

## CHAPTER 2: PYTHON STACK

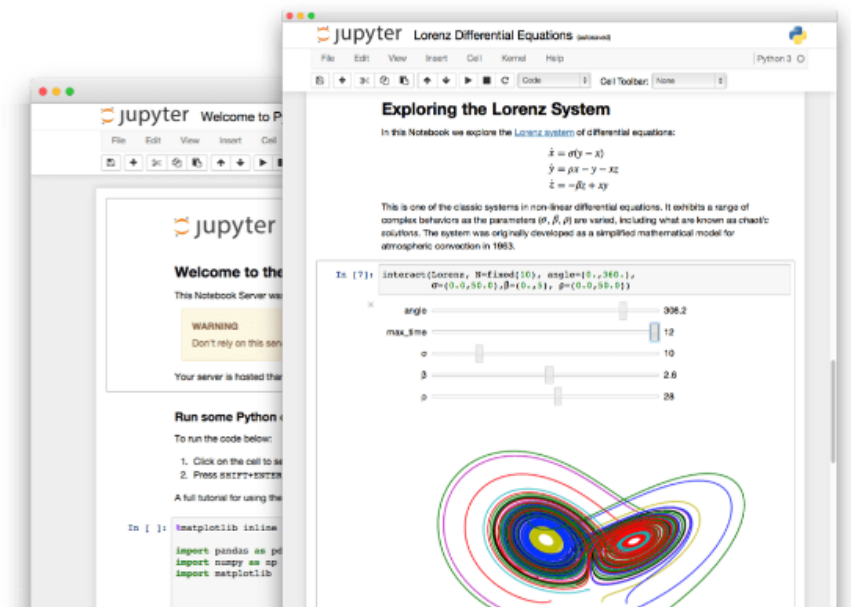
# Python Stack --- FREE

- In order to use text code, you'll need a stack of tools.
- “You will need a number of Python modules: Jupyter, NumPy, Pandas, Matplotlib, and Scikit-Learn.”

# Jupyter Notebook

[Install](#)[About Us](#)[Community](#)[Documentation](#)[NBViewer](#)[Widgets](#)[Blog](#)

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.



## The Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.



### Language of choice

The Notebook has support for over 40 programming languages, including Python, R, Julia, and Scala.



### Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the [Jupyter Notebook Viewer](#).



### Interactive output

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.



### Big data integration

Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, TensorFlow.



## Steering Council

The role of the Jupyter Steering Council is to ensure, through working with and serving the broader Jupyter community, the long-term well-being of the project, both technically and as a community. The Jupyter Steering Council currently consists of the following members (in alphabetical order).



Damian Avila  
Continuum Analytics  
[@damianavila](#) on GitHub



Matthias Bussonnier  
UC Berkeley  
[@carreau](#) on GitHub



Sylvain Corlay  
QuantStack  
[@sylvaincorlay](#) on GitHub



Jonathan Frederic  
Cal Poly, San Luis Obispo  
[@jofreder](#) on GitHub



Brian Granger  
Cal Poly, San Luis Obispo  
[@ellisonbg](#) on GitHub



Jason Grout  
Bloomberg  
[@jasongrout](#) on GitHub



Jessica Hamrick  
UC Berkeley  
[@hamrick](#) on GitHub



Paul Ivanov  
Bloomberg  
[@ivanov](#) on GitHub



Thomas Kluyver  
University of Southampton  
[@takluyver](#) on GitHub



Kyle Kelley  
Netflix  
[@rgbkrk](#) on GitHub



Peter Parente  
MaxPoint  
[@parente](#) on GitHub



Fernando Perez  
UC Berkeley  
[@perez](#) on GitHub



Min Ragan-Kelley  
Simula Research Lab  
[@minrk](#) on GitHub



Steven Silvester  
Continuum Analytics  
[@silnk1073](#) on GitHub



Carol Willing  
Cal Poly  
[@willinc](#) on GitHub

Kxrr Fix incorrect description in chapter 2

91b142c on Nov 24, 2017

4 contributors

1.29 MB

Download History

## Chapter 2 – End-to-end Machine Learning project

Welcome to Machine Learning Housing Corp.! Your task is to predict median house values in Californian districts, given a number of features from these districts.

This notebook contains all the sample code and solutions to the exercises in chapter 2.

**Note:** You may find little differences between the code outputs in the book and in these Jupyter notebooks: these slight differences are mostly due to the random nature of many training algorithms: although I have tried to make these notebooks' outputs as constant as possible, it is impossible to guarantee that they will produce the exact same output on every platform. Also, some data structures (such as dictionaries) do not preserve the item order. Finally, I fixed a few minor bugs (I added notes next to the concerned cells) which lead to slightly different results, without changing the ideas presented in the book.

