David Guy Brizan & Paul Intrevado

Machine Learning

CS 686-02

Spring 2018

Table of Contents

Section 1

About this Class

The Warden

Who Am I?

- Ph.D. Operations Management (McGill University)
  - Research focused on service operations
  - Model and solve optimization problems (e.g., MIPs)
- B.Sc., M.Sc. Industrial Engineering (Purdue University)
- B. Commerce (McGill University)
- Assistant Prof @ USF as of August 2014
- I teach or have taught
  - MSAN 593 - Exploratory Data Analysis with R
  - MSAN 601 - Linear Regression Analysis with R
  - MSAN 605/625/627/632 - Practicum
- MS Analytics Practicum Director [2014 - 2017]
- Founding Associate Director of Data Institute [2016 - 2017]

What I Do

$$\min_{\mathbf{y},\mathbf{z}^+,\mathbf{z}^-} \sum_{j\in\mathbb{J}}\sum_{\nu\in\mathbb{V}}\sum_{t\in\mathbb{T}} r^{(t-1)}\left[\sum_{\omega\in\Omega}\left[f^+_{j\nu\omega}z^+_{j\nu\omega t} - f^-_{j\nu\omega}z^-_{j\nu\omega t}\right] + \gamma_\nu\left(\kappa_{j\nu t} - \rho_{j\nu t} + \sum_{i\in\mathbb{I}} y_{ij\nu t}\right)\right] \tag{4.1}$$

subject to

$$\kappa_{j\nu t} = \kappa_{j\nu(t-1)} + \sum_{\omega\in\Omega} C_{\nu\omega}\left(z^+_{j\nu\omega t} - z^-_{j\nu\omega t}\right) \qquad \forall\ j\in\mathbb{J}, \nu\in\mathbb{V}, t\in\mathbb{T} \tag{4.2}$$

$$\rho_{j\nu t} = \rho_{j\nu(t-1)} + \sum_{\omega\in\Omega} C_{\nu\omega}\left(z^+_{j\nu\omega t} - z^-_{j\nu\omega t}\right) - \sum_{i\in\mathbb{I}} y_{ij\nu(t-1)} + \alpha_{j\nu(t-1)} - \psi^{(\nu+1\to\nu)}_{j(t-1)} + \psi^{(\nu\to\nu-1)}_{j(t-1)}$$
$$\forall\ j\in\mathbb{J}, \nu\in\mathbb{V}, t\in\mathbb{T} \tag{4.3}$$

$$\alpha_{j\nu t} = \mu_{\nu t}\left(\kappa_{j\nu t} - \rho_{j\nu t} + \sum_{i\in\mathbb{I}} y_{ij\nu t}\right) \qquad \forall\ j\in\mathbb{J}, \nu\in\mathbb{V}, t\in\mathbb{T}\setminus|\mathbb{T}| \tag{4.4}$$

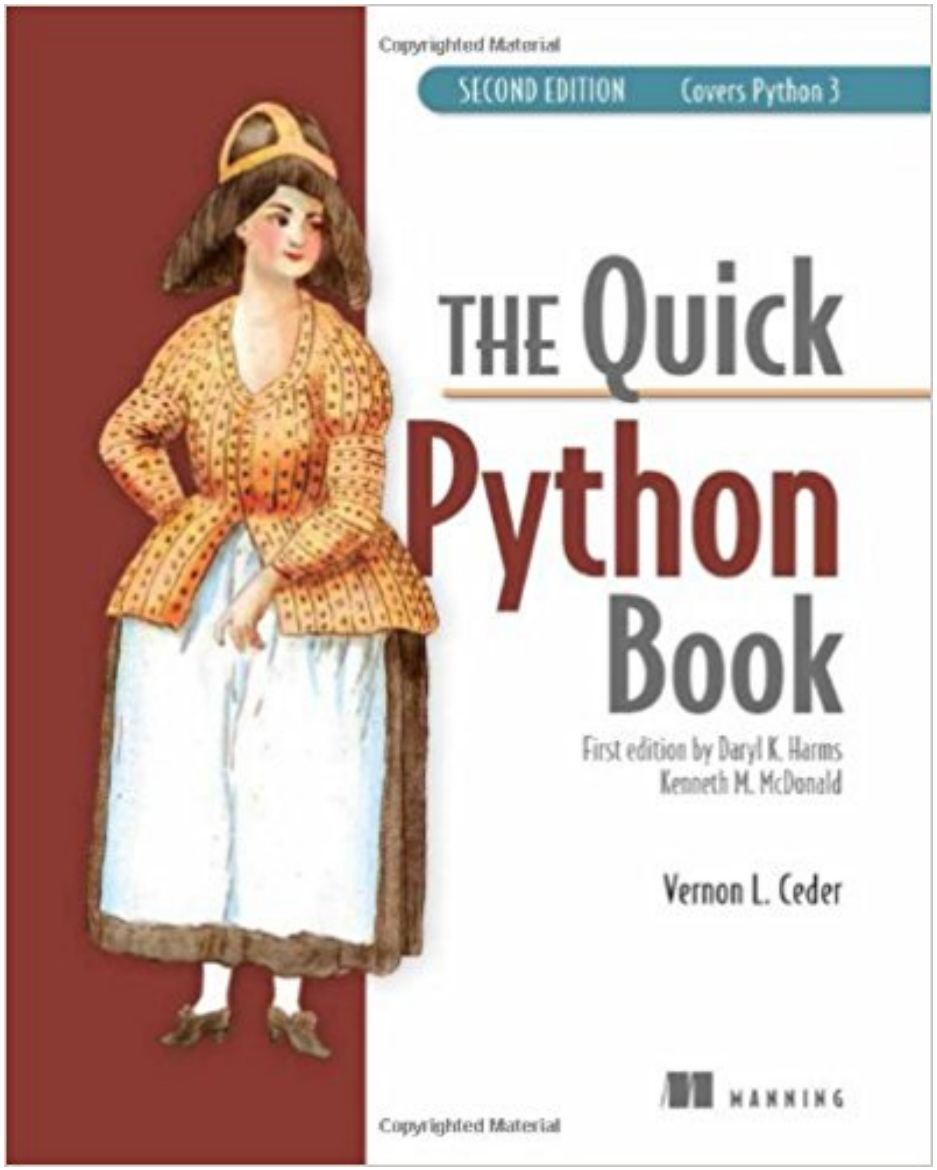
$$\psi^{(\nu\to\nu-1)}_{jt} = \tau^{(\nu\to\nu-1)}_t\left(\kappa_{j\nu t} - \rho_{j\nu t} + \sum_{i\in\mathbb{I}} y_{ij\nu t} - \alpha_{j\nu t}\right) \qquad \forall\ j\in\mathbb{J}, \nu\in\mathbb{V}\setminus\{1\}, t\in\mathbb{T}\setminus|\mathbb{T}| \tag{4.5}$$

