



Data Science Cheat Sheet for Business Leaders

Data Science Basics

Types of Data Science

- **Descriptive Analytics (Business Intelligence):** Get useful data in front of the right people in the form of dashboards, reports, and emails
 - Which customers have churned?
 - Which homes have sold in a given location, and do homes of a certain size sell more quickly?
- **Predictive Analytics (Machine Learning):** Put data science models continuously into production
 - Which customers may churn?
 - How much will a home sell for, given its location and number of rooms?
- **Prescriptive Analytics (Decision Science):** Use data to help a company make decisions
 - What should we do about the particular types of customers that are prone to churn?
 - How should we market a home to sell quickly, given its location and number of rooms?

The Standard Data Science Workflow

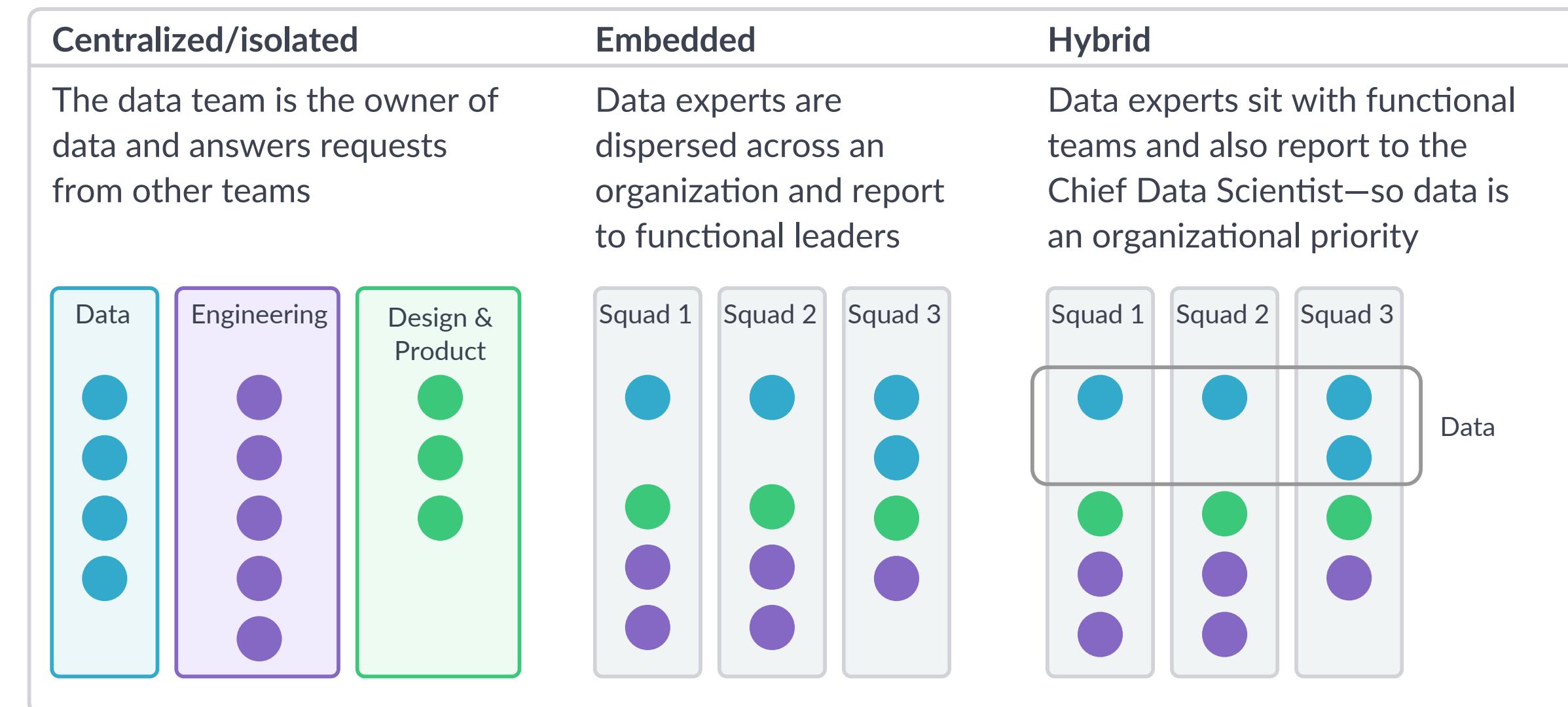
- 1 **Data Collection:** Compile data from different sources and store it for efficient access
- 2 **Exploration and Visualization:** Explore and visualize data through dashboards
- 3 **Experimentation and Prediction:** The buzziest topic in data science—machine learning!

Building a Data Science Team

Your data team members require different skills for different purposes.

Data Engineer	Data Analyst	Machine Learning Engineer	Data Scientist
Store and maintain data	Visualize and describe data	Write production-level code to predict with data	Build custom models to drive business decisions
SQL/Java/Scala/Python	SQL + BI Tools + Spreadsheets	Python/Java/R	Python/R/SQL

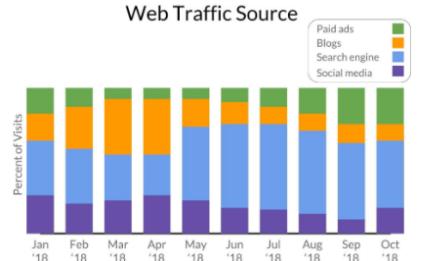
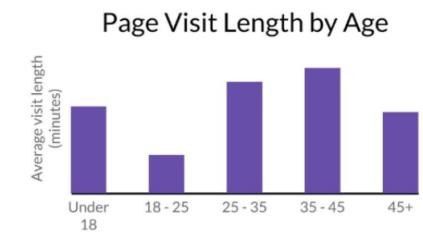
Data Science Team Organizational Models



Exploration and Visualization

The type of dashboard you should use depends on what you'll be using it for.