## Setup

First, let's make sure this notebook works well in both python 2 and 3, import a few common modules, ensure Matplotlib plots figures inline and prepare a function to save the figures:

```
In [1]: # To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

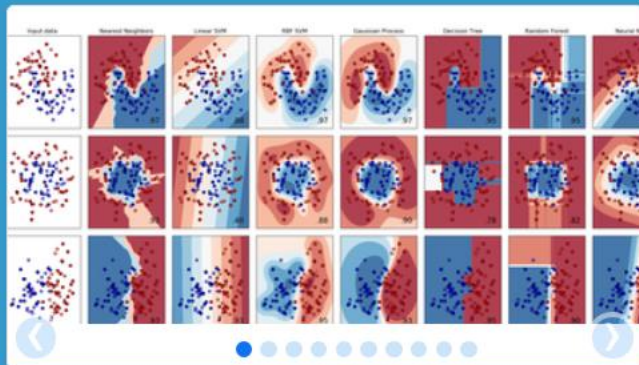
# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

## Get the data

- Course Material in Notebooks on Github





# scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ... [— Examples](#)

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.  
**Algorithms:** SVR, ridge regression, Lasso, ... [— Examples](#)

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes  
**Algorithms:** k-Means, spectral clustering, mean-shift, ... [— Examples](#)

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency  
**Algorithms:** PCA, feature selection, non-negative matrix factorization. [— Examples](#)

## Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning  
**Modules:** grid search, cross validation, metrics. [— Examples](#)

## Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.  
**Modules:** preprocessing, feature extraction. [— Examples](#)



Install



Getting Started



Documentation



Report Bugs



Blogs

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:



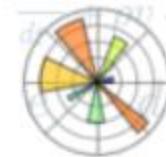
**NumPy**

Base N-dimensional  
array package



**SciPy library**

Fundamental library  
for scientific  
computing



**Matplotlib**

Comprehensive 2D  
Plotting

**IP[y]:**  
IPython

**IPython**

Enhanced Interactive  
Console



**Sympy**

Symbolic  
mathematics



**pandas**

Data structures &  
analysis





# NumPy

Scipy.org

## NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

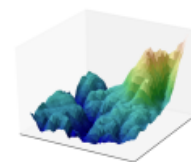
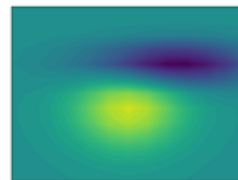
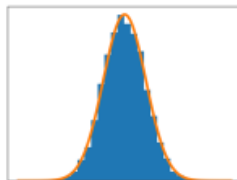
NumPy is licensed under the [BSD license](#), enabling reuse with few restrictions.

# matplotlib

[home](#) | [examples](#) | [tutorials](#) | [pyplot](#) | [docs](#) »

## Introduction

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and **IPython** shell, the **jupyter** notebook, web application servers, and four graphical user interface toolkits.

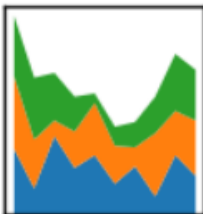
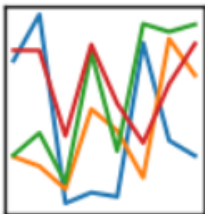


Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the [sample plots](#) and [thumbnail gallery](#).

For simple plotting the `pyplot` module provides a MATLAB-like interface, particularly when combined with `IPython`. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

# pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



[home](#) // [about](#) // [get pandas](#) // [documentation](#) // [community](#) // [talks](#) // [donate](#)

## Python Data Analysis Library

*pandas* is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the [Python](#) programming language.

*pandas* is a [NumFOCUS](#) sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to [donate](#) to the project.

A Fiscally Sponsored Project of

# NUMFOCUS

OPEN CODE = BETTER SCIENCE

### VERSIONS

#### Release

0.22.0 - December 2017

[download](#) // [docs](#) // [pdf](#)

#### Development

0.23.0 - 2018

[github](#) // [docs](#)

# Now Onto Our Notebook

```
In [1]: # To support both python 2 and python 3
        from __future__ import division, print_function, unicode_literals

        # Common imports
        import numpy as np
        import os

        # to make this notebook's output stable across runs
        np.random.seed(42)

        # To plot pretty figures
        %matplotlib inline
        import matplotlib
        import matplotlib.pyplot as plt
        plt.rcParams['axes.labelsize'] = 14
        plt.rcParams['xtick.labelsize'] = 12
        plt.rcParams['ytick.labelsize'] = 12

        # Where to save the figures
        PROJECT_ROOT_DIR = "."
        CHAPTER_ID = "end_to_end_project"
        IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)

        def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
            path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
            print("Saving figure", fig_id)
            if tight_layout:
                plt.tight_layout()
            plt.savefig(path, format=fig_extension, dpi=resolution)
```

# Get Data!

## Get the data

```
In [2]: import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

```
In [3]: fetch_housing_data()
```



# Pandas for Data!

```
In [4]: import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

```
In [5]: housing = load_housing_data()
housing.head()
```

```
Out[5]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_household_income
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	45260.39
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	35860.52
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	35210.21
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	34130.72
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	34220.19

