**Inspecting Python For Data Science** *Cheat Sheet*

**Your Array** NumPy Basics Learn Python for Data Science **Interactively** at www.DataCamp.com

**NumPy**

**Asking For Help** >>> np.info(np.ndarray.dtype) **Sorting Arrays** >>> a.sort() Sort an array >>> c.sort(axis=0) Sort the elements of an array's axis

**DataCamp Learn Python for Data Science Interactively**

**Also see Lists**

**Subsetting** >>> a[2] Select the element at the 2nd index 3 >>> b[1,2] Select the element at row 1 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]] array([[ [ [ [ 1.5, 4. 1.5, 4. , ,5. 2. 5. 2. , , , , 6. 3. 6. 3. , , , , 4. 1.5], 4. 1.5]])

], ], Select and columns a subset of the matrix’s rows

>>> 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**

**Aggregate Functions**

>>> np.loadtxt("myfile.txt") >>> np.genfromtxt("my\_file.csv", delimiter=',') >>> np.savetxt("myarray.txt", a, delimiter=" ")

>>> 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

1 2 3

1.5 2 3

4 5 6

**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

1 2 3

The **NumPy** library is the core library for scientific computing in

1.5 2 3 Python. It provides a high-performance multidimensional array

4 5 6

object, and tools for working with these arrays.

1.5 2 3

4 5 6

Use the following import convention:

>>> import numpy as np 1 2 3

**Creating Arrays**

**Arithmetic Operations**

**NumPy Arrays**

**1D array 2D array 3D array**

1 2 3

axis 1

axis 0

1.5 2 3 4 5 6

axis 2 axis 1axis 0

**Array Manipulation**

**Transposing Array** >>> i = np.transpose(b) Permute array dimensions >>> i.T Permute array dimensions

>>> np.zeros((3,4)) Create an array of zeros >>> np.ones((2,3,4),dtype=np.int16) Create an array of ones

**Changing Array Shape** >>> b.ravel() Flatten the array >>> d = np.arange(10,25,5) Create an array of evenly >>> g.reshape(3,-2) Reshape, but don’t change data

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

**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 >>> np.empty((3,2)) Create an empty array

**I/O**

**Combining Arrays** >>> np.concatenate((a,d),axis=0) Concatenate arrays

array([ 1, 2, 3, 10, 15, 20])

**Saving & Loading On Disk** >>> np.vstack((a,b)) Stack arrays vertically (row-wise) array([[ 1. , 2. , 3. ], [ 1.5, 2. , 3. ], >>> np.save('my\_array', a) >>> np.savez('array.npz', a, b) >>> np.load('my\_array.npy')

[ 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.], **Saving & Loading Text Files**

[ 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

**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

**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.]]])]

>>> np.string\_ Fixed-length string type >>> np.unicode\_ Fixed-length unicode type

2

**Subsetting, Slicing, Indexing**

**Array Mathematics**

>>> 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

>>> 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

F **Data Wrangling**

with pandas M A Cheat Sheet http://pandas.pydata.org

**Syntax** – Creating DataFrames

**Tidy Data** – A foundation for wrangling in pandas

In a tidy data set:

F M A

Each **variable** is saved in its own **column** &

Tidy data complements pandas’s **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

Each **observation** is saved in its own **rowReshaping Data** – Change the layout of a data set

**a b c**

**1** 4 7 10

**2** 5 8 11

**3** 6 9 12 **df = pd.DataFrame(**

**{"a" : [4 ,5, 6], "b" : [7, 8, 9], "c" : [10, 11, 12]}, index = [1, 2, 3])** Specify values for each column.

**df = pd.DataFrame(**

**[[4, 7, 10], [5, 8, 11], [6, 9, 12]], index=[1, 2, 3], columns=['a', 'b', 'c'])** Specify values for each row.

**a b c**

**n v**

**d 1** 4 7 10 **2** 5 8 11

**e 2** 6 9 12 **df = pd.DataFrame(**

**{"a" : [4 ,5, 6], "b" : [7, 8, 9], "c" : [10, 11, 12]}, index = pd.MultiIndex.from\_tuples(**

**[('d',1),('d',2),('e',2)],**

**names=['n','v'])))** Create DataFrame with a MultiIndex

**Method Chaining**

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code. **df = (pd.melt(df)**

**.rename(columns={**

**'variable' : 'var', 'value' : 'val'}) .query('val >= 200') )**

**df[['width','length','species']] df[df.Length > 7]**

Extract rows that meet logical criteria. **df.drop\_duplicates()**

Remove duplicate rows (only considers columns).

**df.sample(frac=0.5)**

Randomly select fraction of rows. **df.sample(n=10)**

Randomly select n rows. **df.iloc[10:20]**

Select rows by position.

Select multiple columns with specific names. **df['width']** *or* **df.width**

Select single column with specific name. **df.filter(regex='*regex*')**

Select columns whose name matches regular expression *regex*.

**df.head(n)**

**df.nlargest(n, 'value')** Select first n rows.

Select and order top n entries. **df.tail(n)**

**df.nsmallest(n, 'value')** Select last n rows.

Select and order bottom n entries.

**Logic in Python (and pandas)**

**<** Less than **!=** Not equal to

**df.loc[:,'x2':'x4'] >** Greater than **df.column.isin(*values*)** Group membership

Select all columns between x2 and x4 (inclusive).

**==** Equals **pd.isnull(*obj*)** Is NaN

**df.iloc[:,[1,2,5]]**

**<=** Less than or equals **pd.notnull(*obj*)** Is not NaN

**>=** Greater than or equals **&,|,~,^,df.any(),df.all()** Logical and, or, not, xor, any, all

**regex (Regular Expressions) Examples**

**'\.'** Matches strings containing a period '.'

**'Length$'** Matches strings ending with word 'Length'

**'^Sepal'** Matches strings beginning with the word 'Sepal'

**'^x[1-5]$'** Matches strings beginning with 'x' and ending with 1,2,3,4,5

**''^(?!Species$).\*'** Matches strings except the string 'Species'

Select columns in positions 1, 2 and 5 (first column is 0). **df.loc[df['a'] > 10, ['a','c']]**

Select rows meeting logical condition, and only the specific columns . http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

M \* A F M \* A **pd.melt(df)** Gather columns into rows. **df.pivot(columns='var', values='val')**

Spread rows into columns.

**pd.concat([df1,df2])**

Append rows of DataFrames

**df.sort\_values('mpg')**

Order rows by values of a column (low to high).

**df.sort\_values('mpg',ascending=False)** Order rows by values of a column (high to low).

**df.rename(columns = {'y':'year'})**

Rename the columns of a DataFrame

**df.sort\_index()**

Sort the index of a DataFrame

**df.reset\_index()**

Reset index of DataFrame to row numbers, moving index to columns.

**pd.concat([df1,df2], axis=1)**

**df.drop(columns=['Length','Height'])** Append columns of DataFrames

Drop columns from DataFrame

**Subset Observations** (Rows) **Subset Variables** (Columns)

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

**Summarize Data**

**Make New Columns**

**Combine Data Sets**

**df['w'].value\_counts()**

Count number of rows with each unique value of variable **len(df)**

# of rows in DataFrame. **df['w'].nunique()**

# of distinct values in a column. **df.describe()**

Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

**sum()**

Sum values of each object. **count()**

Count non-NA/null values of each object. **median()**

Median value of each object. **quantile([0.25,0.75])** Quantiles of each object. **apply(*function*)**

Apply function to each object.

**Handling Missing Data**

**df.dropna()**

Drop rows with any column having NA/null data. **df.fillna(value)**

Replace all NA/null data with value.

**Plotting**

**df.plot.hist()**

Histogram for each column

**adf bdf x1 x2 A 1 B 2 C 3** Standard Joins

**x1 x2 x3 A 1 T B 2 F C 3 NaN**

**x1 x2 x3 A 1.0 T B 2.0 F D NaN T**

**x1 x2 x3 A 1 T B 2 F**

**x1 x2 x3 A 1 T B 2 F C 3 NaN D NaN T**

**x1 x3 A T B F D T**

**pd.merge(adf, bdf,**

**how='left', on='x1')** Join matching rows from bdf to adf.

**df.assign(Area=lambda df: df.Length\*df.Height)**

Compute and append one or more new columns.

**pd.merge(adf, bdf, df['Volume'] = df.Length\*df.Height\*df.Depth**

**how='right', on='x1')** Add single column.

Join matching rows from adf to bdf. **pd.qcut(df.col, n, labels=False)**

Bin column into n buckets. **min()**

Minimum value in each object. **max()**

**pd.merge(adf, bdf,**

**how='inner', on='x1') Vector function**

Join data. Retain only rows in both sets. Maximum value in each object. **mean()**

Mean value of each object. **var()**

**Vector function**

**pd.merge(adf, bdf,** pandas provides a large set of **vector functions** that operate on all

**how='outer', on='x1')** columns of a DataFrame or a single selected column (a pandas

Join data. Retain all values, all rows. Variance of each object. **std()**

Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples: Standard deviation of each

Filtering Joins object.