Common Dashboard Elements

Type	What is it best for?	Example
Time series	Tracking a value over time	
Stacked bar chart	Tracking composition over time	
Bar chart	Categorical comparison	

Popular Dashboard Tools

Spreadsheets	BI Tools	Customized Tools
 Excel	 Power BI	 R Shiny
 Sheets	 Tableau	 d3.js
 Looker		

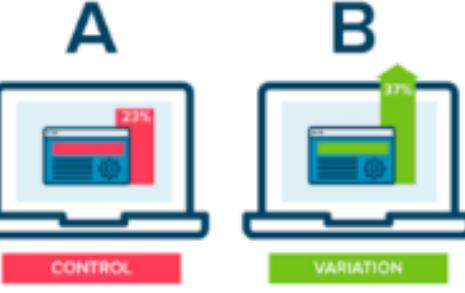
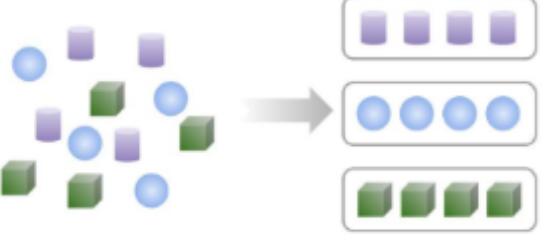
When You Should Request a Dashboard

-  When you'll use it multiple times
-  When you'll need the information updated regularly
-  When the request will always be the same

Experimentation and Prediction

Machine Learning

Machine learning is an application of artificial intelligence (AI) that builds algorithms and statistical models to train data to address specific questions without explicit instructions.

	Supervised Machine Learning	Unsupervised Machine Learning
Purpose	Makes predictions from data with labels and features	Makes predictions by clustering data with no labels into categories
Example	Recommendation systems, email subject optimization, churn prediction	Image segmentation, customer segmentation
A		

Special Topics in Machine Learning

- **Time Series Forecasting** is a technique for predicting events through a sequence of time and can capture seasonality or periodic events.
- **Natural Language Processing (NLP)** allows computers to process and analyze large amounts of natural language data.
 - Text as input data
 - Word counts track the important words in a text
 - Word embeddings create features that group similar words

Deep Learning / Neural Networks enables unsupervised machine learning using data that is unstructured or unlabeled.	Explainable AI is an emerging field in machine learning that applies AI such that results can be easily understood.
Highly accurate predictions Better for "What?"	Understandable by humans Better for "Why?"



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.sas/bdat') 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

<pre>>>> print(mat.keys()) >>> for key in mat.keys(): print(key)</pre> <p>meta quality strain</p> <pre>>>> pickled_data.values() >>> print(mat.items())</pre>	<p>Print dictionary keys Print dictionary keys</p> <p>Return dictionary values Returns items in list format of (key, value) tuple pairs</p>
---	---

Accessing Data Items with Keys

<pre>>>> for key in data['meta'].keys(): print(key)</pre> <p>Description DescriptionURL Detector Duration GRFstart Observatory Type UTCstart</p> <pre>>>> print(data['meta']['Description'].value)</pre>	<p>Explore the HDF5 structure</p> <p>Retrieve the value for a key</p>
--	---

Navigating Your FileSystem

Magic Commands

<pre>!ls %cd .. %pwd</pre>	<p>List directory contents of files and directories Change current working directory Return the current working directory path</p>
------------------------------------	--

os Library

<pre>>>> 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")</pre>	<p>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</p>
---	---



Python For Data Science Cheat Sheet

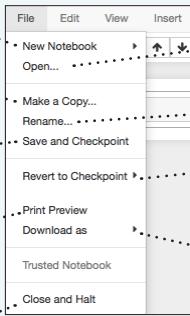
Jupyter Notebook

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



Saving/Loading Notebooks

Create new notebook



Make a copy of the current notebook

Save current notebook and record checkpoint

Preview of the printed notebook

Close notebook & stop running any scripts

Open an existing notebook

Rename notebook

Revert notebook to a previous checkpoint

Download notebook as

- IPython notebook
- Python
- HTML
- Markdown
- reST
- LaTeX
- PDF

Writing Code And Text

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

Edit Cells

Cut currently selected cells to clipboard

Paste cells from clipboard above current cell

Paste cells from clipboard on top of current cell

Revert "Delete Cells" invocation

Merge current cell with the one above

Move current cell up

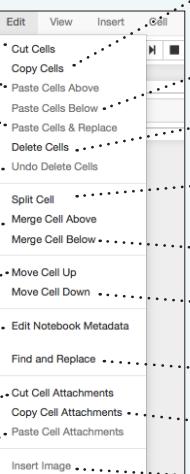
Adjust metadata underlying the current notebook

