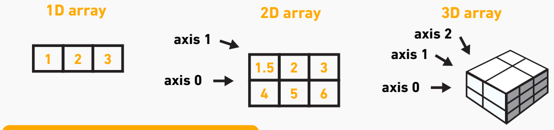
NumPy Basics Cheat Sheet

BecomingHuman.AI

DataCamp

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.



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],dtype = float)
```

Initial Placeholders

>>> np.zeros((3,4))	**Create an array of zeros**
>>> np.ones((2,3,4),dtype=np.int16)	**Create an array of ones**
>>> d = np.arange(10,25,5)	**Create an array of evenly spaced values (step value)**
>>> np.linspace(0,2,9)	**Create an array of evenly spaced values (number of samples)**
>>> e = np.full((2,2),7)	**Create a constant array**
>>> f = np.eye(2)	**Create a 2X2 identity matrix**
>>> np.random.random((2,2))	**Create an array with random values**
>>> np.empty((3,2))	**Create an empty array**

I/O

Saving & Loading On Disk

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

Saving & Loading Text Files

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

Inspecting Your Array

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

Data Types

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

Asking For Help

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

Array Mathematics

Arithmetic Operations

>>> g = a - b array([[-0.5, 0. , 0.], [-3. , -3. , -3.]])	**Subtraction**
>>> np.subtract(a,b)	**Subtraction**
>>> b + a array([[2.5, 4. , 6.], [5. , 7. , 9.]])	**Addition**
>>> np.add(b,a)	**Addition**
>>> a / b array([[0.66666667, 1. , 1.], [0.25 , 0.4 , 0.5]])	**Division**
>>> np.divide(a,b)	**Division**
>>> a * b array([[1.5, 4. , 9.], [4. , 10. , 18.]])	**Multiplication**
>>> np.multiply(a,b)	**Multiplication**
>>> np.exp(b)	**Exponentiation**
>>> np.sqrt(b)	**Square root**
>>> np.sin(a)	**Print sines of an array**
>>> np.cos(b)	**Element-wise cosine**
>>> np.log(a)	**Element-wise natural logarithm**
>>> e.dot(f) array([[7., 7.], [7., 7.]])	**Dot product**

Comparison

>>> a == b array([[False, True, True], [False, False, False]], dtype=bool)	**Element-wise comparison**
>>> a < 2 array([True, False, False], dtype=bool)	**Element-wise comparison**
>>> np.array_equal(a, b)	**Array-wise comparison**

Aggregate Functions

>>> a.sum()	**Array-wise sum**
>>> a.min()	**Array-wise minimum value**
>>> b.max(axis=0)	**Maximum value of an array row**
>>> b.cumsum(axis=1)	**Cumulative sum of the elements**
>>> a.mean()	**Mean**
>>> b.median()	**Median**

Copying Arrays

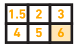
>>> h = a.view()	**Create a view of the array with the same data**
>>> np.copy(a)	**Create a copy of the array**
>>> h = a.copy()	**Create a deep copy of the array**

Sorting Arrays

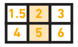
>>> a.sort()	**Sort an array**
>>> c.sort(axis=0)	**Sort the elements of an array's axis**

Subsetting, Slicing, Indexing

Subsetting

>>> a[2] 3		**Select the element at the 2nd index**
>>> b[1,2] 6.0		**Select the element at row 1 column 2 (equivalent to** b[1][2]**)**

Slicing

>>> a[0:2] array([1, 2])		**Select items at index 0 and 1**
>>> b[0:2,1] array([2., 5.])		**Select items at rows 0 and 1 in column 1**
>>> b[:1] array([[1.5, 2., 3.]])		**Select all items at row 0 (equivalent to** b[0:1, :]**)**
>>> c[1,...] array([[[3., 2., 1.], [4., 5., 6.]]])		**Same as** [1,:,:]
>>> a[: :-1] array([3, 2, 1])		**Reversed array** a

Boolean Indexing

>>> a[a<2] array([1])		**Select elements from** a **less than 2**

Fancy Indexing

>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] array([4. , 2. , 6. , 1.5])	**Select elements** (1,0),(0,1),(1,2) **and** (0,0)
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]] array([[4. ,5. , 6. , 4.], [1.5, 2. , 3. , 1.5], [4. , 5. , 6. , 4.], [1.5, 2. , 3. , 1.5]])	**Select a subset of the matrix's rows and columns**

Array Manipulation

Transposing Array

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

Changing Array Shape

>>> b.ravel()	**Flatten the array**
>>> g.reshape(3,-2)	**Reshape, but don't change data**

Adding/Removing Elements

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

Combining Arrays

>>> np.concatenate((a,d),axis=0) array([1, 2, 3, 10, 15, 20])	**Concatenate arrays**
>>> np.vstack((a,b)) array([[1. , 2. , 3.], [1.5, 2. , 3.], [4. , 5. , 6.]])	**Stack arrays vertically (row-wise)**
>>> np.r_[e,f]	**Stack arrays vertically (row-wise)**
>>> np.hstack((e,f)) array([[7., 7., 1., 0.], [7., 7., 0., 1.]])	**Stack arrays horizontally (column-wise)**
>>> np.column_stack((a,d)) array([[1, 10], [2, 15], [3, 20]])	**Create stacked column-wise arrays**
>>> np.c_[a,d]	**Create stacked column-wise arrays**

Splitting Arrays

>>> np.hsplit(a,3) [array([1]),array([2]),array([3])] index	**Split the array horizontally at the 3rd**
>>> np.vsplit(c,2) Split the array [array([[[1.5, 2. , 1.], [4. , 5. , 6.]]]),	**vertically at the 2nd index**

Bokeh Cheat Sheet

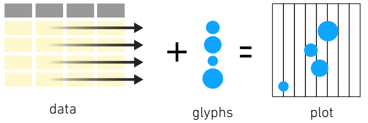
BecomingHuman.AI

DataCamp

Data Types

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

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



The basic steps to creating plots with the bokeh.plotting interface are:

1. **Prepare some data:**
   Python lists, NumPy arrays, Pandas DataFrames and other sequences of values
2. **Create a new plot**
3. **Add renderers for your data, with visual customizations**
4. **Specify where to generate the output**
5. **Show or save the results**

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]          step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",   step 2
               x_axis_label='x',
               y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)   step 3
>>> output_file("lines.html")   step 4
>>> show(p)   step 5
```

Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9,4,65, 'US'],
                                [32.4,4,66, 'Asia'],
                                [21.4,4,109, 'Europe']]),
                      columns=['mpg','cyl', 'hp', 'origin'],
                      index=['Toyota', 'Fiat', 'Volvo'])
```

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

Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
 x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

Show or Save Your Plots

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

Renderers & Visual Customizations

Glyphs

Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```

Line Glyphs

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

Rows & Columns Layout

Rows

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

Columns

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

Nesting Rows & Columns

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

Grid Layout

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

Customized Glyphs

Also see data

Selection and Non-Selection Glyphs

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

Hover Glyphs

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

Colormapping

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

Linked Plots

Also see data

Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

Linked Brushing

```
>>> p4 = figure(plot_width = 100, tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200, tools='box_select,lasso_select')
```

Tabbed Layout

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

Legends

Legend Location

Inside Plot Area

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

Outside Plot Area

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

Legend Orientation

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

Legend Background & Border

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

Output

Output to HTML File

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

Notebook Output

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

Standalone HTML

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

Components

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

Statistical Charts With Bokeh

Also see Data

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

Bar Chart

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

Box Plot

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

Histogram

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

Scatter Plot

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

Keras Cheat Sheet

BecomingHuman.AI

DataCamp

K Keras is a powerfuland 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'])
```

Data

Also see NumPy, Pandas & Scikit-Learn

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

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                                 mnist,
                                 cifar10,
                                 imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Other

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

Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

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

Multi-Class Classification

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

Regression

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

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

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

Model Fine-tuning

Optimization Parameters

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

Early Stopping

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

Compile Model

MLP: Binary Classification

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

MLP: Multi-Class Classification

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

MLP: Regression

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

Recurrent Neural Network

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

Save/ Reload Models

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

Inspect Model

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

Prediction

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

Model Training

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

Evaluate Your Model's Performance

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

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)
```

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)
```

Pandas Basics

Cheat Sheet

BecomingHuman.AI

DataCamp

Use the following import convention: >>> import pandas as pd

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

Pandas Data Structures

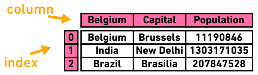
Series

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

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

Data Frame

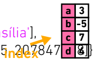
A two-dimensional labeled data structure with columns of potentially different types



column

index

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



Index

a	3
b	-5
c	7
d	4

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

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()	**Sort by labels along an axis**
>>> df.sort_values(by='Country')	**Sort by the values along an axis**
>>> df.rank()	**Assign ranks to entries**

Retrieving Series/ DataFrame Information

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

Summary

>>> df.sum()	**Sum of values**
>>> df.cumsum()	**Cummulative sum of values**
>>> df.min()/df.max()	**Minimum/maximum values**
>>> df.idxmin()/df.idxmax()	**Minimum/Maximum index value**
>>> df.describe()	**Summary statistics**
>>> df.mean()	**Mean of values**
>>> df.median()	**Median of values**

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5
```
Get one element

```
>>> df[1:]
    Country Capital      Population
 1  India     New Delhi  1303171035
 2  Brazil    Brasília   207847528
```
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        Brasília
  Population     207847528
```
Select single row of subset of rows

```
>>> df.ix[:,'Capital']
  0 Brussels
  1 New Delhi
  2 Brasília
```
Select a single column of subset of columns

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

Boolean Indexing

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

Setting

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

Asking For Help

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

Applying Functions

>>> f = lambda x: 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)
```

I/O

Read and Write to CSV

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

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> 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')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql()is **a convenience wrapper around** read_sql_table() and read_sql_query()

```
>>> pd.to_sql('myDf', engine)
```

Pandas
Cheat Sheet

BecomingHuman.AI

Pandas Data Structures

Pivot

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

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

→

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

Pivot Table

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

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

→

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

Melt

