



Kailash Ahirwar [Follow](#)

Co-founder @ Mate Labs | Democratizing Artificial Intelligence

May 28 · 2 min read

Essential Cheat Sheets for Machine Learning and Deep Learning Engineers

Learning machine learning and deep learning is difficult for newbies. As well as deep learning libraries are difficult to understand. I am creating a repository on Github([cheatsheets-ai](#)) with cheat sheets which I collected from different sources. Do visit it and contribute cheat sheets if you have any. Thanks.

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

Data

Also see NumPy, Pandas & Scikit-Learn

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

Keras Data Sets

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

Other

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

Preprocessing

Sequence Padding

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

One-Hot Encoding

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

Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

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

Multi-Class Classification

```
>>> from keras.layers import Dropout
```

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

Regression

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

```
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
```

```
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
```

```
>>> model2.add(Activation('relu'))
```

```
>>> model2.add(Conv2D(32,(3, 3)))
```

```
>>> model2.add(Activation('relu'))
```

```
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
```

```
>>> model2.add(Dropout(0.25))
```

```
>>> model2.add(Conv2D(64,(3, 3), padding='same'))
```

```
>>> model2.add(Activation('relu'))
```

```
>>> model2.add(Conv2D(64,(3, 3)))
```

```
>>> model2.add(Activation('relu'))
```

```
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
```

```
>>> model2.add(Dropout(0.25))
```

```
>>> model2.add(Flatten())
```

```
>>> model2.add(Dense(512))
```

```
>>> model2.add(Activation('relu'))
```

```
>>> model2.add(Dropout(0.5))
```

```
>>> model2.add(Dense(num_classes))
```

```
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
```

```
>>> model3.add(Embedding(20000,128))
```

```
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
```

```
>>> model3.add(Dense(1,activation='sigmoid'))
```

Also see NumPy & Scikit-Learn

Train and Test Sets

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

Standardization/Normalization

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

Inspect Model

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

Model output shape	Model summary representation
Model configuration	List all weight tensors in the model

Compile Model

MLP: Binary Classification	<code>>>> model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])</code>
MLP: Multi-Class Classification	<code>>>> model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])</code>
MLP: Regression	<code>>>> model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])</code>

Recurrent Neural Network

<code>>>> model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])</code>
--

Model Training

<code>>>> model3.fit(x_train4, y_train, batch_size=32, epochs=15, verbose=1, validation_data=(x_test4,y_test4))</code>

Evaluate Your Model's Performance

<code>>>> score = model3.evaluate(x_test, y_test, batch_size=32)</code>
--

Prediction

<code>>>> model3.predict(x_test4, batch_size=32)</code>
--

Save/ Reload Models

<code>>>> from keras.models import load_model >>> model3.save('model.h5') >>> my_model = load_model('my_model.h5')</code>
--

Model Fine-tuning

Optimization Parameters

<code>>>> from keras.optimizers import RMSProp >>> opt = RMSprop(lr=1e-0001, decay=1e-0)</code>
<code>>>> model2.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])</code>

Early Stopping

<code>>>> 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])</code>
--

DataCamp

Learn Python for Data Science [Interactively](#)

Source — <https://www.datacamp.com/community/blog/keras-cheat-sheet#gs.DRKeNMs>

2. Numpy

Python For Data Science Cheat Sheet

NumPy Basics

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



NumPy

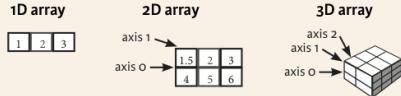
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

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



NumPy Arrays



Creating Arrays

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

Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)
>>> np.linspace(0,2,9)
>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2x2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

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

Saving & Loading Text Files

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

Data Types

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

Inspecting Your Array

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

Asking For Help

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

Array Mathematics

Arithmetic Operations

<code>>>> g = a - b</code>	Subtraction
<code>>>> np.subtract(a,b)</code>	Addition
<code>>>> b + a</code>	Addition
<code>>>> np.add(b,a)</code>	Division
<code>>>> a / b</code>	Multiplication
<code>>>> np.divide(a,b)</code>	Exponentiation
<code>>>> a * b</code>	Square root
<code>>>> np.multiply(a,b)</code>	Print sines of an array
<code>>>> np.exp(b)</code>	Element-wise cosine
<code>>>> np.sqrt(b)</code>	Element-wise natural logarithm
<code>>>> np.sin(a)</code>	Dot product
<code>>>> np.cos(b)</code>	
<code>>>> np.log(a)</code>	
<code>>>> e.dot(f)</code>	
<code>>>> array([[1.5, 4., 9.], [4., 10., 18.]]])</code>	

Comparison

<code>>>> a == b</code>	Element-wise comparison
<code>>>> a != b</code>	Element-wise comparison
<code>>>> a < 2</code>	Element-wise comparison
<code>>>> np.array_equal(a, b)</code>	Array-wise comparison

Aggregate Functions

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

Copying Arrays

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

Sorting Arrays

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

Subsetting, Slicing, Indexing

Also see [Lists](#)

Subsetting

<code>>>> a[2]</code>	Select the element at the 2nd index
<code>>>> b[1, 2]</code>	Select the element at row 0 column 2 (equivalent to <code>b[1][2]</code>)
<code>6.0</code>	Select items at index 0 and 1

Slicing

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

Boolean Indexing

<code>>>> a[a < 2]</code>	Select elements from <code>a</code> less than 2
<code>array([1])</code>	Select elements <code>(1,0), (0,1), (1,2)</code> and <code>(0,0)</code>
<code>>>> b[(1, 0, 1, 0)[:, 0, 1, 2, 0, 1]]</code>	Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

<code>>>> 1 = np.transpose(b)</code>	Permute array dimensions
<code>>>> 1.T</code>	Permute array dimensions

Changing Array Shape

<code>>>> a.ravel()</code>	Flatten the array
<code>>>> g.reshape(3, -2)</code>	Reshape, but don't change data