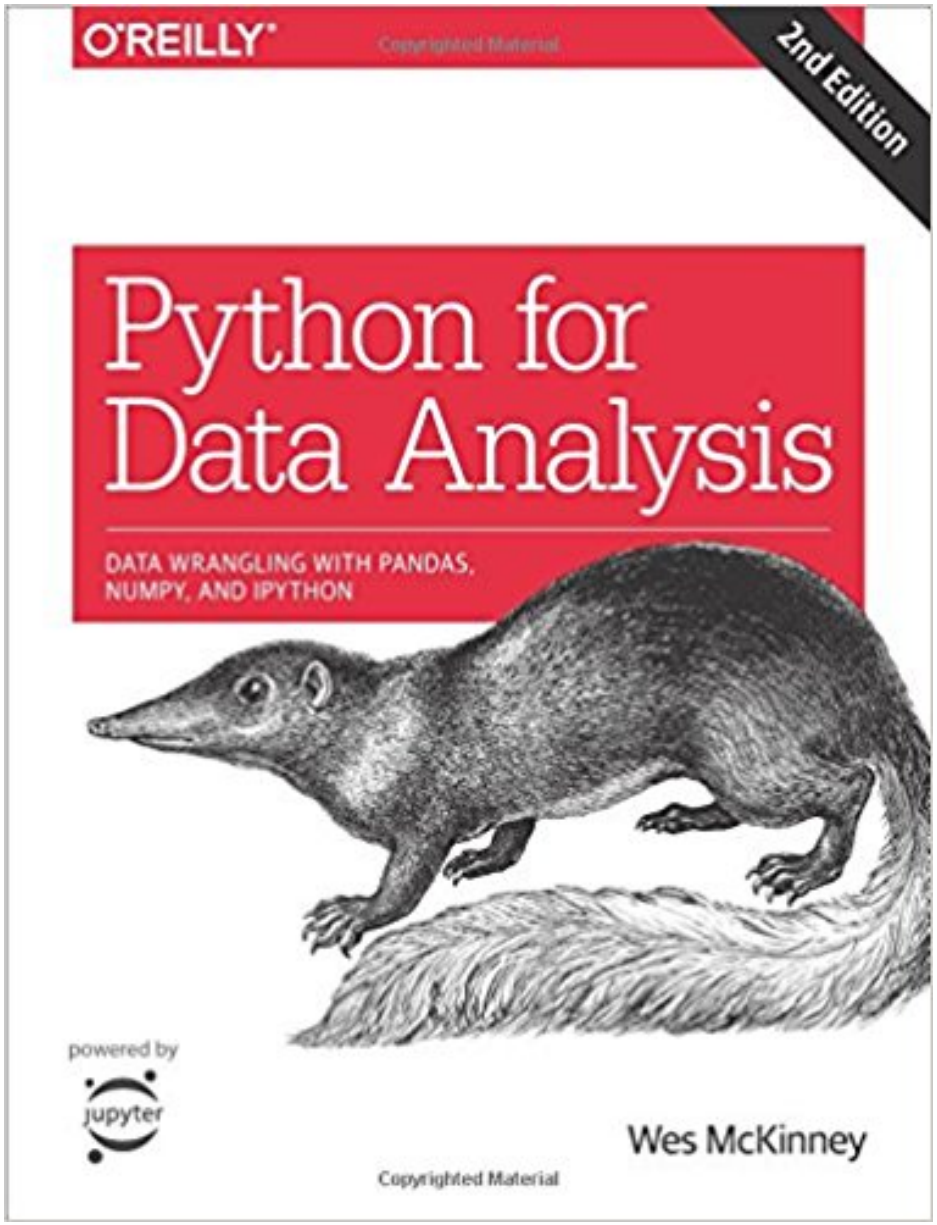
$$\eta_{\nu t}\lambda_{it} + \psi^{(\nu+1\to\nu)}_{i(t-1)} = \sum_{j\in\mathbb{J}} y_{ij\nu t} \qquad \forall\ i\in\mathbb{I}, \nu\in\mathbb{V}, t\in\mathbb{T} \tag{4.6}$$

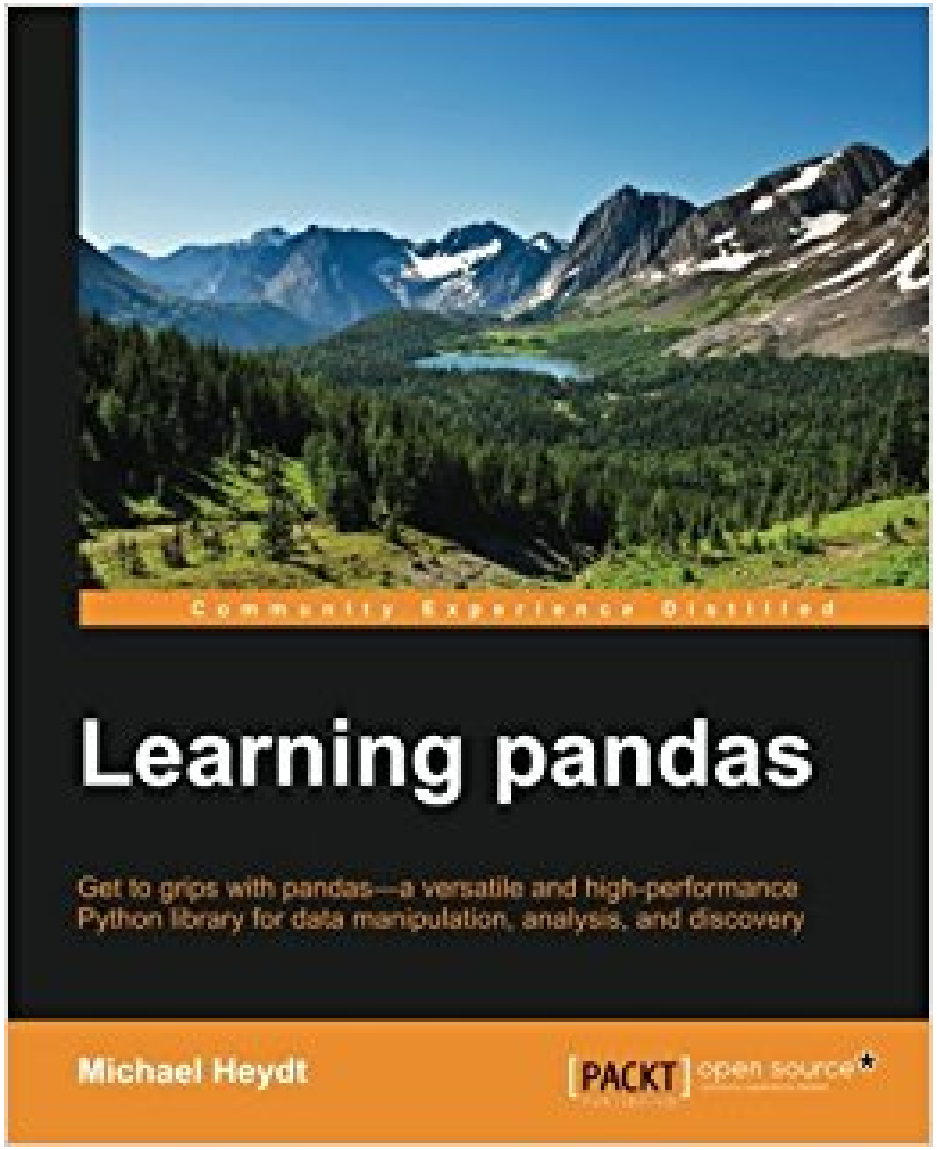
$$\eta_{\nu t}\lambda_{it}\pi_{i\nu t} + \psi^{(\nu+1\to\nu)}_{i(t-1)} \le y_{ij\nu t} \qquad \forall\ i=j\in\mathbb{J}, \nu\in\mathbb{V}, t\in\mathbb{T} \tag{4.7}$$