Remove cell attachments

Paste attachments of current cell

Insert Cells

Add new cell above the current one



Copy cells from clipboard to current cursor position

Paste cells from clipboard below current cell

Delete current cells

Split up a cell from current cursor position

Merge current cell with the one below

Move current cell down

Find and replace in selected cells

Copy attachments of current cell

Insert image in selected cells

Restart kernel

Restart kernel & run all cells

Restart kernel & run all cells



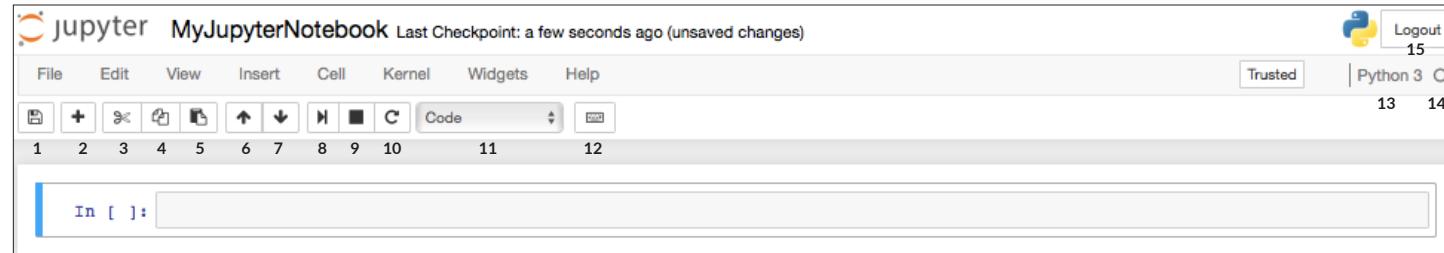
Interrupt kernel

Interrupt kernel & clear all output

Connect back to a remote notebook

Run other installed kernels

Command Mode:



Edit Mode:



Executing Cells

Run selected cell(s)

Run current cells down and create a new one above

Run all cells above the current cell

Change the cell type of current cell

toggle, toggle scrolling and clear all output

Cell Kernel Widgets

Run Cells
Run Cells and Select Below
Run Cells and Insert Below
Run All
Run All Above
Run All Below

Cell Type

Current Outputs

All Output

Run current cells down and create a new one below

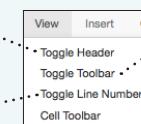
Run all cells

Run all cells below the current cell

toggle, toggle scrolling and clear current outputs

View Cells

Toggle display of Jupyter logo and filename



Toggle line numbers in cells

Toggle display of toolbar

Toggle display of cell action icons:

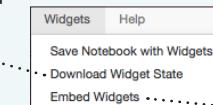
- None
- Edit metadata
- Raw cell format
- Slideshow
- Attachments
- Tags

Widgets

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

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

Download serialized state of all widget models in use

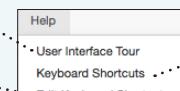


- Save notebook with interactive widgets
- Embed current widgets

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

Asking For Help

Walk through a UI tour



List of built-in keyboard shortcuts

Notebook help topics

Information on unofficial Jupyter Notebook extensions

IPython help topics

SciPy help topics

Sympy help topics

About Jupyter Notebook

Edit the built-in keyboard shortcuts

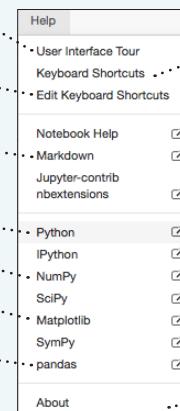
Description of markdown available in notebook

Python help topics

NumPy help topics

Matplotlib help topics

Pandas help topics



Insert Cells



Add new cell below the current one



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]
```

Preprocessing

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)
```

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.layers 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'))
```

Also see NumPy & Scikit-Learn

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)
```

Inspect Model

```
>>> model.output_shape
>>> model.summary()
>>> model.get_config()
>>> model.get_weights()
```

Model output shape
Model summary representation
Model configuration
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])
```



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.



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)
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()  
>>> lines = ax.plot(x, y)  
>>> ax.scatter(x, y)  
>>> axes[0,0].bar([1,2,3],[3,4,5])  
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])  
>>> axes[1,1].axhline(0.45)  
>>> axes[0,1].axvline(0.65)  
>>> ax.fill(x,y,color='blue')  
>>> ax.fill_between(x,y,color='yellow')
```

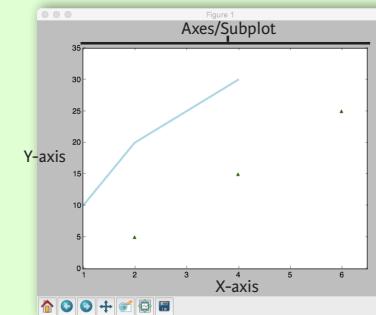
2D Data or Images

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

Colormapped or RGB arrays

Plot Anatomy & Workflow

Plot Anatomy



Figure

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]  
>>> y = [10,20,25,30] Step 1  
>>> fig = plt.figure() Step 2  
>>> ax = fig.add_subplot(111) Step 3  
>>> ax.plot(x, y, color='lightblue', linewidth=3) Step 3.4  
>>> ax.scatter([2,4,6],  
             [5,15,25],  
             color='darkgreen',  
             marker='^')  
>>> ax.set_xlim(1, 6.5)  
>>> plt.savefig('foo.png')  
>>> plt.show() Step 6
```

4 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"),)
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)  
>>> axes[1,1].quiver(y,z)  
>>> axes[0,1].streamplot(X,Y,U,V)
```

Mathtext

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

Limits, Legends & Layouts

```
>>> ax.margins(x=0.0,y=0.1)  
>>> ax.axis('equal')  
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])  
>>> ax.set_xlim(0,10.5)
```

Legends

```
>>> ax.set(title='An Example Axes',  
           ylabel='Y-Axis',  
           xlabel='X-Axis')  
>>> ax.legend(loc='best')
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),  
                  ticklabels=[3,100,-12,"foo"])  
>>> ax.tick_params(axis='y',  
                           direction='inout',  
                           length=10)
```

Subplot Spacing

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

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)  
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible
Move the bottom axis line outward

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()  
>>> plt.clf()  
>>> plt.close()
```

Clear an axis
Clear the entire figure
Close a window



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 to 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")
```

Inspect Data

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

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('phoneType',
               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
Convert df into an RDD
>>> df.toJSON().first()
Convert df into a RDD of string
>>> df.toPandas()
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()
```



Python For Data Science Cheat Sheet

Bokeh

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

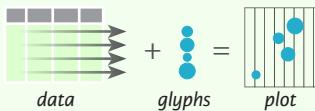


Plotting With Bokeh

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'])
```