Adding/Removing Elements

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

Combining Arrays

<code>>>> np.concatenate((a,d),axis=0)</code>	Concatenate arrays
<code>>>> np.vstack((a,b))</code>	Stack arrays vertically (row-wise)
<code>>>> np.hstack((e,f))</code>	Stack arrays vertically (row-wise)
<code>>>> np.column_stack((a,d))</code>	Stack arrays horizontally (column-wise)
<code>>>> np.c_[a,d]</code>	Create stacked column-wise arrays
<code>>>> np.hsplit(a,3)</code>	Create stacked column-wise arrays

Splitting Arrays

<code>>>> np.hsplit(a,3)</code>	Split the array horizontally at the 3rd index
<code>(array([1]), array([2]), array([3]))</code>	Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science **Interactively**



Source — <https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs.AK5ZBgE>

3. Pandas

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.
```

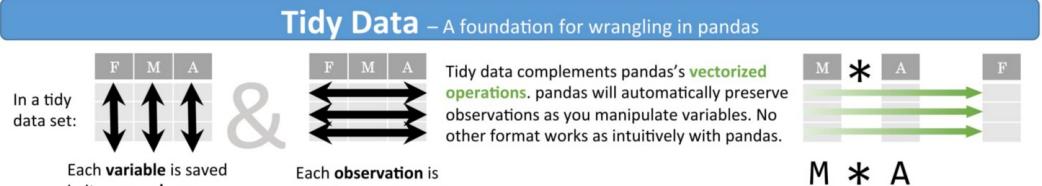
	a	b	c
n	v		
d	1	4	7
e	2	5	10
f	3	6	11
g	4	7	12

```
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
```

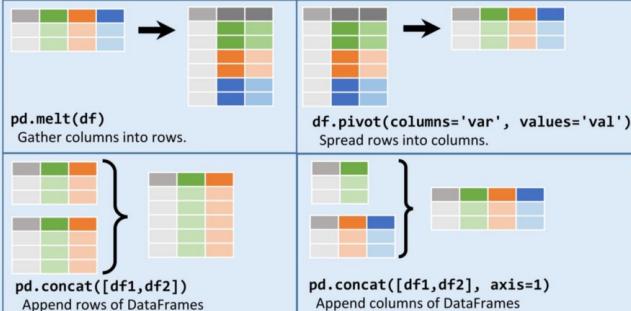
Method Chaining

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



```
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).

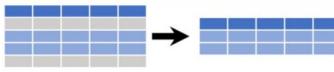
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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)



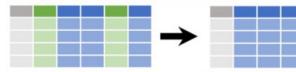
```
df[df.Length > 7]
Extract rows that meet logical criteria.

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'

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

df.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns.
```

Logic in Python (and pandas)

<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

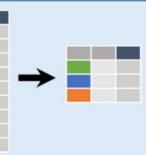
```
df['w'].value_counts()
Count number of rows with each unique value of variable
len(df)
# of rows in DataFrame.
df['w'].nunique()
# of distinct values in a column.
df.describe()
Basic descriptive statistics for each column (or GroupBy)
```



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:

```
sum()           Sum values of each object.
count()          Count non-NA/null values of each object.
median()         Median value of each object.
quantile([0.25,0.75]) Quantiles of each object.
apply(function) Apply function to each object.
```

Group Data



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

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

```
size()           Size of each group.
agg(function)   Aggregate group using function.
```

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:

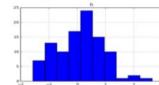
```
max(axis=1)      min(axis=1)
Element-wise max. Element-wise min.
clip(lower=-10,upper=10) abs()
Trim values at input thresholds Absolute value.
```

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.

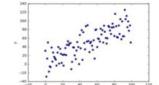
shift(1)	shift(-1)
Copy with values shifted by 1.	Copy with values lagged by 1.
rank(method='dense')	cumsum()
Ranks with no gaps.	Cumulative sum.
rank(method='min')	cummax()
Ranks. Ties get min rank.	Cumulative max.
rank(pct=True)	cummin()
Ranks rescaled to interval [0, 1].	Cumulative min.
rank(method='first')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

Plotting

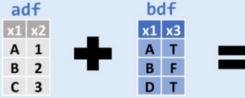
```
df.plot.hist()
Histogram for each column
```



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



Combine Data Sets



Standard Joins

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

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

Standard Joins

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

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

Standard Joins

```
x1 | x2 | x3
A 1 | T
B 2 | F
```

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

Standard Joins

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

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

Filtering Joins

```
x1 | x2
A 1
B 2
```

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

Filtering Joins

```
x1 | x2
C 3
```

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



Set-like Operations

```
x1 | x2
B 2
C 3
```

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

Set-like Operations

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

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

Set-like Operations

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

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
```

```
.query('_merge == "left_only")
```

```
.drop(['_merge'],axis=1)
```

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

<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

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

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.

pandas

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

Index

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.xls')
>>> 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

```
>>> df.ix[1, 'Capital']
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 > 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'])
Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)
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	(rows,columns)
>>> df.index	Describe index
>>> df.columns	Describe DataFrame columns
>>> df.info()	Info on DataFrame
>>> df.count()	Number of non-NA values

Summary

>>> df.sum()	Sum of values
>>> df.cumsum()	Cumulative sum of values
>>> df.min() / df.max()	Minimum/maximum values
>>> df.idmin() / df.idmax()	Minimum/Maximum index value
>>> df.describe()	Summary statistics
>>> df.mean()	Mean of values
>>> df.median()	Median of values

Applying Functions

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

Data Alignment

Internal Data Alignment

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

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

Arithmetic Operations with Fill Methods

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

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

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

Index Tricks

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

Shape Manipulation

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

Polynomials

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

Vectorizing Functions

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

Type Handling

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

Other Useful Functions

>>> np.angle(b,deg=True)	Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)	Create an array of evenly spaced values (number of samples)
>>> g[3:] += np.pi	Unwrap
>>> np.unwrap(g)	Create an array of evenly spaced values (log scale)
>>> np.logspace(0,10,3)	Return values from a list of arrays depending on conditions
>>> np.select([c<4],[c*2])	Factorial
>>> misc.factorial(a)	Combine N things taken at k time
>>> misc.comb(10,3,exact=True)	Weights for N-point central derivative
>>> misc.central_diff_weights(3)	Find the n-th derivative of a function at a point
>>> misc.derivative(myfunc,1.0)	