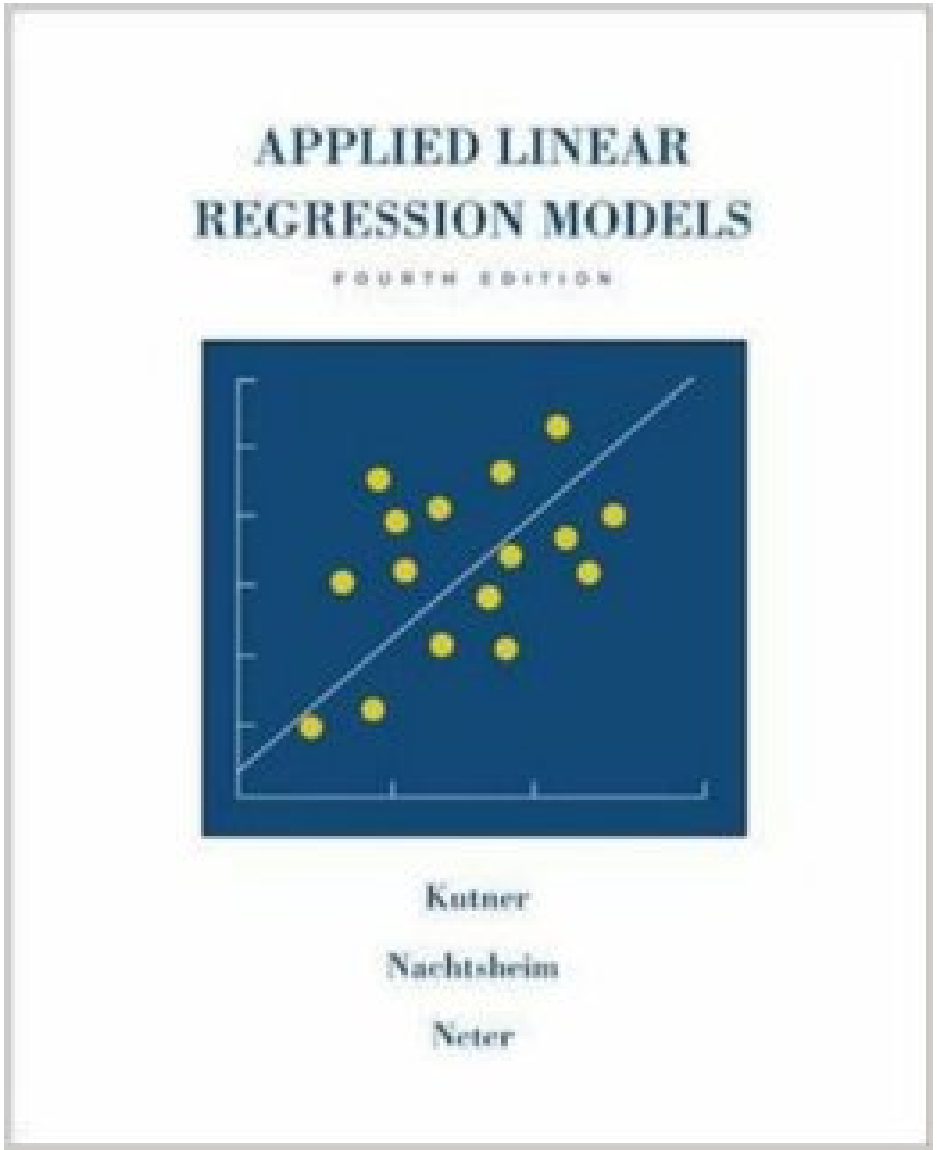
Subsection 3

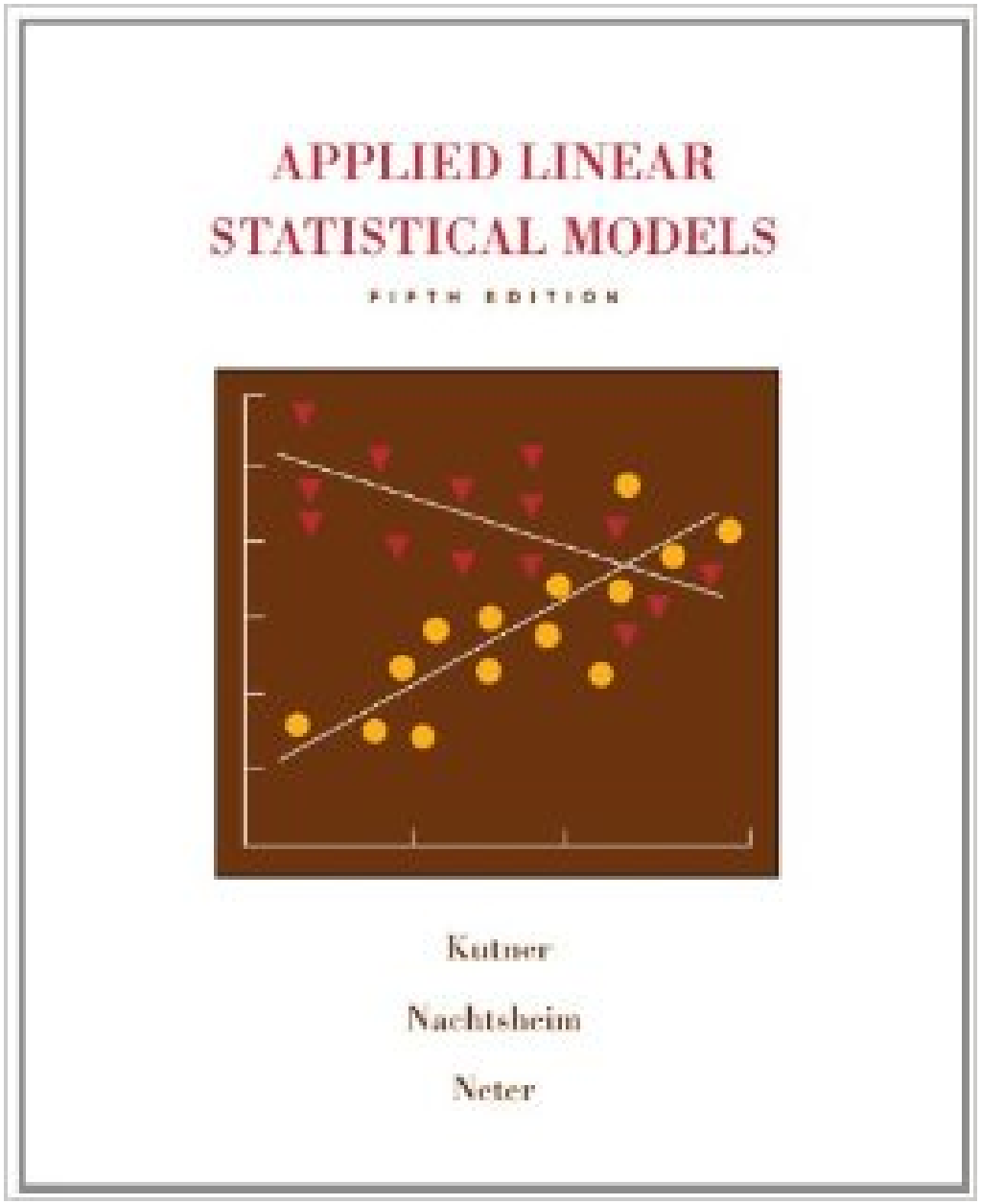
Reference Textbooks

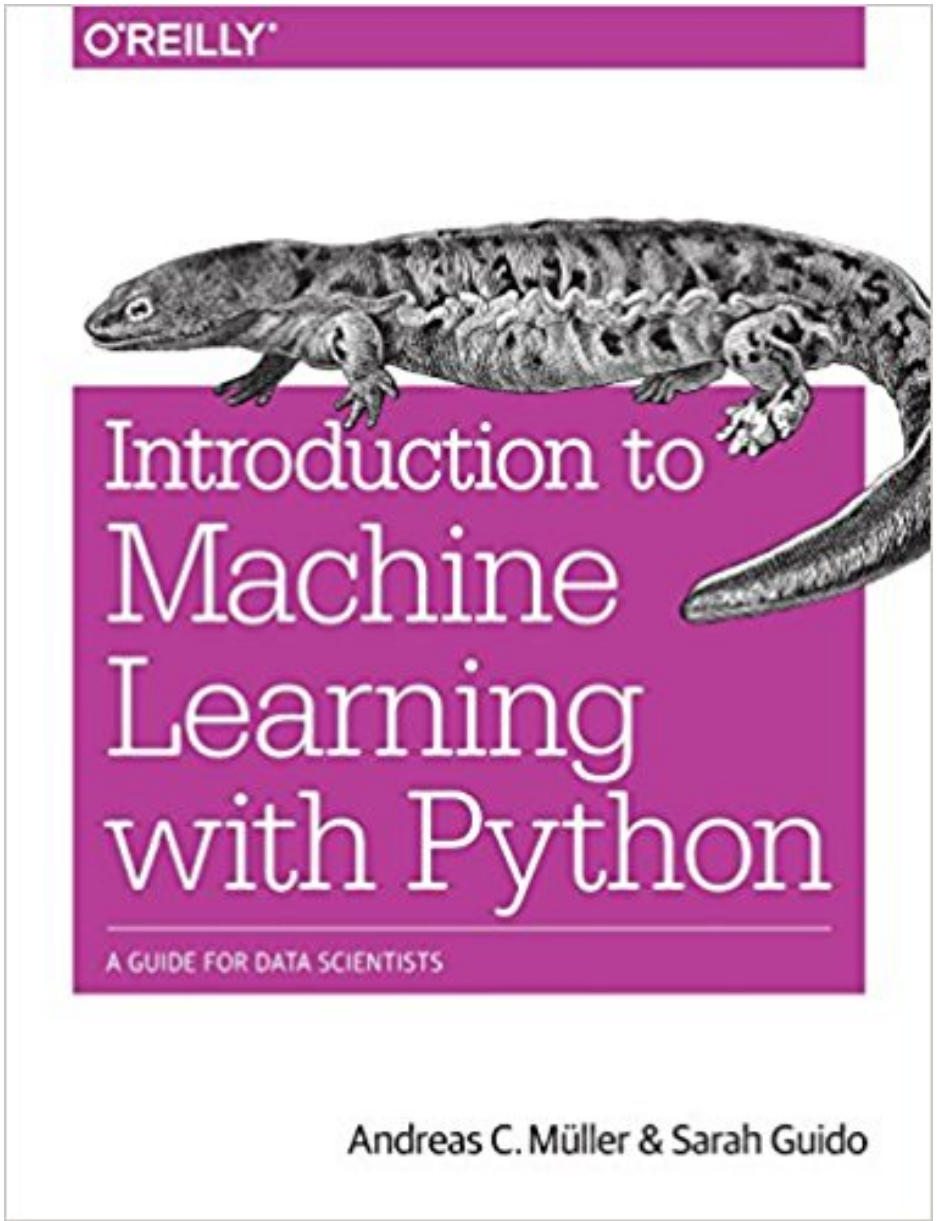

Copyrighted Material
SECOND EDITION
Covers Python 3
THE Quick
Python
Book
First edition by Daryl K. Harms
Kenneth M. McDonald
Vernon L. Ceder
MANNING
Copyrighted Material


O'REILLY
Copyrighted Material
2nd Edition
Python for Data Analysis
DATA WRANGLING WITH PANDAS, NUMPY, AND IPYTHON
powered by
jupyter
Wes McKinney
Copyrighted Material


Community Experience Distilled
Learning pandas
Get to grips with pandas—a versatile and high-performance Python library for data manipulation, analysis, and discovery
Michael Heydt
[PACKT] open source


APPLIED LINEAR
REGRESSION MODELS
FOURTH EDITION
Kutner
Nachtsheim
Neter


APPLIED LINEAR
STATISTICAL MODELS
FIFTH EDITION
Kutner
Nachtsheim
Neter


O'REILLY
Introduction to
Machine
Learning
with Python
A GUIDE FOR DATA SCIENTISTS