**x1 x2 A 1 B 2**

**x1 x2 C 3**

**shift(1)**

Copy with values shifted by 1. **rank(method='dense')**

Ranks with no gaps. **rank(method='min')**

Ranks. Ties get min rank. **rank(pct=True)**

Ranks rescaled to interval [0, 1]. **rank(method='first')**

Ranks. Ties go to first value.

**min(axis=1)**

Element-wise min. **abs()**

Absolute value.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

**Windows**

**df.expanding()**

Return an Expanding object allowing summary functions to be applied cumulatively. **df.rolling(n)**

Return a Rolling object allowing summary functions to be applied to windows of length n.

**max(axis=1)**

Element-wise max. **clip(lower=-10,upper=10)** Trim values at input thresholds

**adf[adf.x1.isin(bdf.x1)] Group Data**

All rows in adf that have a match in bdf.

**df.groupby(by="col")**

**adf[~adf.x1.isin(bdf.x1)]** Return a GroupBy object,

All rows in adf that do not have a match in bdf. grouped by values in column named "col".

**df.groupby(level="ind")**

Return a GroupBy object, grouped by values in index level named "ind".

**x1 x2 A 1 B 2 C 3**

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

**shift(-1)**

**ydf zdf** Copy with values lagged by 1. **cumsum()**

Cumulative sum. **cummax()**

Cumulative max. **cummin()**

Cumulative min. **cumprod()**

Cumulative product.

Set-like Operations **x1 x2 B 2 C 3**

**x1 x2 A 1 B 2 C 3 D 4**

**x1 x2 A 1**

**x1 x2 B 2 C 3 D 4**

**pd.merge(ydf, zdf) size()**

**agg(*function*)**

Rows that appear in both ydf and zdf Size of each group.

Aggregate group using function.

(Intersection).

**pd.merge(ydf, zdf, how='outer')**

Rows that appear in either or both ydf and zdf

**df.plot.scatter(x='w',y='h')**

(Union).

Scatter chart using pairs of points

**pd.merge(ydf, zdf, how='outer',**

**indicator=True) .query('\_merge == "left\_only"') .drop(columns=['\_merge'])**

Rows that appear in ydf but not zdf (Setdiff).

**Dropping Python For Data Science** *Cheat Sheet* Pandas Basics

**Asking For Help** >>> help(pd.Series.loc) Learn Python for Data Science **Interactively** at www.DataCamp.com

>>> s.drop(['a', 'c']) Drop values from rows (axis=0) **Also see NumPy Arrays**

>>> df.drop('Country', axis=1) Drop values from columns(axis=1)

**Sort & Rank Pandas** The **Pandas** library is built on NumPy and provides easy-to-use

>>> df.sort\_index() Sort by labels along an axis >>> df.sort\_values(by='Country') Sort by the values along an axis >>> df.rank() Assign ranks to entries **data structures** and **data analysis** tools for the Python programming language.

Use the following import convention:

**Data Alignment**

>>> s.add(s3, fill\_value=0)

a 10.0 b **-5.0** c 5.0 d 7.0 >>> s.sub(s3, fill\_value=2) >>> s.div(s3, fill\_value=4) >>> s.mul(s3, fill\_value=3) **Selecting, Boolean Indexing & Setting Basic Information**

>>> import pandas as pd **Pandas Data Structures**

**Series**

**Summary** A **one-dimensional** labeled array

a 3 capable of holding any data type

b-5Index c7d4 >>> df.sum() Sum of values >>> df.cumsum() Cummulative sum of values >>> df.min()/df.max() Minimum/maximum values >>> df.idxmin()/df.idxmax() Minimum/Maximum index value >>> df.describe() Summary statistics >>> df.mean() Mean of values >>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])

>>> df.median() Median of values

**DataFrame**

Columns Country Capital

Population

A **two-dimensional** labeled

0Belgium Brussels

11190846

data structure with columns of potentially different types

Index

1India New Delhi

1303171035

2

Brazil Brasília

207847528

**Internal Data Alignment** NA values are introduced in the indices that don’t overlap:

>>> data = {'Country': ['Belgium', 'India', 'Brazil'],

>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd']) 'Capital': ['Brussels', 'New Delhi', 'Brasília'],

>>> s + s3 'Population': [11190846, 1303171035, 207847528]}

a 10.0

>>> df = pd.DataFrame(data,

columns=['Country', 'Capital', 'Population'])

b **NaN** c 5.0 d 7.0 **I/OArithmetic Operations with Fill Methods**

**Read and Write to CSV**

You can also do the internal data alignment yourself with the help of the fill methods: >>> pd.read\_csv('file.csv', header=None, nrows=5) >>> df.to\_csv('myDataFrame.csv')

**Read and Write to Excel**

>>> pd.read\_excel('file.xlsx') >>> pd.to\_excel('dir/myDataFrame.xlsx', sheet\_name='Sheet1')

**Read multiple sheets from the same file** >>> xlsx = pd.ExcelFile('file.xls') >>> df = pd.read\_excel(xlsx, 'Sheet1')

**DataCamp Learn Python for Data Science Interactively**

**Selection**

**Getting** >>> s['b'] Get one element

-5 >>> df[1:] Get subset of a DataFrame

Country Capital Population 1 India New Delhi 1303171035 2 Brazil Brasília 207847528

**By Position** >>> df.iloc[[0],[0]] Select single value by row &

'Belgium' column >>> df.iat([0],[0])

'Belgium' **By Label** >>> df.loc[[0], ['Country']] Select single value by row &

'Belgium' column labels >>> df.at([0], ['Country'])

'Belgium' **By Label/Position** >>> df.ix[2] Select single row of

Country Brazil subset of rows Capital Brasília Population 207847528 >>> df.ix[:,'Capital'] 0 Brussels subset Select a of single columns

column of

1 New Delhi 2 Brasília

>>> df.ix[1,'Capital'] Select rows and columns

'New Delhi' **Boolean Indexing** >>> s[~(s > 1)] Series s where value is not >1 >>> s[(s < -1) | (s > 2)] s where value is <-1 or >2 >>> df[df['Population']>1200000000] Use filter to adjust DataFrame

**Setting** >>> s['a'] = 6 Set index a of Series s to 6

**Read and Write to SQL Query or Database Table**

>>> from sqlalchemy import create\_engine >>> engine = create\_engine('sqlite:///:memory:') >>> pd.read\_sql("SELECT \* FROM my\_table;", engine) >>> pd.read\_sql\_table('my\_table', engine) >>> pd.read\_sql\_query("SELECT \* FROM my\_table;", engine)

read\_sql()is a convenience wrapper around read\_sql\_table() and read\_sql\_query()

>>> pd.to\_sql('myDf', engine)

**Retrieving Series/DataFrame Information**

>>> df.shape (rows,columns) >>> df.index Describe index >>> df.columns Describe DataFrame columns >>> df.info() Info on DataFrame >>> df.count() Number of non-NA values

**Applying Functions**

>>> f = lambda x: x\*2 >>> df.apply(f) Apply function >>> df.applymap(f) Apply function element-wise

**Python For Data Science** *Cheat Sheet*

**Also see NumPy Arrays**

**Combining Data** Pandas

Selecting >>> df3.loc[:,(df3>1).any()] Select cols with any vals >1 Learn Python for Data Science **Interactively** at www.DataCamp.com

>>> df3.loc[:,(df3>1).all()] >>> df3.loc[:,df3.isnull().any()] *data1*

Select cols with vals > 1 Select cols with NaN a**Reshaping Data**

>>> df3.loc[:,df3.notnull().all()] Indexing With isin >>> df[(df.Country.isin(df2.Type))] *data2*

**X1 X2**

**X1 X3**

11.432

a20.784 Select cols without NaN b1.303

bNaN

Find same elements

c

99.906

d

20.784

>>> df3.filter(items=”a”,”b”]) **Pivot**

>>> df.select(lambda x: not x%5) Filter on values

Select specific elements

**Merge** Where >>> df3= df2.pivot(index='Date', Spread rows into columns >>> s.where(s > 0) Subset the data

columns='Type', values='Value')

Query >>> df6.query('second > first') Query DataFrame

>>> pd.merge(data1,

**X1 X2 X3** data2, how='left', on='X1')

abc

**Pivot Table**

>>> df4 = pd.pivot\_table(df2, values='Value', Spread rows into columns >>> pd.merge(data1,

index='Date', columns='Type'])

data2, how='outer', **Stack / Unstack**

on='X1')

**Join**

>>> data1.join(data2, how='right')

**Concatenate**

**Melt**

>>> arrays = [np.array([1,2,3]), np.array([5,4,3])] >>> df5 = pd.DataFrame(np.random.rand(3, >>> tuples = list(zip(\*arrays)) >>> index = pd.MultiIndex.from\_tuples(tuples, >>> df6 = pd.DataFrame(np.random.rand(3, Vertical

2), index=arrays) >>> s.append(s2) Horizontal/Vertical

names=['first', 'second']) 2), index=index)

>>> pd.concat([s,s2],axis=1, keys=['One','Two']) >>> pd.concat([data1, data2], axis=1, join='inner')