Linear Algebra

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

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

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(B)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I
```

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

Transpose

```
>>> A.T
```

```
>>> A.H
```

Trace

```
>>> np.trace(A)
```

Norm

```
>>> linalg.norm(A)
```

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

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

Rank

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

Determinant

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

Solving linear problems

```
>>> linalg.solve(A,b)
```

```
>>> E = np.mat(a).T
```

```
>>> linalg.lstsq(F,E)
```

Generalized inverse

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

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

Frobenius norm

L1 norm (max column sum)

Linf norm (max row sum)

Transpose matrix

Conjugate transposition

Trace

Frobenius norm

L1 norm (max column sum)

Linf norm (max row sum)

Matrix rank

Determinant

Solver for dense matrices

Solver for dense matrices

Least-squares solution to linear matrix equation

Generalized inverse

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

Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

Inverse

```
>>> F = np.eye(3, k=1)
```

```
>>> G = np.mat(np.identity(2))
```

```
>>> C[C > 0.5] = 0
```

```
>>> H = sparse.csr_matrix(C)
```

```
>>> I = sparse.csc_matrix(D)
```

```
>>> J = sparse.dok_matrix(A)
```

```
>>> E.todense()
```

```
>>> sparse.isspmatrix_csc(A)
```

Create a 2x2 identity matrix

Create a 2x2 identity matrix

Compressed Sparse Row matrix

Compressed Sparse Column matrix

Dictionary Of Keys matrix

Sparse matrix to full matrix

Identify sparse matrix

Inverse

Norm

Solver for sparse matrices

Sparse Matrix Functions

Inverse

Norm

Solver for sparse matrices

Sparse matrix exponential

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
```

```
>>> np.info(np.matrix)
```

[Also see NumPy](#)

Matrix Functions

Addition

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

Subtraction

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

Division

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

Multiplication

```
>>> A @ D
```

Multiplication operator (python 3)

Multiplication

Dot product

Vector dot product

Inner product

Outer product

Tensor dot product

Kronecker product

Matrix exponential

Matrix exponential (Taylor Series)

Matrix exponential (eigenvalue decomposition)

Matrix logarithm

Matrix sine

Matrix cosine

Matrix tangent

Hyperbolic Trigonometric Functions

Hyperbolic matrix sine

Hyperbolic matrix cosine

Hyperbolic matrix tangent

Matrix Sign Function

Matrix sign function

Matrix square root

Matrix square root

Arbitrary Functions

Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors

Solve ordinary or generalized eigenvalue problem for square matrix

Unpack eigenvalues

First eigenvector

Second eigenvector

Unpack eigenvalues

Singular Value Decomposition

U,S,Vh = linalg.svd(B)

M,N = B.shape

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

LU Decomposition

F,L,U = linalg.lu(C)

LU Decomposition

Sparse Matrix Decompositions

Eigenvalues and eigenvectors

SVD

DataCamp

Learn Python for Data Science [Interactively](#)

Source — <https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs.JDSg3OI>

5. Matplotlib

Python For Data Science Cheat Sheet

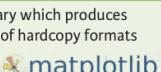
Matplotlib

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

2) Create Plot

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

Figure

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

Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()  
>>> ax1 = fig.add_subplot(221) # row-col-num  
>>> ax3 = fig.add_subplot(212)  
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)  
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3) Plotting Routines

1D Data

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

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 o

Vector Fields

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

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Data Distributions

```
>>> ax1.hist(y)  
>>> ax3.boxplot(y)  
>>> ax3.violinplot(z)
```

Plot a histogram
Make a box and whisker plot
Make a violin plot

2D Data or Images

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

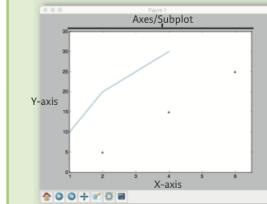
Colormapped or RGB arrays

```
>>> axes2[0].pcolor(data2)  
>>> axes2[0].pcolormesh(data)  
>>> CS = plt.contourf(Y,X,U)  
>>> axes2[2].contourf(data)  
>>> axes2[2].clabel(CS)
```

Pseudocolor plot of 2D array
Pseudocolor plot of 2D array
Plot contours
Plot filled contours
Label a contour plot

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare data
 - 2 Create plot
 - 3 Plot
 - 4 Customize plot
 - 5 Save plot
 - 6 Show plot
- ```
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(0, 6)
>>> y = [10,20,25,30] Step 1
>>> fig = plt.figure() Step 2
>>> ax = fig.add_subplot(111) Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3) Step 3.4
>>> ax.scatter([2,4,6],
 [5,15,25],
 color='darkgreen',
 marker='^')
>>> ax.set_xlim(1, 6.5) Step 4
>>> plt.savefig('foo.png') Step 5
>>> plt.show() Step 6
```

Figure

### 4) Customize Plot

#### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha=.4)
>>> ax.plot(x, y, c='r')
>>> 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")
>>> ax.setp(lines,color='r',linewidth=4.0)
```

#### LineStyles

```
>>> plt.plot(x,y,linewidth=4.0)
```

```
>>> plt.plot(x,y,ls='solid')
```

```
>>> plt.plot(x,y,ls='--')
```

```
>>> plt.plot(x,y,'-.',y**2,'.-')
```

```
>>> plt.setp(lines,color='r',linewidth=4.0)
```

#### Text & Annotations

```
>>> ax.text(1,-2,1,
 'Example Graph',
 style='italic')
>>> ax.annotate("Sine",
 xy=(8, 0),
 xycoords='data',
 xytext=(10.5, 0),
 textcoords='data',
 arrowprops=dict(arrowstyle="->",
 connectionstyle="arc3"))
```

#### Subplot Spacing

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

#### 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',
 xlabel='Y-Axis',
 ylabel='X-Axis')
>>> ax.legend(loc='best')
```

#### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
 ticklabels=[3,100,-12,"foo"])
```

```
>>> ax.tick_params(axis='y',
 direction='inout',
 length=10)
```

#### Subplot Spacing

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

#### 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.clf()
```

```
>>> plt.cla()
```

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

Clear an axis.