Andreas C. Müller & Sarah Guido

Section 2

A Brief Introduction to Python

Python 3

- Python 2 will be not longer be supported as of 2020
- We will **exclusively** use Python 3 for this course
- Installing Python
  - `https://www.python.org/`
  - `https://anaconda.org/`
- `https://www.python.org/` Python 3 distributions include the `pip3` package manager
- `https://anaconda.org/` Anaconda 3 distributions include the `conda` package manager

n.b. This class will **exclusively** support `pip3`

n.b. Don't conflate `pip3` with `pip`; the latter installs packages to Python 2

Jupyter Notebooks

- Once you have Python 3 installed, open a terminal window and run the following command to install Jupyter Notebooks

```
Pauls-iMac:~ paul$ pip3 install jupyter
```

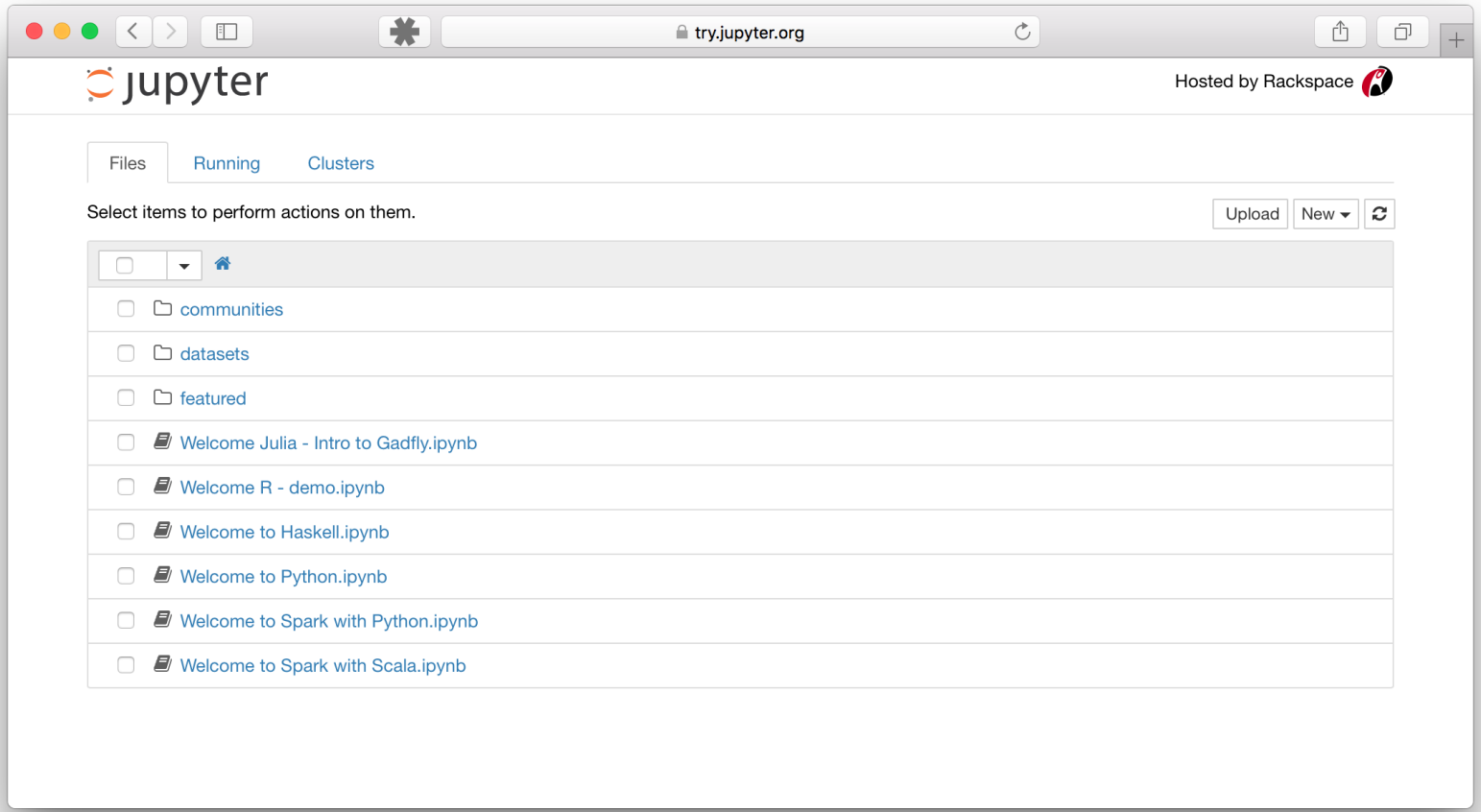
- Opening an instance of Jupyter Notebook is achieved by running the following command in a terminal window

```
Pauls-iMac:~ paul$ jupyter notebook
```