```
>>> pd.melt(df2,                                                Gather columns into rows
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

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

→

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

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()]                  Select cols with any vals >1
>>> df3.loc[:,(df3>1).all()]                     Select cols with vals > 1
>>> df3.loc[:,df3.isnull().any()]                    Select cols with NaN
>>> df3.loc[:,df3.notnull().all()]                Select cols without NaN
```

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]                    Find same elements
>>> df3.filter(items="a","b"])                           Filter on values
>>> df.select(lambda x: not x%5)                 Select specific elements
```

Where

```
>>> s.where(s > 0)                                        Subset the data
```

Query

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

Setting/Resetting Index

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

Reindexing

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

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
Country Capital Population
 0  Belgium   Brussels    11190846
 1  India     New Delhi   1303171035
 2  Brazil    Brasília    207847528
 3  Brazil    Brasília    207847528
```

Forward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
 0   3
 1   3
 2   3
 3   3
 4   3
```

MultiIndexing

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

Duplicate Data

```
>>> s3.unique()                                      Return unique values
>>> df2.duplicated('Type')                               Check duplicates
>>> df2.drop_duplicates('Type', keep='last')              Drop duplicates
>>> df.index.duplicated()                                 Drop duplicates
```

Grouping Data

Aggregation

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

Transformation

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

Missing Data

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

Combining Data

data1

X1	X2
a	11.432
b	1.303
c	99.906

data2

X1	X3
a	20.784
b	NaN
d	20.784

Pivot

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

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

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

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

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

X1	X2	X3
a	11.432	20.784
b	1.303	NaN

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

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

Join

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

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

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

Dates

```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1', periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

Visualization

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

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

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

Data Wrangling with pandas Cheat Sheet

BecomingHuman.AI

Syntax Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
            {"a" : [4 ,5, 6],
             "b" : [7, 8, 9],
             "c" : [10, 11, 12]},
index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
      [[4, 7, 10],
       [5, 8, 11],
       [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```

Specify values for each row.

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

```
df = pd.DataFrame(
             {"a" : [4 ,5, 6],
              "b" : [7, 8, 9],
              "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
            [('d',1),('d',2),('e',2)],
              names=['n','v']))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                    'variable' : 'var',
                    'value' : 'val'})
        .query('val >= 200')
)
```

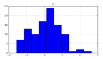
Windows

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

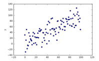
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.

Windows

df.plot.hist()
Histogram for each column

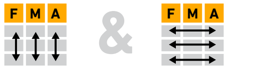


df.plot.scatter(x='w',y='h')
Scatter chart using pairs of points



Tidy Data A foundation for wrangling in pandas

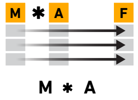
In a tidy data set:



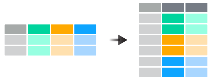
Each variable is saved in its own column

Each observation is saved in its own row

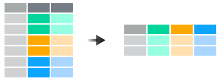
Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas



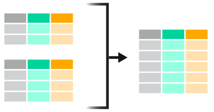
Reshaping Data Change the layout of a data set



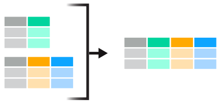
pd.melt(df)
Gather columns into rows.



df.pivot(columns='var', values='val')
Spread rows into columns.



pd.concat([df1,df2])
Append rows of DataFrames



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

df.sort_values('mpg')
Order rows by values of a column (low to high).

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

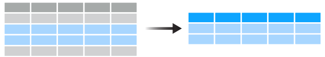
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

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

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

Subset Observations (Rows)



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

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

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

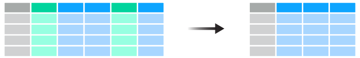
df.nlargest(n, 'value')
Select and order top n entries.

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

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equal to	pd.isnull(obj)	Is NaN
<=	Less than or equal to	pd.notnull(obj)	Is not NaN
>=	Greater than or equal to	&,|,~,^,df.any(),df.all(	Logical and, or, not, xor, any, all

Subset Variables (Columns)



df[['width','length','species']]
Select multiple columns with specific names.

df['width'] or df.width
Select single column with specific name.

df.filter(regex='regex')
Select columns whose name matches regular expression regex.

Logic in Python (and pandas)

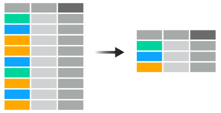
'\.'	Matches strings containing a period '.'
'Length$	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

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 .

Windows



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

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

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

size()
Size of each group.

agg(function)
Aggregate group using function.

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

shift(1)
Copy with values shifted by 1.

rank(method='dense')
Ranks with no gaps.

rank(method='min')
Ranks. Ties get min rank.

rank(pct=True)
Ranks rescaled to interval [0, 1].

rank(method='first')
Ranks. Ties go to first value.

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

cumsum()
Cumulative sum.

cummax()
Cumulative max.

cummin()
Cumulative min.

cumprod()
Cumulative product

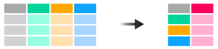
Summarise Data

df['w'].value_counts()
Count number of rows with each unique value of variable

len(df)
# of rows in DataFrame.

df['w'].nunique()
# of distinct values in a column.

df.describe()
Basic descriptive statistics for each column (or GroupBy)



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

sum()
Sum values of each object.

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

median()
Median value of each object.

quantile([0.25,0.75])
Quantiles of each object.

apply(function)
Apply function to each object

min()
Minimum value in each object.

max()
Maximum value in each object.

mean()
Mean value of each object.

var()
Variance of each object.

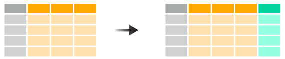
std()
Standard deviation of each object.

Handling Missing Data

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

df.fillna(value)

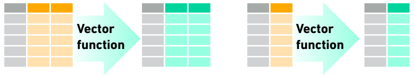
Make New Columns



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 allcolumns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

max(axis=1)
Element-wise max.

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

min(axis=1)
Element-wise min.

abs()
Absolute value.

Combine Data Sets

ydf

x1	x2
A	1
B	2
C	3

+ zdf

x1	x2
B	2
C	3
D	4

=

Set Operations

x1	x2
B	2
C	3

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

x1	x2
A	1
B	2
C	3
D	4

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

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
           indicator=True)
           .query('_merge == "left_only"')
          .drop(columns=['_merge'])
```

