

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Essential Cheat Sheets for Machine Learning and Deep Learning Engineers



Kailash Ahirwar · [Follow](#)

Published in Startups & Venture Capital · 4 min read · May 28, 2017

25K

107



Machine learning is complex. For newbies, starting to learn machine learning can be painful if they don't have right resources to learn from. Most of the machine learning libraries are difficult to understand and learning curve can be a bit frustrating. I am creating a repository on [Github\(cheatsheets-ai\)](#) containing cheatsheets for different machine learning frameworks, gathered from different sources. Do visit the Github repository, also, contribute cheat sheets if you have any. Thanks.

List of Cheatsheets:

1. Keras
2. Numpy
3. Pandas
4. Scipy
5. Matplotlib
6. Scikit-learn

7. Neural Networks Zoo

8. ggplot2

9. PySpark

10. R Studio

11. Jupyter Notebook

12. Dask

1. Keras

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.randint(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_train,X_test,y_train,y_test = train_test_split(x,
    y,
    test_size=0.33,
    random_state=42)
```

Standardization/Normalization

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

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])
```

2. Numpy

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:

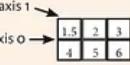
```
>>> import numpy as np
```



NumPy Arrays

1D array

1	2	3
4	5	6



axis 0

axis 1

axis 0

Data Wrangling

with pandas

Cheat Sheet

<http://pandas.pydata.org>

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.
```

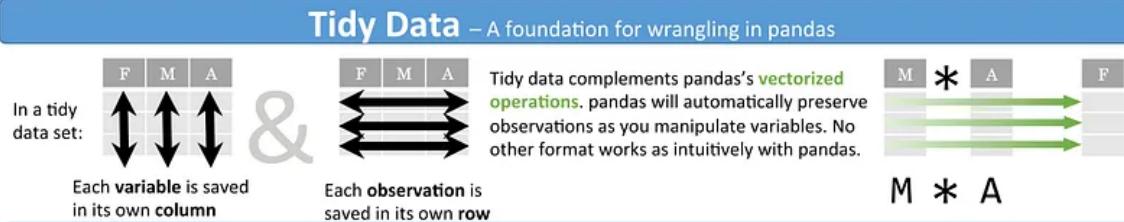
	n	v	a	b	c
d	1	4	7	10	
d	2	5	8	11	
e	2	6	9	12	

```
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(['Length','Height'], axis=1)
Drop columns from DataFrame
```

Subset Observations (Rows)



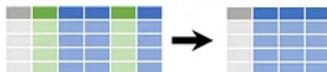
```
df[df.Length > 7]
Extract rows that meet logical criteria.

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

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.
```

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.
```

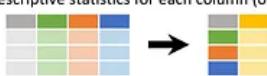
regex (Regular Expressions) Examples	
'.'	Matches strings containing a period '.'
'Length\$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]\$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species\$)/*'	Matches strings except the string 'Species'

Logic in Python (and pandas)		
<	Less than	!=
>	Greater than	df.column.isin(values)
==	Equals	pd.isnull(obj)
<=	Less than or equals	pd.notnull(obj)
>=	Greater than or equals	&, , ~, ^, df.any(), df.all()

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

Summarize Data

```
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:

<code>sum()</code>	Sum values of each object.
<code>count()</code>	Count non-NA/null values of each object.
<code>median()</code>	Median value of each object.
<code>quantile([0.25, 0.75])</code>	Quantiles of each object.
<code>apply(function)</code>	Apply function to each object.
<code>min()</code>	Minimum value in each object.
<code>max()</code>	Maximum value in each object.
<code>mean()</code>	Mean value of each object.
<code>var()</code>	Variance of each object.
<code>std()</code>	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:

<code>size()</code>	Size of each group.
<code>agg(function)</code>	Aggregate group using function.

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.
```

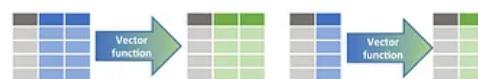
Handling Missing Data

```
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:

<code>max(axis=1)</code>	<code>min(axis=1)</code>
Element-wise max.	Element-wise min.
<code>clip(lower=-10,upper=10)</code>	<code>abs()</code>
Trim values at input thresholds	Absolute value.

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

<code>shift(1)</code>	<code>shift(-1)</code>
Copy with values shifted by 1.	Copy with values lagged by 1.
<code>rank(method='dense')</code>	<code>cumsum()</code>
Ranks with no gaps.	Cumulative sum.
<code>rank(method='min')</code>	<code>cummax()</code>
Ranks. Ties get min rank.	Cumulative max.
<code>rank(pct=True)</code>	<code>cummin()</code>
Ranks rescaled to interval [0, 1].	Cumulative min.
<code>rank(method='first')</code>	<code>cumprod()</code>
Ranks. Ties go to first value.	Cumulative product.

Plotting

<code>df.plot.hist()</code>	Histogram for each column
<code>df.plot.scatter(x='w',y='h')</code>	Scatter chart using pairs of points

Combine Data Sets

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



Standard Joins

<code>x1 x2 x3</code>	<code>pd.merge(adf, bdf,</code>
A 1 T	how='left', on='x1')
B 2 F	Join matching rows from bdf to adf.
C 3 NaN	

<code>x1 x2 x3</code>	<code>pd.merge(adf, bdf,</code>
A 1.0 T	how='right', on='x1')
B 2.0 F	Join matching rows from adf to bdf.
D NaN T	

<code>x1 x2 x3</code>	<code>pd.merge(adf, bdf,</code>
A 1 T	how='inner', on='x1')
B 2 F	Join data. Retain only rows in both sets.
C 3 NaN	

<code>x1 x2 x3</code>	<code>pd.merge(adf, bdf,</code>
A 1 T	how='outer', on='x1')
B 2 F	Join data. Retain all values, all rows.
C 3 NaN	

Filtering Joins

<code>x1 x2</code>	<code>adf[adf.x1.isin(bdf.x1)]</code>
A 1	All rows in adf that have a match in bdf.
B 2	

<code>x1 x2</code>	<code>adf[~adf.x1.isin(bdf.x1)]</code>
C 3	All rows in adf that do not have a match in bdf.

ydf	zdf
x1 x2	x1 x2
A 1	B 2
B 2	C 3
C 3	D 4



Set-like Operations

<code>x1 x2</code>	<code>pd.merge(ydf, zdf)</code>
B 2	Rows that appear in both ydf and zdf (Intersection).
C 3	

<code>x1 x2</code>	<code>pd.merge(ydf, zdf, how='outer')</code>
A 1	Rows that appear in either or both ydf and zdf (Union).
B 2	
C 3	
D 4	

<code>x1 x2</code>	<code>pd.merge(ydf, zdf, how='outer', indicator=True)</code>
----------------------	--------------------------------------------------------------

<code>A 1</code>	.query('_merge == "left_only"')
------------------	---------------------------------

<code>A 1</code>	.drop(['_merge'], axis=1)
------------------	---------------------------

<code>A 1</code>	Rows that appear in ydf but not zdf (Setdiff).
------------------	------------------------------------------------

Source — <https://www.datacamp.com/community/blog/pandas-cheat-sheet-python#gs.HPFoRlc>

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

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



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

DataFrame

Columns

	Country	Capital	Population
1	Belgium	Brussels	11190846
2	India	New Delhi	1303171035
3	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)
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

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

Read multiple sheets from the same file

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

Asking For Help

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

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
-> 5
>>> df[1:]
   Country    Capital  Population
1  India      New Delhi     1303171035
2  Brazil     Brasilia     207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

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

Select single value by row & column

By Label

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

Select single value by row & column labels

By Label/Position

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

Select single row of subset of rows

```
>>> df.ix[:, 'Capital']
```

Select a single column of subset of columns

```
0  Brussels
1  New Delhi
2  Brasilia
```

Select rows and columns

Boolean Indexing

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

Series s where value is not >1
s where value is <-1 or >2
Use filter to adjust DataFrame

Setting

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

Set index a of Series s to 6

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(by='Country')
>>> s.order()
>>> df.rank()
```

Sort by row or column index
Sort a series by its values
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
Cumulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x**2
>>> 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)
   a    8.0
   b    -7.0
   c    3.0
   d    5.0
>>> s.div(s3, fill_value=4)
   a    2.0
   b    -0.5
   c    0.75
   d    1.75
>>> s.mul(s3, fill_value=3)
   a    21.0
   b    -6.0
   c    9.0
   d    21.0
```

DataCamp

Learn Python for Data Science [Interactively](#)



Source — <https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#gs.oundfxM>

4. Scipy

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([[1,2,3], [4,5,6], [3,2,1], [4,5,6]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]  
>>> a = np.array([1,2,3])  
>>> np.ogrid[0:2,0:2]  
>>> np.e_[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)
```

Permute array dimensions
Flatten the array
Stack arrays horizontally (column-wise)
Stack arrays vertically (row-wise)
Split the array horizontally at the 2nd index
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)  
>>> np.imag(b)  
>>> 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
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`.