>>> df2.set\_index(["Date", "Type"])

>>> pd.melt(df2, Gather columns into rows id\_vars=["Date"],

value\_vars=["Type", "Value"], value\_name="Observations")

>>> s.plot() >>> plt.show()

**Missing Data**

>>> df.dropna() Drop NaN values >>> df3.fillna(df3.mean()) Fill NaN values with a predetermined value >>> df2.replace("a", "f") Replace values with others

**DataCamp Learn Python for Data Science Interactively**

11.432

1.303 99.906

20.784

NaN

Date Type Value

NaN

012345

2016-03-01 a

11.432

Type

**Setting/Resetting Index**

**X1 X2 X3**

2016-03-02 b

13.031

Date

>>> df.set\_index('Country') Set the index 2016-03-01 c

20.784

2016-03-01 2016-03-03 a

2016-03-02 2016-03-02 a

2016-03-03 2016-03-03 c

a b11.432

NaN

1.303

13.031

99.906

NaN

c

20.784

NaN

20.784

>>> df4 = df.reset\_index() Reset the index >>> df = df.rename(index=str, columns={"Country":"cntry", "Capital":"cptl", "Population":"ppltn"})

Rename DataFrame >>> pd.merge(data1, data2, how='inner', on='X1')

**Duplicate Data**

>>> s3.unique() Return unique values >>> df2.duplicated('Type') Check duplicates >>> df2.drop\_duplicates('Type', keep='last') Drop duplicates >>> df.index.duplicated() Check index duplicates **Grouping Data**

**Aggregation** >>> df2.groupby(by=['Date','Type']).mean() >>> df4.groupby(level=0).sum() >>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x), **Transformation** 'b': np.sum}) >>> customSum = lambda x: (x+x%2) >>> df4.groupby(level=0).transform(customSum)

>>> pd.merge(data1, data2, how='right', on='X1')

abd

11.432 1.303

NaN

20.784

NaN

99.906

20.784

1.303

20.784

**X1 X2 X3**

ab

abc

11.432 1.303

20.784

NaN

**X1 X2 X3**

11.432

20.784 1.303

NaN 99.906

NaN d NaN 20.784 >>> stacked = df5.stack() Pivot a level of column labels >>> stacked.unstack() Pivot a level of index labels

0

1

1500.233482

150.233482

0.390959

10.390959

**MultiIndexing**

240.184713

0.237102

2400.184713

3

3

0.433522

0.429401

10.237102 *Unstacked*

3

3

00.433522

1

0.429401 *Stacked*

**Dates**

Date Type Value

0Date Variable Observations 02016-03-01 a 112016-03-02 b 222016-03-01 c 3342016-03-01 Type 2016-03-02 Type 42016-03-01 Type

**Visualization**

5

2016-03-01 Value 5

2016-03-02 Value 2016-03-01 Value a11.432 b13.031 c

20.784

2016-03-03 Type

2016-03-03 a

2016-03-02 a

2016-03-02 Type 2016-03-03 Type

2016-03-03 c

a**Also see Matplotlib** 99.906

>>> import matplotlib.pyplot as plt 1.303

>>> df2.plot() >>> plt.show() ac

20.784

611.432 713.031 820.784 92016-03-03 Value

99.906 102016-03-02 Value

1.303 11

2016-03-03 Value

20.784 **Iteration**

>>> df.iteritems() >>> df.iterrows() (Column-index, Series) pairs

(Row-index, Series) pairs

**Advanced Indexing**

**Reindexing**

>>> s2 = s.reindex(['a','c','d','e','b'])

Forward Filling Backward Filling >>> df.reindex(range(4),

>>> s3 = s.reindex(range(5), method='ffill')

method='bfill') Country Capital Population

0 3 0 Belgium Brussels 11190846

1 3 1 India New Delhi 1303171035

2 3 2 Brazil Brasília 207847528

3 3 3 Brazil Brasília 207847528

4 3

>>> df2['Date']= pd.to\_datetime(df2['Date']) >>> df2['Date']= pd.date\_range('2000-1-1', periods=6, freq='M') >>> dates = [datetime(2012,5,1), datetime(2012,5,2)] >>> index = pd.DatetimeIndex(dates) >>> index = pd.date\_range(datetime(2012,2,1), end, freq='BM')

**Python For Data Science** *Cheat Sheet*

**Plot Anatomy & Workflow** Matplotlib Learn Python **Interactively** at www.DataCamp.com

Axes/Subplot

**Matplotlib**

Y-axis **Matplotlib** is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. 1**Prepare The Data Also see Lists & NumPy**

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system. >>> fig.add\_axes() >>> ax1 = fig.add\_subplot(221) # row-col-num >>> ax3 = fig.add\_subplot(212) >>> fig3, axes = plt.subplots(nrows=2,ncols=2) >>> fig4, axes2 = plt.subplots(ncols=3)

**DataCamp Learn Python for Data Science Interactively** X-axis

**1D Data** >>> import numpy as np >>> x = np.linspace(0, 10, 100) >>> y = np.cos(x) >>> z = np.sin(x)

**2D Data or Images**

**Customize Plot**

**Colors, Color Bars & Color Maps**

>>> plt.plot(x, x, x, x\*\*2, x, x\*\*3) >>> ax.plot(x, y, alpha = 0.4) >>> ax.plot(x, y, c='k') >>> data = 2 \* np.random.random((10, 10)) >>> data2 = 3 \* np.random.random((10, 10))

>>> >>> fig.colorbar(im, im = ax.imshow(img, orientation='horizontal') cmap='seismic')

>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j] >>> U = -1 - X\*\*2 + Y

**Markers**

>>> V = 1 + X - Y\*\*2 >>> from matplotlib.cbook import get\_sample\_data >>> img = np.load(get\_sample\_data('axes\_grid/bivariate\_normal.npy'))

>>> fig, ax = plt.subplots() >>> ax.scatter(x,y,marker=".") >>> ax.plot(x,y,marker="o") 2

**Create Plot Linestyles**

>>> import matplotlib.pyplot as plt

**Figure** >>> fig = plt.figure() >>> fig2 = plt.figure(figsize=plt.figaspect(2.0)) **Axes**

>>> fig, ax = plt.subplots() >>> lines = ax.plot(x,y) Draw points with lines or markers connecting them >>> ax.scatter(x,y) Draw unconnected points, scaled or colored >>> axes[0,0].bar([1,2,3],[3,4,5]) Plot vertical rectangles (constant width) >>> axes[1,0].barh([0.5,1,2.5],[0,1,2]) Plot horiontal rectangles (constant height) >>> axes[1,1].axhline(0.45) Draw a horizontal line across axes >>> axes[0,1].axvline(0.65) Draw a vertical line across axes >>> ax.fill(x,y,color='blue') Draw filled polygons >>> ax.fill\_between(x,y,color='yellow') Fill between y-values and 0

**Close & Clear** >>> >>> >>> 3**Plotting Routines 1D Data 2D Data or Images** >>> >>> fig, im = ax ax.imshow(img, = plt.subplots() cmap='gist\_earth', Colormapped or RGB arrays plt.cla() plt.clf() plt.close() Clear Clear Close an the a window

axis entire figure interpolation='nearest', vmin=-2, vmax=2)

**Plot Anatomy Workflow**

The basic **1** Prepare steps to data creating **2** Create plots plot with matplotlib **3** Plot **4** are: Customize plot **5** Save plot **6** Show plot

>>> import matplotlib.pyplot as plt >>> x = [1,2,3,4] >>> y = [10,20,25,30]

Step 1

>>> fig = plt.figure() Step 2>>> ax = fig.add\_subplot(111)

Step 3 >>> ax.plot(x, y, color='lightblue', linewidth=3)

Step 3, 4 >>> ax.scatter([2,4,6],

[5,15,25], color='darkgreen', marker='^') >>> ax.set\_xlim(1, 6.5) >>> plt.savefig('foo.png') >>> plt.show()

Step 6 4

**Mathtext**

**Limits & Autoscaling** >>> ax.margins(x=0.0,y=0.1) Add padding to a plot >>> ax.axis('equal') Set the aspect ratio of the plot to 1 >>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5]) Set limits for x-and y-axis >>> ax.set\_xlim(0,10.5) Set limits for x-axis **Legends** >>> ax.set(title='An Example Axes', Set a title and x-and y-axis labels ylabel='Y-Axis', xlabel='X-Axis') >>> ax.legend(loc='best') No overlapping plot elements **Ticks** >>> ax.xaxis.set(ticks=range(1,5), Manually set x-ticks ticklabels=[3,100,-12,"foo"]) >>> ax.tick\_params(axis='y', Make y-ticks longer and go in and out direction='inout', **Text & Annotations**

**Subplot Spacing** length=10) >>> fig3.subplots\_adjust(wspace=0.5, Adjust the spacing between subplots hspace=0.3, left=0.125, right=0.9, top=0.9, >>> fig.tight\_layout() bottom=0.1) Fit subplot(s) in to the figure area **Axis Spines** >>> ax1.spines['top'].set\_visible(False) Make the top axis line for a plot invisible >>> ax1.spines['bottom'].set\_position(('outward',10)) Move the bottom axis line outward >>> plt.title(r'$sigma\_i=15$', fontsize=20)