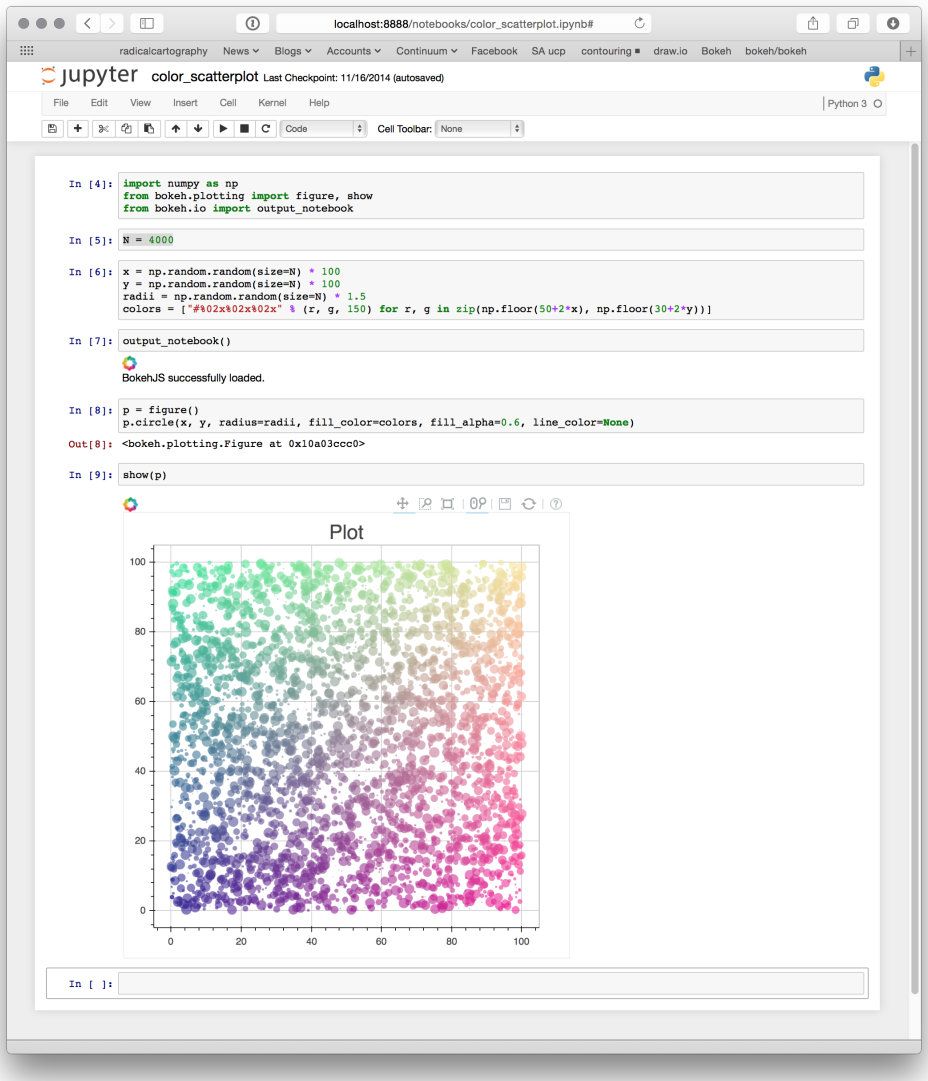
- Jupyter, an interactive programming environment, will open in your browser to the Jupyter Notebook Dashboard

Jupyter Notebook Dashboard

In the top right of the Dashboard, click on the button labeled `New` to run a new Python 3 kernel and open a Notebook



Jupyter Notebook using a Python 3 Kernel



Why use Jupyter Notebooks?

- Interactive environment works well with interpreted languages such as Python
- Run code snippets and observe output immediately, line by line
- Facilitates the exploration of data
- Great for interactive data visualization
- Can include Markdown and LaTeX
- Can be used with various kernels (not only Python)
  - The name *Jupyter* is an imperfect aggregation of the languages that inspired its creation: **Ju**lia, **Pyt**hon and **R**

n.b. This class will **exclusively** use Jupyter Notebooks for examples, in-class labs, and all deliverables

Jupyter Notebook Cheat Sheet

Python For Data Science *Cheat Sheet*

Jupyter Notebook

Learn More Python for Data Science Interactively at www.DataCamp.com

Saving/Loading Notebooks

Create new notebook
Make a copy of the current notebook
Save current notebook and record checkpoint
Preview of the printed notebook
Close notebook & stop running any scripts
Open an existing notebook
Rename notebook
Revert notebook to a previous checkpoint
Download notebook as
- IPython notebook
- Python
- HTML
- Markdown
- reST
- LaTeX
- PDF

Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells

Cut currently selected cells to clipboard
Paste cells from clipboard above current cell
Paste cells from clipboard on top of current cel
Revert *"Delete Cells"* invocation
Merge current cell with the one above
Move current cell up
Adjust metadata underlying the current notebook
Remove cell attachments
Paste attachments of current cell
Copy cells from clipboard to current cursor position
Paste cells from clipboard below current cell
Delete current cells
Split up a cell from current cursor position
Merge current cell with the one below
Move current cell down
Find and replace in selected cells
Copy attachments of current cell
Insert image in selected cells

Insert Cells

Add new cell above the current one
Add new cell below the current one

Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython | IRkernel | IJulia

Installing Jupyter Notebook will automatically install the IPython kernel.

