VIP Cheatsheet: Machine Learning Tips

Afshine Amidi and Shervine Amidi

September 9, 2018

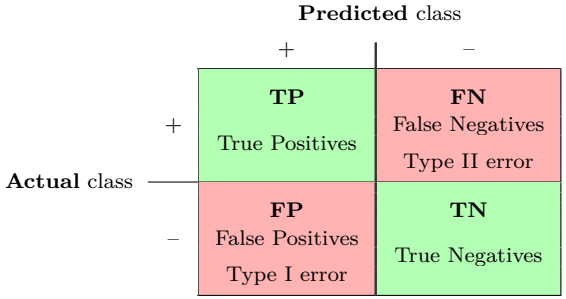
Metrics

Given a set of data points $\{x^{(1)}, ..., x^{(m)}\}$, where each $x^{(i)}$ has n features, associated to a set of outcomes $\{y^{(1)}, ..., y^{(m)}\}$, we want to assess a given classifier that learns how to predict y from x.

Classification

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

❒ **Confusion matrix** – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:



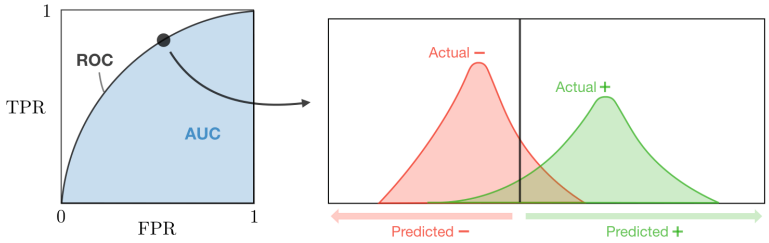
❒ **Main metrics** – The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$	Overall performance of model
Precision	$\frac{\text{TP}}{\text{TP}+\text{FP}}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Coverage of actual positive sample
Specificity	$\frac{\text{TN}}{\text{TN}+\text{FP}}$	Coverage of actual negative sample
F1 score	$\frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}}$	Hybrid metric useful for unbalanced classes

❒ **ROC** – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Recall, sensitivity
False Positive Rate FPR	$\frac{\text{FP}}{\text{TN}+\text{FP}}$	1-specificity

❒ **AUC** – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Regression

❒ **Basic metrics** – Given a regression model f, the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$\text{SS}_{\text{tot}} = \sum_{i=1}^{m}(y_i - \overline{y})^2$	$\text{SS}_{\text{reg}} = \sum_{i=1}^{m}(f(x_i) - \overline{y})^2$	$\text{SS}_{\text{res}} = \sum_{i=1}^{m}(y_i - f(x_i))^2$

❒ **Coefficient of determination** – The coefficient of determination, often noted R^2 or r^2, provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}$$

❒ **Main metrics** – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{\text{SS}_{\text{res}} + 2(n+1)\widehat{\sigma^2}}{m}$	$2\big[(n+2) - \log(L)\big]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

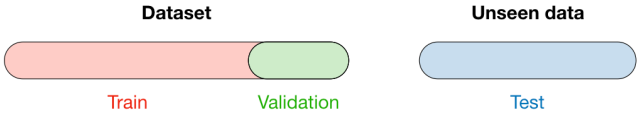
where L is the likelihood and $\widehat{\sigma}^2$ is an estimate of the variance associated with each response.

Model selection

❒ **Vocabulary** – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

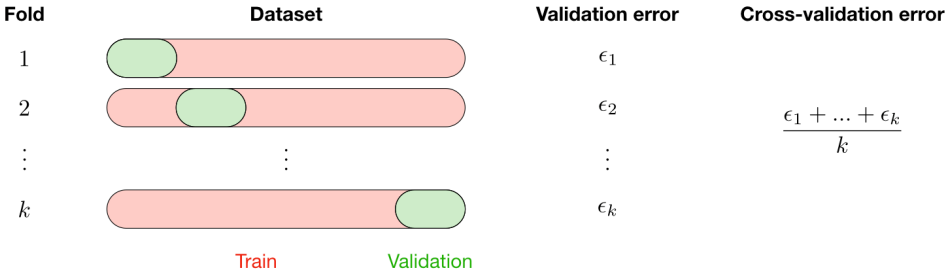
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



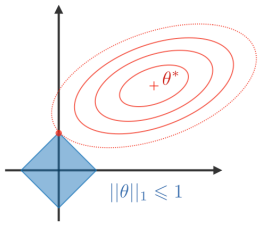
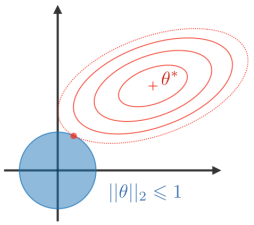
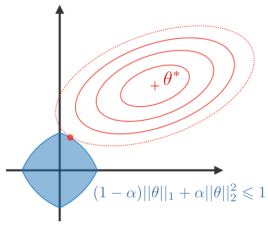
❒ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

k-fold	Leave-p-out
- Training on $k-1$ folds and assessment on the remaining one - Generally $k=5$ or 10	- Training on $n-p$ observations and assessment on the p remaining ones - Case $p=1$ is called leave-one-out

The most commonly used method is called k-fold cross-validation and splits the training data into k folds to validate the model on one fold while training the model on the $k-1$ other folds, all of this k times. The error is then averaged over the k folds and is named cross-validation error.