>>> >>> ax.text(1, ax.annotate("Sine", 'Example style='italic') -2.1, xy=(8, xycoords='data', xytext=(10.5, textcoords='data', arrowprops=dict(arrowstyle="->", Graph', 0), connectionstyle="arc3"),)

0), **Limits, Legends & Layouts** >>> plt.plot(x,y,linewidth=4.0) >>> plt.plot(x,y,ls='solid') >>> plt.plot(x,y,ls='--') >>> plt.plot(x,y,'--',x\*\*2,y\*\*2,'-.') >>> plt.setp(lines,color='r',linewidth=4.0)

**Vector Fields**

>>> axes[0,1].arrow(0,0,0.5,0.5) Add an arrow to the axes >>> axes[1,1].quiver(y,z) Plot a 2D field of arrows >>> axes[0,1].streamplot(X,Y,U,V) Plot a 2D field of arrows

**Data Distributions** >>> ax1.hist(y) Plot a histogram >>> ax3.boxplot(y) Make a box and whisker plot >>> ax3.violinplot(z) Make a violin plot

>>> axes2[0].pcolor(data2) Pseudocolor plot of 2D array >>> axes2[0].pcolormesh(data) Pseudocolor plot of 2D array >>> CS = plt.contour(Y,X,U) Plot contours >>> axes2[2].contourf(data1) Plot filled contours >>> axes2[2]= ax.clabel(CS) Label a contour plot

Figure

**Save** 5

**Plot Save figures** >>> plt.savefig('foo.png')

**Save transparent figures** >>> plt.savefig('foo.png', transparent=True) 6**Show Plot** >>> plt.show()

Matplotlib 2.0.0 - Updated on: 02/2017

**Lists Python For Data Science** *Cheat Sheet* Python Basics

>>> a = 'is' >>> b = 'nice'

**Libraries**

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

>>> my\_list = ['my', 'list', a, b] >>> my\_list2 = [[4,5,6,7], [3,4,5,6]]

**Variable Assignment**

**Subset** >>> my\_list[1] >>> my\_list[-3]

**Slice** >>> my\_list[1:3] >>> my\_list[1:] >>> my\_list[:3] >>> my\_list[:]

**Subset Lists of Lists** >>> my\_list2[1][0] >>> my\_list2[1][:2]

**Strings**

>>> my\_string.upper() >>> my\_string.lower() >>> my\_string.count('w') >>> my\_string.replace('e', 'i') >>> my\_string.strip()

**Also see NumPy Arrays**

>>> 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()

**Import libraries** >>> import numpy >>> import numpy as np

**Selective import**

**Variables and Data Types**

**Index starts at 0**

>>> from math import pi

Select item at index 1 Select 3rd last item

Select items at index 1 and 2 Select items after index 0 Select items before index 3 Copy my\_list

my\_list[list][itemOfList]

str() '5', '3.45', 'True'

int() 5, 3, 1

float() 5.0, 1.0

bool() True, True, True

>>> help(str)

**Install Python**

>>> x=5 >>> x 5**Calculations With Variables**

>>> x+2 Sum of two variables 7 >>> x-2 Subtraction of two variables 3>>> x\*2 Multiplication of two variables >>> 10 x\*\*2 Exponentiation of a variable

25 >>> x%2 Remainder of a variable 1>>> x/float(2) Division of a variable

2.5

Create and share documents with live code, visualizations, text, ...

**Types and Type Conversion**

**String Operations**

**Numpy Arrays**

**List Operations**

>>> my\_list = [1, 2, 3, 4] >>> my\_array = np.array(my\_list) >>> my\_list + my\_list

>>> my\_2darray = np.array([[1,2,3],[4,5,6]])

['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\_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])

**Index starts at 0**

>>> my\_array.shape >>> np.append(other\_array)

**String Operations**

>>> my\_string \* 2

**String Methods** 'thisStringIsAwesomethisStringIsAwesome' >>> my\_string + 'Innit' 'thisStringIsAwesomeInnit'

>>> 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) >>> 'm' in my\_string

True

**Selecting Numpy Array Elements Index starts at 0**

Variables to strings

Get the index of an item Variables to integers

Count an item Append an item at a time

Variables to floats

Remove an item Remove an item

**Numpy Array Operations** Variables to booleans

Reverse the list Append an item **Asking For Help**

Remove an item Insert an item Sort the list

>>> my\_string = 'thisStringIsAwesome'

**Numpy Array Functions** >>> my\_string 'thisStringIsAwesome'

>>> my\_string[3] >>> my\_string[4:9]

**Selecting List Elements**

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

**Subset** >>> my\_array[1]

2 **Slice** >>> my\_array[0:2]

array([1, 2]) **Subset 2D Numpy arrays** >>> my\_2darray[:,0]

array([1, 4])

Leading open data science platform powered by Python

Data analysis

Machine learning

Scientific computing

2D plotting

Free IDE that is included with Anaconda

**Also see Lists**

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**

Select item at index 1

Select items at index 0 and 1

my\_2darray[rows, columns]

**Python For Data Science** *Cheat Sheet* Importing Data Learn Python for data science **Interactively** at www.DataCamp.com

**Importing Data in Python**

Most of the time, you’ll use either **NumPy** or **pandas** to import your data:

To access the sheet names, use the sheet\_names attribute:

>>> import numpy as np >>> import pandas as pd

**Help**

>>> np.info(np.ndarray.dtype) >>> help(pd.read\_csv)

**Text Files**

**Exploring Dictionaries**

**Accessing Elements with Functions Plain Text Files**

>>> print(mat.keys()) Print dictionary keys >>> filename = 'huck\_finn.txt' >>> file = open(filename, mode='r') Open the file for reading >>> text = file.read() Read a file’s contents >>> print(file.closed) Check whether file is closed >>> file.close() Close file >>> print(text)

>>> for key in data.keys(): Print dictionary keys

print(key) meta quality strain >>> pickled\_data.values() Return dictionary values >>> print(mat.items()) Returns items in list format of (key, value)

tuple pairs **Using the context manager** with >>> with open('huck\_finn.txt', 'r') as file:

**Accessing Data Items with Keys** print(file.readline()) Read a single line print(file.readline()) print(file.readline())

Use the table\_names() method to fetch a list of table names:

>>> for key in data ['meta'].keys() Explore the HDF5 structure

print(key) Description **Table Data: Flat Files**

DescriptionURL Detector Duration **Importing Flat Files with numpy**

**Using the context manager** with

GPSstart Observatory **Files with one data type**

Type UTCstart >>> filename = ‘mnist.txt’

>>> print(data['meta']['Description'].value) Retrieve the value for a key >>> data = np.loadtxt(filename,

delimiter=',', String used to separate values skiprows=2, Skip the first 2 lines usecols=[0,2], Read the 1st and 3rd column dtype=str) The type of the resulting array **Files with mixed data types**

**Magic Commands**

>>> filename = 'titanic.csv' >>> data = np.genfromtxt(filename,

delimiter=',', names=True, dtype=None)

Look for column header **Exploring Your Data**

os **Library**

>>> data\_array = np.recfromcsv(filename)

>>> data\_array.dtype Data type of array elements The default dtype of the np.recfromcsv() function is None.

**Importing Flat Files with pandas** >>> filename = 'winequality-red.csv' !ls List directory contents of files and directories

%cd .. Change current working directory %pwd Return the current working directory path

**NumPy Arrays**

>>> data\_array.shape Array dimensions >>> len(data\_array) Length of array

**pandas DataFrames**

>>> data = pd.read\_csv(filename,

nrows=5, Number of rows of file to read header=None, Row number to use as col names sep='\t', Delimiter to use comment='#', Character to split comments na\_values=[""]) String to recognize as NA/NaN

**DataCamp Learn R for Data Science Interactively**

**Pickled Excel Spreadsheets**

**Files**

>>> file = 'urbanpop.xlsx' >>> data = pd.ExcelFile(file)

>>> import pickle >>> with open('pickled\_fruit.pkl', 'rb') as file: >>> df\_sheet2 = data.parse('1960-1966',

pickled\_data = pickle.load(file) skiprows=[0], names=['Country', >>> df\_sheet1 = data.parse(0, 'AAM: War(2002)'])

**HDF5 Files** parse\_cols=[0], skiprows=[0],

>>> import h5py names=['Country'])

>>> filename = 'H-H1\_LOSC\_4\_v1-815411200-4096.hdf5' >>> data = h5py.File(filename, 'r')

>>> data.sheet\_names

**Matlab Files SAS Files**

>>> import scipy.io >>> filename = 'workspace.mat' >>> from sas7bdat import SAS7BDAT

>>> mat = scipy.io.loadmat(filename) >>> with SAS7BDAT('urbanpop.sas7bdat') as file:

df\_sas = file.to\_data\_frame()

**Stata Files**

>>> data = pd.read\_stata('urbanpop.dta')