Restart kernel
Restart kernel & run all cells
Restart kernel & run all cells
Interrupt kernel
Interrupt kernel & clear all output
Connect back to a remote notebook
Run other installed kernels

Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

Download serialized state of all widget models in use
Save notebook with interactive widgets
Embed current widgets

Command Mode:

Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

Executing Cells

Run selected cell(s)
Run current cells down and create a new one above
Run all cells above the current cell
Change the cell type of current cell
toggle, toggle scrolling and clear all output
Run current cells down and create a new one below
Run all cells
Run all cells below the current cell
toggle, toggle scrolling and clear current outputs

View Cells

Toggle display of Jupyter logo and filename
Toggle line numbers in cells
Toggle display of toolbar
Toggle display of cell action icons:
- None
- Edit metadata
- Raw cell format
- Slideshow
- Attachments
- Tags

Asking For Help

Walk through a UI tour
Edit the built-in keyboard shortcuts
Description of markdown available in notebook
Python help topics
NumPy help topics
Matplotlib help topics
Pandas help topics
List of built-in keyboard shortcuts
Notebook help topics
Information on unofficial Jupyter Notebook extensions
IPython help topics
SciPy help topics
SymPy help topics
About Jupyter Notebook

DataCamp
Learn Python for Data Science Interactively

Useful Keyboard Shortcuts in Jupyter

Action	`Shortcut`
Run cell	`ctrl-Enter`
Run cell, select below	`shift-Enter`
Convert cell to code	`Y`
Convert cell to Markdown	`M`
Insert cell above	`A`
Insert cell below	`B`
Cut selected cell	`X`
Delete selected cell	`D,D`
Merge with cell below	`shift-M`

Markdown Cheat Sheet

HEADERS

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

EMPHASIS

```
*This text will be italic*
_This will also be italic_

**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

LISTS

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

IMAGES

```
![GitHub Logo](/images/logo.png)

Format: ![Alt Text](url)
```

LINKS

```
http://github.com - automatic!

[GitHub](http://github.com)
```

BLOCKQUOTES

```
As Grace Hopper said:

> I've always been more interested
> in the future than in the past.
```

As Grace Hopper said:

> I've always been more interested
> in the future than in the past.

BACKSLASH ESCAPES

Markdown allows you to use backslash escapes to generate literal characters which would otherwise have special meaning in Markdown's formatting syntax.

```
\*literal asterisks\*
```

literal asterisks

Markdown provides backslash escapes for the following characters:

\ backslash	() parentheses
` backtick	# hash mark
* asterisk	+ plus sign
_ underscore	- minus sign (hyphen)
{} curly braces	. dot
[] square brackets	! exclamation mark

Useful LaTeX Commands

- Inline commands

Code	Result
`$\frac{a}{b}$`	$\frac{a}{b}$
`Section $\S 1.3$`	Section §1.3
`$\Phi^{\phi + 1} = \pi$`	$\Phi^{\phi+1} = \pi$
`$\sum_{i = 1}^{n} = x_i / n = \bar{x}$`	$\sum_{i=1}^{n} = x_i/n = \bar{x}$

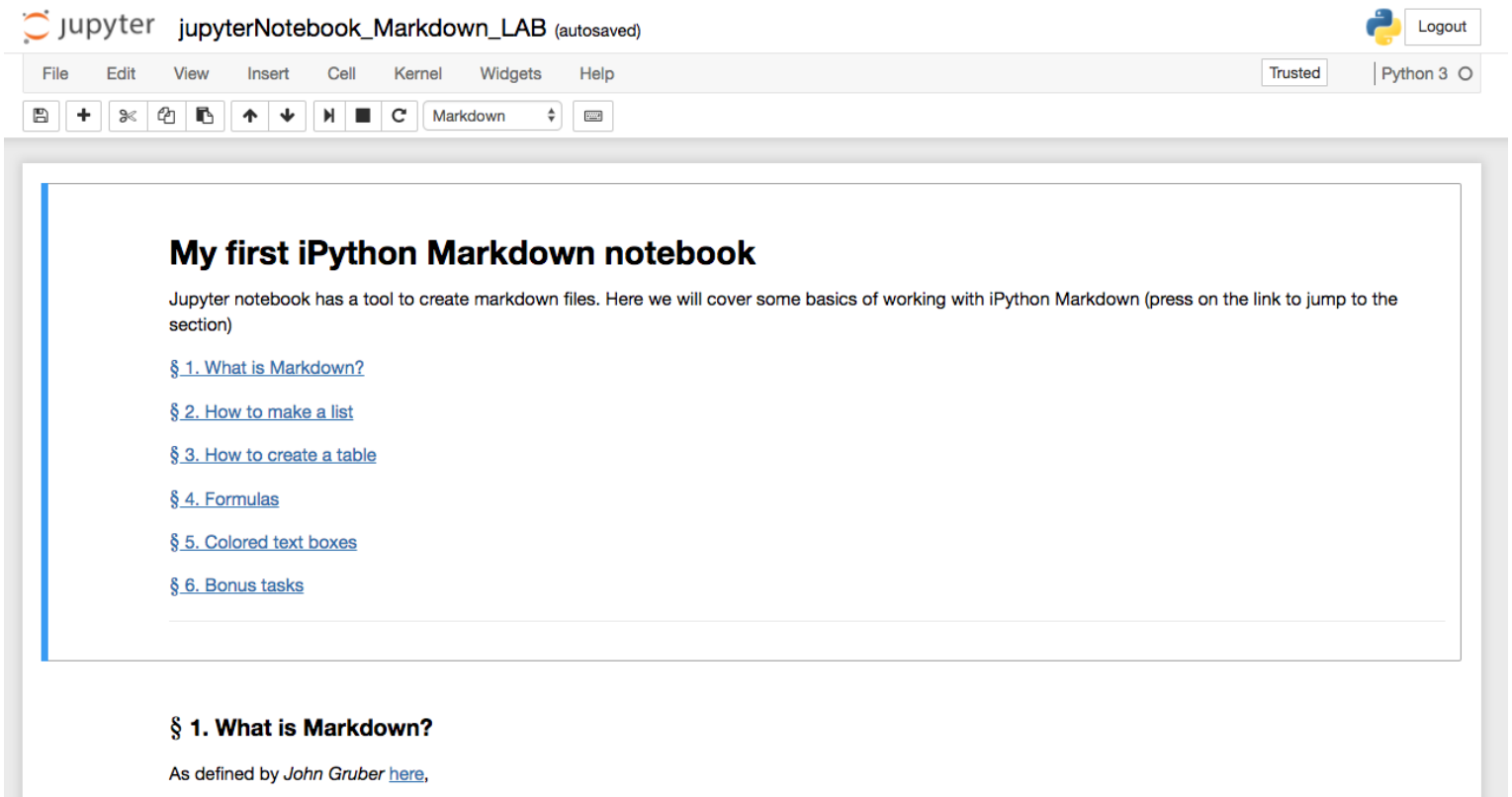
- Mathematical equations

```
\begin{equation}
  e^{i \pi} + 1 = 0
\end{equation}
```

$$e^{i\pi} + 1 = 0 \tag{1}$$

LaTeX, Markdown, Jupyter Lab

`jupyterNotebook_Markdown_LAB.ipynb`



Lists

- Lists are like arrays, do not need to be declared (type or size), and can contain a mixture of other types as its elements, including
  - strings
  - tuples
  - other lists
  - dictionaries
  - etc.
- A list can be indexed from the front or the back

```
>>> x = ["1st", "2nd", "3rd", "4th"]

>>> x[0]
'1st'

>>> x[2]
'3rd'