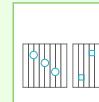
```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

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

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



Colormapping

```
>>> from bokeh.models import CategoricalColorMapper
>>> 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')
```

Legend Location

Inside Plot Area

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

Outside Plot Area

```
>>> from bokeh.models import Legend
>>> 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"
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

Columns

```
>>> from bokeh.layouts import column
>>> 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

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)
```

4) Output & Export

Notebook

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

HTML

Standalone HTML

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

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

Components

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

PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

5) Show or Save Your Plots

```
>>> show(p1)
>>> save(p1)
```

```
>>> show(layout)
>>> save(layout)
```



Python For Data Science Cheat Sheet

Pandas

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



Reshaping Data

Pivot

```
>>> df3 = df2.pivot(index='Date',  
                   columns='Type',  
                   values='Value')
```

Spread rows into columns

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Type	a	b	c
2016-03-01		11.432	NaN	20.784
2016-03-02		1.303	13.031	NaN
2016-03-03		99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,  
                       values='Value',  
                       index='Date',  
                       columns='Type')
```

Spread rows into columns

Stack / Unstack

```
>>> stacked = df5.stack()  
>>> stacked.unstack()
```

Pivot a level of column labels
Pivot a level of index labels

	0	1
1	0.233482	0.390959
2	0.184713	0.237102
3	0.433522	0.429401
Unstacked		

	5	0	2.233482
1	5	1	0.390959
2	4	0	0.184713
3	3	1	0.237102
4	2	0	0.433522
Stacked			

Melt

```
>>> pd.melt(df2,  
            id_vars=['Date'],  
            value_vars=['Type', 'Value'],  
            value_name='Observations')
```

Gather columns into rows

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

Iteration

```
>>> df.iteritems()  
>>> df.iterrows()
```

(Column-index, Series) pairs
(Row-index, Series) pairs

Advanced Indexing

Selecting

```
>>> df3.loc[:, (df3>1).any()]  
>>> df3.loc[:, (df3>1).all()]  
>>> df3.loc[:,df3.isnull().any()]  
>>> df3.loc[:,df3.notnull().all()]
```

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]  
>>> df.filter(items=["a","b"])  
>>> df.select(lambda x: not x%5)
```

Where

```
>>> s.where(s > 0)
```

Query

```
>>> df6.query('second > first')
```

Also see NumPy Arrays

Select cols with any vals >1
Select cols with vals >1
Select cols with NaN
Select cols without NaN

Find same elements
Filter on values
Select specific elements

Subset the data
Query DataFrame

Combining Data

X1	X2
a	11.432
b	1.303
c	99.906

X1	X3
a	20.784
b	NaN
d	20.784

Merge

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

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	20.784

Setting/Resetting Index

```
>>> df.set_index('Country')  
>>> df4 = df.reset_index()  
>>> df = df.rename(index=str,  
                   columns={"Country":"cntry",  
                             "Capital":"cptl",  
                             "Population":"ppltn"})
```

Set the index
Reset the index
Rename DataFrame

Reindexing

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

Forward Filling

```
>>> df.reindex(range(4),  
               method='ffill')
```

Backward Filling

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

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),  
             np.array([5,4,3])]  
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)  
>>> tuples = list(zip(*arrays))  
>>> index = pd.MultiIndex.from_tuples(tuples,  
                                         names=['first', 'second'])  
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)  
>>> df2.set_index(['Date', 'Type'])
```

Duplicate Data

```
>>> s3.unique()  
>>> df2.duplicated('Type')  
>>> df2.drop_duplicates('Type', keep='last')  
>>> df.index.duplicated()
```

Return unique values
Check duplicates
Drop duplicates
Check index duplicates

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date', 'Type']).mean()  
>>> df4.groupby(level=0).sum()  
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x),  
                           'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)  
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

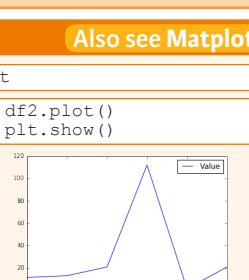
```
>>> df.dropna()  
>>> df3.fillna(df3.mean())  
>>> df2.replace("a", "f")
```

Drop NaN values
Fill NaN values with a predetermined value
Replace values with others

Visualization

```
>>> import matplotlib.pyplot as plt  
>>> s.plot()  
>>> plt.show()
```

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



DataCamp
Learn Python for Data Science Interactively!

Python For Data Science Cheat Sheet

Pandas Basics

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



Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.



Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type

a	3
b	-5
c	7
d	4

Index

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame

Index	Columns		
	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   >>>          'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
   >>>          'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
   >>>                      columns=['Country', 'Capital', 'Population'])
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')


#### Read multiple sheets from the same file


>>> xlsx = pd.ExcelFile('file.xlsx')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Getting

>>> s['b'] -5	Get one element
>>> df[1:] Country Capital Population 1 India New Delhi 1303171035 2 Brazil Brasilia 207847528	Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0], [0]]  
'Belgium'  
>>> df.iat[[0], [0]]  
'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']]  
'Belgium'  
>>> df.at[[0], ['Country']]  
'Belgium'
```

By Label/Position

```
>>> df.ix[2]  
Country Brazil  
Capital Brasilia  
Population 207847528
```

```
>>> df.ix[:, 'Capital']  
0 Brussels  
1 New Delhi  
2 Brasilia
```

```
>>> df.ix[1, 'Capital']  
'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)]  
>>> s[(s < -1) | (s > 2)]  
>>> df[df['Population'] > 1200000000]
```

Setting

```
>>> s['a'] = 6
```

Also see NumPy Arrays

Dropping

```
>>> s.drop(['a', 'c'])  
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)

Drop values from columns (axis=1)

Sort & Rank