**Relational Databases**

>>> from sqlalchemy import create\_engine >>> engine = create\_engine('sqlite://Northwind.sqlite')

>>> table\_names = engine.table\_names()

**Querying Relational Databases**

>>> con = engine.connect() >>> rs = con.execute("SELECT \* FROM Orders") >>> df = pd.DataFrame(rs.fetchall()) >>> df.columns = rs.keys() >>> con.close()

>>> with engine.connect() as con:

rs = con.execute("SELECT OrderID FROM Orders") df = pd.DataFrame(rs.fetchmany(size=5)) df.columns = rs.keys()

**Navigating Your FileSystem Querying relational databases with pandas**

>>> df = pd.read\_sql\_query("SELECT \* FROM Orders", engine)

>>> import os >>> path = "/usr/tmp" >>> wd = os.getcwd() Store the name of current directory in a string >>> os.listdir(wd) Output contents of the directory in a list >>> os.chdir(path) Change current working directory >>> os.rename("test1.txt", Rename a file

>>> df.head() >>> df.tail() >>> df.index Return first DataFrame rows Return last DataFrame rows Describe index

"test2.txt") >>> os.remove("test1.txt") Delete an existing file >>> os.mkdir("newdir") Create a new directory

>>> df.columns Describe DataFrame columns >>> df.info() Info on DataFrame >>> data\_array = data.values Convert a DataFrame to an a NumPy array

IRkernel IJulia

**Saving/Loading Notebooks**

Interrupt kernel

Create new notebook

Interrupt kernel & Open an existing

clear all output

Make a copy of the notebook current notebook Rename notebook

Connect back to a remote notebook

Run other installed kernels Save current notebook and record checkpoint

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 Copy cells from

clipboard to current

Paste cells from cursor position clipboard above current cell Paste cells from clipboard below Paste cells from current cell clipboard on top of current cel Delete current cells

Revert *“Delete Cells”* invocation

Insert Cells

Revert notebook to a previous checkpoint

Preview of the printed Download notebook as - IPython notebook notebook - Python - HTML Close notebook & stop running any scripts

**Writing Code And Text**

Split up a cell from current cursor position Merge current cell with the one above Merge current cell

with the one below Move current cell up Move current cell Adjust metadata down underlying the

Find and replace current notebook

in selected cells

Insert image in selected cells - Markdown - reST - LaTeX - PDF

View Cells

Add new cell above the current one

Remove cell attachments Paste attachments of current cell

**Working Python For Data Science** *Cheat Sheet*

**with Different Programming Languages Widgets** Jupyter Notebook Learn More Python for Data Science Interactively at www.DataCamp.com

**Asking For Help**

Add new cell below the current one

**Command Mode:**

**Edit Mode:**

**15**

Copy attachments of current cell

**1 2 3 4 5 6 7 8 9 10 11 12**

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

Installing Jupyter Notebook will automatically install the IPython kernel.

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.

Restart kernel

Restart kernel & run all cells

Restart kernel & run all cells

Executing Cells

Run all cells above the

Run all cells below current cell

the current cell Change the cell type of current cell toggle, toggle

scrolling and clear toggle, toggle current outputs scrolling and clear all output

Toggle display of Jupyter logo and filename Toggle display of toolbar Toggle display of cell action icons: - None - Edit metadata Toggle line numbers in cells Run selected cell(s) Run current cells down

and create a new one below Run current cells down and create a new one above Run all cells

- Raw cell format - Slideshow - Attachments - Tags

Download serialized state of all widget models in use

Embed current widgets

**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

Walk through a UI tour

List of built-in keyboard Edit the built-in shortcuts keyboard shortcuts Notebook help topics Description of markdown available

Information on in notebook

unofficial Jupyter

Python help topics Notebook extensions IPython help topics NumPy help topics

Matplotlib help topics

SciPy help topics

SymPy help topics Pandas help topics About Jupyter Notebook

**DataCamp Learn Python for Data Science Interactively**

**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

Save notebook with interactive widgets

**13 14**

**Python For Data Science** *Cheat Sheet* SciPy - Linear Algebra Learn More Python for Data Science Interactively at www.datacamp.com

**Linear Algebra**

You’ll use the linalg and sparse modules. Note that scipy.linalg contains and expands on numpy.linalg.

>>> from scipy import linalg, sparse

**Creating Matrices**

>>> A = np.matrix(np.random.random((2,2))) **SciPy**

>>> B = np.asmatrix(b) >>> C = np.mat(np.random.random((10,5))) The **SciPy** library is one of the core packages for

>>> D = np.mat([[3,4], [5,6]]) scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python. **Interacting With NumPy Also see NumPy**

>>> import numpy as np >>> a = np.array([1,2,3]) >>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)]) >>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])

**Index Tricks**

>>> np.mgrid[0:5,0:5] Create a dense meshgrid >>> np.ogrid[0:2,0:2] Create an open meshgrid >>> np.r\_[3,[0]\*5,-1:1:10j] Stack arrays vertically (row-wise) >>> np.c\_[b,c] Create stacked column-wise arrays

**Shape Manipulation**

>>> np.transpose(b) >>> b.flatten() Permute Flatten the array array dimensions >>> np.hstack((b,c)) Stack arrays horizontally (column-wise) >>> np.vstack((a,b)) Stack arrays vertically (row-wise) >>> np.hsplit(c,2) Split the array horizontally at the 2nd index >>> np.vpslit(d,2) Split the array vertically at the 2nd index

**Polynomials**

>>> from numpy import poly1d >>> p = poly1d([3,4,5]) Create a polynomial object

**Vectorizing Functions**

>>> def myfunc(a):

if a < 0: return a\*2 else: return a/2 >>> np.vectorize(myfunc) Vectorize functions

**Type Handling**

>>> np.real(b) Return the real part of the array elements >>> np.imag(b) Return the imaginary part of the array elements >>> np.real\_if\_close(c,tol=1000) Return a real array if complex parts close to 0 >>> np.cast['f'](np.pi) Cast object to a data type

>>> np.angle(b,deg=True) Return the angle of the complex argument >>> g = np.linspace(0,np.pi,num=5) Create an array of evenly spaced values >>> g [3:] += np.pi (number of samples) >>> np.unwrap(g) Unwrap >>> np.logspace(0,10,3) >>> np.select([c<4],[c\*2]) Create Return an values array from of evenly a list of spaced arrays values depending (log scale)

on conditions >>> misc.factorial(a) Factorial >>> misc.comb(10,3,exact=True) Combine N things taken at k time >>> misc.central\_diff\_weights(3) Weights for Np-point central derivative >>> misc.derivative(myfunc,1.0) Find the n-th derivative of a function at a point

>>> help(scipy.linalg.diagsvd) >>> np.info(np.matrix) **Other Useful Functions**

**DataCamp Learn Python for Data Science Interactively**

**Also see NumPy**

**Basic Matrix Routines Inverse** >>> A.I Inverse >>> linalg.inv(A) Inverse

**Transposition** >>> A.T >>> A.H Tranpose Conjugate matrix transposition

**Trace** >>> np.trace(A) Trace

**Norm** >>> linalg.norm(A) Frobenius norm >>> linalg.norm(A,1) L1 norm (max column sum) >>> linalg.norm(A,np.inf) L inf norm (max row sum)

**Rank** >>> np.linalg.matrix\_rank(C) Matrix rank

**Determinant** >>> linalg.det(A) Determinant

**Solving linear problems** >>> linalg.solve(A,b) Solver for dense matrices >>> E = np.mat(a).T Solver for dense matrices >>> linalg.lstsq(F,E) Least-squares solution to linear matrix equation **Generalized inverse** >>> linalg.pinv(C) Compute the pseudo-inverse of a matrix (least-squares solver) >>> linalg.pinv2(C) Compute the pseudo-inverse of a matrix (SVD)

**Matrix Functions**

**Asking For Help**

**Addition** >>> np.add(A,D) Addition **Subtraction** >>> np.subtract(A,D) Subtraction **Division** >>> np.divide(A,D) Division **Multiplication** >>> A @ D Multiplication operator

(Python 3) >>> np.multiply(D,A) Multiplication >>> np.dot(A,D) Dot product >>> np.vdot(A,D) Vector dot product >>> np.inner(A,D) Inner product >>> np.outer(A,D) Outer product >>> np.tensordot(A,D) Tensor dot product >>> np.kron(A,D) Kronecker product

**Exponential Functions** >>> linalg.expm(A) Matrix exponential >>> linalg.expm2(A) Matrix exponential (Taylor Series) >>> linalg.expm3(D) Matrix exponential (eigenvalue decomposition)

**Logarithm Function** >>> linalg.logm(A) Matrix logarithm

**Trigonometric Functions** >>> linalg.sinm(D) Matrix sine >>> linalg.cosm(D) Matrix cosine >>> linalg.tanm(A) Matrix tangent

**Hyperbolic Trigonometric Functions** >>> linalg.sinhm(D) Hypberbolic matrix sine >>> linalg.coshm(D) Hyperbolic matrix cosine >>> linalg.tanhm(A) Hyperbolic matrix tangent