## Table Of Contents

- What's New
- Installation
- Contributing to pandas
- Package overview
- 10 Minutes to pandas
- Tutorials
- Cookbook
- Intro to Data Structures
- Essential Basic Functionality
- Working with Text Data
- Options and Settings
- Indexing and Selecting Data
- Multindex / Advanced Indexing
- Computational tools
- Working with missing data
- Group By: split-apply-combine
- Merge, join, and concatenate
- Reshaping and Pivot Tables
- Time Series / Date functionality
- Time Deltas
- Categorical Data
- Visualization
- Styling
- IO Tools (Text, CSV, HDF5, ...)
- CSV & Text files
  - Parsing options
    - Basic
      - Column and Index Locations and Names
    - General Parsing Configuration
    - NA and Missing Data Handling
    - Datetime Handling
    - Iteration
    - Quoting

## IO Tools (Text, CSV, HDF5, ...)

The pandas I/O API is a set of top level reader functions accessed like `pd.read_csv()` that generally return a pandas object. The corresponding writer functions are object methods that are accessed like `df.to_csv()`

Format Type	Data Description	Reader	Writer
text	CSV	<a href="#">read_csv</a>	<a href="#">to_csv</a>
text	JSON	<a href="#">read_json</a>	<a href="#">to_json</a>
text	HTML	<a href="#">read_html</a>	<a href="#">to_html</a>
text	Local clipboard	<a href="#">read_clipboard</a>	<a href="#">to_clipboard</a>
binary	MS Excel	<a href="#">read_excel</a>	<a href="#">to_excel</a>
binary	HDF5 Format	<a href="#">read_hdf</a>	<a href="#">to_hdf</a>
binary	Feather Format	<a href="#">read_feather</a>	<a href="#">to_feather</a>
binary	Parquet Format	<a href="#">read_parquet</a>	<a href="#">to_parquet</a>
binary	Msgpack	<a href="#">read_msgpack</a>	<a href="#">to_msgpack</a>
binary	Stata	<a href="#">read_stata</a>	<a href="#">to_stata</a>
binary	SAS	<a href="#">read_sas</a>	
binary	Python Pickle Format	<a href="#">read_pickle</a>	<a href="#">to_pickle</a>
SQL	SQL	<a href="#">read_sql</a>	<a href="#">to_sql</a>
SQL	Google Big Query	<a href="#">read_gbq</a>	<a href="#">to_gbq</a>

Here is an informal performance comparison for some of these IO methods.

**Note:** For examples that use the `StringIO` class, make sure you import it according to your Python version, i.e. `from StringIO import StringIO` for Python 2 and `from io import StringIO` for Python 3.

## CSV &amp; Text files

The two workhorse functions for reading text files (a.k.a. flat files) are `read_csv()` and `read_table()`. They both use the same parsing code to intelligently convert tabular data into a DataFrame object. See the [cookbook](#) for some advanced strategies.

# Some Pandas Basics

[pandas 0.22.0 documentation »](#)

## Table Of Contents

- What's New
- Installation
- Contributing to pandas
- Package overview
- 10 Minutes to pandas
- Tutorials
- Cookbook
- Intro to Data Structures
- Essential Basic Functionality
  - Head and Tail
  - Attributes and the raw ndarray(s)
  - Accelerated operations
  - Flexible binary operations
    - Matching / broadcasting behavior
    - Missing data / operations with fill values
    - Flexible Comparisons
    - Boolean Reductions
    - Comparing if objects are equivalent
    - Comparing array-like objects
    - Combining overlapping data sets
    - General DataFrame Combine
  - Descriptive statistics
    - Summarizing data: describe
    - Index of Min/Max Values
    - Value counts (histogramming) / Mode
    - Discretization and quantiling

## Essential Basic Functionality

Here we discuss a lot of the essential functionality common to the pandas data structures. Here's how to create some of the objects used in the examples from the previous section:

```
In [1]: index = pd.date_range('1/1/2000', periods=8)
In [2]: s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
In [3]: df = pd.DataFrame(np.random.randn(8, 3), index=index,
...:                      columns=['A', 'B', 'C'])
...:
In [4]: wp = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],
...:                 major_axis=pd.date_range('1/1/2000', periods=5),
...:                 minor_axis=['A', 'B', 'C', 'D'])
...:
```

## Head and Tail

To view a small sample of a Series or DataFrame object, use the `head()` and `tail()` methods. The default number of elements to display is five, but you may pass a custom number.

```
In [5]: long_series = pd.Series(np.random.randn(1000))
In [6]: long_series.head()
Out[6]:
0    0.229453
1    0.304418
2    0.736135
3   -0.859631
4   -0.424100
dtype: float64
```

# read\_csv() head()

```
In [4]: import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

```
In [5]: housing = load_housing_data()
housing.head()
```

```
Out[5]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_housing_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200