```
>>> df.sort_index()  
>>> df.sort_values(by='Country')  
>>> df.rank()
```

Sort by labels along an axis

Sort by the values along an axis

Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape  
>>> df.index  
>>> df.columns  
>>> df.info()  
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()  
>>> df.cumsum()  
>>> df.min() / df.max()  
>>> df.idxmin() / df.idxmax()  
>>> df.describe()  
>>> df.mean()  
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*x2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a    10.0
b    NaN
c     5.0
d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c     5.0
d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```



Python For Data Science Cheat Sheet

Python Basics

Learn More Python for Data Science [Interactively](#) at www.datacamp.com



Variables and Data Types

Variable Assignment

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

Calculations With Variables

	Sum of two variables
>>> x+2 7	Subtraction of two variables
>>> x-2 3	Multiplication of two variables
>>> x*2 10	Exponentiation of a variable
>>> x**2 25	Remainder of a variable
>>> x%2 1	Division of a variable
>>> x/float(2) 2.5	

Types and Type Conversion

str()	'5', '3.45', 'True'	Variables to strings
int()	5, 3, 1	Variables to integers
float()	5.0, 1.0	Variables to floats
bool()	True, True, True	Variables to booleans

Asking For Help

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

Strings

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

String Operations

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

Lists

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

Selecting List Elements

Index starts at 0

Subset

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

Slice

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

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

Subset Lists of Lists

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

Select item at index 1
Select 3rd last item

Select items at index 1 and 2
Select items after index 0

Select items before index 3

Copy my_list

```
my_list[list][itemOfList]
```

List Operations

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

List Methods

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

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

String Operations

Index starts at 0

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

String Methods

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

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

Also see NumPy Arrays

Libraries

Import libraries

```
>>> import numpy  
>>> import numpy as np  
Selective import  
>>> from math import pi
```

pandas 
Data analysis

Machine learning

NumPy 
Scientific computing

matplotlib 
2D plotting

Install Python



ANACONDA

Leading open data science platform
powered by Python



Free IDE that is included
with Anaconda



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

Numpy Arrays

Also see Lists

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

Selecting Numpy Array Elements

Index starts at 0

Subset
>>> my_array[1]
2

Select item at index 1

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

Select items at index 0 and 1

Subset 2D Numpy arrays

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

my_2darray[rows, columns]

Numpy Array Operations

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

Numpy Array Functions

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



Python For Data Science Cheat Sheet

Also see NumPy

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]  
>>> np.ogrid[0:2,0:2]  
>>> np.r_[3,[0]*5,-1:1:10j]  
>>> np.c_[b,c]
```

Create a dense meshgrid
Create an open meshgrid
Stack arrays vertically (row-wise)
Create stacked column-wise arrays

Shape Manipulation

```
>>> np.transpose(b)  
>>> b.flatten()  
>>> np.hstack((b,c))  
>>> np.vstack((a,b))  
>>> np.hsplit(c,2)  
>>> np.vsplit(d,2)
```

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(c)  
>>> np.imag(c)  
>>> np.real_if_close(c,tol=1000)  
>>> np.cast['f'](np.pi)
```

Return the real part of the array elements
Return the imaginary part of the array elements
Return a real array if complex parts close to 0
Cast object to a data type

Other Useful Functions

```
>>> np.angle(b,deg=True)  
>>> g = np.linspace(0,np.pi,num=5)  
>>> g[3:] += np.pi  
>>> np.unwrap(g)  
>>> np.logspace(0,10,3)  
>>> np.select([c<4],[c*2])  
  
>>> misc.factorial(a)  
>>> misc.comb(10,3,exact=True)  
>>> misc.central_diff_weights(3)  
>>> misc.derivative(myfunc,1.0)
```

Return the angle of the complex argument
Create an array of evenly spaced values
(number of samples)
Unwrap
Create an array of evenly spaced values (log scale)
Return values from a list of arrays depending on conditions
Factorial
Combine N things taken at k time
Weights for N-point central derivative
Find the n-th derivative of a function at a point

Linear Algebra

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

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))  
>>> 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  
>>> linalg.inv(A)  
>>> A.T  
>>> A.H  
>>> np.trace(A)
```

Norm

```
>>> linalg.norm(A)  
>>> linalg.norm(A,1)  
>>> linalg.norm(A,np.inf)
```

Rank

```
>>> np.linalg.matrix_rank(C)
```

Determinant

```
>>> linalg.det(A)
```

Solving linear problems

```
>>> linalg.solve(A,b)  
>>> E = np.mat(a).T  
>>> linalg.lstsq(D,E)
```

Generalized inverse

```
>>> linalg.pinv(C)  
>>> linalg.pinv2(C)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)  
>>> G = np.mat(np.identity(2))  
>>> C[C > 0.5] = 0  
>>> H = sparse.csr_matrix(C)  
>>> I = sparse.csc_matrix(D)  
>>> J = sparse.dok_matrix(A)  
>>> E.todense()  
>>> sparse.isspmatrix_csc(A)
```

Inverse
Inverse
Transpose matrix
Conjugate transposition
Trace

Frobenius norm
L1 norm (max column sum)
L inf norm (max row sum)

Matrix rank

Determinant

Solver for dense matrices
Solver for dense matrices
Least-squares solution to linear matrix equation

Compute the pseudo-inverse of a matrix
(least-squares solver)
Compute the pseudo-inverse of a matrix
(SVD)

Create a 2x2 identity matrix
Create a 2x2 identity matrix

Compressed Sparse Row matrix
Compressed Sparse Column matrix
Dictionary Of Keys matrix
Sparse matrix to full matrix
Identify sparse matrix