**Matrix Sign Function** >>> np.signm(A) Matrix sign function

**Matrix Square Root** >>> linalg.sqrtm(A) Matrix square root **Creating Sparse Matrices**

**Arbitrary Functions** >>> linalg.funm(A, lambda x: x\*x) Evaluate matrix function >>> F = np.eye(3, k=1) Create a 2X2 identity matrix >>> G = np.mat(np.identity(2)) Create a 2x2 identity matrix >>> C[C > 0.5] = 0 >>> H = sparse.csr\_matrix(C) Compressed Sparse Row matrix >>> I = sparse.csc\_matrix(D) Compressed Sparse Column matrix >>> J = sparse.dok\_matrix(A) Dictionary Of Keys matrix >>> E.todense() Sparse matrix to full matrix >>> sparse.isspmatrix\_csc(A) Identify sparse matrix

**Inverse** >>> sparse.linalg.inv(I) Inverse

**Norm** >>> sparse.linalg.norm(I) Norm

**Solving linear problems** >>> sparse.linalg.spsolve(H,I) Solver for sparse matrices

**Decompositions**

**Eigenvalues and Eigenvectors** >>> la, v = linalg.eig(A) Solve ordinary or generalized eigenvalue problem for square matrix >>> l1, l2 = la Unpack eigenvalues >>> v[:,0] First eigenvector **Sparse Matrix Routines**

>>> v[:,1] Second eigenvector >>> linalg.eigvals(A) Unpack eigenvalues

**Singular Value Decomposition** >>> U,s,Vh = linalg.svd(B) Singular Value Decomposition (SVD) >>> M,N = B.shape >>> Sig = linalg.diagsvd(s,M,N) Construct sigma matrix in SVD **LU Decomposition** >>> P,L,U = linalg.lu(C) LU Decomposition

**Sparse Matrix Functions**

>>> sparse.linalg.expm(I) Sparse matrix exponential

**Sparse Matrix Decompositions**

>>> la, v = sparse.linalg.eigs(F,1) Eigenvalues and eigenvectors >>> sparse.linalg.svds(H, 2) SVD

**Python For Data Science** *Cheat Sheet* Scikit-Learn

**Supervised Learning Estimators**

Learn Python for data science **Interactively** at www.DataCamp.com

**Scikit-learn Scikit-learn** is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

**A Basic Example**

>>> from sklearn import neighbors, datasets, preprocessing >>> from sklearn.model\_selection import train\_test\_split

**Unsupervised Learning Estimators** >>> from sklearn.metrics import accuracy\_score >>> iris = datasets.load\_iris() >>> X, y = iris.data[:, :2], iris.target >>> X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=33)

**Principal Component Analysis (PCA)** >>> from sklearn.decomposition import PCA >>> pca = PCA(n\_components=0.95) >>> scaler = preprocessing.StandardScaler().fit(X\_train) >>> X\_train = scaler.transform(X\_train) >>> X\_test = scaler.transform(X\_test) >>> knn = neighbors.KNeighborsClassifier(n\_neighbors=5)

**K Means** >>> from sklearn.cluster import KMeans >>> k\_means = KMeans(n\_clusters=3, random\_state=0) >>> knn.fit(X\_train, y\_train) >>> y\_pred = knn.predict(X\_test) >>> accuracy\_score(y\_test, y\_pred)

**Loading The Data Also see NumPy & Pandas**

Fit the model to the data

**Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.** >>> import numpy as np Fit the model to the data

Fit to data, then transform it >>> X = np.random.random((10,5)) >>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F']) >>> X[X < 0.7] = 0

**Training And Test Data**

**Supervised Estimators** >>> y\_pred = svc.predict(np.random.random((2,5))) >>> y\_pred = lr.predict(X\_test) >>> from sklearn.model\_selection import train\_test\_split >>> X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

>>> y\_pred = knn.predict\_proba(X\_test)

**Unsupervised Estimators** >>> y\_pred = k\_means.predict(X\_test)

**Preprocessing The Data Standardization**

>>> from sklearn.preprocessing import StandardScaler >>> scaler = StandardScaler().fit(X\_train) >>> standardized\_X = scaler.transform(X\_train) >>> standardized\_X\_test = scaler.transform(X\_test)

>>> from sklearn.preprocessing >>> enc = LabelEncoder() >>> y = enc.fit\_transform(y)

Predict labels Predict labels Estimate probability of a label

Predict labels in clustering algos

**Encoding Categorical Features**

import LabelEncoder

**Normalization**

**Imputing Missing Values**

>>> from sklearn.preprocessing import Normalizer >>> scaler = Normalizer().fit(X\_train) >>> normalized\_X = scaler.transform(X\_train) >>> normalized\_X\_test = scaler.transform(X\_test)

>>> from sklearn.preprocessing import Imputer >>> imp = Imputer(missing\_values=0, strategy='mean', axis=0) >>> imp.fit\_transform(X\_train)

**Binarization**

**Generating Polynomial Features**

>>> from sklearn.preprocessing import Binarizer

>>> from sklearn.preprocessing import PolynomialFeatures >>> binarizer = Binarizer(threshold=0.0).fit(X)

>>> poly = PolynomialFeatures(5) >>> binary\_X = binarizer.transform(X)

>>> poly.fit\_transform(X)

**DataCamp Learn Python for Data Science Interactively**

**Create Your Model**

**Evaluate Your Model’s Performance Classification Metrics**

**Linear Regression**

**Accuracy Score** >>> from sklearn.linear\_model import LinearRegression >>> knn.score(X\_test, y\_test) >>> lr = LinearRegression(normalize=True)

>>> from sklearn.metrics import accuracy\_score **Support Vector Machines (SVM)** >>> from sklearn.svm import SVC >>> accuracy\_score(y\_test, y\_pred)

**Classification Report** >>> svc = SVC(kernel='linear')

**Naive Bayes** >>> from sklearn.naive\_bayes import GaussianNB >>> gnb = GaussianNB() **KNN** >>> from sklearn.metrics import classification\_report >>> print(classification\_report(y\_test, y\_pred))

**Confusion Matrix** >>> from sklearn.metrics import confusion\_matrix >>> print(confusion\_matrix(y\_test, y\_pred))

>>> from sklearn import neighbors >>> knn = neighbors.KNeighborsClassifier(n\_neighbors=5)

**Model Fitting**

**Supervised learning** >>> lr.fit(X, y) >>> knn.fit(X\_train, y\_train) >>> svc.fit(X\_train, y\_train) **Unsupervised Learning** >>> k\_means.fit(X\_train) >>> pca\_model = pca.fit\_transform(X\_train)

**Prediction**

**Cross-Validation**

>>> from sklearn.cross\_validation import cross\_val\_score >>> print(cross\_val\_score(knn, X\_train, y\_train, cv=4)) >>> print(cross\_val\_score(lr, X, y, cv=2))

**Tune Your Model Grid Search**

>>> from sklearn.grid\_search import GridSearchCV >>> params = {"n\_neighbors": np.arange(1,3), "metric": ["euclidean", "cityblock"]} >>> grid = GridSearchCV(estimator=knn, param\_grid=params) >>> grid.fit(X\_train, y\_train) >>> print(grid.best\_score\_) >>> print(grid.best\_estimator\_.n\_neighbors) **Randomized Parameter Optimization**

>>> from sklearn.grid\_search import RandomizedSearchCV >>> params = {"n\_neighbors": range(1,5), >>> rsearch = "weights": RandomizedSearchCV(estimator=knn, ["uniform", cv=4, n\_iter=8, param\_distributions=params, "distance"]} random\_state=5) >>> rsearch.fit(X\_train, y\_train) >>> print(rsearch.best\_score\_) Estimator score method Metric scoring functions

Precision, recall, f1-score and support

**Regression Metrics Mean Absolute Error** >>> from sklearn.metrics import mean\_absolute\_error >>> y\_true = [3, -0.5, 2] >>> mean\_absolute\_error(y\_true, y\_pred)

**Mean Squared Error** >>> from sklearn.metrics import mean\_squared\_error >>> mean\_squared\_error(y\_test, y\_pred) **R2 Score** >>> from sklearn.metrics import r2\_score >>> r2\_score(y\_true, y\_pred)

**Clustering Metrics Adjusted Rand Index** >>> from sklearn.metrics import adjusted\_rand\_score >>> adjusted\_rand\_score(y\_true, y\_pred) >>> **Homogeneity** from sklearn.metrics import homogeneity\_score >>> homogeneity\_score(y\_true, y\_pred) **V-measure** >>> from sklearn.metrics import v\_measure\_score >>> metrics.v\_measure\_score(y\_true, y\_pred)

3 **Python For Data Science** *Cheat Sheet*

**Plotting With Seaborn** Seaborn

**Axis Grids**

Learn Data Science **Interactively** at www.DataCamp.com

**Statistical Data Visualization With Seaborn**

>>> import matplotlib.pyplot as plt >>> import seaborn as sns >>> tips = sns.load\_dataset("tips")