Clear the entire figure.

Close a window.

DataCamp

Learn Python for Data Science interactively

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](https://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', 'F', 'M', 'F', 'P', '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 = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

#### Normalization

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

#### Binarization

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

## Create Your Model

### Supervised Learning Estimators

**Linear Regression**  
>>> from sklearn.linear\_model import LinearRegression  
>>> lr = LinearRegression(normalize=True)  
**Support Vector Machines (SVM)**  
>>> from sklearn.svm import SVC  
>>> svc = SVC(kernel='linear')  
**Naive Bayes**  
>>> from sklearn.naive\_bayes import GaussianNB  
>>> gnb = GaussianNB()  
**KNN**  
>>> from sklearn import neighbors  
>>> knn = neighbors.KNeighborsClassifier(n\_neighbors=5)

### Unsupervised Learning Estimators

**Principal Component Analysis (PCA)**  
>>> from sklearn.decomposition import PCA  
>>> pca = PCA(n\_components=0.95)  
**K Means**  
>>> from sklearn.cluster import KMeans  
>>> k\_means = KMeans(n\_clusters=3, random\_state=0)

### Model Fitting

#### Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

#### Unsupervised Learning

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

Fit the model to the data

Fit to data, then transform it

### Prediction

#### Supervised Estimators

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

Predict labels

Predict labels

Estimate probability of a label

#### Unsupervised Estimators

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

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
```

Precision, recall, f1-score  
and support

#### Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
```

)

```
>>> print(confusion_matrix(y_test, y_pred))
```

Precision, recall, f1-score  
and support

### Regression Metrics

#### Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

#### Mean Squared Error

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

Mean squared error

#### R<sup>2</sup> Score

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

R<sup>2</sup> score

### Clustering Metrics

#### Adjusted Rand Index

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

adjusted rand score(y\_true, y\_pred)

#### Homogeneity

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

homogeneity\_score(y\_true, y\_pred)

#### V-measure

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

metrics.v\_measure\_score(y\_true, y\_pred)

metrics.v\_measure\_score(y\_true, y\_pred)

metrics.v\_measure\_score(y\_true, y\_pred)

### Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
```

cross\_val\_score(X\_train, y\_train, cv=4))

print(cross\_val\_score(knn, X\_train, y\_train, cv=4))

print(cross\_val\_score(lr, X, y, cv=2))

cross\_val\_score(X\_train, y\_train, cv=4))

print(cross\_val\_score(knn, X\_train, y\_train, cv=4))

print(cross\_val\_score(lr, X, y, cv=2))

### Tune Your Model

#### Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
```

params = {"n\_neighbors": np.arange(1,3), "metric": ["euclidean", "cityblock"]}

```
>>> grid = GridSearchCV(estimator=knn, param_grid=params)
```

grid.fit(X\_train, y\_train)

print(grid.best\_score\_)

print(grid.best\_estimator\_.n\_neighbors)

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
```

params = {"n\_neighbors": range(1,5), "weights": ["uniform", "distance"]}