Sparse Matrix Routines

Inverse

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

Norm

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

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Inverse

Norm

Solver for sparse matrices

Sparse Matrix Functions

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

Matrix Functions

Addition

```
>>> np.add(A,D)
```

Subtraction

```
>>> np.subtract(A,D)
```

Division

```
>>> np.divide(A,D)
```

Multiplication

```
>>> np.multiply(D,A)  
>>> np.dot(A,D)  
>>> np.vdot(A,D)  
>>> np.inner(A,D)  
>>> np.outer(A,D)  
>>> np.tensordot(A,D)  
>>> np.kron(A,D)
```

Exponential Functions

```
>>> linalg.expm(A)  
>>> linalg.expm2(A)  
>>> linalg.expm3(D)
```

Logarithm Function

```
>>> linalg.logm(A)
```

Trigonometric Functions

```
>>> linalg.sinm(D)  
>>> linalg.cosm(D)  
>>> linalg.tanm(A)
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)  
>>> linalg.coshm(D)  
>>> linalg.tanhm(A)
```

Matrix Sign Function

```
>>> np.signm(A)
```

Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Arbitrary Functions

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

Addition

Subtraction

Division

Multiplication
Dot product
Vector dot product
Inner product
Outer product
Tensor dot product
Kronecker product

Matrix exponential
Matrix exponential (Taylor Series)
Matrix exponential (eigenvalue decomposition)

Matrix logarithm

Matrix sine
Matrix cosine
Matrix tangent

Hypberbolic matrix sine
Hyperbolic matrix cosine
Hyperbolic matrix tangent

Matrix sign function

Matrix square root

Solve ordinary or generalized eigenvalue problem for square matrix
Unpack eigenvalues
First eigenvector
Second eigenvector
Unpack eigenvalues

Singular Value Decomposition (SVD)
Construct sigma matrix in SVD

LU Decomposition

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)  
  
>>> l1, l2 = la  
>>> v[:,0]  
>>> v[:,1]  
>>> linalg.eigvals(A)
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)  
>>> M,N = B.shape  
>>> Sig = linalg.diagsvd(s,M,N)
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C)
```

Sparse Matrix Decompositions

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

Eigenvalues and eigenvectors
SVD

Asking For Help

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

DataCamp

Learn Python for Data Science [Interactively](#)



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', '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)
```

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)
```

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)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit the model to the data
Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels
Predict labels
Estimate probability of a label
Predict labels in clustering algos

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

Estimator score method

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)
```

R² Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

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=kn,
...                                param_distributions=params,
...                                cv=4,
...                                n_iter=8,
...                                random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```



Python For Data Science Cheat Sheet

Seaborn

Learn Data Science interactively at www.DataCamp.com



Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on `matplotlib` and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt  
>>> import seaborn as sns  
>>> tips = sns.load_dataset("tips")  
>>> sns.set_style("whitegrid")  
>>> g = sns.lmplot(x="tip",  
y="total_bill",  
data=tips,  
aspect=2)  
>>> g.set_axis_labels("Tip", "Total bill(USD)")  
set(xlim=(0,10), ylim=(0,100))  
>>> plt.title("title")  
>>> plt.show(g)
```

Step 1
Step 2
Step 3
Step 4
Step 5

1) Data

Also see [Lists, NumPy & Pandas](#)

```
>>> import pandas as pd  
>>> import numpy as np  
>>> uniform_data = np.random.rand(10, 12)  
>>> data = pd.DataFrame({'x':np.arange(1,101),  
y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")  
>>> iris = sns.load_dataset("iris")
```

2) Figure Aesthetics

Seaborn styles

```
>>> sns.set()  
>>> sns.set_style("whitegrid")  
>>> sns.set_style("ticks",  
{"xtick.major.size":8,  
"ytick.major.size":8})  
>>> sns.axes_style("whitegrid")
```

(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters
Return a dict of params or use with
with to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")  
>>> sns.set_context("notebook",  
font_scale=1.5,  
rc={"lines.linewidth":2.5})
```

Color Palette

```
>>> sns.set_palette("husl",3)  
>>> sns.color_palette("husl")  
>>> flatui = ["#9b59b6","#3498db","#95a5e6","#e74c3c","#34495e","#2ecc71"]  
>>> sns.set_palette(flatui)
```

3) Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,  
col="survived",  
row="sex")  
>>> g.map(plt.hist,"age")  
>>> sns.factorplot(x="pclass",  
y="survived",  
hue="sex",  
data=titanic)  
>>> sns.lmplot(x="sepal_width",  
y="sepal_length",  
hue="species",  
data=iris)
```

Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)  
>>> h = h.map(plt.scatter)  
>>> sns.pairplot(iris)  
>>> i = sns.JointGrid(x="x",  
y="y",  
data=data)  
>>> i = i.plot(sns.regplot,  
sns.distplot)  
>>> sns.jointplot("sepal_length",  
"sepal_width",  
data=iris,  
kind='kde')
```

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Categorical Plots

Scatterplot
`>>> sns.stripplot(x="species",
y="petal_length",
data=iris)
>>> sns.swarmplot(x="species",
y="petal_length",
data=iris)`

Bar Chart

```
>>> sns.barplot(x="sex",  
y="survived",  
hue="class",  
data=titanic)
```

Count Plot

```
>>> sns.countplot(x="deck",  
data=titanic,  
palette="Greens_d")
```

Point Plot

```
>>> sns.pointplot(x="class",  
y="survived",  
hue="sex",  
data=titanic,  
palette={"male":"g",  
"female":"m"},  
markers=["^","o"],  
linestyles=["-","--"])
```

Boxplot

```
>>> sns.boxplot(x="alive",  
y="age",  
hue="adult_male",  
data=titanic)
```

Violinplot

```
>>> sns.violinplot(x="age",  
y="sex",  
hue="survived",  
data=titanic)
```

Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

Show point estimates and confidence intervals with scatterplot glyphs

Show count of observations

Show point estimates and confidence intervals as rectangular bars

Boxplot

Boxplot with wide-form data

Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",  
y="sepal_length",  
data=iris,  
ax=ax)
```

Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,  
kde=False,  
color="b")
```

Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```

Heatmap

4) Further Customizations

Also see [Matplotlib](#)

Axisgrid Objects

```
>>> g.despine(left=True)  
>>> g.set_ylabels("Survived")  
>>> g.set_xticklabels(rotation=45)  
>>> g.set_axis_labels("Survived",  
"Sex")  
>>> h.set(xlim=(0,5),  
ylim=(0,5),  
xticks=[0,2.5,5],  
yticks=[0,2.5,5])
```

Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")  
>>> plt.ylabel("Survived")  
>>> plt.xlabel("Sex")  
>>> plt.ylim(0,100)  
>>> plt.xlim(0,10)  
>>> plt.setp(ax,yticks=[0,5])  
>>> plt.tight_layout()
```

Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5) Show or Save Plot

Also see [Matplotlib](#)

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

Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

```
>>> plt.cla()  
>>> plt.clf()  
>>> plt.close()
```

Clear an axis
Clear an entire figure
Close a window



Python for Data Science Cheat Sheet spaCy

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



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'
```

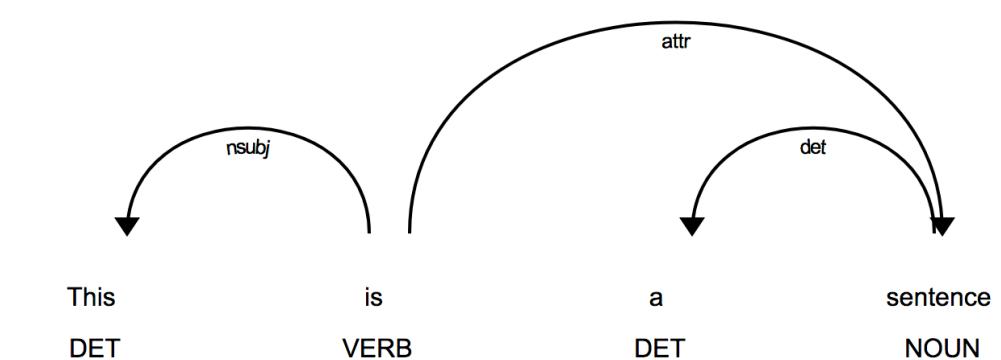
Visualizing

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

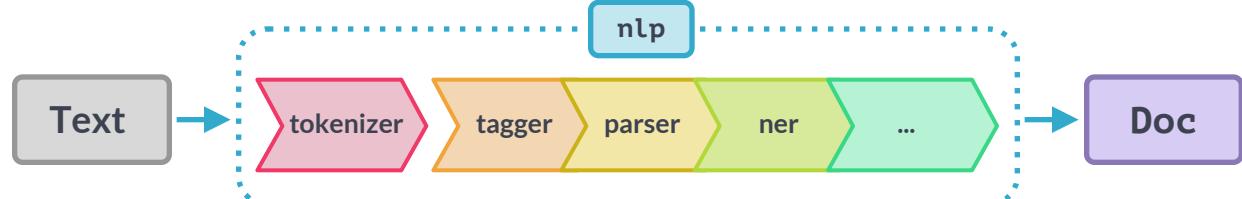
```
doc1 = nlp("I like cats")
doc2 = nlp("I like dogs")
# Compare 2 documents
doc1.similarity(doc2)
# Compare 2 tokens
doc1[2].similarity(doc2[2])
# Compare tokens and spans
doc1[0].similarity(doc2[1:3])
```

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.

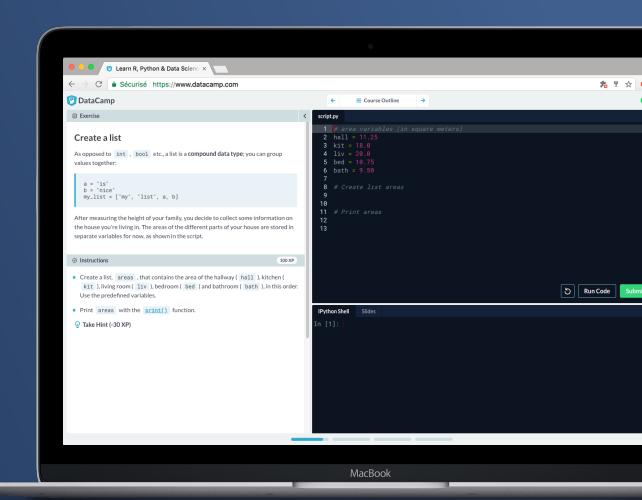
- ! Negate pattern and match exactly 0 times.
- ? Make pattern optional and match 0 or 1 times.
- + Require pattern to match 1 or more times.
- * Allow pattern to match 0 or more times.

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.



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



R For Data Science Cheat Sheet

Tidyverse for Beginners

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



Tidyverse

The **tidyverse** is a powerful collection of R packages that are actually data tools for transforming and visualizing data. All packages of the tidyverse share an underlying philosophy and common APIs.

The core packages are:



- **ggplot2**, which implements the grammar of graphics. You can use it to visualize your data.



- **dplyr** is a grammar of data manipulation. You can use it to solve the most common data manipulation challenges.



- **tidyr** helps you to create tidy data or data where each variable is in a column, each observation is a row and each value is a cell.



- **readr** is a fast and friendly way to read rectangular data.



- **purrr** enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors.



- **tibble** is a modern re-imaging of the data frame.