❒ **Regularization** – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$...+\lambda\|\|\theta\|\|_1$ $\lambda\in\mathbb{R}$	$...+\lambda\|\|\theta\|\|_2^2$ $\lambda\in\mathbb{R}$	$...+\lambda\left[(1-\alpha)\|\|\theta\|\|_1+\alpha\|\|\theta\|\|_2^2\right]$ $\lambda\in\mathbb{R}$, $\alpha\in[0,1]$

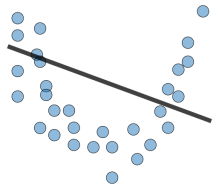
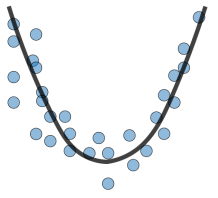
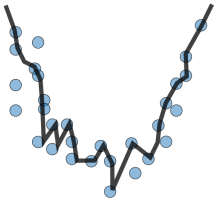
❒ **Model selection** – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

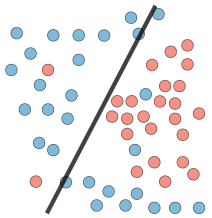
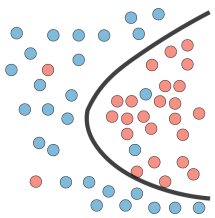
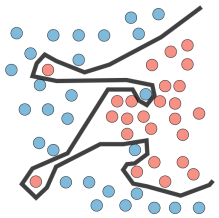
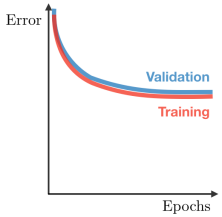
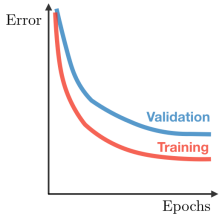
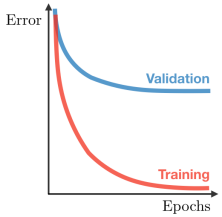
Diagnostics

❒ **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

❒ **Variance** – The variance of a model is the variability of the model prediction for given data points.

❒ **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			

Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

❒ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

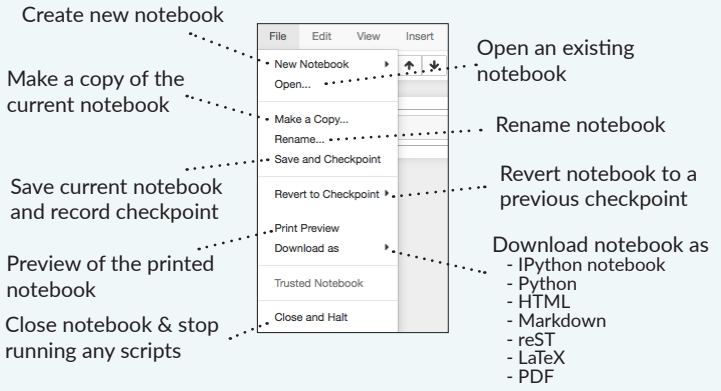
❒ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

Python For Data Science *Cheat Sheet*

Jupyter Notebook

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

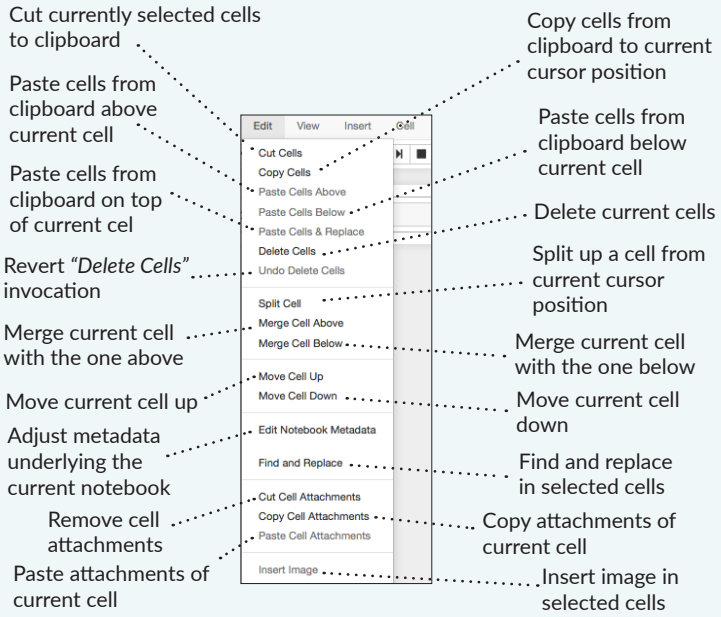
Saving/Loading Notebooks



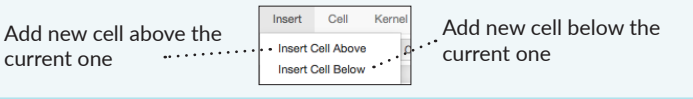
Writing Code And Text

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

Edit Cells



Insert Cells

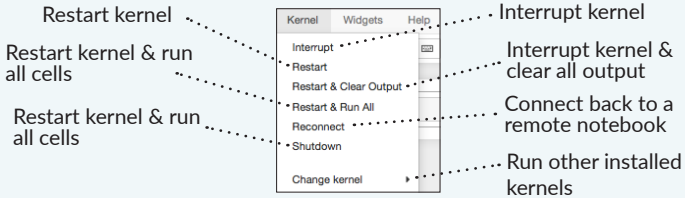


Working with Different Programming Languages

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

IPython IRkernel IJulia

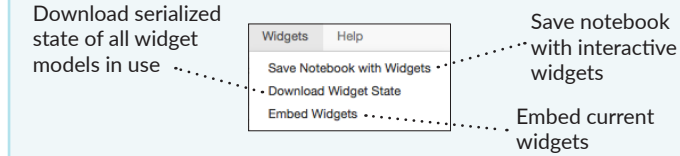
Installing Jupyter Notebook will automatically install the IPython kernel.



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.