```
>>> rsearch = RandomizedSearchCV(estimator=knn, param_distributions=params,
```

cv=5, n\_iter=8, random\_state=5)

rsearch.fit(X\_train, y\_train)

print(rsearch.best\_score\_)

DataCamp

Learn Python for Data Science interactively



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

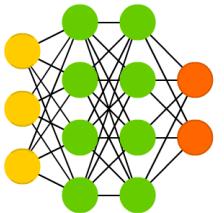
## 7. Neural Networks Zoo

A mostly complete chart of  
**Neural Networks**

©2016 Fjodor van Veen - [asimovinstitute.org](http://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

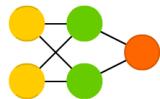
Deep Feed Forward (DFF)



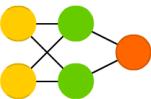
Perceptron (P)



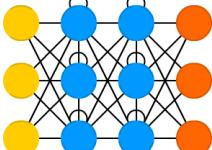
Feed Forward (FF)



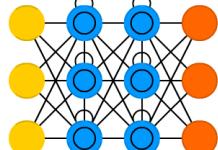
Radial Basis Network (RBF)



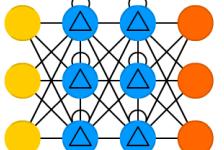
Recurrent Neural Network (RNN)



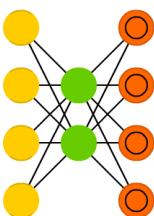
Long / Short Term Memory (LSTM)



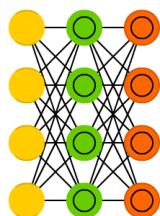
Gated Recurrent Unit (GRU)



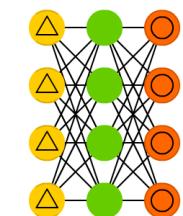
Auto Encoder (AE)



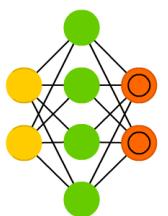
Variational AE (VAE)



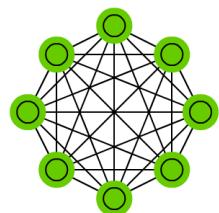
Denoising AE (DAE)



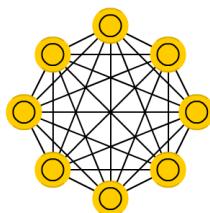
Sparse AE (SAE)



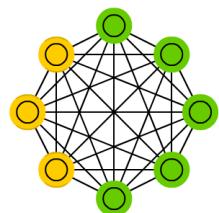
Markov Chain (MC)



Hopfield Network (HN)



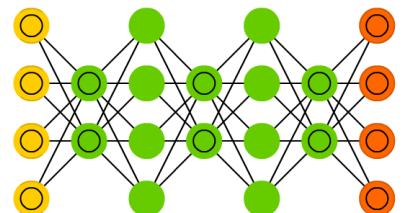
Boltzmann Machine (BM)



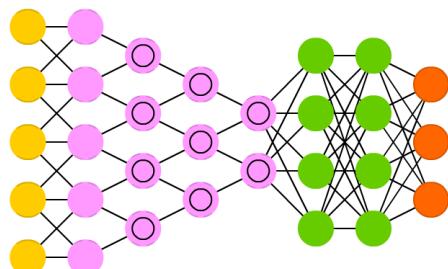
Restricted BM (RBM)



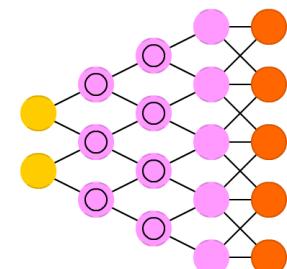
Deep Belief Network (DBN)



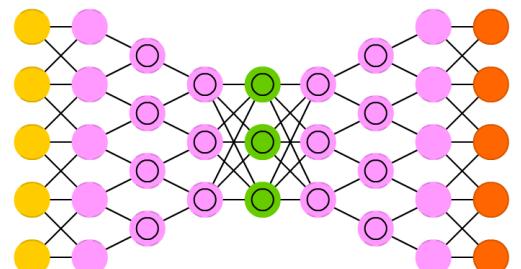
Deep Convolutional Network (DCN)



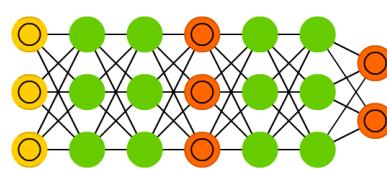
Deconvolutional Network (DN)



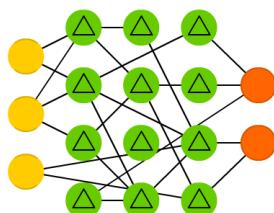
Deep Convolutional Inverse Graphics Network (DCIGN)



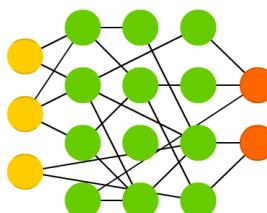
Generative Adversarial Network (GAN)



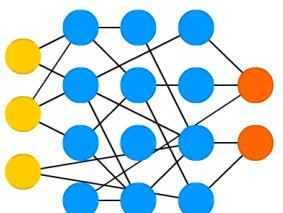
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



Echo State Network (ESN)



Deep Residual Network (DRN)



Kohonen Network (KN)

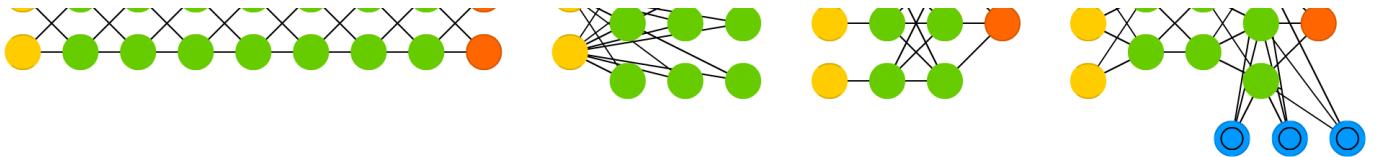


Support Vector Machine (SVM)



Neural Turing Machine (NTM)





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

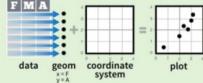
## 8. ggplot2

### Data Visualization with ggplot2 Cheat Sheet

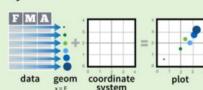


#### Basics

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()**

**qplot(x = cty, 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 = cty, y = hwy))**

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

**ggplot(mpg, aes(hwy, cty)) +  
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.

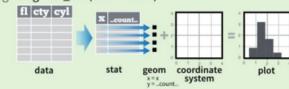
RStudio® is a trademark of RStudio, Inc. • [CC BY](http://creativecommons.org/licenses/by/3.0/) RStudio • [info@rstudio.com](mailto:info@rstudio.com) • 844-448-1212 • [rstudio.com](http://rstudio.com)

| Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>One Variable</b> <ul style="list-style-type: none"> <li><b>Continuous</b></li> </ul> <pre>a &lt;- ggplot(mpg, aes(hwy)) a + geom_area(stat = "bin") a + geom_density(kernel = "gaussian") a + geom_dotplot() a + geom_freqpoly() a + geom_histogram(binwidth = 5)</pre>                                                                                                                                                                                                                                             | <b>Two Variables</b> <ul style="list-style-type: none"> <li><b>Continuous X, Continuous Y</b></li> </ul> <pre>f &lt;- ggplot(mpg, aes(cty, hwy)) f + geom_blank() f + geom_jitter() f + geom_point() f + geom_quantile() f + geom_rug(sides = "bl") f + geom_smooth(model = lm) f + geom_text(aes(label = cyl))</pre> | <b>Continuous Bivariate Distribution</b> <pre>i + geom_bin2d(binwidth = c(5, 0.5)) i + geom_density2d() i + geom_hex()</pre>                                                                                                                                                  |
| <b>Discrete</b> <pre>b &lt;- ggplot(mpg, aes(f)) b + geom_bar()</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <b>AB</b>                                                                                                                                                                                                                                                                                                             | <b>Continuous Function</b> <pre>j &lt;- ggplot(economics, aes(date, unemploy)) j + geom_area() j + geom_line() j + geom_step(direction = "hv")</pre>                                                                                                                          |
| <b>Graphical Primitives</b> <pre>c &lt;- ggplot(map, aes(long, lat)) c + geom_polygon(aes(group = group)) d &lt;- ggplot(economics, aes(date, unemploy)) d + geom_path(lineend = "butt", linejoin = "round", linemt = 1) d + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)) e &lt;- ggplot(seals, aes(x = long, y = lat)) e + geom_segment(aes( xend = long + delta_long, yend = lat + delta_lat)) e + geom_rect(aes(xmin = long, ymin = lat, xmax = long + delta_long, ymax = lat + delta_lat))</pre> | <b>Discrete X, Continuous Y</b> <pre>g &lt;- ggplot(mpg, aes(class, hwy)) g + geom_bar(stat = "identity") g + geom_boxplot() g + geom_dotplot(binaxis = "y", stackdir = "center") g + geom_violin(scale = "area")</pre>                                                                                               | <b>Visualizing error</b> <pre>df &lt;- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2) k &lt;- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se)) k + geom_crossbar(fatten = 2) k + geom_errorbar() k + geom_linerange() k + geom_pointrange()</pre>              |
| <pre>h + geom_jitter() h + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE) h + geom_contour(aes(z = z)) h + geom_tile(aes(fill = z))</pre>                                                                                                                                                                                                                                                                                                                                                   | <b>Discrete X, Discrete Y</b> <pre>h &lt;- ggplot(diamonds, aes(cut, color)) h + geom_jitter()</pre>                                                                                                                                                                                                                  | <b>Maps</b> <pre>data &lt;- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests))) map &lt;- map_data("state") l &lt;- ggplot(data, aes(fill = murder)) l + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat)</pre> |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <b>Three Variables</b>                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                               |

Learn more at [docs.ggplot2.org](http://docs.ggplot2.org) • ggplot2 0.9.3.1 • Updated: 3/15

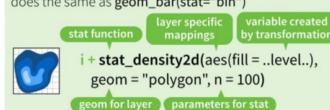
## Stats - An alternative way to build a layer

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.

stats 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..., density..., ..density...
a + stat_bindot(binwidth = 1, binaxis = "x")
x, y | ...count..., ..density...
a + stat_density2d(density = 1, kernel = "gaussian")
x, y | ...density..., ..scaled...
f + stat_bin2d(bins = 30, drop = TRUE) 2D distributions
x, y, fill | ...count..., ..density...
f + stat_hex(bins = 30)
x, y, fill | ...count..., ..density...
f + stat_contour(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, yend, y, xend | ...x..., ..yend...
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..., ..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(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(d = 5))
sample, x, y | ...x..., ..y...
f + stat_sum()
x, y, size | ..size...
f + stat_summary(fun.data = "mean_cl_boot")
f + stat_unique()
```