[Also see NumPy](#)

Matrix Functions

Addition

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

Addition

Subtraction

Subtraction

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

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 product

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

Kronecker product

Exponential Functions

```
>>> linalg.exp(A)
```

Matrix exponential

```
>>> linalg.expm2(A)
```

Matrix exponential (Taylor Series)

```
>>> linalg.expm3(D)
```

Matrix logarithm

Logarithm Function

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

Trigonometric Functions

```
>>> linalg.sinm(D)
```

Matrix sine

```
>>> linalg.cosm(D)
```

Matrix cosine

```
>>> linalg.tanm(A)
```

Matrix tangent

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(B)
```

Hyperbolic matrix sine

```
>>> linalg.coshm(B)
```

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
```

Construct sigma matrix in SVD

```
>>> Sig = linalg.diagsvd(s,M,N)
```

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

DataCamp
Learn Python for Data Science [interactively](#)



Source — <https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs.JDSg3O1>

5. Matplotlib

Python For Data Science Cheat Sheet

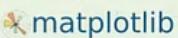
Matplotlib

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]  
>>> V = 1 + X**2 - 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.hilf(x,y,color='blue')  
>>> ax.hilf_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them
Draw unconnected points, scaled or colored
Plot vertical rectangles (constant width)
Plot horizontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and 0

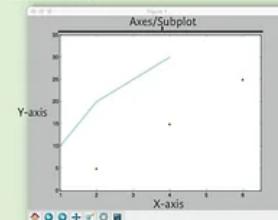
2D Data or Images

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

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(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(arrowsize=15,  
 connectionstyle="arc3"),)
```

Vector Fields

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

Data Distributions

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

MathText

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

Limits, Legends & Layouts

```
>>> ax.margins(x=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=[3,10,0,-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

Source — <https://www.datacamp.com/community/blog/python-matplotlib-cheat-sheet#gs.uEKySpY>

6. Scikit-learn

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.cross_validation 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', 'M', 'F', 'F', 'F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.cross_validation 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_train = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

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

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.5).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')
```

Naïve 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))
```

Confusion matrix

and support

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
```

```
>>> y_true = [3, -0.5, 2]
```

```
>>> mean_absolute_error(y_true, y_pred)
```

Mean absolute error

Mean squared error

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
```

```
>>> mean_squared_error(y_test, y_pred)
```

Mean squared error

R² Score

```
>>> from sklearn.metrics import r2_score
```

```
>>> r2_score(y_true, y_pred)
```

R² score

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
```

```
>>> adjusted_rand_score(y_true, y_pred)
```

Adjusted rand index

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
```

```
>>> homogeneity_score(y_true, y_pred)
```

Homogeneity

V-measure

```
>>> from sklearn.metrics import v_measure_score
```

```
>>> metrics.v_measure_score(y_true, y_pred)
```

V-measure

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))
```

Cross-validation

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)
```