4 **Further Customizations** >>> sns.set\_style("whitegrid") >>> g = sns.lmplot(x="tip", y="total\_bill", data=tips,

aspect=2) >>> g = (g.set\_axis\_labels("Tip","Total bill(USD)"). set(xlim=(0,10),ylim=(0,100))) >>> plt.title("title") >>> plt.show(g)

**Show or Save Plot**

>>> f, ax = plt.subplots(figsize=(5,6)) Create a figure and one subplot >>> sns.set() >>> sns.set\_style("whitegrid") >>> sns.set\_style("ticks", (Re)set Set Set the the >>> g = sns.FacetGrid(titanic, Subplot grid for plotting conditional

>>> h = sns.PairGrid(iris) col="survived", relationships >>> h = h.map(plt.scatter) row="sex") >>> sns.pairplot(iris) >>> g = g.map(plt.hist,"age") >>> i = sns.JointGrid(x="x", >>> sns.factorplot(x="pclass", Draw a categorical plot onto a

y="y", Subplot grid for plotting pairwise relationships Plot pairwise bivariate distributions Grid for bivariate plot with marginal univariate plots y="survived", Facetgrid

data=data) hue="sex",

>>> i = i.plot(sns.regplot, data=titanic)

sns.distplot) The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing

>>> sns.lmplot(x="sepal\_width", Plot data and regression model fits

>>> sns.jointplot("sepal\_length", Plot bivariate distribution y="sepal\_length", across a FacetGrid

"sepal\_width", hue="species",

data=iris, attractive statistical graphics.

data=iris)

kind='kde')

Make use of the following aliases to import the libraries:

**Categorical Plots Regression Plots Scatterplot** >>> sns.stripplot(x="species", Scatterplot with one

y="petal\_length", categorical variable

The basic steps to creating plots with Seaborn are:

data=iris) >>> sns.swarmplot(x="species", 1. Prepare some data 2. Control figure aesthetics

y="petal\_length", **Bar Chart**

data=iris) Categorical scatterplot with

non-overlapping points

3. Plot with Seaborn 4. Further customize your plot

>>> sns.barplot(x="sex", y="survived", Show point estimates and

confidence intervals with

**Matrix Plots** hue="class", **Count Plot**

data=titanic) scatterplot glyphs

>>> sns.heatmap(uniform\_data,vmin=0,vmax=1) Heatmap

>>> sns.countplot(x="deck", Show count of observations

data=titanic,

**Point Plot** palette="Greens\_d") >>> sns.pointplot(x="class", Show point estimates and

y="survived", confidence intervals as hue="sex", rectangular bars data=titanic, palette={"male":"g",

"female":"m"}, markers=["^","o"],

**Boxplot** linestyles=["-","--"]) >>> sns.boxplot(x="alive", Boxplot

y="age", >>> import pandas as pd hue="adult\_male", >>> import numpy as np >>> uniform\_data = np.random.rand(10, 12) >>> data = pd.DataFrame({'x':np.arange(1,101), 'y':np.random.normal(0,4,100)})

>>> sns.regplot(x="sepal\_width", Plot data and a linear regression >>> import matplotlib.pyplot as plt

y="sepal\_length", model fit >>> import seaborn as sns

data=iris, ax=ax) **Distribution Plots**

>>> g.despine(left=True) Remove left spine >>> g.set\_ylabels("Survived") Set the labels of the y-axis >>> g.set\_xticklabels(rotation=45) Set the tick labels for x >>> g.set\_axis\_labels("Survived", Set the axis labels

"Sex") >>> h.set(xlim=(0,5), ylim=(0,5), Set x-and the y-axis limit and ticks of the xticks=[0,2.5,5],

yticks=[0,2.5,5])

data=titanic) >>> sns.boxplot(data=iris,orient="h") Boxplot with wide-form data

**Violinplot** >>> sns.violinplot(x="age", Violin plot

y="sex", hue="survived", data=titanic)

the matplotlib matplotlib seaborn parameters parameters

default

{"xtick.major.size":8, >>> sns.axes\_style("whitegrid") "ytick.major.size":8})

with Return to a temporarily dict of params set the or use style

with

**Close & Clear** >>> plt.cla() >>> plt.clf() >>> plt.close() >>> plot = sns.distplot(data.y, kde=False, Plot univariate distribution color="b") Step 1

**Also see Matplotlib**

Step 5

**Context Functions**

**Seaborn styles**

Clear Clear Close an an a window axis entire figure **DataCamp Learn Python for Data Science Interactively** Step 2

Step 3

**Axisgrid Objects**

Step 4

1

**Data**

**Also see Lists, NumPy & Pandas**

**Plot**

>>> plt.title("A Title") Add plot title >>> plt.ylabel("Survived") Adjust the label of the y-axis >>> plt.xlabel("Sex") Adjust the label of the x-axis >>> plt.ylim(0,100) Adjust the limits of the y-axis

Seaborn also offers built-in data sets:

>>> plt.xlim(0,10) Adjust the limits of the x-axis >>> plt.setp(ax,yticks=[0,5]) Adjust a plot property >>> titanic = sns.load\_dataset("titanic")

>>> plt.tight\_layout() Adjust subplot params

2

>>> iris = sns.load\_dataset("iris")

**Also see Matplotlib** 5 >>> plt.show() >>> plt.savefig("foo.png") >>> >>> sns.set\_context("talk") sns.set\_context("notebook", font\_scale=1.5, rc={"lines.linewidth":2.5}) override Set Set Scale context context font param elements to to "talk"

"notebook", mapping and **Figure Aesthetics**

**Also see Matplotlib**

>>> plt.savefig("foo.png", Show Save Save the transparent the plot plot as a figure figure transparent=True)

**Also see Matplotlib**

**Color Palette** >>> sns.set\_palette("husl",3) >>> sns.color\_palette("husl") Define Use with the with color to palette temporarily set palette >>> >>> flatui sns.set\_palette(flatui) = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]

Set your own color palette

3 **Python For Data Science** *Cheat Sheet*

**Renderers & Visual Customizations** Bokeh

**Glyphs**

Learn Bokeh **Interactively** at www.DataCamp.com, taught by Bryan Van de Ven, core contributor

**Scatter Markers** >>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),

fill\_color='white') >>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],

**Plotting With Bokeh**

**Line Glyphs** color='blue', size=1) >>> p1.line([1,2,3,4], [3,4,5,6], line\_width=2) The Python interactive visualization library **Bokeh** enables high-performance visual presentation of

>>> p2.multi\_line(pd.DataFrame([[1,2,3],[5,6,7]]),

pd.DataFrame([[3,4,5],[3,2,1]]), color="blue") large datasets in modern web browsers.

**Customized Glyphs**

**Also see Data** Bokeh’s mid-level general purpose bokeh.plotting

**Selection and Non-Selection Glyphs** interface is centered around two main components: data and glyphs.

>>> p = figure(tools='box\_select') >>> p.circle('mpg', 'cyl', source=cds\_df,

selection\_color='red', nonselection\_alpha=0.1) + =

**Hover Glyphs** >>> from bokeh.models import HoverTool *data glyphs plot* The basic steps to creating plots with the bokeh.plotting

>>> hover = HoverTool(tooltips=None, mode='vline') >>> p3.add\_tools(hover)

interface are:

1. Prepare some data: Python lists, NumPy arrays, Pandas DataFrames and other sequences of values

2. Create a new plot 3. Add renderers for your data, with visual customizations

**Colormapping** >>> from bokeh.models import CategoricalColorMapper >>> color\_mapper = CategoricalColorMapper(

factors=['US', 'Asia', 'Europe'],

4 **Output & Export** palette=['blue', 'red', 'green']) >>> p3.circle('mpg', 'cyl', source=cds\_df, 4. Specify where to generate the output 5. Show or save the resultscolor=dict(field='origin',

transform=color\_mapper), legend='Origin')

>>> from bokeh.io import output\_notebook, show >>> output\_notebook() >>> from bokeh.plotting import figure >>> from bokeh.io import output\_file, show

**HTML** >>> x = [1, 2, 3, 4, 5] >>> y = [6, 7, 2, 4, 5]

Step 1

>>> p = figure(title="simple line example",

Step 2 x\_axis\_label='x', y\_axis\_label='y') >>> p.line(x, y, legend="Temp.", line\_width=2)

Step 3

>>> from bokeh.io import output\_file, show >>> output\_file("lines.html")

Step 4 >>> show(p)

Step 5

>>> output\_file('my\_bar\_chart.html', mode='cdn')

1**Data Also see Lists, NumPy & Pandas**

**Components** >>> from bokeh.embed import components >>> script, div = components(p) Under the hood, your data is converted to Column Data Sources. You can also do this manually: >>> import numpy as np >>> import pandas as pd >>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'], [32.4,4,66, 'Asia'], [21.4,4,109, 'Europe']]), columns=['mpg','cyl', 'hp', 'origin'], index=['Toyota', 'Fiat', 'Volvo'])

>>> from bokeh.models import ColumnDataSource >>> cds\_df = ColumnDataSource(df) 2