- **stringr** provides a cohesive set of functions designed to make working with strings as easy as possible



- **forcats** provide a suite of useful tools that solve common problems with factors.

You can install the complete tidyverse with:

```
> install.packages("tidyverse")
```

Then, load the core tidyverse and make it available in your current R session by running:

```
> library(tidyverse)
```

Note: there are many other tidyverse packages with more specialised usage. They are not loaded automatically with `library(tidyverse)`, so you'll need to load each one with its own call to `library()`.

Useful Functions

```
> tidyverse_conflicts()
```

Conflicts between tidyverse and other packages

```
> tidyverse_deps()
```

List all tidyverse dependencies

```
> tidyverse_logo()
```

Get tidyverse logo, using ASCII or unicode characters

```
> tidyverse_packages()
```

List all tidyverse packages

```
> tidyverse_update()
```

Update tidyverse packages

Loading in the data

```
> library(datasets)
```

Load the datasets package

```
> library(gapminder)
```

Load the gapminder package

```
> attach(iris)
```

Attach iris data to the R search path

dplyr

Filter

`filter()` allows you to select a subset of rows in a data frame.

```
> iris %>%  
  filter(Species=="virginica")  
> iris %>%  
  filter(Species=="virginica",  
         Sepal.Length > 6)
```

Select iris data of species "virginica"
Select iris data of species "virginica" and sepal length greater than 6.

Arrange

`arrange()` sorts the observations in a dataset in ascending or descending order based on one of its variables.

```
> iris %>%  
  arrange(Sepal.Length)  
> iris %>%  
  arrange(desc(Sepal.Length))
```

Sort in ascending order of sepal length
Sort in descending order of sepal length

Combine multiple dplyr verbs in a row with the pipe operator `%>%`:

```
> iris %>%  
  filter(Species=="virginica") %>%  
  arrange(desc(Sepal.Length))
```

Filter for species "virginica" then arrange in descending order of sepal length

Mutate

`mutate()` allows you to update or create new columns of a data frame.

```
> iris %>%  
  mutate(Sepal.Length=Sepal.Length*10)  
> iris %>%  
  mutate(SLMm=Sepal.Length*10)
```

Change Sepal.Length to be in millimeters
Create a new column called SLMm

Combine the verbs `filter()`, `arrange()`, and `mutate()`:

```
> iris %>%  
  filter(Species=="Virginica") %>%  
  mutate(SLMm=Sepal.Length*10) %>%  
  arrange(desc(SLMm))
```

Summarize

`summarize()` allows you to turn many observations into a single data point.

```
> iris %>%  
  summarize(medianSL=median(Sepal.Length))  
> iris %>%  
  filter(Species=="virginica") %>%  
  summarize(medianSL=median(Sepal.Length))
```

Summarize to find the median sepal length
Filter for virginica then summarize the median sepal length

You can also summarize multiple variables at once:

```
> iris %>%  
  filter(Species=="virginica") %>%  
  summarize(medianSL=median(Sepal.Length),  
            maxSL=max(Sepal.Length))
```

`group_by()` allows you to summarize within groups instead of summarizing the entire dataset:

```
> iris %>%  
  group_by(Species) %>%  
  summarize(medianSL=median(Sepal.Length),  
            maxSL=max(Sepal.Length))  
> iris %>%  
  filter(Sepal.Length>6) %>%  
  group_by(Species) %>%  
  summarize(medianPL=median(Petal.Length),  
            maxPL=max(Petal.Length))
```

Find median and max sepal length of each species
Find median and max petal length of each species with sepal length > 6

ggplot2

Scatter plot

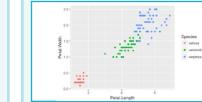
Scatter plots allow you to compare two variables within your data. To do this with ggplot2, you use `geom_point()`

```
> iris_small <- iris %>%  
  filter(Sepal.Length > 5)  
> ggplot(iris_small, aes(x=Petal.Length,  
                         y=Petal.Width)) +  
  geom_point()
```

Compare petal width and length

Additional Aesthetics

• Color



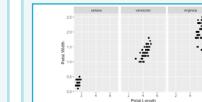
```
> ggplot(iris_small, aes(x=Petal.Length,  
                         y=Petal.Width,  
                         color=Species)) +  
  geom_point()
```

• Size



```
> ggplot(iris_small, aes(x=Petal.Length,  
                         y=Petal.Width,  
                         color=Species,  
                         size=Sepal.Length)) +  
  geom_point()
```

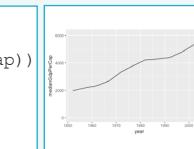
Faceting



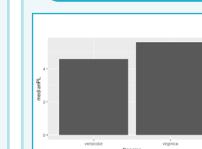
```
> ggplot(iris_small, aes(x=Petal.Length,  
                         y=Petal.Width)) +  
  geom_point() +  
  facet_wrap(~Species)
```

Line Plots

```
> by_year <- gapminder %>%  
  group_by(year) %>%  
  summarize(medianGdpPerCap=median(gdpPerCap))  
> ggplot(by_year, aes(x=year,  
                         y=medianGdpPerCap)) +  
  geom_line() +  
  expand_limits(y=0)
```



Bar Plots



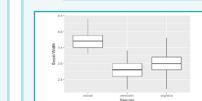
```
> by_species <- iris %>%  
  filter(Sepal.Length>6) %>%  
  group_by(Species) %>%  
  summarize(medianPL=median(Petal.Length))  
> ggplot(by_species, aes(x=Species,  
                         y=medianPL)) +  
  geom_col()
```

Histograms

```
> ggplot(iris_small, aes(x=Petal.Length)) +  
  geom_histogram()
```



Box Plots



```
> ggplot(iris_small, aes(x=Species,  
                         y=Sepal.Width)) +  
  geom_boxplot()
```