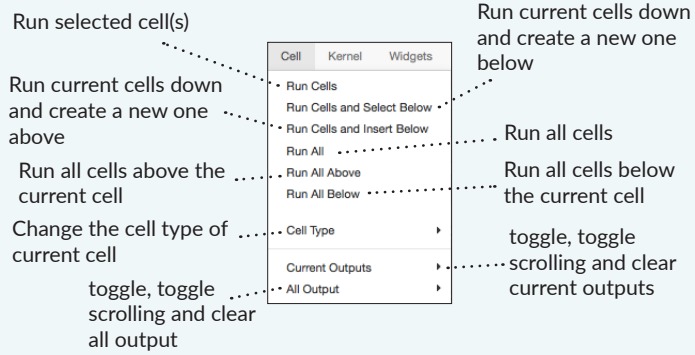


Command Mode:

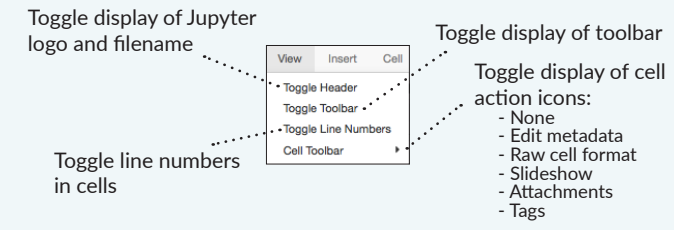
Edit Mode:

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

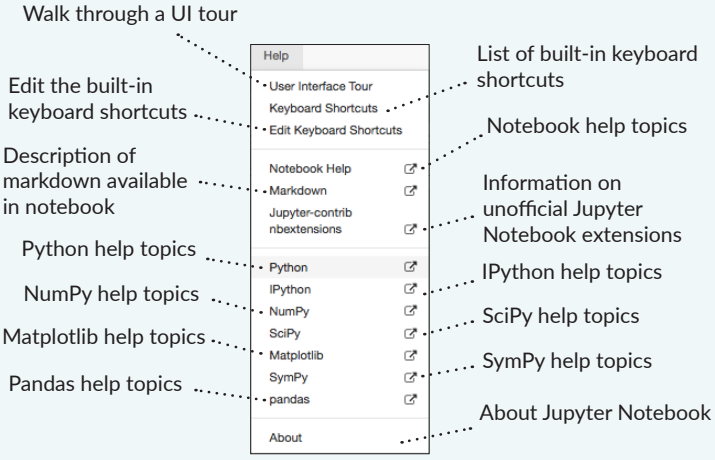
Executing Cells



View Cells



Asking For Help



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:

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

Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r')
>>> text = file.read()
>>> print(file.closed)
>>> file.close()
>>> print(text)
```

Open the file for reading
Read a file's contents
Check whether file is closed
Close file

Using the context manager `with`

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline())
        print(file.readline())
        print(file.readline())
```

Read a single line

Table Data: Flat Files

Importing Flat Files with numpy

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',',
                      skiprows=2,
                      usecols=[0,2],
                      dtype=str)
```

String used to separate values
Skip the first 2 lines
Read the 1st and 3rd column
The type of the resulting array

Files with mixed data types

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True,
                         dtype=None)
```

Look for column header

```
>>> data_array = np.recfromcsv(filename)
```

The default `dtype` of the `np.recfromcsv()` function is `None`.

Importing Flat Files with pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5,
                       header=None,
                       sep='\t',
                       comment='#',
                       na_values[""])
```

Number of rows of file to read
Row number to use as col names
Delimiter to use
Character to split comments
String to recognize as NA/NaN

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                                  'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

SAS Files

```
>>> from sas7bdat import SAS7BDAT
>>> 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')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> 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()
```

Using the context manager `with`

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype
>>> data_array.shape
>>> len(data_array)
```

Data type of array elements
Array dimensions
Length of array

pandas DataFrames

```
>>> df.head()
>>> df.tail()
>>> df.index
>>> df.columns
>>> df.info()
>>> data_array = data.values
```

Return first DataFrame rows
Return last DataFrame rows
Describe index
Describe DataFrame columns
Info on DataFrame
Convert a DataFrame to an a NumPy array

Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

Exploring Dictionaries

Accessing Elements with Functions

```
>>> print(mat.keys())
>>> for key in data.keys():
        print(key)
meta
quality
strain
>>> pickled_data.values()
>>> print(mat.items())
```

Print dictionary keys
Print dictionary keys

Return dictionary values
Returns items in list format of (key, value) tuple pairs

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys()
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
>>> print(data['meta']['Description'].value)
```

Explore the HDF5 structure

Retrieve the value for a key

Navigating Your FileSystem

Magic Commands

```
!ls
%cd ..
%pwd
```

List directory contents of files and directories
Change current working directory
Return the current working directory path

`os` Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd()
>>> os.listdir(wd)
>>> os.chdir(path)
>>> os.rename("test1.txt",
              "test2.txt")
>>> os.remove("test1.txt")
>>> os.mkdir("newdir")
```

Store the name of current directory in a string
Output contents of the directory in a list
Change current working directory
Rename a file

Delete an existing file
Create a new directory

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.