**Plotting** >>> from bokeh.plotting import figure >>> p1 = figure(plot\_width=300, tools='pan,box\_zoom') >>> p2 = figure(plot\_width=300, plot\_height=300,

x\_range=(0, 8), y\_range=(0, 8))

**DataCamp** >>> p3 = figure()

**Learn Python for Data Science Interactively**

**Grid Layout**

>>> from bokeh.layouts import gridplot >>> row1 = [p1,p2] >>> row2 = [p3] >>> layout = gridplot([[p1,p2],[p3]])

**Tabbed Layout**

>>> from bokeh.models.widgets import Panel, Tabs >>> tab1 = Panel(child=p1, title="tab1") >>> tab2 = Panel(child=p2, title="tab2") >>> layout = Tabs(tabs=[tab1, tab2])

**Linked Plots**

**Inside Plot Area** >>> p.legend.location = 'bottom\_left'

>>> >>> >>> >>> **Outside** from r1 r2 legend = = >>> p.add\_layout(legend, **Plot** bokeh.models p2.asterisk(np.array([1,2,3]), p2.line([1,2,3,4], = **Area** Legend(items=[("One" location=(0, import 'right')

[3,4,5,6]) Legend -30)) ,[p1, np.array([3,2,1]) r1]),("Two",[r2])], **Linked Axes**

>>> p2.x\_range = p1.x\_range >>> p2.y\_range = p1.y\_range

**Linked Brushing** >>> p4 = figure(plot\_width = 100,

tools='box\_select,lasso\_select') >>> p4.circle('mpg', 'cyl', source=cds\_df) >>> p5 = figure(plot\_width = 200,

tools='box\_select,lasso\_select') >>> p5.circle('mpg', 'hp', source=cds\_df) >>> layout = row(p4,p5)

**Notebook**

**Legend Location**

**Standalone HTML** >>> from bokeh.embed import file\_html >>> from bokeh.resources import CDN >>> html = file\_html(p, CDN, "my\_plot")

**Legend Orientation**

**Legend Background & Border**

**Rows & Columns Layout Rows** >>> from bokeh.layouts import row >>> layout = row(p1,p2,p3) >>> **Columns** from bokeh.layouts import columns >>> layout = column(p1,p2,p3) >>> p.legend.orientation = "horizontal"

**PNG** >>> p.legend.orientation = "vertical"

>>> from bokeh.io import export\_png >>> export\_png(p, filename="plot.png")

>>> p.legend.border\_line\_color = "navy" >>> p.legend.background\_fill\_color = "white"

**SVG**

>>> from bokeh.io import export\_svgs >>> p.output\_backend = "svg" >>> export\_svgs(p, filename="plot.svg") 5

**Show or Save Your Plots** >>> show(p1) >>> show(layout) >>> save(p1) >>> save(layout) **Nesting Rows & Columns** >>>layout = row(column(p1,p2), p3)

USAsia

Europe

**Python For Data Science** *Cheat Sheet*

**Duplicate Values**

**GroupBy** PySpark - SQL Basics Learn Python for data science **Interactively** at www.DataCamp.com

>>> df = df.dropDuplicates() **Queries**

>>> df.groupBy("age")\ Group by age, count the members .count() \ in the groups .show()

>>> **Select** from pyspark.sql import functions as F >>> df.select("firstName").show() Show all entries in firstName column >>> df.select("firstName","lastName") \ .show() >>> df.select("firstName", Show all entries in firstName, age "age", and type

explode("phoneNumber") \ .alias("contactInfo")) \ .select("contactInfo.type", "firstName", "age") \ .show() >>> df.select(df["firstName"],df["age"]+ 1) Show all entries in firstName and age, .show() add 1 to the entries of age >>> **When** df.select(df['age'] > 24).show() Show all entries where age >24 >>> df.select("firstName", Show firstName and 0 or 1 depending F.when(df.age > 30, 1) \ on age >30 .otherwise(0)) \ .show() >>> df[df.firstName.isin("Jane","Boris")] Show firstName if in the given options **Like** .collect() **Filter PySpark & Spark SQL**

>>> df.filter(df["age"]>24).show() Filter entries of age, only keep those records of which the values are >24

**Spark SQL** is Apache Spark's module for working with structured data.

**Sort**

**Initializing SparkSession** >>> peopledf.sort(peopledf.age.desc()).collect() >>> df.sort("age", ascending=False).collect() >>> df.orderBy(["age","city"],ascending=[0,1])\ A SparkSession can be used create DataFrame, register DataFrame as tables,

.collect() execute SQL over tables, cache tables, and read parquet files. >>> from pyspark.sql import SparkSession >>> spark = SparkSession \ **Missing & Replacing Values**

.builder \ .appName("Python Spark SQL basic example") \ .config("spark.some.config.option", "some-value") \ .getOrCreate()

>>> df.na.fill(50).show() >>> df.na.drop().show() >>> df.na \ .replace(10, 20) \ .show() Replace Return Return another new new null df df values omitting replacing rows one value with null with values **Creating DataFrames From RDDs**

>>> from pyspark.sql.types import \*

>>> df.select("firstName", Show firstName, and lastName is df.lastName.like("Smith")) \ TRUE if lastName is like Smith **Repartitioning Startswith** .show() **- Endswith** >>> df.repartition(10)\ df with 10 partitions **Infer Schema** >>> sc = spark.sparkContext >>> lines = sc.textFile("people.txt") >>> df.select("firstName", Show firstName, and TRUE if df.lastName \ lastName starts with Sm .startswith("Sm")) \ >>> parts = lines.map(lambda l: l.split(",")) >>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1]))) >>> peopledf = spark.createDataFrame(people) **Specify Schema** .rdd \ .getNumPartitions() >>> df.coalesce(1).rdd.getNumPartitions() df with 1 partition .show() >>> people = parts.map(lambda p: Row(name=p[0],

>>> df.select(df.lastName.endswith("th")) **Substring** .show() >>> df.select(df.firstName.substr(1, \ Show last names ending in th **Running SQL Queries Programmatically**

3) \ Return substrings of firstName .alias("name")) \ **Registering DataFrames as Views**

>>> schemaString = "name age" >>> fields = [StructField(field\_name, field\_name in schemaString.split()]

age=int(p[1].strip()))) **Between** .collect() StringType(), True) for

>>> df.select(df.age.between(22, .show() >>> peopledf.createGlobalTempView("people") 24)) \ Show age: values are TRUE if between 22 and 24

>>> df.createTempView("customer") >>> df.createOrReplaceTempView("customer")

>>> schema = StructType(fields) >>> +--------+---+ | +--------+---+ | | |Jonathan| +--------+---+

spark.createDataFrame(people, Filip| name|age| Mine| 28| 29| 30| schema).show() **Add, Update & Remove Columns**

**Query Views**

>>> df5 = spark.sql("SELECT \* FROM customer").show() **Adding Columns**

>>> peopledf2 = spark.sql("SELECT \* FROM global\_temp.people")\ .show() >>> df = df.withColumn('city',df.address.city) \ **From Spark Data Sources**

.withColumn('postalCode',df.address.postalCode) \ .withColumn('state',df.address.state) \ .withColumn('streetAddress',df.address.streetAddress) \ .withColumn('telePhoneNumber', explode(df.phoneNumber.number)) \ **Output JSON Data Structures** >>> df = spark.read.json("customer.json") >>> +--------------------+---+---------+--------+--------------------+ | +--------------------+---+---------+--------+--------------------+ |[New |[New df.show() York,10021,N...| York,10021,N...| address|age|firstName 25| 21| |lastName| phoneNumber| John| Jane| Smith|[[212 Doe|[[322 555-1234,ho...| 888-1234,ho...|

.withColumn('telePhoneType', explode(df.phoneNumber.type)) >>> rdd1 = df.rdd Convert df into an RDD >>> df.toJSON().first() Convert df into a RDD of string **Updating Columns**

>>> df.toPandas() Return the contents of df as Pandas

DataFrame

+--------------------+---+---------+--------+--------------------+ >>> df2 = spark.read.load("people.json", format="json")

>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')

**Write & Save to Files Parquet files** >>> df3 = spark.read.load("users.parquet")

**Removing Columns**

**TXT files** >>> df4 = spark.read.text("people.txt")

>>> df = df.drop("address", "phoneNumber") >>> df = df.drop(df.address).drop(df.phoneNumber) >>> df.select("firstName", "city")\ .write \ .save("nameAndCity.parquet") >>> df.select("firstName", "age") \ .write \ **Inspect Data**

.save("namesAndAges.json",format="json")

>>> df.dtypes >>> df.show() >>> df.head() >>> df.first() >>> df.take(2) >>> df.schema Return df column names and data types Display the content of df Return first n rows Return first row Return the first n rows Return the schema of df

>>> df.describe().show() >>> df.columns >>> df.count() >>> df.distinct().count() >>> df.printSchema() >>> df.explain() Compute summary statistics Return the columns of df Count the number of rows in df Count the number of distinct rows in df Print the schema of df Print the (logical and physical) plans

**Stopping SparkSession**

>>> spark.stop() **DataCamp**

**Learn Python for Data Science Interactively**