>>> x[-1]
'4th'
```

Lists [CONT'D]

- Lists can be sliced to generate a subset
  - Slice an array using the `:` operator
  - Subset a list from the *beginning*, starting with 0
  - Subset a list from the *end*, starting with -1
  - When slicing an array from index `a` to `b`, the left most index is *included* in the slice, but the rightmost index is *excluded*, i.e., `[a, b)`

`x = [`		`"1st",`		`"2nd",`		`"3rd",`		`"4th"`		`]`
Positive Indices	0		1		2		3			
Negative Indices	-4		-3		-2		-1			

YOU TRY IT

```
>>> x = ["1st", "2nd", "3rd", "4th"]

>>> x[1:-1]

>>> x[0:3]

>>> x[-2:-1]

>>> x[:3]

>>> x[-2:]
```

SOLUTION

```
>>> x = ["1st", "2nd", "3rd", "4th"]

>>> x[1:-1]
['2nd', '3rd']

>>> x[0:3]
['1st', '2nd', '3rd']

>>> x[-2:-1]
['3rd']

>>> x[:3]
['1st', '2nd', '3rd']

>>> x[-2:]
['3rd', '4th']
```

Slicing Lists and Data Type-Matching

```
>>> myList = list(range(1, 9))    # create a list
>>> print(myList)
[1, 2, 3, 4, 5, 6, 7, 8]

>>> type(myList[3])
int

>>> myList[3] = -99               # replace scalar with scalar
>>> print(myList)
[1, 2, 3, -99, 5, 6, 7, 8]

>>> type(myList[:4])
list

>>> myList[:4] = 555              # can't replace a list with a scalar...
<...error code truncated...>
TypeError: can only assign an iterable

>>> myList[:4] = [555]            # ...but can replace list with another list,
                                  # even if list is of different length
>>> print(myList)
[555, 5, 6, 7, 8]

>>> myList[0:1] = list(range(101, 103))  # scalar can also be replaced with...
>>> print(myList)                        # ... a list so long as you select the...
[101, 102, 6, 7, 8]                      # ... scalar as a list before assignment
```

Common List Operators & Operations

List operation	Explanation	Example
`[]`	Creates an empty list	`x = []`
`len`	Returns the length of a list	`len(x)`
`append`	Adds a single element to the end of a list	`x.append('y')`
`insert`	Inserts a new element at a given position in the list	`x.insert(0, 'y')`
`del`	Removes a list element or slice	`del(x[0])`
`remove`	Searches for and removes a given value from a list	`x.remove('y')`
`reverse`	Reverses a list in place	`x.reverse()`

- `.extend()`

Common List Operators & Operations [CONT'D]

List operation	Explanation	Example
`sort`	Sorts a list in place	`x.sort()`
`+`	Adds two lists together	`x1 + x2`
`*`	Replicates a list	`x = ['y'] * 3`
`min`	Returns the smallest element in a list	`min(x)`
`max`	Returns the largest element in a list	`max(x)`
`index`	Returns the position of a value in a list	`x.index['y']`
`count`	Counts the number of times a value occurs in a list	`x.count('y')`
`in`	Returns whether an item is in a list	`'y' in x`

Tuples

Very similar to lists save for one **BIG** difference

- Lists are **MUTABLE**
- Tuples are **IMMUTABLE**

```
>>> myList = [1, 2, 3, "four"]              # create a list

>>> myList[2] = "three"                     # change third entry of list
>>> print("myList = " + str(myList))

myList = [1, 2, 'three', 'four']

>>> myTuple = (1, 2, 3, "four")             # create a tuple, note round braces

myTuple = (1, 2, 3, 'four')

>>> myTuple[2] = "three"                    # try to change third entry of list

Traceback (most recent call last):
  File "/Users/paul/Desktop/myFile.py", line 13, in <module>
    myTuple[2] = "three" # try to change third entry of list
TypeError: 'tuple' object does not support item assignment
```

Dictionaries

A dictionary may feel like an associative array or hash table

- In a list
  - values are accessed by means of an integer index
  - values are implicitly ordered by their position in the list
- In a dictionary
  - values are accessed by means of a *key*, which can be integers, strings, or other Python objects
  - values are not implicitly ordered

A dictionary is a way of mapping from one set of arbitrary objects to an associated but equally arbitrary set of objects

Dictionaries [CONT'D]

Let's make an actual translational dictionary using Python's dictionary data structure

```
>>> eng_to_french = {}                          # initialize empty dictionary
                                                # note curly braces for dictionaries

>>> eng_to_french['blue'] = 'bleu'              # create a single key-value pair

>>> eng_to_french['red'] = 'rouge'              # create a single key-value pair

>>> eng_to_french['yellow'] = 'jaune'           # create a single key-value pair

>>> print("In French, the colour yellow is", eng_to_french['yellow'])

In French, the colour yellow is jaune

  # one may alternatively build a dictionary in a single line of code

>>> eng_to_french = {'blue':'bleu', 'red':'rouge', 'yellow':'jaune'}
```

Common Dictionary Operators & Operations

Dictionary operation	Explanation	Example
`{}`	Creates an empty dictionary	`x = {}`
`len`	Returns the number of entries in a dictionary	`len(x)`
`keys`	Returns a view of all keys in a dictionary	`x.keys()`

Dictionary operation	Explanation	Example
`values`	Returns a view of all values in a dictionary	`x.values()`
`items`	Returns a view of all items in a dictionary	`x.items()`
`del`	Removes an entry from a dictionary	`del(x[key])`
`in`	Tests whether a key exists in a dictionary	`'y' in x`
`get`	Returns the value of a key or a configurable default	`x.get('y',None)`
`setdefault`	Returns the value if the key is in the dictionary; otherwise, sets the value for the key to the default and returns the value	`x.setdefault('y', None)`
`copy`	Makes a copy of a dictionary	`y = x.copy()`
`update`	Combines the entries of two dictionaries	`x.update(z)`

Python Basics Cheat Sheet

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

```
>>> x+2
7
>>> x-2
3
>>> x*2
10
>>> x**2
25
>>> x%2
1
>>> x/float(2)
2.5
```

Sum of two variables
Subtraction of two variables
Multiplication of two variables
Exponentiation of a variable
Remainder of a variable
Division of a variable

Types and Type Conversion

```
str()     '5', '3.45', 'True'
int()     5, 3, 1
float()   5.0, 1.0
bool()    True, True, True
```

Variables to strings
Variables to integers
Variables to floats
Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1]
>>> my_list[-3]
```

Select item at index 1
Select 3rd last item

Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list

Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a)
>>> my_list.count(a)
>>> my_list.append('!')
>>> my_list.remove('!')
>>> del(my_list[0:1])
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> my_list.sort()
```

Get the index of an item
Count an item
Append an item at a time
Remove an item
Remove an item
Reverse the list
Append an item
Remove an item
Insert an item
Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
Strip whitespaces

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis
learn — Machine learning
NumPy — Scientific computing
matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python
spyder — Free IDE that is included with Anaconda
jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1]
2
```

Select item at index 1

Slice

```
>>> my_array[0:2]
array([1, 2])
```

Select items at index 0 and 1

Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
array([1, 4])
```

my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape
>>> np.append(other_array)
>>> np.insert(my_array, 1, 5)
>>> np.delete(my_array,[1])
>>> np.mean(my_array)
>>> np.median(my_array)
>>> my_array.corrcoef()
>>> np.std(my_array)
```

Get the dimensions of the array
Append items to an array
Insert items in an array
Delete items in an array
Mean of the array
Median of the array
Correlation coefficient
Standard deviation