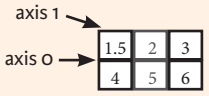
Use the following import convention:

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

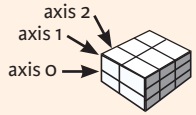
NumPy Arrays

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]
  3
```

Select the element at the 2nd index


```
>>> b[1,2]
  6.0
```

Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing


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

Select items at index 0 and 1


```
>>> b[0:2,1]
  array([ 2.,  5.])
```

Select items at rows 0 and 1 in column 1


```
>>> b[:1]
  array([[1.5, 2., 3.]])
```

Select all items at row 0 (equivalent to `b[0:1, :]`)

```
>>> c[1,...]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
```

Same as `[1,:,:]`

```
>>> a[ : :-1]
  array([3, 2, 1])
```

Reversed array `a`

Boolean Indexing


```
>>> a[a<2]
  array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
  array([ 4. , 2. , 6. , 1.5])
```

Select elements $(1,0)$, $(0,1)$, $(1,2)$ and $(0,0)$

```
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
  array([[ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Select a subset of the matrix's rows and columns

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 index
  [array([1]),array([2]),array([3])]
>>> 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.]]])]
```

Python For Data Science *Cheat Sheet*

Scikit-Learn

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
>>> 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)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> 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

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
>>> 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

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

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)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```
Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```
Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

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)
```

R^2 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)
```

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),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Data Wrangling

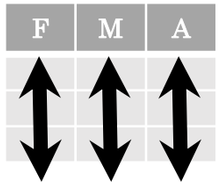
with pandas

Cheat Sheet

http://pandas.pydata.org

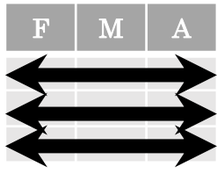
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



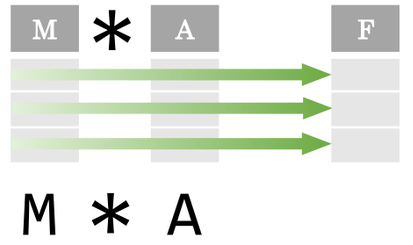
Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

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



Syntax – Creating DataFrames

	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')
    )
```

Reshaping Data – Change the layout of a data set

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

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

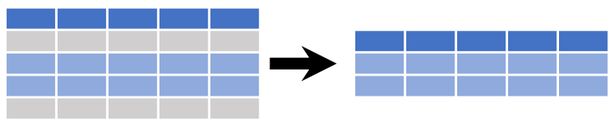
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

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.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	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

Subset Variables (Columns)

df[['width','length','species']]
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*.

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'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

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

Summarize Data

`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.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data

`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

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

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

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.

`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.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

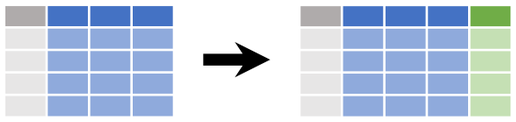
`cumprod()`
Cumulative product.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

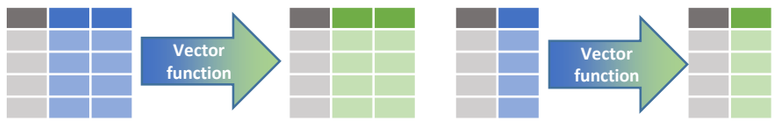
Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

`abs()`
Absolute value.

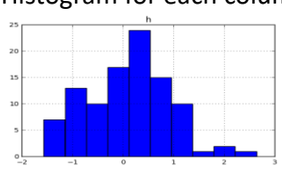
Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

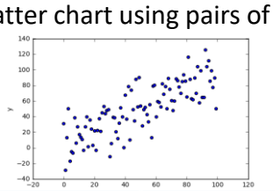
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

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

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

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

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

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

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

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

SciPy

The **SciPy** library is one of the core packages for 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)        Permute array dimensions
>>> b.flatten()            Flatten the array
>>> 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
```

Other Useful Functions

```
>>> 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
                           (number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g)           Unwrap
>>> np.logspace(0,10,3)    Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2]) Return values from a list of arrays depending 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
```

Linear Algebra

Also see NumPy

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)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I                    Inverse
>>> linalg.inv(A)          Inverse
```

Transposition

```
>>> A.T                    Tranpose matrix
>>> A.H                    Conjugate 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)
```

Creating Sparse Matrices

```
>>> 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
```

Sparse Matrix Routines

Inverse

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

Norm

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

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) Solver for sparse matrices
```

Sparse Matrix Functions

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

Matrix Functions

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)        Hyperbolic 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
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x) Evaluate matrix function
```

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
>>> 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 Decompositions

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

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

Python For Data Science *Cheat Sheet*

Keras

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

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

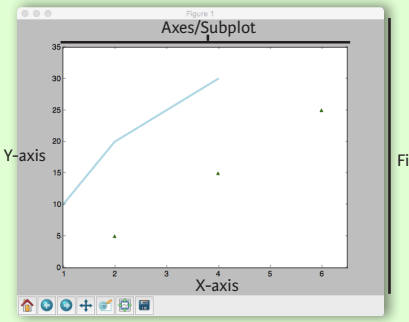
Axes

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)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> 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 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')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

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

Linestyles

```
>>> 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)
```

Text & Annotations

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

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

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',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                       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
```

3 Plotting Routines

1D Data

```
>>> 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 o
```

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 2D vector fields
```

Data Distributions

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

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> 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
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

plotly

GETTING STARTED

1. Install

In the terminal

```
sudo pip install plotly
```

2. Sign Up & Configure

http://www.plot.ly/python/getting-started

3. Boilerplate Imports

```
import plotly.plotly as py
import plotly.graph_objs as go
```

4. A Hello World Figure

```
trace = { 'x' : [1, 2] , 'y' : [1, 2] }
data = [ trace ]
layout = { }
fig = go.Figure (
    data = data, layout = layout )
```

5. Plot the Figure!

In the terminal:

```
plot_url = py.plot ( fig )
```

Or in the IPython notebook:

```
py.iplot ( fig )
```

BASIC CHARTS

Line Plots

```
trace1 = go.Scatter (
   x = [ 1, 2 ] , y = [ 1, 2 ] )
trace2 = go.Scatter (
   x = [ 1, 2 ] , y = [ 2, 1 ] )
py.iplot ( [ trace1, trace2 ] )
```