Rows that appear in ydf but not zdf (Setdiff)

adf

x1	x2
A	1
B	2
C	3

+ bdf

x1	x3
A	T
B	F
C	T

=

Standard Joins

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

```
dpd.merge(adf, bdf,
             how='left', on='x1')
```

Join matching rows from bdf to adf.

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

```
pd.merge(adf, bdf,
            how='right', on='x1')
```

Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

```
pd.merge(adf, bdf,
            how='inner', on='x1')
```

Join data. Retain only rows in both sets.

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

```
pd.merge(adf, bdf,
            how='outer', on='x1')
```

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

Data Wrangling with dplyr and tidyr

Cheat Sheet

BecomingHuman.AI

Syntax Helpful conventions for wrangling

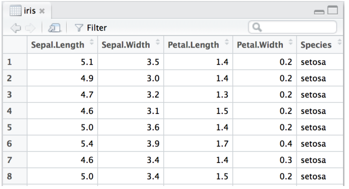
dplyr::tbl_df(iris)
Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen

```
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
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::%>%
Passes object on lef hand side as first argument (or . argument) of function on righthand side.

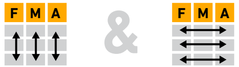
x %>% f(y) is the same as **f(x, y)**
y %>% f(x, ., z) is the same as **f(x, y, z)**

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

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

Tidy Data A foundation for wrangling in R

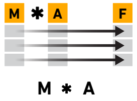
In a tidy data set:



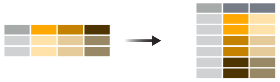
Each **variable** is saved in its own **column**

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

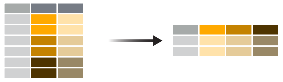
Tidy data complements R's vectorized operations. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R



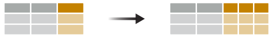
Reshaping Data Change the layout of a data set



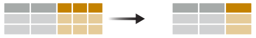
tidyr::gather(cases, "year", "n", 2:4)
Gather columns into rows.



tidyr::spread(pollution, size, amount)
Spread rows into columns



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



tidyr::unite(data, col, ..., sep)
Unite several columns into one.

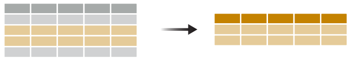
dplyr::data_frame(a = 1:3, b = 4:6)
Combine vectors into data frame (optimized).

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.

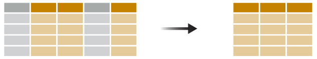
dplyr::sample_n(iris, 10, replace = TRUE)
Randomly select n rows.

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).

	Logic in R - ?	Comparison, ?base	::Logic
<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&,|,!,xor,any,all	Boolean operators

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).

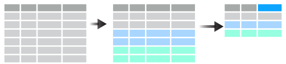
select(iris, -Species)
Select all columns except Species.

Group Data

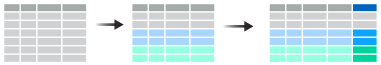
dplyr::group_by(iris, Species)
Group data into rows with the same value of Species.

dplyr::ungroup(iris)
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(...)
Compute separate summary row for each group.



iris %>% group_by(Species) %>% mutate(...)
Compute new variables by group.

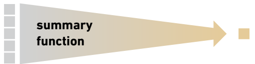


Summarise Data

dplyr::summarise(iris, avg = mean(Sepal.Length))
Summarise data into single row of values.

dplyr::summarise_each(iris, funs(mean))
Apply summary function to each column.