DataCamp
Learn Python for Data Science Interactively

An Introduction to NumPy

- The Numerical Python (NumPy) package is designed for high-performance scientific computing and analysis of data
- NumPy is foundational, and it serves as a basis on which many other scientific and analytic tools are built
- Notably, NumPy incorporates very fast vectorized array operations

np.array

- Just as it is possible to create *n*-dimensional lists, the `ndarray` object creates an *n*-dimensional array
- Unlike lists, an `ndarray` objects contain only *homogeneous* data
- `ndarrays` have, amongst others,
  - `shapes`, dimensions of the `ndarray`
  - `dtypes`, the data type of the `ndarray`

```
>>> import numpy as np
>>> x = np.array([[1, 2, 3], [-99, -98, -97]])
>>> print(x)

[[  1   2   3]
 [-99 -98 -97]]

>>> print(x.shape)

(2, 3)

>>> print(x.dtype)

int64
```

Vectorization of `np.arrays`

- As `ndarray` objects contain homogeneous data, fast, vectorized mathematical operations are possible, and differ significantly from how lists behave

```
>>> x = np.array([[1, 2, 3], [-99, -98, -97]])   # create array
>>> y = x.tolist()                               # convert array to list

>>> print(x + x)  # array operation

[[   2    4    6]
 [-198 -196 -194]]

>>> print(y + y)  # list operation

[[1, 2, 3], [-99, -98, -97], [1, 2, 3], [-99, -98, -97]]
```

n.b. `np.zeros` creates an n-dimensional array of zeros, but `np.empty` creates an n-dimensional array filled with a random mix of zeros, very small or very large numbers

Mixing Vectors and Scalars with `np.arrays`

- NumPy will automatically vectorize operations between scalars and arrays
- The '.' after an integer implies that is it of type `float`; can also be used for array initialization

```
>>> y = np.array([1, 2, 3])       # create an array
>>> y.dtype                       # numpy creates an int64 type array
dtype('int64')

>>> z = 1/y
>>> print(z)
[ 1.          0.5         0.33333333]

>>> z.dtype                       # numpy automatically converts to float type array
dtype('float64')
```

Slicing np.arrays

- NumPy has its own range function, np.arange (note the single 'r'), which operates similarly to the range function
- NumPy likes to vectorize, so array manipulation is different from list manipulation

```
>>> y = np.arange(5)        # create an array
>>> print(y)

[0 1 2 3 4]

>>> y[1:3] = 99             # numpy automatically vectorizes the scalar to the length
>>> print(y)                # of what is being replaced

[ 0 99 99  3  4]
```

Memory Allocation with NumPy

- As NumPy is designed to deal with large amounts of data, it seeks to be as efficient with memory as possible
- One of the results of this design approach is that array slices of an ndarray object are *views* of the original array
- An implication of the behaviour: modifying data in a view changes the value(s) in the parent (original) array
- Copies of slices can be made by appending `.copy()` to a slice

```
>>> y = np.arange(5)
>>> myNumPySlice = y[1:3]
>>> myNumPySlice[:] = 789
>>> print(myNumPySlice)

array([789, 789])

>>> print(y)

array([  0, 789, 789,   3,   4])
```

Logical Operations on `np.arrays`

- Use the logical operators >, >=, ==, =<, <, | (`OR`) and & (`AND`) to test conditions in a vectorized fashion
- The result in a boolean vector of `True` and `False` values
- `True` and `False` values are treated as 0's and 1's, upon which mathematical calculations can be completed
- Vectors can also be compared against other vectors

Logical Operations on np.arrays [CONT'D]

EXAMPLES

```
>>> x = np.arange(5)
>>> x > 2
array([False, False, False,  True,  True], dtype=bool)

>>> (x > 2) | (x < 1)
array([ True, False, False,  True,  True], dtype=bool)

>>> y = (x > 2) | (x < 1)                    # n.b. fails without brackets
>>> y.sum()                                  # arithmetic operations on booleans
3

>>> y.mean()
0.59999999999999998

>>> y.astype(int)                            # convert boolean to numeric type
array([0, 0, 1, 0, 0])

>>> np.random.seed(1))
# 5 random normal vars,  mu = 1, sig^2 = 36
>>> z = np.random.normal(1, 6, 5)
array([ 10.74607218,  -2.67053848,  -2.16903051,  -5.43781173,   6.19244578])

>>> x > z                                    # comparing two arrays
array([False,  True,  True,  True, False], dtype=bool)
```

NumPy Cheat Sheet

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science Interactively at www.DataCamp.com

NumPy

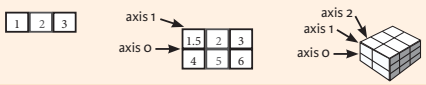
The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays

1D array | 2D array | 3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                 Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) Create an array of ones
>>> d = np.arange(10,25,5)          Create an array of evenly
                                    spaced values (step value)
>>> np.linspace(0,2,9)              Create an array of evenly
                                    spaced values (number of samples)
>>> e = np.full((2,2),7)            Create a constant array
>>> f = np.eye(2)                   Create a 2X2 identity matrix
>>> np.random.random((2,2))         Create an array with random values
>>> np.empty((3,2))                 Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64      Signed 64-bit integer types
>>> np.float32    Standard double-precision floating point
>>> np.complex    Complex numbers represented by 128 floats
>>> np.bool       Boolean type storing TRUE and FALSE values
>>> np.object     Python object type
>>> np.string_    Fixed-length string type
>>> np.unicode_   Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape       Array dimensions
>>> len(a)        Length of array
>>> b.ndim        Number of array dimensions
>>> e.size        Number of array elements
>>> b.dtype       Data type of array elements
>>> b.dtype.name  Name of data type
>>> b.astype(int) Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                       Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                Subtraction
>>> b + a                           Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                     Addition
>>> a / b                           Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                  Division
>>> a * b                           Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                Multiplication
>>> np.exp(b)                       Exponentiation
>>> np.sqrt(b)                      Square root
>>> np.sin(a)                       Print sines of an array
>>> np.cos(b)                       Element-wise cosine
>>> np.log(a)                       Element-wise natural logarithm
>>> e.dot(f)                        Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                          Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                           Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)            Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                         Array-wise sum
>>> a.min()                         Array-wise minimum value
>>> b.max(axis=0)                   Maximum value of an array row
>>> b.cumsum(axis=1)                Cumulative sum of the elements
>>> a.mean()                        Mean
>>> b.median()                      Median
>>> a.corrcoef()                    Correlation coefficient
>>> np.std(b)                       Standard deviation
```

Copying Arrays

```
>>> h = a.view()                    Create a view of the array with the same data
>>> np.copy(a)                      Create a copy of the array
>>> h = a.copy()                    Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()                        Sort an array
>>> c.sort(axis=0)                  Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                            Select the element at the 2nd index
  3
>>> b[1,2]                          Select the element at row 0 column 2
  6.0                               (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]                          Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                        Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                           Select all items at row 0
  array([[1.5, 2., 3.]])            (equivalent to b[0:1, :])
>>> c[1,...]                        Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                       Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                          Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]   Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]   Select a subset of the matrix's rows
  array([[ 4. ,5. , 6. , 4. ],      and columns
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)             Permute array dimensions
>>> i.T                             Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()                       Flatten the array
>>> g.reshape(3,-2)                 Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))                 Return a new array with shape (2,6)
>>> np.append(h,g)                  Append items to an array
>>> np.insert(a, 1, 5)              Insert items in an array
>>> np.delete(a,[1])                Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)    Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                      Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))          Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                      Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                  Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])] index
>>> np.vsplit(c,2)                  Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

DataCamp
Learn Python for Data Science Interactively