Bubble Charts

```
trace = go.Scatter (
  x = [ 1, 2, 3 ] , y = [ 1, 2, 3 ] ,
  marker = dict (
    color = [ 'red', 'blue' ,
    'green' ]
    size = [ 30, 80, 200 ] ) ,
    mode = 'markers' )
py.iplot ( [ trace ] )
```

Scatter Plots

```
trace1 = go.Scatter (
   x = [ 1, 2, 3 ] , y = [ 1 ,2, 3 ] ,
   text = [ 'A','B','C' ] ,
   textposition = 'top center'
   mode = 'markers+text' )
mode = [ trace ]
py.iplot ( data )
```

Heatmaps

```
trace = go.Heatmap (
   z = [ [ 1, 2, 3, 4 ] ,
       [ 5, 6, 7, 8 ] ] )
data = [ trace ]
py.iplot ( data )
```

Bar Charts

```
trace = go.Bar (
   x = [ 1, 2 ] , y = [ 1, 2 ] )
data = [ trace ]
py.iplot ( data )
```

Area Plots

```
trace = go.Scatter (
   x = [ 1, 2 ] , y = [ 1, 2 ] ,
   fill = 'tonexty' )
data = [ trace ]
py.iplot ( data )
```

LAYOUT

Legends

```
trace1 = go.Scatter (
   name = 'Calvin'
   x = [ 1, 2 ] , y = [ 1, 2 ] )

trace2 = go.Scatter (
   name = 'Hobbes'
   x = [ 2, 1 ] , y = [ 2, 1 ] )

layout = go.Layout (
   showlegend = True ,
   legend = dict (
      x = 0.2 , y = 0.5 )
   )

data =  [ trace1 , trace2 ]
fig = go.Figure (
   data = data ,
   layout = layout )
py.iplot ( fig )
```

Axes

```
trace = go.Scatter (
   x = [ 1, 2, 3, 4 ] ,
   y = [ 1, 2, 3, 6 ] )

axis_template = dict (
   showgrid = False ,
   zeroline = False ,
   nticks = 20 ,
   showline = True ,
   title = 'X AXIS'
   mirror = 'all' )
layout = go.Layout (
   xaxis = axis_template ,
   yaxis = axis_template ,
   )

data = [ trace ]
fig = go.Figure (
   data = data
   layout = layout
py.iplot ( fig )
```

PYTHON CLIENT **PLOT.LY/PYTHON** ALL LAYOUTS **PLOT.LY/PYTHON/REFERENCE/#LAYOUT**

STATISTICAL CHARTS

Histograms

```
trace = go.Histogram (
   x = [ 1, 2, 3, 3, 3, 4, 5 ] )
data = [ trace ]
py.iplot ( data )
```

Box Plots

```
trace = go.Box (
   x = [ 1, 2, 3, 3, 3, 4, 5 ] )
data = [ trace ]
py.iplot ( data )
```

2D Histogram

```
trace = go.Historgram2d (
   x = [ 1, 2, 3, 3, 3, 4, 5 ] ,
   x = [ 1, 2, 3, 3, 3, 4, 5 ] )
data = [ trace ]
py.iplot ( data )
```

MAPS

Bubble Map

```
trace = dict (
  type = 'scattergeo' ,
  lon = [ 100, 400 ] , lat = [ 0, 0] ,
  marker = dict (
    marker = [ 'red', 'blue' ]
    size = [ 30, 50 ] ) ,
  mode = 'markers' )
py.iplot ( [ trace ] )
```

Choropleth Map

```
trc = dict (
   type = 'choropleth' ,
   locations = [ 'AZ' , 'CA' , 'VT' ] ,
   locationmode = 'USA-states' ,
   colorscale = [ 'Viridis' ] ,
   z = [ 10, 20, 40 ] )
lyt = dict ( geo = dict ( scope = 'usa' ) )
map = go.Figure ( data = [ trc ] ,
   layout = lyt )
py.iplot ( map)
```

Scatter Map

```
trace = dict (
   type = 'scattergeo' ,
   Ion = [ 42, 39 ] , lat = [ 12, 22 ] ,
   marker = [ 'Rome' , 'Greece' ] ,
   mode = 'markers' )
py.iplot ( [ trace ] )
```

3D CHARTS

3D Surface Plots

```
trace = go.Surface (
   colorscale = 'Viridis' ,
   z = [ [ 3, 5, 8, 13] ,
       [ 21, 13, 8, 5 ] )
data =  [ trace ]
py.iplot ( data )
```

3D Line Plots

```
trace = go.Scatter3D (
   x = [ 9, 8, 5, 1] , y = [ 1, 2, 4, 8 ] ,
   z = [ 11, 8, 15, 3 ] ,
   mode = 'lines' )
data = [ trace ]
py.iplot ( data )
```

3D Scatter Plots

```
trace = go.Scatter3D (
   x = [ 9, 8, 5, 1] , y = [ 1, 2, 4, 8 ] ,
   z = [ 11, 8, 15, 3 ] ,
   mode = 'markers' )
data = [ trace ]
py.iplot ( data )
```

FIGURE HIERARCHY

Figure { }

```
DATA [ ]
  TRACE { }
    x, y, z [ ]
    color, text, size [ ]
    colorscale ABC or [ ]
     MARKER { }
       color ABC
       symbol ABC
       LINE { }
         color ABC
         width 123
LAYOUT { }
  title ABC
  XAXIS, YAXIS { }
  SCENE { }
    XAXIS, YAXIS, ZAXIS { }
  GEO { }
  LEGEND { }
  ANNOTATIONS { }
```