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

## 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()) - map discrete values to manually chosen visual values

### 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. Use `tz` argument to specify time zone.

`scale_x_datetime`() - treat x values as date times. 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                                                                                                         | Continuous                                                                                                                                                 |
| <code>n &lt;- b + geom_bar(aes(fill = f))</code>                                                                 | <code>o &lt;- a + geom_dotplot(aes(fill = x))</code>                                                                                                       |
| <code>n + scale_fill_brewer(palette = "Blues")</code><br>For palette choices: RColorBrewer::display.brewer.all() | <code>o + scale_fill_gradient(low = "red", high = "blue")</code>                                                                                           |
| <code>n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")</code>                                       | <code>o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 0.5)</code>                                                           |
|                                                                                                                  | <code>o + scale_fill_gradientn(colors = terrain.colors(6))</code><br>Albers rainbow, heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal() |

### Shape scales

|                                                                                                                     |                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| Manual shape values                                                                                                 | Shapes                                                                                                                          |
| <code>0 □ 6 ▽ 12 ▢ 18 ◆ 24 ▲</code>                                                                                 | <code>0 □ 1 ○ 2 △ 3 + 4 × 5 ◇ 6 ◁ 7 ▨ 8 * 9 ▹ 10 □ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code> |
| <code>1 ○ 7 ▨ 13 ▨ 19 ● 25 ▽</code>                                                                                 | <code>1 ○ 2 △ 3 + 4 × 5 ◇ 6 ◁ 7 ▨ 8 * 9 ▹ 10 □ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>     |
| <code>2 △ 8 * 14 ▨ 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>                                                                  | <code>3 + 4 × 5 ◇ 6 ◁ 7 ▨ 8 * 9 ▹ 10 □ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>             |
| <code>3 + 4 × 5 ◇ 6 ◁ 7 ▨ 8 * 9 ▹ 10 □ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code> | <code>3 + 4 × 5 ◇ 6 ◁ 7 ▨ 8 * 9 ▹ 10 □ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>             |
| <code>4 × 10 ▫ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>                         | <code>4 × 10 ▫ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>                                     |
| <code>5 ◇ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>                              | <code>5 ◇ 11 ▨ 12 ▨ 13 ▨ 14 ▨ 15 ▨ 16 ▨ 17 ▨ 18 ▨ 19 ● 20 ▪ 21 ▨ 22 ▨ 23 ▨ 24 ▨</code>                                          |

### Size scales

|                                                      |                                                            |
|------------------------------------------------------|------------------------------------------------------------|
| <code>q &lt;- f + geom_point(aes(size = cyl))</code> | <code>q + scale_size_area(max = 6, aes(size = cyl))</code> |
|------------------------------------------------------|------------------------------------------------------------|

## Coordinate Systems

`r <- b + geom_bar()`  
`coord_cartesian(xlim = c(0, 5), ylim = c(0, 5))`

The default cartesian coordinate system  
`coord_fixed(ratio = 1/2)`  
Cartesian coordinates with fixed aspect ratio between x and y units

`coord_flip()`  
Flipped Cartesian coordinates

`coord_polar(theta = "x", direction = 1)`  
Polar coordinates

`coord_trans(ytrans = "sqrt")`  
Transformed cartesian coordinates. Set extra and strains to the name of a window function.