dplyr::count(iris, Species, wt = Sepal.Length)
Count number of rows with each unique value of variable (with or without weights).



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

dplyr::first
First value of a vector.

dplyr::last
Last value of a vector.

dplyr::nth
Nth value of a vector.

dplyr::n
# of values in a vector.

dplyr::n_distinct
# of distinct values in a vector.

IQR
IQR of a vector

min
Minimum value in a vector.

max
Maximum value in a vector.

mean
Mean value of a vector.

median
Median value of a vector.

var
Variance of a vector.

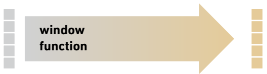
sd
Standard deviation of a vector.

Make New Variables

dplyr::mutate(iris, sepal = Sepal.Length + Sepal. Width)
Compute and append one or more new columns.

dplyr::mutate_each(iris, funs(min_rank))
Apply window function to each column.

dplyr::transmute(iris, sepal = Sepal.Length + Sepal. Width)
Compute one or more new columns. Drop original columns



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

dplyr::lead
Copy with values shifed by 1.

dplyr::lag
Copy with values lagged by 1.

dplyr::dense_rank
Ranks with no gaps.

dplyr::min_rank
Ranks. Ties get min rank.

dplyr::percent_rank
Ranks rescaled to [0, 1].

dplyr::row_number
Ranks. Ties got to first value.

dplyr::ntile
Bin vector into n buckets.

dplyr::between
Are values between a and b?

dplyr::cume_dist
Cumulative distribution.

dplyr::cumall
Cumulative all

dplyr::cumany
Cumulative any

dplyr::cummean
Cumulative mean

cumsum
Cumulative sum

cummax
Cumulative max

cummin
Cumulative min

cumprod
Cumulative prod

pmax
Element-wise max

pmin
Element-wise min

Combine Data Sets

A

x1	x2
A	1
B	2
C	3

+ B

x1	x2
A	T
B	F
C	T

=

Y

x1	x2
A	1
B	2
C	3

+ Z

x1	x2
B	2
C	3
D	4

=

Mutating Joins

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

dplyr::lef_join(a, b, by = "x1")
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
C	T	NA

dplyr::right_join(a, b, by = "x1")
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::inner_join(a, b, by = "x1")
Join data. Retain only rows in both sets.

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

dplyr::full_join(a, b, by = "x1")
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::semi_join(a, b, by = "x1")
All rows in a that have a match in b.

x1	x2
C	3

dplyr::anti_join(a, b, by = "x1")
All rows in a that do not have a match in b

Set Operations

x1	x2
B	2
C	3

dplyr::intersect(y, z)
Rows that appear in both y and z.

x1	x2
A	1
B	2
C	3
D	4

dplyr::union(y, z)
Rows that appear in either or both y and z.

x1	x2
A	1

dplyr::setdiff(y, z)
Rows that appear in y but not z.

Binding

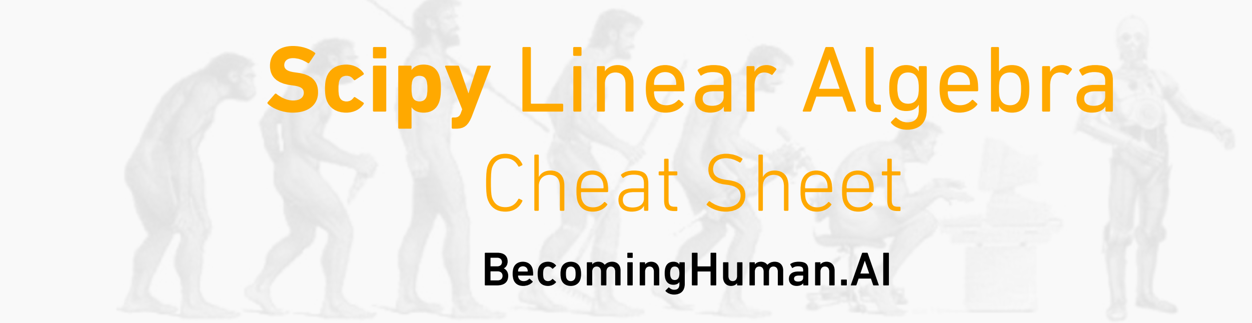
x1	x2
A	1
B	2
C	3
B	2
C	3
D	4

dplyr::bind_rows(y, z)
Append z to y as new rows.

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

dplyr::bind_cols(y, z)
Append z to y as new columns.
Caution: matches rows by position.

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.



Scipy Linear Algebra Cheat Sheet

BecomingHuman.AI

SciPy

DataCamp

Interacting With NumPy

Also see NumPy

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

Index Tricks

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

Shape Manipulation

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

Polynomials

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

Vectorizing Functions