Grid search

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_)
```

DataCamp

Learn Python for Data Science interactively

7. Neural Networks Zoo

A mostly complete chart of
Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

○ Backfed Input Cell

○ Input Cell

△ Noisy Input Cell

● Hidden Cell

○ Probabilistic Hidden Cell

△ Spiking Hidden Cell

● Output Cell

○ Match Input Output Cell

● Recurrent Cell

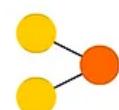
○ Memory Cell

△ Different Memory Cell

● Kernel

○ Convolution or Pool

Perceptron (P)



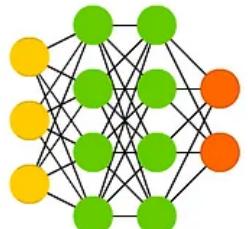
Feed Forward (FF)



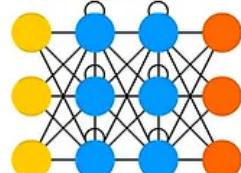
Radial Basis Network (RBF)



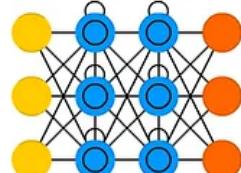
Deep Feed Forward (DFF)



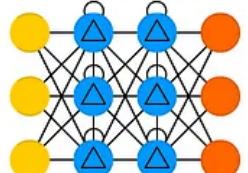
Recurrent Neural Network (RNN)



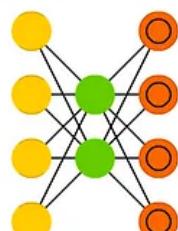
Long / Short Term Memory (LSTM)



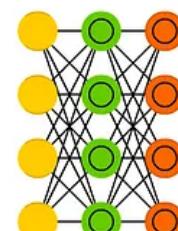
Gated Recurrent Unit (GRU)



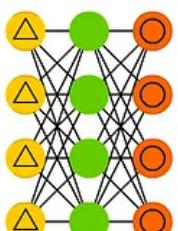
Auto Encoder (AE)



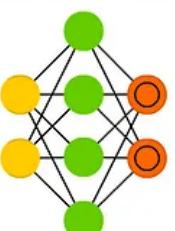
Variational AE (VAE)



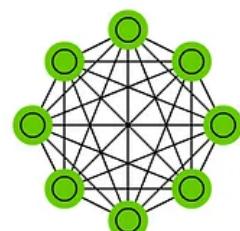
Denoising AE (DAE)



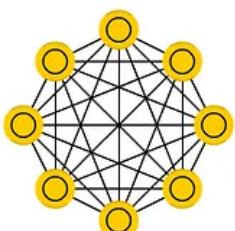
Sparse AE (SAE)



Markov Chain (MC)



Hopfield Network (HN)



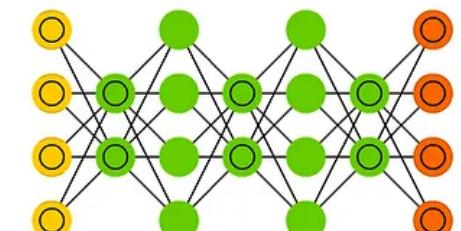
Boltzmann Machine (BM)



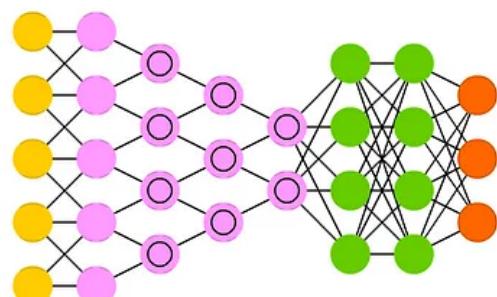
Restricted BM (RBM)



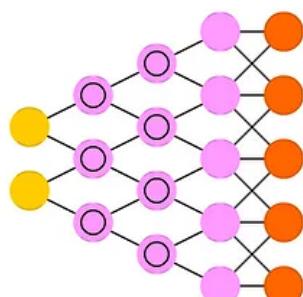
Deep Belief Network (DBN)



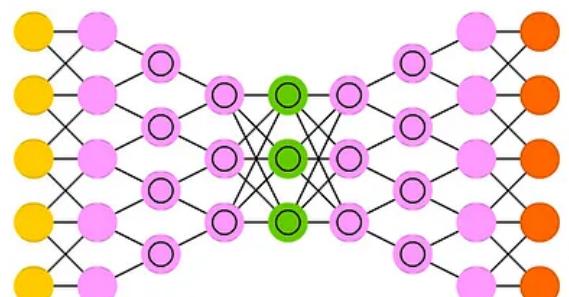
Deep Convolutional Network (DCN)



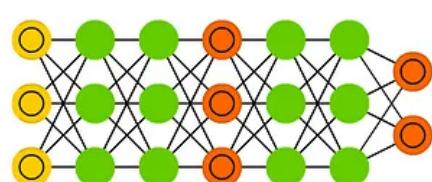
Deconvolutional Network (DN)



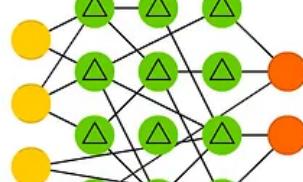
Deep Convolutional Inverse Graphics Network (DCIGN)



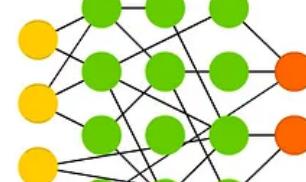
Generative Adversarial Network (GAN)



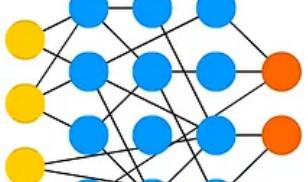
Liquid State Machine (LSM)



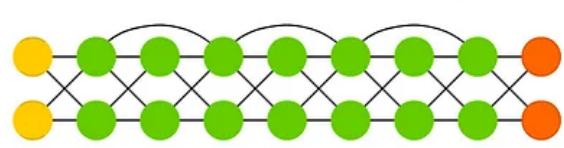
Extreme Learning Machine (ELM)



Echo State Network (ESN)



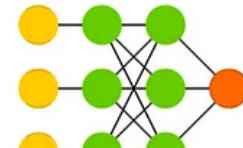
Deep Residual Network (DRN)



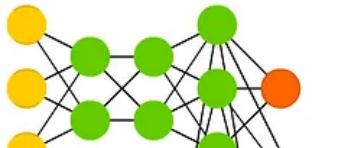
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)





Source — <http://www.asimovinstitute.org/neural-network-zoo/>

8. ggplot2

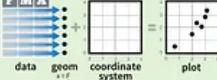
Data Visualization with ggplot2

Cheat Sheet

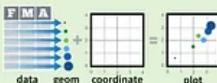


Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data set**, a set of **geoms**—visual marks that represent data points, and a **coordinate system**.



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings **data** **geom**
qplot(x = cyl, y = hwy, color = cyl, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, aes(x = cyl, y = hwy))

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data **add layers, elements with +** **additional elements**
ggplot(mpg, aes(hwy, cyl)) + geom_point(aes(color = cyl)) + geom_smooth(method = "lm") + coord_cartesian() + scale_color_gradient() + theme_bw()

Add a new layer to a plot with a **geom_*** or **stat_*** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5'x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.	
<h4>One Variable</h4> <p>Continuous</p> <pre>a <- ggplot(mpg, aes(hwy)) a + geom_area(stat = "bin") x, y, alpha, color, fill, linetype, size b + geom_area(aes(y = ..density..), stat = "bin") a + geom_density(kernel = "gaussian") x, y, alpha, color, fill, linetype, size, weight b + geom_density(aes(y = ..count..)) a + geom_dotplot() x, y, alpha, color, fill</pre> <p>Discrete</p> <pre>b <- ggplot(mpg, aes(fct_lubridate::month)) b + geom_bar()</pre>	<h4>Two Variables</h4> <p>Continuous X, Continuous Y</p> <pre>f <- ggplot(mpg, aes(cty, hwy)) f + geom_blank()</pre> <p>Continuous Bivariate Distribution</p> <pre>i <- ggplot(movies, aes(year, rating)) i + geom_bin2d(binwidth = c(5, 0.5)) xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight i + geom_density2d() x, y, alpha, colour, linetype, size i + geom_hex() x, y, alpha, colour, fill, size</pre> <p>Continuous Function</p> <pre>j <- ggplot(economics, aes(date, unemploy)) j + geom_area()</pre>
<p>Graphical Primitives</p> <pre>c <- ggplot(map, aes(long, lat)) c + geom_polygon(aes(group = group)) x, y, alpha, color, fill, linetype, size</pre> <p>Discrete X, Continuous Y</p> <pre>g <- ggplot(mpg, aes(class, hwy)) g + geom_bar(stat = "identity")</pre>	<p>Discrete X, Continuous Y</p> <pre>g + geom_boxplot() lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight g + geom_dotplot(binaxis = "y", stackdir = "center") x, y, alpha, color, fill g + geom_violin(scale = "area")</pre> <p>Discrete X, Discrete Y</p> <pre>h <- ggplot(diamonds, aes(cut, color)) h + geom_jitter()</pre>
<p>Maps</p> <pre>seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)) m <- ggplot(seals, aes(long, lat)) m + geom_raster(aes(fill = z), hijst=0.5, vjust=0.5, interpolate=FALSE) x, y, alpha, fill m + geom_tile(aes(fill = z)) x, y, alpha, color, fill, linetype, size</pre>	<p>Three Variables</p> <pre>m + geom_raster(aes(fill = z), hijst=0.5, vjust=0.5, interpolate=FALSE) x, y, alpha, fill m + geom_tile(aes(fill = z)) x, y, alpha, color, fill, linetype, size</pre>

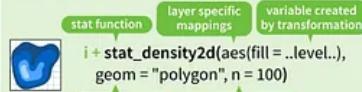
Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. `a + geom_bar(stat = "bin")`



Each stat creates additional variables to map aesthetics to. These variables use a common `..name..` syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. `stat_bin(geom="bar")` does the same as `geom_bar(stat="bin")`



`a + stat_bin(binwidth = 1, origin = 10)` 1D distributions

`x, y | ..count, ..ncount, ..density, ..ndensity..`

`a + stat_bindot(binwidth = 1, binaxis = "x")`

`x, y | ..count, ..ncount..`

`a + stat_density(adjust = 1, kernel = "gaussian")`

`x, y | ..count, ..density, ..scaled..`

`f + stat_bin2d(bins = 30, drop = TRUE)` 2D distributions

`x, y, fill | ..count, ..density..`

`f + stat_binnedx(bins = 30)`

`x, y, fill | ..count, ..density..`

`f + stat_density2d(contour = TRUE, n = 100)`

`x, y, color, size | ..level..`

`m + stat_contour(aes(z = z))` 3 Variables

`x, y, z, order | ..level..`

`m + stat_spoke(aes(radius = z, angle = z))`

`angle, radius, x, xend, y, yend | ..x, ..yend, ..y..`

`m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)`

`x, y, z, fill | ..value..`

`m + stat_summary2d(aes(z = z), bins = 30, fun = mean)`

`x, y, z, fill | ..value..`

`g + stat_boxplot(coef = 1.5)` Comparisons

`x, y | ..lower, ..middle, ..upper, ..outliers..`

`g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")`

`x, y | ..density, ..scaled, ..count, ..n, ..violinwidth, ..width..`

`f + stat_ecdf(n = 40)` Functions

`x, y | ..x, ..y..`

`f + stat_quantile(quartiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")`

`x, y | ..quartile, ..x, ..y..`

`f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)`

`x, y | ..se, ..x, ..y, ..ymin, ..ymax..`

`ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd = 0.5))` General Purpose

`x | ..y..`

`f + stat_identity()`

`ggplot() + stat_qq(aes(sample = 1:100), distribution = qt, dparams = list(df = 5))`

`sample, x, y | ..x, ..y..`

`f + stat_sum()`

`x, y, size | ..size..`

`f + stat_summary(fun.data = "mean_cl_boot")`

`f + stat_unique()`

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



General Purpose scales

Use with any aesthetic: alpha, color, fill, linetype, shape, size

`scale_*_continuous()` - map cont values to visual values

`scale_*_discrete()` - map discrete values to visual values

`scale_*_identity()` - use data values as visual values

`scale_*_manual(values = c(), name = "fuel", labels = c("D", "E", "P", "R"))`

range of values to include in mapping

title to use in legend/axis

labels to use in legend/axis

breaks to use in legend/axis

X and Y location scales

Use with x or y aesthetics (x shown here)

`scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks"))` - treat x values as dates. See ?strptime for label formats.

`scale_x_datetime()` - treat x values as dates. Use same arguments as `scale_x_date()`.

`scale_x_log10()` - Plot x on log10 scale

`scale_x_reverse()` - Reverse direction of x axis

`scale_x_sqrt()` - Plot x on square root scale

Color and fill scales

Discrete vs Continuous

`l + <- b + geom_bar(aes(fill = fl))`

`n + scale_fill_brewer(palette = "Blues")`

For palette choices: library(RColorBrewer); display.brewer.all()

`n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")`

Continuous

`o + <- a + geom_dotplot(aes(fill = ..x..))`

`o + scale_fill_gradient(low = "red", high = "yellow")`

`o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)`

`o + scale_fill_gradientn(colours = terrain.colors(6))`

Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape scales

Manual shape values

`o + <- f + geom_point(aes(shape = fl))`

`f + scale_shape(solid = FALSE)`

`f + scale_shape_manual(values = c(3:7))`

Shape values shown in chart on right

Size scales

`q + <- f + geom_point(aes(size = cyl))`

`q + scale_size_area(max = 6)`

Value mapped to area of circle (not radius)

Coordinate Systems

`r + coord_cartesian(xlim = c(0, 5))`

`xlim, ylim`

The default cartesian coordinate system

`r + coord_fixed(ratio = 1/2)`

ratio, xlim, ylim

Cartesian coordinates with fixed aspect ratio between x and y units

`r + coord_flip()`

`xlim, ylim`

Flipped Cartesian coordinates

`r + coord_polar(theta = "x", direction = 1)`

theta, start, direction

Polar coordinates

`r + coord_trans(ytrans = "sqrt")`

`ytrans, ylim, xlim, limy`

Transformed cartesian coordinates. Set extras and strains to the name of a window function.

`z + coord_map(projection = "ortho", orientation = c(41, -74, 0))`

`projection, orientation, xlim, ylim`

Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

`s <- ggplot(mpg, aes(fl, fill = drv))`

`s + geom_bar(position = "dodge")`

Arrange elements side by side

`s + geom_bar(position = "fill")`

Stack elements on top of one another, normalize height

`s + geom_bar(position = "stack")`

Stack elements on top of one another

`f + geom_point(position = "jitter")`

Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual width and height arguments

`s + geom_bar(position = position_dodge(width = 1))`

Themes

`r + theme_bw()`

White background with grid lines

`r + theme_classic()`

White background no gridlines

`r + theme_minimal()`

Minimal theme

`ggthemes` - Package with additional ggplot2 themes

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

`t <- ggplot(mpg, aes(cty, hwy)) + geom_point()`

`t + facet_grid(~ fl)`

facet into columns based on fl

`t + facet_grid(year ~ .)`

facet into rows based on year

`t + facet_grid(year ~ fl)`

facet into both rows and columns

`t + facet_wrap(~ fl)`

wrap facets into a rectangular layout

Set scales to let axis limits vary across facets

`t + facet_grid(~ x, scales = "free")`

x and y axis limits adjust to individual facets

- "free_x" - x axis limits adjust
- "free_y" - y axis limits adjust

Set labeller to adjust facet labels

`t + facet_grid(~ fl, labeller = label_both)`

fl: c fl: d fl: e fl: p fl: r

`t + facet_grid(~ fl, labeller = label_bquote(alpha ^ ..x..))`

projection, orientation, xlim, ylim

`t + facet_grid(~ fl, labeller = label_parsed)`

c d e p r

Labels

`t + ggtitle("New Plot Title")`

Add a main title above the plot

`t + xlab("New X label")`

Change the label on the X axis

`t + ylab("New Y label")`

Change the label on the Y axis

`t + labs(title = "New title", x = "New x", y = "New y")`

All of the above

Use scale functions to update legend labels

Legends

`t + theme(legend.position = "bottom")`

Place legend at "bottom", "top", "left", or "right"

`t + guides(color = "none")`

Set legend type for each aesthetic: colorbar, legend, or none (no legend)

`t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))`

Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)

`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`

With clipping (removes unseen data points)

`t + xlim(0, 100) + ylim(10, 20)`

`t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`

Learn more at docs.ggplot2.org/ggplot2.0.9.3.1.html

Source — <https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

9. PySpark

Python For Data Science Cheat Sheet

PySpark 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
>>> sc.pythonVer
>>> sc.master
>>> str(sc.sparkHome)
>>> str(sc.sparkUser())
>>> sc.appName
>>> sc.applicationId
>>> sc.defaultParallelism
>>> sc.defaultMinPartitions
```

