code>%run -t my_script.py</code>
<code>%run -d</code>	Run script under the Python debugger.	<code>%run -d my_script.py</code>
<code>%run -p</code>	Run a script under the profiler.	<code>%run -p my_script.py</code>
<code>%run -i</code>	Run a script, giving it access to the variables from the current IPython namespace.	<code>%run -i my_script.py</code>
<code>%debug</code>	Enters the interactive debugger.	<code>%debug</code>

Next Steps

We believe that, regardless of your experience level, it's easy to get started analyzing larger datasets straight from within Jupyter Notebooks.

1. Sign up for Treasure Data and get your master API key from [Treasure Data Console](#).
2. Get [Condas](#) as an environment manager, and install dependencies.
3. Install Treasure Data Pandas Connector and launch your first notebook.

You can also [request a demo](#) from Treasure Data. We're here to help you get started!