```
>>> def myfunc(a):
      if a < 0:
        return a*2
      else:
        return a/2
```

>>> np.vectorize(myfunc)	**Vectorize functions**

Type Handling

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

Other Useful Functions

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

Linear Algebra

Also see NumPy

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

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

Creating Matrices

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

Basic Matrix Routines

Inverse

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

Transposition

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

Trace

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

Norm

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

Rank

>>> np.linalg.matrix_rank(C)	**Matrix rank**

Determinant

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

Solving linear problems

>>> linalg.solve(A,b)	**Solver for dense matrices**
>>> E = np.mat(a).T	**Solver for dense matrices**
>>> linalg.lstsq(F,E)	**Least-squares solution to linear matrix**

Generalized inverse

>>> linalg.pinv(C)	**Compute the pseudo-inverse of a matrix (least-squares solver)**
>>> linalg.pinv2(C)	**Compute the pseudo-inverse of a matrix (SVD)**

Creating Matrices

>>> F = np.eye(3, k=1)	**Create a 2X2 identity matrix**
>>> G = np.mat(np.identity(2))	**Create a 2x2 identity matrix**
>>> C[C > 0.5] = 0	
>>> H = sparse.csr_matrix(C)	**Compressed Sparse Row matrix**
>>> I = sparse.csc_matrix(D)	**Compressed Sparse Column matrix**
>>> J = sparse.dok_matrix(A)	**Dictionary Of Keys matrix**
>>> E.todense()	**Sparse matrix to full matrix**
>>> sparse.isspmatrix_csc(A)	**Identify sparse matrix**

Matrix Functions

Addition

>>> np.add(A,D)	**Addition**

Subtraction

>>> np.subtract(A,D)	**Subtraction**

Division

>>> np.divide(A,D)	**Division**

Multiplication

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

Exponential Functions

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

Logarithm Function

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

Trigonometric Functions

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

Hyperbolic Trigonometric Functions

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

Matrix Sign Function

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

Matrix Square Root

>>> linalg.sqrtm(A)	**Matrix square root**

Arbitrary Functions

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

Sparse Matrix Routines

Inverse

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

Norm

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

Solving linear problems

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

Sparse Matrix Functions

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

Decompositions

Eigenvalues and Eigenvectors

>>> la, v = linalg.eig(A)	**Solve ordinary or generalized**
>>> l1, l2 = la	**eigenvalue problem for square matrix**
>>> v[:,0]	**First eigenvector**
>>> v[:,1]	**Second eigenvector**
>>> linalg.eigvals(A)	**Unpack eigenvalues**

Singular Value Decomposition

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

LU Decomposition

>>> P,L,U = linalg.lu(C)	**LU Decomposition**

Sparse Matrix Decompositions

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

Asking For Help

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

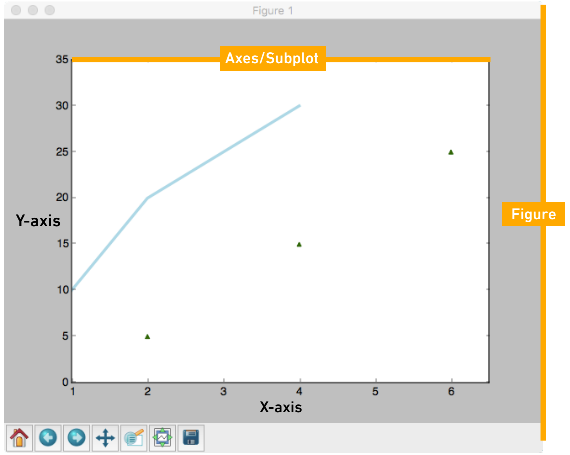
Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

Matplotlib Cheat Sheet

BecomingHuman.AI

Anatomy & Workflow

Plot Anatomy



Workflow

01 Prepare data

02 Create plot

03 Plot

04 Customize plot

05 Save plot

06 Show plot

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

Prepare The Data

Also see Lists & NumPy

Index Tricks

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

2D Data or Images

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

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)
```

Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

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

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

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

Mathtext

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

Limits, Legends & Layouts

Limits & Autoscaling

`>>> ax.margins(x=0.0,y=0.1)`	**Add padding to a plot**
`>>> ax.axis('equal')`	**Set the aspect ratio of the plot to 1**
`>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])`	**Set limits for x-and y-axis**
`>>> ax.set_xlim(0,10.5)`	**Set limits for x-axis**

Legends

`>>> ax.set(title='An Example Axes',`
`ylabel='Y-Axis',`
`xlabel='X-Axis')`
	Set a title and x-and y-axis labels

`>>> ax.legend(loc='best')`	**No overlapping plot elements**

Ticks

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

`direction='inout',`
`length=10)`
	Make y-ticks longer and go in and out

Subplot Spacing

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

Axis Spines

`>>> ax1.spines['top'=].set_visible(False)`	**Make the top axis line for a plot invisible**
`>>> ax1.spines['bottom'].set_position(('outward',10))`	**Move the bottom axis line outward**

Plotting Routines

1D Data

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

2D Data

`>>> fig, ax = plt.subplots()`	**Colormapped or RGB**

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

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

Vector Fields

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

Data Distributions

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

Save Plot

Save figures

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

Save transparent figures

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

Show Plot

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

Close & Clear

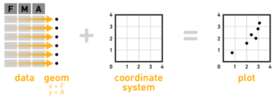
```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Content Copyright by DataCamp.com. Design Copyright by BecomingHuman.Ai. **See Original here.**

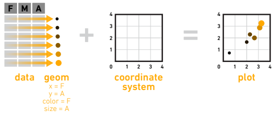
Data Visualisation with ggplot2 Cheat Sheet

Basics

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



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



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

aesthetic mappings · data · geom

qplot(x = 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().

data · add layers, elements with + · layer = geom + default stat + layer specific mappings · additional elements

```
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.