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), ("a", 1), ("b", 1))])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([(("a", ["x", "y", "z"]),
...     ("b", ["p", "q", "r"]))])
```

External Data

Read either one text file from HDFS, a local file system 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()
>>> rdd.count()
>>> rdd.countByKey()
defaultdict<type 'int'>, {'a':2, 'b':1}
>>> rdd.countByValue()
defaultdict<type 'int'>, {('b',2):1, ('a',2):1, ('a',7):1}
>>> rdd.collectAsMap()
{'a': 2, 'b': 2}
>>> rdd.sum()
4950
>>> sc.parallelize([]).isEmpty()
```

List the number of partitions
Count RDD instances
Count RDD instances by key
Count RDD instances by value
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

Summary

```
>>> rdd3.max()
99
>>> rdd3.min()
0
>>> rdd3.mean()
49.5
>>> rdd3.stdev()
28.86607004772218
>>> rdd3.variance()
833.25
>>> rdd3.histogram(3)
[(0,33,66,99),(33,33,34)]
>>> rdd3.stats()
```

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins
Summary statistics (count, mean, stdev, max & min)

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))
[(('a', 7), ('a', 1), ('a', 2, 2, 'a'), ('b', 2, 2, 'b'))]
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
>>> rdd5.collect()
[('a', 7, 'a', 'a', 2, 2, 'a', 'b', 2, 2, 'b')]
>>> rdd4.flatMapValues(lambda x: x)
>>> rdd4.collect()
[('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'p'), ('b', 'r')]
```

Apply a function to each RDD element
Apply a function to each RDD element and flatten the result
Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys

Selecting Data

```
>>> rdd.collect()
[('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)
[('a', 7), ('a', 2)]
>>> rdd.first()
('a', 7)
>>> rdd.top(2)
[('b', 2), ('a', 7)]

```

Return a list with all RDD elements
Take first 2 RDD elements
Take first RDD element
Take top 2 RDD elements
Return sampled subset of rdd3

Sampling

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

Filtering

```
>>> rdd.filter(lambda x: "a" in x)
```

```
.collect()
```

```
[('a', 7), ('a', 2)]
```

```
>>> rdd5.distinct().collect()
```

```
[('a', 2, 'b', 7)]
```

```
>>> rdd.keys().collect()
```

```
[('a', 'a', 'b')]
```

Filter the RDD

Return distinct RDD values

Return (key,value) RDD's keys

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)
('a', 7)
('b', 2)
('a', 2)
```

Apply a function to all RDD elements

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)
.collect()
[('a',7),('b',2)]
>>> rdd.reduceByKey(lambda a, b: a + b)
('a',7,'a',2,'b',2)
```

Merge the rdd values for each key

Merge the rdd values

Grouping by

```
>>> rdd3.groupByKey(lambda x: x * 2)
.mapValues(list)
.collect()
>>> rdd3.groupByKey()
.mapValues(list)
.collect()
[('a', [17, 2]), ('b', [2])]
```

Return RDD of grouped values

Group rdd by key

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)
(4950, 100)
>>> rdd3.aggregateByKey((0,0),seqOp,combOp)
.collect()
[('a', (9,2)), ('b', (2,1))]
>>> rdd3.fold(0,add)
4950
>>> rdd3.foldByKey(0, add)
.collect()
[('a', 9), ('b', 2)]
>>> rdd3.keyBy(lambda x: x+x)
.collect()
```

Aggregate RDD elements of each partition and then the results

Aggregate values of each RDD key

Aggregate the elements of each partition, and then the results

Merge the values for each key

Create tuples of RDD elements by applying a function

Mathematical Operations

```
>>> rdd.subtract(rdd2)
```

Return each rdd value not contained in rdd2

```
.collect()
[('b',2),('a',7)]
```