{ } = dictionary
[ ] = list
ABC = string
123 = number

Python For Data Science *Cheat Sheet*

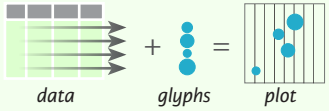
Bokeh

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

Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` 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
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]                          Step 1
>>> y = [6, 7, 2, 4, 5]
>>> 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
>>> output_file("lines.html")                    Step 4
>>> show(p)                                      Step 5
```

1 Data

Also see Lists, NumPy & Pandas

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'])
```

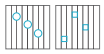
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

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))
>>> p3 = figure()
```

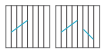
3 Renderers & Visual Customizations

Glyphs



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],
              color='blue', size=1)
```



Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
             selection_color='red',
             nonselection_alpha=0.1)
```

Hover Glyphs

```
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

Colormapping

```
>>> color_mapper = CategoricalColorMapper(
                   factors=['US', 'Asia', 'Europe'],
                   palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
              color=dict(field='origin',
                         transform=color_mapper),
                         legend='Origin'))
```

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)
```

Nesting Rows & Columns

```
>>>layout = row(column(p1,p2), p3)
```

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

Also see Data

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)
```

Legends

Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" , [p1, r1]),("Two" , [r2])], location=(0, -30))
>>> p.add_layout(legend, 'right')
```

Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

4 Output

Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Notebook Output

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

Embedding

Standalone HTML

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, "my_plot")
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

5 Show or Save Your Plots

```
>>> show(p1)
>>> show(layout)
```

```
>>> save(p1)
>>> save(layout)
```

Statistical Charts With Bokeh

Also see Data

Bokeh's high-level `bokeh.charts` interface is ideal for quickly creating statistical charts

Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red','blue'])
```

Box Plot

```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
                legend='bottom_right')
```

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y ='hp', marker='square',
                xlabel='Miles Per Gallon',
                ylabel='Horsepower')
```

Python For Data Science *Cheat Sheet*

PySpark - RDD Basics

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

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version                  Retrieve SparkContext version
>>> sc.pythonVer                Retrieve Python version
>>> sc.master                   Master URL to connect to
>>> str(sc.sparkHome)           Path where Spark is installed on worker nodes
>>> str(sc.sparkUser())         Retrieve name of the Spark User running SparkContext
>>> sc.appName                  Return application name
>>> sc.applicationId            Retrieve application ID
>>> sc.defaultParallelism       Return default level of parallelism
>>> sc.defaultMinPartitions     Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()      List the number of partitions
>>> rdd.count()                 Count RDD instances
 3
>>> rdd.countByKey()            Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()          Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()          Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum()                  Sum of RDD elements
 4950
>>> sc.parallelize([]).isEmpty()  Check whether RDD is empty
 True
```

Summary

```
>>> rdd3.max()                  Maximum value of RDD elements
  99
>>> rdd3.min()                  Minimum value of RDD elements
  0
>>> rdd3.mean()                 Mean value of RDD elements
  49.5
>>> rdd3.stdev()                Standard deviation of RDD elements
  28.866070047722118
>>> rdd3.variance()             Compute variance of RDD elements
  833.25
>>> rdd3.histogram(3)           Compute histogram by bins
  ([0,33,66,99],[33,33,34])
>>> rdd3.stats()                Summary statistics (count, mean, stdev, max & min)
```

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element
       .collect()
  [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))   Apply a function to each RDD element and flatten the result

>>> rdd5.collect()
  ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
>>> rdd4.flatMapValues(lambda x: x)   Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
       .collect()
  [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Selecting Data

Getting

```
>>> rdd.collect()               Return a list with all RDD elements
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                 Take first 2 RDD elements
  [('a', 7), ('a', 2)]
>>> rdd.first()                 Take first RDD element
  ('a', 7)
>>> rdd.top(2)                  Take top 2 RDD elements
  [('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()   Return sampled subset of rdd3
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x)   Filter the RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()   Return distinct RDD values
  ['a',2,'b',7]
>>> rdd.keys().collect()        Return (key,value) RDD's keys
  ['a', 'a', 'b']
```

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)              Apply a function to all RDD elements
  ('a', 7)
  ('b', 2)
  ('a', 2)
```

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)   Merge the rdd values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)   Merge the rdd values
  ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)   Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()            Group rdd by key
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)   Aggregate RDD elements of each partition and then the results
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)   Aggregate values of each RDD key
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)            Aggregate the elements of each partition, and then the results
  4950
>>> rdd.foldByKey(0, add)       Merge the values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)   Create tuples of RDD elements by applying a function
       .collect()
```

Mathematical Operations

```
>>> rdd.subtract(rdd2)          Return each rdd value not contained in rdd2
       .collect()
  [('b',2),('a',7)]
>>> rdd2.subtractByKey(rdd)     Return each (key,value) pair of rdd2 with no matching key in rdd
       .collect()
  [('d', 1)]
>>> rdd.cartesian(rdd2).collect()   Return the Cartesian product of rdd and rdd2
```

Sort

```
>>> rdd2.sortBy(lambda x: x[1])   Sort RDD by given function
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()            Sort (key, value) RDD by key
       .collect()
  [('a',2),('b',1),('d',1)]
```

Repartitioning

```
>>> rdd.repartition(4)          New RDD with 4 partitions
>>> rdd.coalesce(1)             Decrease the number of partitions in the RDD to 1
```

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

Python For Data Science *Cheat Sheet*

PySpark - SQL Basics

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

PySpark & Spark SQL

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

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

Creating DataFrames

From RDDs

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

Infer Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> 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