`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(f, 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))`

## 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()`

|                                       |                                       |
|---------------------------------------|---------------------------------------|
| <code>t + facet_grid(~ f)</code>      | facet into columns based on f         |
| <code>t + facet_grid(~ .)</code>      | facet into rows based on year         |
| <code>t + facet_grid(year ~ f)</code> | facet into both rows and columns      |
| <code>t + facet_wrap(~ f)</code>      | wrap facets into a rectangular layout |

Set `scales` to let axis limits vary across facets

|                                                   |                                                 |
|---------------------------------------------------|-------------------------------------------------|
| <code>t + facet_grid(~ x, scales = "free")</code> | x and y axis limits adjust to individual facets |
| <code>• "free_x"</code>                           | - x axis limits adjust                          |
| <code>• "free_y"</code>                           | - y axis limits adjust                          |

Set `labeler` to adjust facet labels

|                                                                        |                              |                              |                              |                              |                              |
|------------------------------------------------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|
| <code>t + facet_grid(~ f, labeler = label_both)</code>                 |                              |                              |                              |                              |                              |
| <code>  f: c</code>                                                    | <code>  f: d</code>          | <code>  f: e</code>          | <code>  f: p</code>          | <code>  f: r</code>          |                              |
| <code>t + facet_grid(~ f, labeler = label_bquote(alpha ^ .(x)))</code> | <code>  α<sup>c</sup></code> | <code>  α<sup>d</sup></code> | <code>  α<sup>e</sup></code> | <code>  α<sup>p</sup></code> | <code>  α<sup>r</sup></code> |
| <code>t + facet_grid(~ f, labeler = label_parsed)</code>               | <code>c</code>               | <code>d</code>               | <code>e</code>               | <code>p</code>               | <code>r</code>               |

## 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

## 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.

## Themes

`r + theme_bw()`  
White background with grid lines

`r + theme_classic()`  
White background no gridlines

`r + theme_grey()`  
Grey background

`r + theme_minimal()`  
Minimal theme

`ggthemes - Package with additional ggplot2 themes`

## 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.ggpolt2.org](https://docs.ggpolt2.org) • ggpolt2 0.9.3.1 • Updated: 3/15

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](http://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

|                                                   |                                                 |
|---------------------------------------------------|-------------------------------------------------|
| <code>&gt;&gt;&gt; sc.version</code>              | Retrieve SparkContext version                   |
| <code>&gt;&gt;&gt; sc.pythonVer</code>            | Retrieve Python version                         |
| <code>&gt;&gt;&gt; sc.master</code>               | Master URL to connect to                        |
| <code>&gt;&gt;&gt; str(sc.sparkHome)</code>       | Path where Spark is installed on worker nodes   |
| <code>&gt;&gt;&gt; str(sc.sparkUser())</code>     | Username of the Spark User running SparkContext |
| <code>&gt;&gt;&gt; sc.appName</code>              | Return application name                         |
| <code>&gt;&gt;&gt; sc.applicationId</code>        | Retrieve application ID                         |
| <code>&gt;&gt;&gt; sc.defaultParallelism</code>   | Return default level of parallelism             |
| <code>&gt;&gt;&gt; sc.defaultMinPartitions</code> | 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
```

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

#### Loading Data

##### Parallelized Collections

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

##### External Data

Read either one text file from HDFS, a local file system or 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

|                                                                                                            |                                          |
|------------------------------------------------------------------------------------------------------------|------------------------------------------|
| <code>&gt;&gt;&gt; rdd.getNumPartitions()</code>                                                           | List the number of partitions            |
| <code>&gt;&gt;&gt; rdd.count()</code>                                                                      | Count RDD instances                      |
| <code>&gt;&gt;&gt; rdd.countByKey()</code>                                                                 | Count RDD instances by key               |
| <code>&gt;&gt;&gt; rdd.countByValue()</code>                                                               | Count RDD instances by value             |
| <code>&gt;&gt;&gt; rdd.collectAsMap()</code>                                                               | Return (key,value) pairs as a dictionary |
| <code>(['a': 2, 'b': 2) &gt;&gt;&gt; rdd3.sum() 4950 &gt;&gt;&gt; sc.parallelize([]).isEmpty() True</code> | Sum of RDD elements                      |
|                                                                                                            | Check whether RDD is empty               |

### Summary

|                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&gt;&gt;&gt; rdd3.max() 99 &gt;&gt;&gt; rdd3.min() 0 &gt;&gt;&gt; rdd3.mean() 49.5 &gt;&gt;&gt; rdd3.stdev() 28.866070047722118 &gt;&gt;&gt; rdd3.variance() 833.0 &gt;&gt;&gt; rdd3.histogram(3) ([0, 33, 66, 99], [33, 33, 34]) &gt;&gt;&gt; rdd3.stats()</code> | Maximum value of RDD elements<br>Minimum value of RDD elements<br>Mean value of RDD elements<br>Standard deviation of RDD elements<br>Compute variance of RDD elements<br>Compute histogram by bins<br>Summary statistics (count, mean, stdev, max & min) |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Applying Functions

|                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&gt;&gt;&gt; rdd.map(lambda x: x+(x[1],x[0])) ...     .collect() [('a',7,'a'), ('a',2,2,'a'), ('b',2,2,'b')] &gt;&gt;&gt; rdd5.collect() [('a',7,'a'), ('a',2,2,'a'), ('b',2,2,'b')] &gt;&gt;&gt; rdd4.flatMapValues(lambda x: x) ...     .collect() [('a','z'), ('a','y'), ('a','z'), ('b','p'), ('b','r')]</code> | Apply a function to each RDD element<br>Apply a function to each RDD element and flatten the result<br>Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Selecting Data

|                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&gt;&gt;&gt; rdd.collect() [('a', 7), ('a', 2), ('b', 2)] &gt;&gt;&gt; rdd.take(2) [('a', 7), ('a', 2)] &gt;&gt;&gt; rdd.first() ('a', 7) &gt;&gt;&gt; rdd.top(2) [('b', 2), ('a', 7)] Sampling &gt;&gt;&gt; rdd3.sample(False, 0.15, 81).collect() [3, 4, 27, 31, 40, 41, 42, 43, 60, 76, 79, 80, 86, 97]</code> | Return a list with all RDD elements<br>Take first 2 RDD elements<br>Take first RDD element<br>Take top 2 RDD elements<br>Return sampled subset of rdd3<br>Filter the RDD<br>Return distinct RDD values<br>Return (key,value) RDD's keys |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Iterating

|                                                                                                     |                                      |
|-----------------------------------------------------------------------------------------------------|--------------------------------------|
| <code>&gt;&gt;&gt; def g(x): print(x) &gt;&gt;&gt; rdd.foreach(g) ('a', 7) ('b', 2) ('a', 2)</code> | Apply a function to all RDD elements |
|-----------------------------------------------------------------------------------------------------|--------------------------------------|

## Reshaping Data

### Reducing

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

Merge the rdd values for each key

Merge the rdd values

### Grouping by