Return each (key,value) pair of rdd2 with no matching key in rdd

```
>>> rdd2.subtractByKey(rdd)
.collect()
[('d', 1)]
```

Return the Cartesian product of rdd and rdd2

Sort

```
>>> rdd2.sortBy(lambda x: x[1])
.collect()
```

Sort RDD by given function

```
[('a',1),('b',1),('a',2)]
```

Sort (key,value) RDD by key

```
>>> rdd2.sortByKey()
.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("namenodehost/parent/child",
...     "org.apache.hadoop.mapred.TextOutputFormat")
```

Stopping SparkContext

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

Execution

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

DataCamp

Learn Python for Data Science interactively



Source — <https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#gs.L=J1zxQ>

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
>>> sc.pythonVer
>>> sc.master
>>> str(sc.sparkHome)
>>> str(sc.sparkUser())
>>> sc.appName
>>> sc.applicationId
>>> sc.defaultParallelism
>>> sc.defaultMinPartitions
```

Retrieve SparkContext version
Retrieve Python version
Master URL to connect to
Path where Spark is installed on worker nodes
Retrieve name of the Spark User running SparkContext
Return application name
Retrieve application ID
Return default level of parallelism
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', 1), ('a', 2), ('b', 1)])
>>> rdd2 = sc.parallelize([('a', 2), ('d', 1), ('b', 1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([('a', 'x'), ('y', 'z'), ('b', 'p'), ('b', 'r')])
```

External Data

Read either one text file from HDFS, a local file system 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()
>>> rdd.count()
3
>>> rdd.countByKey()
defaultdict(<type 'int'>, {'a':2, 'b':1})
>>> rdd.countByValue()
defaultdict(<type 'int'>, {'b':2, 1, ('a',2,1), ('a',1,1)})
>>> rdd.collectAsMap()
{'a': 2, 'b': 2}
>>> rdd.sum()
4950
>>> sc.parallelize([]).isEmpty()
True
```

List the number of partitions
Count RDD instances
Count RDD instances by key
Count RDD instances by value
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

Summary

```
>>> rdd3.max()
99
>>> rdd3.min()
0
>>> rdd3.mean()
49.5
>>> rdd3.stdev()
28.866070047722118
>>> rdd3.variance()
833.25
>>> rdd3.histogram(3)
[(0,33,66,99), [33,33,34]]
>>> rdd3.stats()
```

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins
Summary statistics (count, mean, stdev, max & min)

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0]))
...     .collect()
...
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
...
>>> rdd5.collect()
[('a',7,7),('a',1),('a',2,2,'a'),('b',2,2,'b')]
>>> rdd4.flatMapValues(lambda x: x)
...
>>> rdd4.collect()
[('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

Apply a function to each RDD element
Apply a function to each RDD element and flatten the result
Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys

Selecting Data

```
>>> Getting
>>> rdd.collect()
[('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)
[('a', 7), ('a', 2)]
>>> rdd.first()
('a', 7)
>>> rdd.top(2)
[('b', 2), ('a', 7)]
...
>>> Sampling
>>> rdd3.sample(False, 0.15, 81).collect()
[3,4,27,31,40,41,42,43,60,76,79,80,86,97]
...
>>> Filtering
>>> rdd.filter(lambda x: "a" in x)
...
>>> rdd.collect()
[('a',7),('a',2)]
...
>>> rdd5.distinct().collect()
[('a',2),('b',7)]
...
>>> rdd.keys().collect()
[('a', 'a', 'b')]
```

Return a list with all RDD elements
Take first 2 RDD elements
Take first RDD element
Take top 2 RDD elements
Return sampled subset of rdd3
Filter the RDD
Return distinct RDD values
Return (key,value) RDD's keys

Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g)
('a', 7)
('a', 2)
('a', 1)
```

Apply a function to all RDD elements

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)
...
[('a',9),('b',2)]
...
>>> rdd.reduce(lambda a, b: a + b)
('a',7,'a',2,'b',2)
```

Merge the rdd values for each key
Merge the rdd values

Grouping by

```
>>> rdd3.groupByKey()
...
[('a',[7,2]),('b',[2])]
```

Return RDD of grouped values
Group rdd by key

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)
(4950,100)
...
>>> rdd.aggregateByKey((0,0),seqOp,combOp)
...
[('a',(9,2)), ('b',(2,1))]
...
>>> rdd3.fold(0,(0,add))
4950
...
>>> rdd.foldByKey(0,(0,add))
...
[('a',9),('b',2)]
...
>>> rdd3.keyBy(lambda x: x+x).collect()
```

Aggregate RDD elements of each partition and then the results
Aggregate values of each RDD key
Aggregate the elements of each partition, and then the results
Merge the values for each key
Create tuples of RDD elements by applying a function

Mathematical Operations

```
>>> rdd.subtract(rdd2)
...
[('b',2),('a',7)]
...
>>> rdd2.subtractByKey(rdd)
...
[('d', 1)]
...
>>> rdd.cartesian(rdd2).collect()
```

Return each rdd value not contained in rdd2
Return each (key,value) pair of rdd2 with no matching key in rdd
Return the Cartesian product of rdd and rdd2

Sort

```
>>> rdd2.sortBy(lambda x: x[1])
...
[('d',1),('b',1),('a',2)]
...
>>> rdd2.sortByKey()
...
[('a',2),('b',1),('d',1)]
```

Sort RDD by given function
Sort (key,value) RDD by key

Repartitioning

```
>>> rdd.repartition(4)
...
>>> rdd.coalesce(1)
```

New RDD with 4 partitions
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
```

DataCamp
Learn Python for Data Science interactively



Source — <https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#gs.L=J1zxQ>

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 \\\n    .builder \\\n    .appName("Python Spark SQL basic example") \\\n    .config("spark.some.config.option", "some-value") \\\n    .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])))  
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 |  
+-----+-----+-----+  
| Alice | 21 |  
| Bob | 22 |  
| Diane | 23 |  
| Eric | 24 |  
| Fred | 25 |  
| Greta | 26 |  
| Howard | 27 |  
| Irene | 28 |  
| Jerry | 29 |  
| Kramer | 30 |  
| Leo | 31 |  
+-----+-----+-----+
```

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... | 26 | Jane | Doe | [321 555-1234, 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 >>> 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
------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------

Duplicate Values

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

Queries

```
>>> from pyspark.sql import functions as F  
Select  
>>> df.select("firstName").show()  
>>> df.select("firstName", "lastName") \\\n    .show()  
>>> df.select("firstName",  
                "age",  
                explode("phoneNumber") \\\n                .alias("contactInfo")) \\\n    .select("contactInfo.type",  
            "firstName",  
            "age") \\\n    .show()  
>>> df.select(df["firstName"],df["age"]+ 1) \\\n    .show()  
>>> df.select(df["age"] > 24).show()  
When  
>>> df.select("firstName",  
                F.when(df.age > 30, 1) \\\n                .otherwise(0)) \\\n    .show()  
>>> df[df.firstName.isin("Jane","Boris")] \\\n    .collect()  
Like  
>>> df.select("firstName",  
                df.lastName.like("Smith")) \\\n    .show()  
Startswith - Endswith  
>>> df.select("firstName",  
                df.lastName \\\n                .startswith("Sm")) \\\n    .show()  
Substring  
>>> df.select(df.firstName.substr(1, 3) \\\n                .alias("name")) \\\n    .collect()  
Between  
>>> df.select(df.age.between(22, 24)) \\\n    .show()
```

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \\\n    .withColumn('postalCode',df.address.postalCode) \\\n    .withColumn('state',df.address.state) \\\n    .withColumn('streetAddress',df.address.streetAddress) \\\n    .withColumn('telephoneNumber',  
                explode(df.phoneNumber.number)) \\\n    .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)
```

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 O 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

GroupBy

```
>>> df.groupBy("age") \\\n    .count() \\\n    .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]) \\\n    .collect()
```

Missing & Replacing Values

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

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

Repartitioning

```
>>> df.repartition(10) \\\n    .rdd \\\n    .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("customers")  
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()  
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people") \\\n    .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") \\\n    .write \\\n    .save("nameAndCity.parquet")  
>>> df.select("firstName", "age") \\\n    .write \\\n    .save("namesAndAges.json",format="json")
```

Stopping SparkSession

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

DataCamp

Learn Python for Data Science interactively

Source — <https://www.datacamp.com/community/blog/pyspark-sql-cheat-sheet>

10. R Studio (dplyr and tidyverse)

Data Wrangling with dplyr and tidyr

Cheat Sheet



Syntax - Helpful conventions for wrangling

`dplyr::tbl_df(iris)`

Converts data to `tbl` class. `tbl`'s are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]
  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3
4          4.6         3.1          1.5
5          5.0         3.6          1.4
..          ...
Variables not shown: Petal.Width (dbl), Species (fctr)
```

`dplyr::glimpse(iris)`

Information dense summary of `tbl` data.