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
  +--------+---+
  |    name|age|
  +--------+---+
  |    Mine| 28|
  |   Filip| 29|
  |Jonathan| 30|
  +--------+---+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
 +--------------------+---+---------+--------+--------------------+
 |             address|age|firstName |lastName|         phoneNumber|
 +--------------------+---+---------+--------+--------------------+
 |[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
 |[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
 +--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet files

```
>>> df3 = spark.read.load("users.parquet")
```

TXT files

```
>>> df4 = spark.read.text("people.txt")
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show()
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName",
              "age",
               explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
               "firstName",
               "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)
      .show()
>>> df.select(df['age'] > 24).show()
```

When

```
>>> df.select("firstName",
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")]
                  .collect()
```

Like

```
>>> df.select("firstName",
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName",
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th")) \
      .show()
```

Substring

```
>>> df.select(df.firstName.substr(1, 3) \
                         .alias("name")) \
      .collect()
```

Between

```
>>> df.select(df.age.between(22, 24)) \
      .show()
```

Show all entries in `firstName` column

Show all entries in `firstName`, `age` and `type`

Show all entries in `firstName` and `age`, add `1` to the entries of `age`
Show all entries where `age` >24

Show `firstName` and 0 or 1 depending on `age` >30

Show `firstName` if in the given options

Show `firstName`, and `lastName` is `TRUE` if `lastName` is like `Smith`

Show `firstName`, and `TRUE` if `lastName` starts with `Sm`

Show last names ending in `th`

Return substrings of `firstName`

Show `age`: values are `TRUE` if between `22` and `24`

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
           .withColumn('postalCode',df.address.postalCode) \
           .withColumn('state',df.address.state) \
           .withColumn('streetAddress',df.address.streetAddress) \
           .withColumn('telePhoneNumber',
                       explode(df.phoneNumber.number)) \
           .withColumn('telePhoneType',
                       explode(df.phoneNumber.type))
```

Updating Columns

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

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age")\
      .count() \
      .show()
```

Group by `age`, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show()
```

Filter entries of `age`, only keep those records of which the values are `>24`

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
      .collect()
```

Missing & Replacing Values

```
>>> df.na.fill(50).show()
>>> df.na.drop().show()
>>> df.na \
      .replace(10, 20) \
      .show()
```

Replace null values
Return new `df` omitting rows with null values
Return new `df` replacing one value with another

Repartitioning

```
>>> df.repartition(10)\
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

`df` with 10 partitions

`df` with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                 .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert `df` into an RDD
Convert `df` into a RDD of string
Return the contents of `df` as Pandas `DataFrame`

Write & Save to Files

```
>>> df.select("firstName", "city")\
      .write \
      .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
      .write \
      .save("namesAndAges.json",format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```

Inspect Data

```
>>> 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

Python for Data Science Cheat Sheet spaCy

Learn more Python for data science interactively at www.datacamp.com

DataCamp

About spaCy

spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It's designed specifically for production use and helps you build applications that process and "understand" large volumes of text. **Documentation:** spacy.io

```
$ pip install spacy
```

```
import spacy
```

Statistical models

Download statistical models

Predict part-of-speech tags, dependency labels, named entities and more. See here for available models: spacy.io/models

```
$ python -m spacy download en_core_web_sm
```

Check that your installed models are up to date

```
$ python -m spacy validate
```

Loading statistical models

```
import spacy
# Load the installed model "en_core_web_sm"
nlp = spacy.load("en_core_web_sm")
```

Documents and tokens

Processing text

Processing text with the `nlp` object returns a `Doc` object that holds all information about the tokens, their linguistic features and their relationships

```
doc = nlp("This is a text")
```

Accessing token attributes

```
doc = nlp("This is a text")
# Token texts
[token.text for token in doc]
# ['This', 'is', 'a', 'text']
```

Spans

Accessing spans

Span indices are **exclusive**. So `doc[2:4]` is a span starting at token 2, up to – but not including! – token 4.

```
doc = nlp("This is a text")
span = doc[2:4]
span.text
# 'a text'
```

Creating a span manually

```
# Import the Span object
from spacy.tokens import Span
# Create a Doc object
doc = nlp("I live in New York")
# Span for "New York" with label GPE (geopolitical)
span = Span(doc, 3, 5, label="GPE")
span.text
# 'New York'
```

Linguistic features

Attributes return label IDs. For string labels, use the attributes with an underscore. For example, `token.pos_` .

Part-of-speech tags

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("This is a text.")
# Coarse-grained part-of-speech tags
[token.pos_ for token in doc]
# ['DET', 'VERB', 'DET', 'NOUN', 'PUNCT']
# Fine-grained part-of-speech tags
[token.tag_ for token in doc]
# ['DT', 'VBZ', 'DT', 'NN', '.']
```

Syntactic dependencies

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("This is a text.")
# Dependency labels
[token.dep_ for token in doc]
# ['nsubj', 'ROOT', 'det', 'attr', 'punct']
# Syntactic head token (governor)
[token.head.text for token in doc]
# ['is', 'is', 'text', 'is', 'is']
```

Named entities

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("Larry Page founded Google")
# Text and label of named entity span
[(ent.text, ent.label_) for ent in doc.ents]
# [('Larry Page', 'PERSON'), ('Google', 'ORG')]
```

Syntax iterators

Sentences

USUALLY NEEDS THE DEPENDENCY PARSER

```
doc = nlp("This a sentence. This is another one.")
# doc.sents is a generator that yields sentence spans
[sent.text for sent in doc.sents]
# ['This is a sentence.', 'This is another one.']
```

Base noun phrases

NEEDS THE TAGGER AND PARSER

```
doc = nlp("I have a red car")
# doc.noun_chunks is a generator that yields spans
[chunk.text for chunk in doc.noun_chunks]
# ['I', 'a red car']
```