Coordinate Systems

r <- b + geo m_bar()

r + **coord_cartesian(**xlim = c(0, 5)**)**
xlim, ylim
The default cartesian coordinate system

r + **coord_fixed(**ratio = 1/2**)**
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + **coord_flip()**
xlim, ylim
Flipped Cartesian coordinates

r + **coord_polar(**theta = "x", direction=1 **)**
theta, start, direction
Polar coordinates

r + **coord_trans(**ytrans = "sqrt"**)**
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

z + **coord_map(**projection = "ortho", orientation=c(41, -74, 0)**)**
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Geoms Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + **geom_area(**stat = "bin"**)**
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + **geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + **geom_dotplot()**
x, y, alpha, color, fill

a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + **geom_histogram(**binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + **geom_bar()**
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + **geom_polygon(**aes(group = group)**)**
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

d + **geom_path(**lineend="butt", linejoin="round', linemitre=1)
x, y, alpha, color, linetype, size

d + **geom_ribbon(**aes(ymin=unemploy - 900, ymax=unemploy + 900)**)**
x, ymax, ymin, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

e + **geom_segment(**aes(
xend = long + delta_long,
yend = lat + delta_lat))
x, xend, y, yend, alpha, color, linetype, size

e + **geom_rect(**aes(xmin = long, ymin = lat,
xmax= long + delta_long,
ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

m + **geom_contour(**aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

m + **geom_raster(**aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

m + **geom_tile(**aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

Faceting

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

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

t + **facet_grid(. ~ fl)**
facet into columns based on fl

t + **facet_grid(year ~ .)**
facet into rows based on year

t + **facet_grid(year ~ fl)**
facet into both rows and columns

t + **facet_wrap(~ fl)**
wrap facets into a rectangular layoutof one or more discrete variables.

Set **scales** to let axis limits vary across facets
t + facet_grid(y ~ x, **scales = "free"**)
x and y axis limits adjust to individual facets
- "free_x" - x axis limits adjust
- "free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, **labeller = label_both**)

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

t + facet_grid(. ~ fl, **labeller = label_both**)

α^c	α^d	α^e	α^p	α^r

t + facet_grid(. ~ fl, **labeller = label_both**)

c	d	e	p	r

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + **geom_blank()**

f + **geom_jitter()**
x, y, alpha, color, fill, shape, size

f + **geom_point()**
x, y, alpha, color, fill, shape, size

f + **geom_quantile()**
x, y, alpha, color, linetype, size, weight

f + **geom_rug(**sides = "bl"**)**
x, y, alpha, color, linetype, size, weight

f + **geom_smooth(**model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + **geom_text(**aes(label = cty)**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + **geom_bar(**stat = "identity"**)**
x, y, alpha, color, fill, linetype, size, weight

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + **geom_violin(**scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + **geom_jitter()**
x, y, alpha, color, fill, linetype, size,

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + **geom_bin2d(**binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

i + **geom_hex()**
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + **geom_area()**
x, y, alpha, color, fill, linetype, size

j + **geom_line()**
x, y, alpha, color, linetype, size

j + **geom_step(**direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + **geom_crossbar(**fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + **geom_errorbar()**
x, ymax, ymin, alpha, color, linetype, size, width (also geom_errorbarh())

k + **geom_linerange()**
x, ymin, ymax, alpha, color, linetype, size

k + **geom_pointrange()**
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + **geom_map(**aes(map_id = state), map = map**) + expand_limits(**x = map$long, y = map$lat**)**
x, y, alpha, color, fill, linetype, size,

Position Adjustments

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

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

s + geom_bar(**position = "dodge"**)
Arrange elements side by side

s + geom_bar(**position = "fill"**)
Stack elements on top of one another, normalize height

s + geom_bar(**position = "stack"**)
Stack elements on top of one another

f + geom_point(**position = "jitter"**)
Add random noise to X and Y position of each element to avoid overplotting

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

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

Labels

t + ggtitle("New Plot Title ")
Add a main title above the plot

t + xlab("New X label")
Change the label on the X axis

t + ylab("New Y label")
Change the label on the Y axis

t + labs(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

Legends

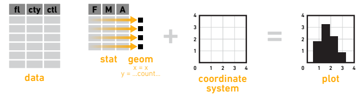
t + **theme(**legend.position = "bottom"**)**
Place legend at "bottom", "top", "left", or "right"

t + **guides(**color = "none"**)**
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

t + **scale_fill_discrete(**name = "Title", labels = c("A", "B", "C")**)**
Set legend title and labels with a scale function.