`utils::View(iris)`

View data set in spreadsheet-like display (note capital V).

iris					
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

`dplyr::%>%`

Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
x %>% f(y) is the same as f(x, y)
y %>% f(x, ., z) is the same as f(x, y, z)
```

"Piping" with `%>%` makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Tidy Data - A foundation for wrangling in R

In a tidy data set:



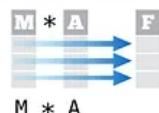
&



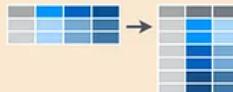
Each variable is saved in its own column

Each observation is saved in its own row

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



Reshaping Data - Change the layout of a data set



`tidy::gather(cases, "year", "n", 2:4)`

Gather columns into rows.



`tidy::spread(pollution, size, amount)`

Spread rows into columns.



`tidy::separate(storms, date, c("y", "m", "d"))`

Separate one column into several.



`tidy::unite(data, col, ..., sep)`

Unite several columns into one.

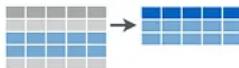
`dplyr::data_frame(a = 1:3, b = 4:6)`
Combine vectors into data frame (optimized).

`dplyr::arrange(mtcars, mpg)`
Order rows by values of a column (low to high).

`dplyr::arrange(mtcars, desc(mpg))`
Order rows by values of a column (high to low).

`dplyr::rename(tb, y = year)`
Rename the columns of a data frame.

Subset Observations (Rows)



`dplyr::filter(iris, Sepal.Length > 7)`

Extract rows that meet logical criteria.

`dplyr::distinct(iris)`

Remove duplicate rows.

`dplyr::sample_frac(iris, 0.5, replace = TRUE)`

Randomly select fraction of rows.

`dplyr::sample_n(iris, 10, replace = TRUE)`

Randomly select n rows.

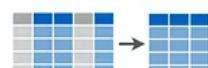
`dplyr::slice(iris, 10:15)`

Select rows by position.

`dplyr::top_n(storms, 2, date)`

Select and order top n entries (by group if grouped data).

Subset Variables (Columns)



`dplyr::select(iris, Sepal.Width, Petal.Length, Species)`

Select columns by name or helper function.

Helper functions for select - ?select

`select(iris, contains("."))`

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches("t.*"))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of(c("Species", "Genus")))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepal"))`

Select columns whose name starts with a character string.

`select(iris, Sepal.Length:Petal.Width)`

Select all columns between Sepal.Length and Petal.Width (inclusive).

`select(iris, -Species)`

Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with `browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.4.0 • tidyverse 0.2.0 • Updated: 1/15

Summarise Data



`dplyr::summarise(iris, avg = mean(Sepal.Length))`

Summarise data into single row of values.

`dplyr::summarise_each(iris, funs(mean))`

Apply summary function to each column.

`dplyr::count(iris, Species, wt = Sepal.Length)`

Count number of rows with each unique value of variable (with or without weights).



Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

`dplyr::first`

First value of a vector.

`dplyr::last`

Last value of a vector.

`dplyr::nth`

Nth value of a vector.

`dplyr::n`

of values in a vector.

`dplyr::n_distinct`

of distinct values in a vector.

`IQR`

IQR of a vector.

`min`

Minimum value in a vector.

`max`

Maximum value in a vector.

`mean`

Mean value of a vector.

`median`

Median value of a vector.

`var`

Variance of a vector.

`sd`

Standard deviation of a vector.

`dplyr::lead`

Copy with values shifted by 1.

`dplyr::lag`

Copy with values lagged by 1.

`dplyr::dense_rank`

Ranks with no gaps.

`dplyr::min_rank`

Ranks. Ties get min rank.

`dplyr::percent_rank`

Ranks rescaled to [0, 1].

`dplyr::row_number`

Ranks. Ties get to first value.

`dplyr::ntile`

Bin vector into n buckets.

`dplyr::between`

Are values between a and b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::cumall`

Cumulative all

`dplyr::cumany`

Cumulative any

`dplyr::cummean`

Cumulative mean

`cumsum`

Cumulative sum

`cummax`

Cumulative max

`cummin`

Cumulative min

`cumprod`

Cumulative prod

`pmax`

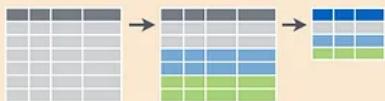
Element-wise max

`pmin`

Element-wise min

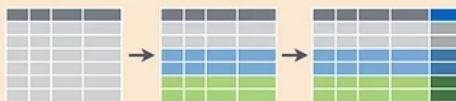
`iris %>% group_by(Species) %>% summarise(...)`

Compute separate summary row for each group.



`iris %>% group_by(Species) %>% mutate(...)`

Compute new variables by group.

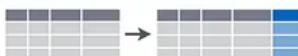


`devtools::install_github("rstudio/EDAWR")` for data sets

Learn more with

`browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.4.0 • tidyverse 0.2.0 • Updated: 1/15

Make New Variables



`dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)`

Compute and append one or more new columns.

`dplyr::mutate_each(iris, funs(min_rank))`

Apply window function to each column.

`dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)`

Compute one or more new columns. Drop original columns.



Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

`dplyr::lead`

Copy with values shifted by 1.

`dplyr::lag`

Copy with values lagged by 1.

`dplyr::dense_rank`

Ranks with no gaps.

`dplyr::min_rank`

Ranks. Ties get min rank.

`dplyr::percent_rank`

Ranks rescaled to [0, 1].

`dplyr::row_number`

Ranks. Ties get to first value.

`dplyr::ntile`

Bin vector into n buckets.

`dplyr::between`

Are values between a and b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::cumall`

Cumulative all

`dplyr::cumany`

Cumulative any

`dplyr::cummean`

Cumulative mean

`cumsum`

Cumulative sum

`cummax`

Cumulative max

`cummin`

Cumulative min

`cumprod`

Cumulative prod

`pmax`

Element-wise max

`pmin`

Element-wise min

Combine Data Sets

a	x2	b	x3
A	1	A	
B	2	B	F
C	3	C	T



a	x2	b	x3
A	1	A	
B	2	B	F
D	1	D	T



Mutating Joins

`dplyr::left_join(a, b, by = "x1")`

Join matching rows from b to a.

`dplyr::right_join(a, b, by = "x1")`

Join matching rows from a to b.

`dplyr::inner_join(a, b, by = "x1")`

Join data. Retain only rows in both sets.

`dplyr::full_join(a, b, by = "x1")`

Join data. Retain all values, all rows.

Filtering Joins

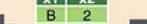
`dplyr::semi_join(a, b, by = "x1")`

All rows in a that have a match in b.

`dplyr::anti_join(a, b, by = "x1")`

All rows in a that do not have a match in b.

y	x2	z	x2
A	1	B	2
B	2	C	3
C	3	D	4



y	x2	z	x2
A	1	B	2
B	2	C	3



Set Operations

`dplyr::intersect(y, z)`

Rows that appear in both y and z.

`dplyr::union(y, z)`

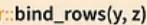
Rows that appear in either or both y and z.

`dplyr::setdiff(y, z)`

Rows that appear in y but not z.

Binding

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



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



`dplyr::bind_rows(y, z)`

Append z to y as new rows.

`dplyr::bind_cols(y, z)`

Append z to y as new columns.

Caution: matches rows by position.

11. Jupyter Notebook

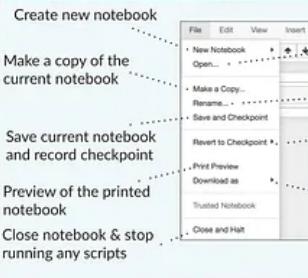
Python For Data Science Cheat Sheet

Jupyter Notebook

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



Saving/Loading Notebooks



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

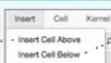
Adjust metadata underlying the current notebook

Remove cell attachments

Paste attachments of current cell

Insert Cells

Add new cell above the current one



Add new cell below 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

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

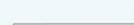
Change the cell type of current cell

toggle, toggle scrolling and clear all output

View Cells

Toggle display of Jupyter logo and filename

Toggle line numbers in cells

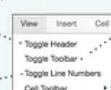


Run current cells down and create a new one below

Run all cells

Run all cells below the current cell

toggle, toggle scrolling and clear current outputs



Toggle display of toolbar

Toggle display of cell action icons:

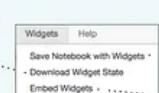
- None
- Edit metadata
- Raw cell format
- Slideshow
- Attachments
- Tags

Widgets

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

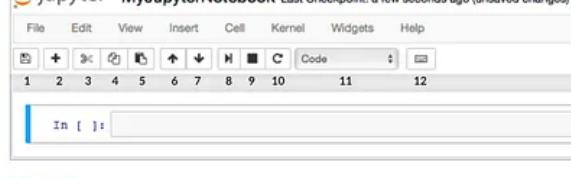
You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

Download serialized state of all widget models in use



Save notebook with interactive widgets
Embed current widgets

Command Mode:



Logout
15
Trusted
Python 3 O
13 14

Asking For Help

Walk through a UI tour

Edit the built-in keyboard shortcuts

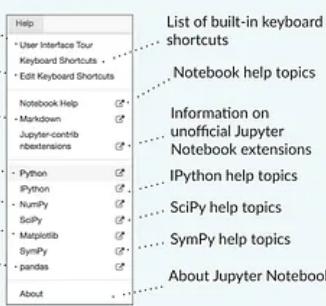
Description of markdown available in notebook

Python help topics

NumPy help topics

Matplotlib help topics

Pandas help topics



List of built-in keyboard shortcuts

Notebook help topics

Information on unofficial Jupyter Notebook extensions

IPython help topics

SciPy help topics

Sympy help topics

About Jupyter Notebook

Source — <https://www.datacamp.com/community/blog/jupyter-notebook-cheat-sheet>

12. Dask



DASK FOR PARALLEL COMPUTING CHEAT SHEET

See full Dask documentation at: <http://dask.pydata.org/>

These instructions use the conda environment manager. Get yours at <http://bit.ly/getconda>

DASK QUICK INSTALL

Install Dask with conda