Label explanations

```
spacy.explain("RB")
# 'adverb'
spacy.explain("GPE")
# 'Countries, cities, states'
```

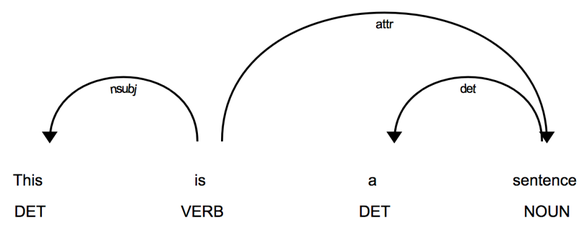
Visualizing

If you're in a Jupyter notebook, use `displacy.render` . Otherwise, use `displacy.serve` to start a web server and show the visualization in your browser.

```
from spacy import displacy
```

Visualize dependencies

```
doc = nlp("This is a sentence")
displacy.render(doc, style="dep")
```



Visualize named entities

```
doc = nlp("Larry Page founded Google")
displacy.render(doc, style="ent")
```

Larry Page PERSON founded Google ORG

Word vectors and similarity

To use word vectors, you need to install the larger models ending in `md` or `lg`, for example `en_core_web_lg`.

Comparing similarity

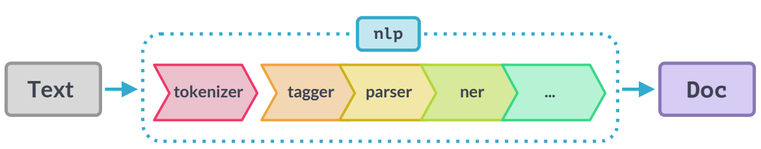
```
doc1 = nlp("I like cats")
doc2 = nlp("I like dogs")
# Compare 2 documents
doc1.similarity(doc2)
# Compare 2 tokens
doc1[2].similarity(doc2[2])
# Compare tokens and spans
doc1[0].similarity(doc2[1:3])
```

Accessing word vectors

```
# Vector as a numpy array
doc = nlp("I like cats")
# The L2 norm of the token's vector
doc[2].vector
doc[2].vector_norm
```

Pipeline components

Functions that take a `Doc` object, modify it and return it.



Pipeline information

```
nlp = spacy.load("en_core_web_sm")
nlp.pipe_names
# ['tagger', 'parser', 'ner']
nlp.pipeline
# [('tagger', <spacy.pipeline.Tagger>),
#  ('parser', <spacy.pipeline.DependencyParser>),
#  ('ner', <spacy.pipeline.EntityRecognizer>)]
```

Custom components

```
# Function that modifies the doc and returns it
def custom_component(doc):
    print("Do something to the doc here!")
    return doc

# Add the component first in the pipeline
nlp.add_pipe(custom_component, first=True)
```

Components can be added `first`, `last` (default), or `before` or `after` an existing component.

Extension attributes

Custom attributes that are registered on the global `Doc`, `Token` and `Span` classes and become available as `._`.

```
from spacy.tokens import Doc, Token, Span
doc = nlp("The sky over New York is blue")
```

Attribute extensions

WITH DEFAULT VALUE

```
# Register custom attribute on Token class
Token.set_extension("is_color", default=False)
# Overwrite extension attribute with default value
doc[6]._.is_color = True
```

Property extensions

WITH GETTER & SETTER

```
# Register custom attribute on Doc class
get_reversed = lambda doc: doc.text[::-1]
Doc.set_extension("reversed", getter=get_reversed)
# Compute value of extension attribute with getter
doc._.reversed
# 'eulb si kroY weN revo yks ehT'
```

Method extensions

CALLABLE METHOD

```
# Register custom attribute on Span class
has_label = lambda span, label: span.label_ == label
Span.set_extension("has_label", method=has_label)
# Compute value of extension attribute with method
doc[3:5].has_label("GPE")
# True
```

Rule-based matching

Using the matcher

```
# Matcher is initialized with the shared vocab
from spacy.matcher import Matcher
# Each dict represents one token and its attributes
matcher = Matcher(nlp.vocab)
# Add with ID, optional callback and pattern(s)
pattern = [{"LOWER": "new"}, {"LOWER": "york"}]
matcher.add("CITIES", None, pattern)
# Match by calling the matcher on a Doc object
doc = nlp("I live in New York")
matches = matcher(doc)
# Matches are (match_id, start, end) tuples
for match_id, start, end in matches:
    # Get the matched span by slicing the Doc
    span = doc[start:end]
    print(span.text)
# 'New York'
```

Rule-based matching

Token patterns

```
# "love cats", "loving cats", "loved cats"
pattern1 = [{"LEMMA": "love"}, {"LOWER": "cats"}]
# "10 people", "twenty people"
pattern2 = [{"LIKE_NUM": True}, {"TEXT": "people"}]
# "book", "a cat", "the sea" (noun + optional article)
pattern3 = [{"POS": "DET", "OP": "?"}, {"POS": "NOUN"}]
```

Operators and quantifiers

Can be added to a token dict as the `"OP"` key.

`!`	Negate pattern and match **exactly 0 times**.
`?`	Make pattern optional and match **0 or 1 times**.
`+`	Require pattern to match **1 or more times**.
`*`	Allow pattern to match **0 or more times**.

Glossary

Tokenization	Segmenting text into words, punctuation etc.
Lemmatization	Assigning the base forms of words, for example: "was" → "be" or "rats" → "rat".
Sentence Boundary Detection	Finding and segmenting individual sentences.
Part-of-speech (POS) Tagging	Assigning word types to tokens like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Named Entity Recognition (NER)	Labeling named "real-world" objects, like persons, companies or locations.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Statistical model	Process for making predictions based on examples.
Training	Updating a statistical model with new examples.