```
>>> rdd3.groupByKey().mapValues(list)
... .collect()
[('a',9), ('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)
```

Aggregate RDD elements of each partition and then the results

Aggregate values of each RDD key

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

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

### Subtract

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

## 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).

|    | 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  |
| .. | ...          | ...         | ...          | ...         | ...     |
| 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

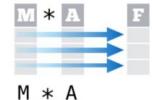
In a tidy data set:



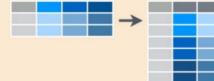
Each **variable** is saved in its own **column**

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).

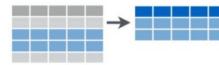
`dplyr::arrange(mtcars, desc(mpg))`

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.

### Logic in R - ?Comparison, ?base::Logic

|                    |                          |                                         |                   |
|--------------------|--------------------------|-----------------------------------------|-------------------|
| <                  | Less than                | <code>!=</code>                         | Not equal to      |
| >                  | Greater than             | <code>%in%</code>                       | Group membership  |
| <code>==</code>    | Equal to                 | <code>is.na</code>                      | Is NA             |
| <code>&lt;=</code> | Less than or equal to    | <code>!is.na</code>                     | Is not NA         |
| <code>&gt;=</code> | Greater than or equal to | <code>&amp;,  , !, xor, any, all</code> | Boolean operators |

Learn more with `browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.4.0 • tidy 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:

|                                   |                                 |
|-----------------------------------|---------------------------------|
| <code>dplyr::first</code>         | <code>min</code>                |
| First value of a vector.          | Minimum value in a vector.      |
| <code>dplyr::last</code>          | <code>max</code>                |
| Last value of a vector.           | Maximum value in a vector.      |
| <code>dplyr::nth</code>           | <code>mean</code>               |
| Nth value of a vector.            | Mean value of a vector.         |
| <code>dplyr::n</code>             | <code>median</code>             |
| # of values in a vector.          | Median value of a vector.       |
| <code>dplyr::n_distinct</code>    | <code>var</code>                |
| # of distinct values in a vector. | Variance of a vector.           |
| <code>IQR</code>                  | <code>sd</code>                 |
| IQR of a vector.                  | Standard deviation of a vector. |

### 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:

|                                  |                             |
|----------------------------------|-----------------------------|
| <code>dplyr::lead</code>         | <code>dplyr::cumall</code>  |
| Copy with values shifted by 1.   | Cumulative <b>all</b>       |
| <code>dplyr::lag</code>          | <code>dplyr::cumany</code>  |
| Copy with values lagged by 1.    | Cumulative <b>any</b>       |
| <code>dplyr::dense_rank</code>   | <code>dplyr::cummean</code> |
| Ranks with no gaps.              | Cumulative <b>mean</b>      |
| <code>dplyr::min_rank</code>     | <code>cumsum</code>         |
| Ranks. Ties get min rank.        | Cumulative <b>sum</b>       |
| <code>dplyr::percent_rank</code> | <code>cummax</code>         |
| Ranks rescaled to [0, 1].        | Cumulative <b>max</b>       |
| <code>dplyr::row_number</code>   | <code>cummin</code>         |
| Ranks. Ties got to first value.  | Cumulative <b>min</b>       |
| <code>dplyr::ntile</code>        | <code>cumprod</code>        |
| Bin vector into n buckets.       | Cumulative <b>prod</b>      |
| <code>dplyr::between</code>      | <code>pmax</code>           |
| Are values between a and b?      | Element-wise <b>max</b>     |
| <code>dplyr::cume_dist</code>    | <code>pmin</code>           |
| Cumulative distribution.         | Element-wise <b>min</b>     |

### Combine Data Sets

`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.

**Mutating Joins**

|                           |                      |
|---------------------------|----------------------|
| <code>x1   x2   x3</code> | <code>x1   x2</code> |
| A 1 T                     | A 1                  |
| B 2 F                     | B 2                  |
| C 3 NA                    | C 3                  |

**Filtering Joins**

|                      |                      |
|----------------------|----------------------|
| <code>x1   x2</code> | <code>x1   x2</code> |
| A 1                  | A 1                  |
| B 2                  | B 2                  |

**Set Operations**

|                      |                      |
|----------------------|----------------------|
| <code>y</code>       | <code>z</code>       |
| <code>x1   x2</code> | <code>x1   x2</code> |
| A 1                  | A 1                  |
| B 2                  | B 2                  |
| C 3                  | C 3                  |

`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**

|                      |                      |
|----------------------|----------------------|
| <code>x1   x2</code> | <code>x1   x2</code> |
| A 1                  | A 1                  |
| B 2                  | B 2                  |
| C 3                  | C 3                  |
| D 4                  | D 4                  |

`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.

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

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

Source — <https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

## About

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

## Big Announcement: Mateverse is in Public Beta

Tell us about more such awesome cheat sheets on Twitter or learn about more [by email](#).