```
conda install dask
```

Install Dask with pip

```
pip install dask[complete]
```

DASK COLLECTIONS

EASY TO USE BIG DATA COLLECTIONS

DASK DATAFRAMES

PARALLEL PANDAS DATAFRAMES FOR LARGE DATA

Import

```
import dask.dataframe as dd
```

Read CSV data

```
df = dd.read_csv('my-data.*.csv')
```

Read Parquet data

```
df = dd.read_parquet('my-data.parquet')
```

Filter and manipulate data with Pandas syntax

```
df['z'] = df.x + df.y
```

Standard groupby aggregations, joins, etc.

```
result = df.groupby(df.z).y.mean()
```

Compute result as a Pandas dataframe

```
out = result.compute()
```

Or store to CSV, Parquet, or other formats

```
result.to_parquet('my-output.parquet')
```

EXAMPLE

```
df = dd.read_csv('filenames.*.csv')
df.groupby(df.timestamp.day) \
    .value.mean().compute()
```

DASK ARRAYS

PARALLEL NUMPY ARRAYS FOR LARGE DATA

Import

```
import dask.array as da
```

Create from any array-like object

```
import h5py
dataset = h5py.File('my-data.hdf5')[ '/group/dataset' ]
x = da.from_array(dataset, chunks=(1000, 1000))
```

Including HDF5, NetCDF, or other on-disk formats.

Alternatively generate an array from a random distribution.

```
da.random.uniform(shape=(1e4, 1e4), chunks=(100, 100))
```

Perform operations with NumPy syntax

```
y = x.dot(x.T - 1) - x.mean(axis=0)
```

Compute result as a NumPy array

```
result = y.compute()
```

Or store to HDF5, NetCDF or other on-disk format

```
out = f.create_dataset(... )
x.store(out)
```

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf

EXAMPLE

```
with h5py.File('my-data.hdf5') as f:  
    x = da.from_array(f['/path'], chunks=(1000, 1000))  
    x -= x.mean(axis=0)  
    out = f.create_dataset(...)  
    x.store(out)
```

DASK BAGS

Import

PARELLEL LISTS FOR UNSTRUCTURED DATA

Create Dask Bag from a sequence

```
import dask.bag as db  
  
b = db.from_sequence(seq, npartitions)
```

Or read from text formats

```
b = db.read_text('my-data.*.json')
```

Map and filter results

```
import json  
records = b.map(json.loads)  
           .filter(lambda d: d["name"] == "Alice")
```

Compute aggregations like mean, count, sum

```
records.pluck('key-name').mean().compute()
```

Or store results back to text formats