Stats An alternative way to build a layer

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



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax. stat functions and geom functions both combine a stat with a geom to make a layer, i.e. **stat_bin(geom="bar")** does the same as **geom_bar(stat="bin")**

stat function · layer specific mappings · variable created by transformation

i + **stat_density2d(**aes(fill = ..level..), geom = "polygon", n = 100**)**

geom for layer · parameters for stat

a + **stat_bin(**binwidth = 1, origin = 10**)**
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

a + **stat_bindot(**binwidth = 1, binaxis = "x"**)**
x, y, | ..count.., ..ncount..

a + **stat_density(**adjust = 1, kernel = "gaussian"**)**
x, y, | ..count.., ..density.., ..scaled..

f + **stat_bin2d(**bins = 30, drop = TRUE**)**
x, y, fill | ..count.., ..density..

f + **stat_binhex(**bins = 30**)**
x, y, fill | ..count.., ..density..

f + **stat_density2d(**contour = TRUE, n = 100**)**
x, y, color, size | ..level..

m + **stat_contour(**aes(z = z)**)**
x, y, z, order | ..level..

m+ **stat_spoke(**aes(radius= z, angle = z)**)**
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..

m + **stat_summary_hex(**aes(z = z), bins = 30, fun = mean**)**
x, y, z, fill | ..value..

g + **stat_boxplot(**coef = 1.5**)**
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)**
x, y | ..x.., ..y..

f + **stat_quantile(**quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq"**)**
x, y | ..quantile.., ..x.., ..y..

f + **stat_smooth(**method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95**)**
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

f + **stat_ecdf(n = 40)**
x, y | ..x.., ..y..

f + **stat_quantile(**quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq"**)**
x, y | ..quantile.., ..x.., ..y..

f + **stat_smooth(**method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95**)**
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + **stat_function(**aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5)**)**
x | ..y..

f + **stat_identity()**

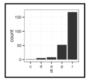
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..x.., ..y..

f + **stat_sum()**
x, y, size | ..size..

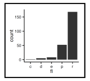
f + **stat_summary(**fun.data = "mean_cl_boot"**)**

f + **stat_unique()**

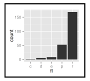
Themes



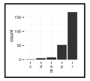
r + **theme_bw()**
White background with grid lines



r + **theme_classic()**
White background no gridlines



r + **theme_grey()**
Grey background (default theme)



r + **theme_minimal()**
Minimal theme

ggthemes - Package with additional ggplot2 themes

Scales

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

n <- b + **geom_bar(**aes(fill = fl)**)**
n

scale_ · aesthetic to adjust · prepackaged scale to use

n + **scale_fill_manual(**
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), **breaks** =c("d", "e", "p", "r"),
name = "fuel", **labels** = c("D", "E", "P", "R"))

range of values to include in mapping · title to use in legend/axis · labels to use in legend/axis · breaks to use in legend/axis

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. See ?strptime for label formats.

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

Use with x or y aesthetics (x shown here)

n <- b + **geom_bar(**
aes(fill = fl)**)**

n + **scale_fill_brewer(**
palette = "Blues"**)**
For palette choices:
library(RcolorBrewer)
display.brewer.all()

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

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

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

o + **scale_fill_gradient2(**
low = "red", hight = "blue",
mid = "white", midpoint = 25**)**

o + **scale_fill_gradientn(**
colours = terrain.colors(6)**)**
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape scales

p <- f + **geom_point(**
aes(shape = fl)**)**

p + **scale_shape(**
solid = FALSE)

p + **scale_shape_manual(**
values = c(3:7)**)**
Shape values shown in chart on right

0 □	6 ▽	12 ⊞	18 ◆	24 △
1 ○	7 ⊠	13 ⊗	19 ●	25 ▽
2 △	8 ✳	14 ◬	20 •	* ✱
3 +	9 ⟐	15 ■	21 ●	. ▪
4 ×	10 ⊕	16 ●	22 ■	0 O
5 ◇	11 ✡	17 ▲	23 ◆	0 O

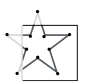
Size scales

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

q + **scale_size_area(**max = 6**)**
Value mapped to area of circle (not radius)

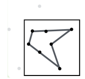
Zooming

Without clipping (preferred)



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

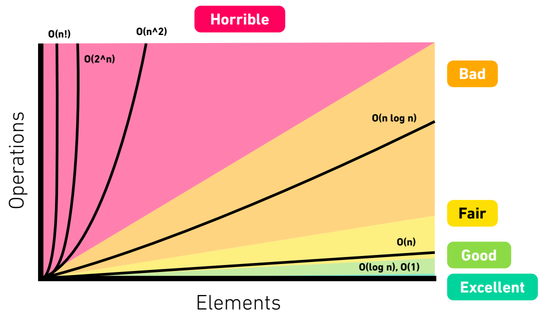
With clipping (removes unseen data points)



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

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

Big-O Complexity Chart



Big-O Cheat Sheet

BecomingHuman.AI

Data Structure Operation

	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Θ(n)	Θ(n)	Θ(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)	Θ(n)	Θ(n)	Θ(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	Θ(n)	Θ(n)	Θ(n)	Θ(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)	Θ(n)	Θ(n)	Θ(n)	Θ(n)
Cartesian Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	Θ(n)	Θ(n)	Θ(n)	Θ(n)
B-Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(n)	Θ(n)	Θ(n)	Θ(n)	Θ(n)

Array Sorting Algorithms

	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n log(n))
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	Θ(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	Θ(n)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	Θ(n)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	Ω(n^2)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n log(n))
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(n log(n))
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	Ω(n+k)
Radix Sort	Ω(n+k)	Θ(n+k)	Ω(n+k)	Ω(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	Ω(n+k)	Ω(n+k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n log(n))

Originally created by bigocheatsheet.com **See original here.**