```
records.to_textfiles('output.*.json')
```

EXAMPLE

```
db.read_text('s3://bucket/my-data.*.json')  
           .map(json.loads)  
           .filter(lambda d: d["name"] == "Alice")  
           .to_textfiles('s3://bucket/output.*.json')
```



CONTINUED ON BACK →

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf

[Open in app](#)

Search



Write



CUSTOM COMPUTATIONS	FOR CUSTOM CODE AND COMPLEX ALGORITHMS
DASK DELAYED	LAZY PARALLELISM FOR CUSTOM CODE
Import	<code>import dask</code>
Wrap custom functions with the <code>@dask.delayed</code> annotation	<code>@dask.delayed</code> <code>def load(filename):</code> ... <code>@dask.delayed</code> <code>def process(data):</code> ... <code>load = dask.delayed(load)</code> <code>process = dask.delayed(process)</code>
Delayed functions operate lazily, producing a task graph rather than executing immediately	
Passing delayed results to other delayed functions creates dependencies between tasks	
Call functions in normal code	<code>data = [load(fn) for fn in filenames]</code> <code>results = [process(d) for d in data]</code>
Compute results to execute in parallel	<code>dask.compute(results)</code>
CONCURRENT.FUTURES	ASYNCHRONOUS REAL-TIME PARALLELISM
Import	<code>from dask.distributed import Client</code>
Start local Dask Client	<code>client = Client()</code>
Submit individual task asynchronously	<code>future = client.submit(func, *args, **kwargs)</code>
Block and gather individual result	<code>result = future.result()</code>
Process results as they arrive	<code>for future in as_completed(futures):</code> ...
EXAMPLE	<code>L = [client.submit(read, fn) for fn in filenames]</code> <code>L = [client.submit(process, future) for future in L]</code> <code>future = client.submit(sum, L)</code> <code>result = future.result()</code>

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf

SET UP CLUSTER	HOW TO LAUNCH ON A CLUSTER
MANUALLY	
Start scheduler on one machine	\$ dask-scheduler Scheduler started at SCHEDULER_ADDRESS:8786
Start workers on other machines Provide address of the running scheduler	host1\$ dask-worker SCHEDULER_ADDRESS:8786 host2\$ dask-worker SCHEDULER_ADDRESS:8786
Start Client from Python process	from dask.distributed import Client client = Client('SCHEDULER_ADDRESS:8786')
ON A SINGLE MACHINE	
Call Client() with no arguments for easy setup on a single host	client = Client()
CLOUD DEPLOYMENT	
See dask-kubernetes project for Google Cloud	pip install dask-kubernetes
See dask-ec2 project for Amazon EC2	pip install dask-ec2
MORE RESOURCES	
User Documentation	dask.pydata.org
Technical documentation for distributed scheduler	distributed.readthedocs.org
Report a bug	github.com/dask/dask/issues



anaconda.com · info@anaconda.com · 512-776-1066
8/20/2017

Source — http://docs.dask.org/en/latest/_downloads/daskcheatsheet.pdf

Thank you for reading.

*If you want to get into contact, you can reach out to me at
ahikailash1@gmail.com*

About Me:

I am a Co-Founder of [MateLabs](#), where we have built [Mateverse](#), an ML Platform which enables everyone to easily build and train Machine Learning Models, without writing a single line of code.

Note: Recently, I published a book on GANs titled “Generative Adversarial Networks Projects”, in which I covered most of the widely popular GAN architectures and their implementations. DCGAN, StackGAN, CycleGAN, Pix2pix, Age-cGAN, and 3D-GAN have been covered in details at the implementation level. Each architecture has a chapter dedicated to it. I have explained these networks in a very simple and descriptive language using Keras framework with Tensorflow

backend. If you are working on GANs or planning to use GANs, give it a read and share your valuable feedback with me at ahikailash1@gmail.com

Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow...

Explore various Generative Adversarial Network architectures using the Python ecosystem Key Features Use different...

[www.amazon.com](http://www.amazon.com/Generative-Adversarial-Networks-Projects-next-generation/dp/1789136679)

You can grab a copy of the book from <http://www.amazon.com/Generative-Adversarial-Networks-Projects-next-generation/dp/1789136679> <https://www.amazon.in/Generative-Adversarial-Networks-Projects-next-generation/dp/1789136679?fbclid=IwAR0X2pDk4CTxn5GqWmBbKIgiB38WmFX-sqCpBNI8k9Z8IKCQ7VWRpJXm7I> https://www.packtpub.com/big-data-and-business-intelligence/generative-adversarial-networks-projects?fbclid=IwAR2OtU21faMFPM4suH_HJmy_DRQxOVwJZB0kz3ZiSbFb_MW7INYCqqV7U0c



Triplebyte — <https://triplebyte.com/a/ZYAvvEc/d>

Triplebyte helps programmers find great companies to work at. They'll go through a technical interview with you, match you with companies that are looking for people with your specific skill sets, and then fast track you through their hiring processes. Looking for a new job? Take Triplebyte's quiz and get a job at top companies!

Machine Learning

Artificial Intelligence

Deep Learning

Technology

Computer Science



Written by Kailash Ahirwar

5.8K Followers · Writer for Startups & Venture Capital

Decentralizing Artificial Intelligence | Co-founder - Raven Protocol | Mate Labs | Author - Generative Adversarial Networks Projects

Follow

More from Kailash Ahirwar and Startups & Venture Capital

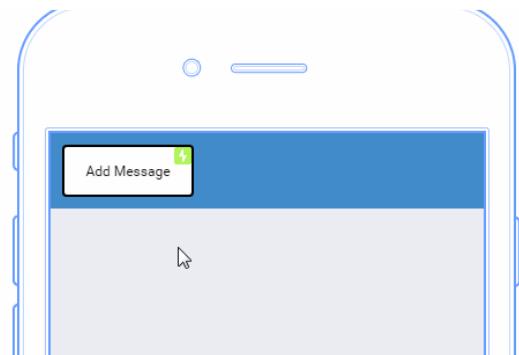


 Kailash Ahirwar

A Very Short Introduction to Diffusion Models

Artificial Intelligence is constantly evolving to solve hard and complex problems. Image...

4 min read · Sep 26



 Sophie Paxton in Startups & Venture Capital

Your UI isn't a Disney Movie

My previous short article about gratuitous animation really struck a chord with people....

5 min read · Oct 6, 2015

129



+

...

4.5K



113

+

...



 Charles R Poliquin in Startups & Venture Capital

10 Household Items That Are Dropping Your Testosterone Level...

We live in a toxic world, and this toxicity affects androgen levels at an epidemic level

7 min read · Jan 19, 2018

879



+

...

85



+

...

 Kailash Ahirwar

A Very Short Introduction to Inception Score(IS)

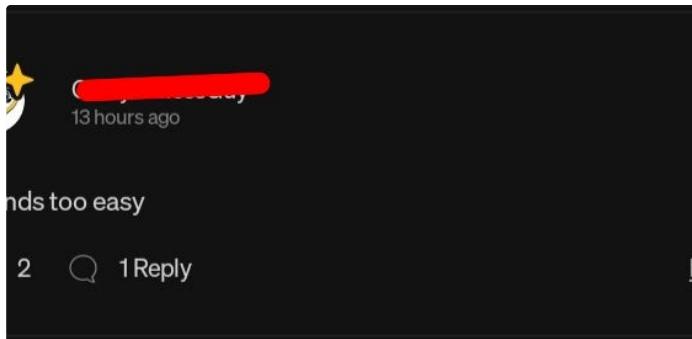
Generative Adversarial Networks or GANs for short were successful in generating high-...

4 min read · Feb 24, 2021

[See all from Kailash Ahirwar](#)

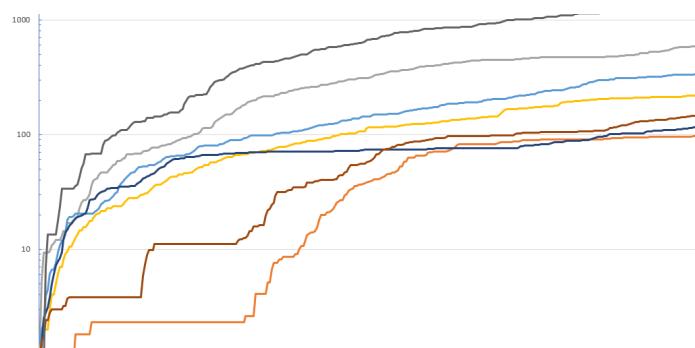
[See all from Startups & Venture Capital](#)

Recommended from Medium



ands too easy

2 1 Reply



I Found A Very Profitable AI Side Hustle

And it's perfect for beginners

6 min read · Oct 19

 12.7K  222

My Life Stats: I Tracked My Habits for a Year, and This Is What I...

I measured the time I spent on my daily activities (studying, doing sports, socializing...)

12 min read · Nov 21

 3.1K  57

Lists



Predictive Modeling w/ Python

20 stories · 659 saves



AI Regulation

6 stories · 208 saves



Natural Language Processing

924 stories · 441 saves



Practical Guides to Machine Learning

10 stories · 739 saves



 Benoit Ruiz in Better Programming

Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21

 12.4K  238

 ...



 Data Scian by Imad Adrees

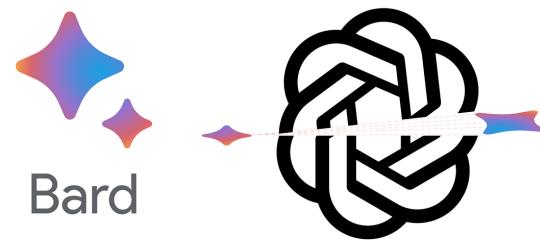
Best Portfolio Projects for Data Science

“How can I showcase my data skills to the world?” you may be asking. Fear not, for the...

5 min read · Sep 19

 611  5

 ...



 AL Anany 

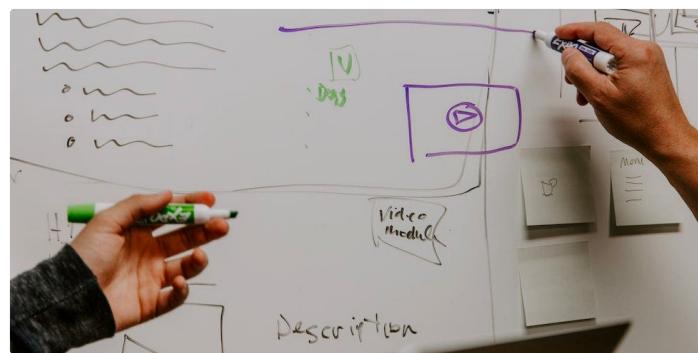
The ChatGPT Hype Is Over—Now Watch How Google Will Kill...

It never happens instantly. The business game is longer than you know.

 · 6 min read · Sep 1

 20K  615

 ...



 Carlos Arguelles

My favorite coding question to give candidates

A coding question, from the viewpoint of an Google/Amazon/Microsoft interviewer

11 min read · Nov 12

 4.5K  49

 ...

See more recommendations