Data Wrangling

with pandas Cheat Sheet

http://pandas.pydata.org

Pandas API Reference Pandas User Guide

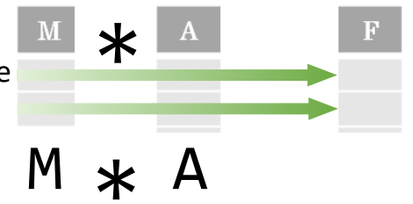
Tidy Data – A foundation for wrangling in pandas

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.



Creating DataFrames

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

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

Specify values for each column.

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

Specify values for each row.

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

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

Create DataFrame with a MultiIndex

Method Chaining

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

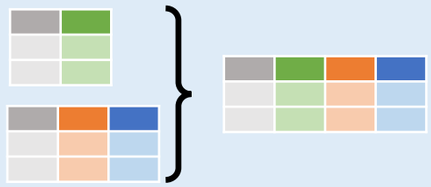
```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

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

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

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

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

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*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use **df.loc[]** and **df.iloc[]** to select only rows, only columns or both.
Use **df.at[]** and **df.iat[]** to access a single value by row and column.
First index selects rows, second index columns.

df.iloc[10:20]
Select rows 10-20.

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

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

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

df.iat[1, 2] Access single value by index

df.at[4, 'A'] Access single value by label

Logic in Python (and pandas)

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

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'

Cheatsheet for pandas (http://pandas.pydata.org/ originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

Summarize Data

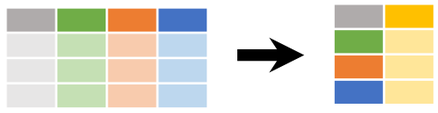
`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df.shape`
Tuple of # of rows, # of columns in DataFrame.

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

`df.describe()`
Basic descriptive and 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.

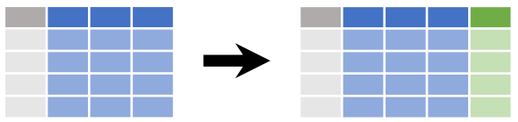
`std()`
Standard deviation of each object.

Handling Missing Data

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

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

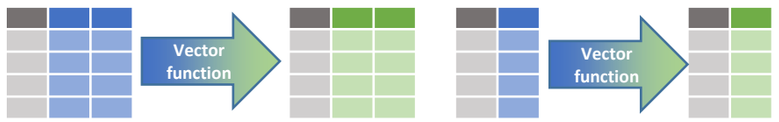
Make New Columns



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

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

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



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

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

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

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

`abs()`
Absolute value.

Group Data

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

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

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

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

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

`shift(1)`
Copy with values shifted by 1.

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

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

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

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

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

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

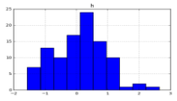
Windows

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

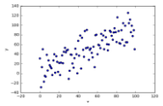
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

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



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



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

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

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

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

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

x1	x2	x3
A	1	T
B	2	F

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

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

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

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

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

x1	x2
A	1
B	2
C	3
D	4

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

x1	x2
A	1

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

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

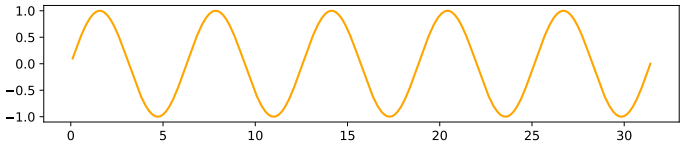
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

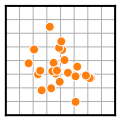
4 Observe



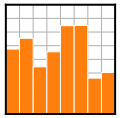
Choose

Matplotlib offers several kind of plots (see Gallery):

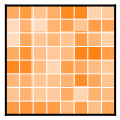
```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

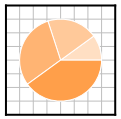


```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

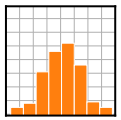


```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

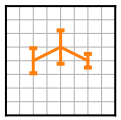
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```



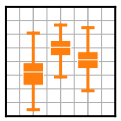
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



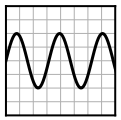
```
Z = np.random.normal(0, 1, (100,3))

ax.boxplot(Z)
```



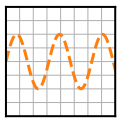
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

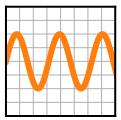
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



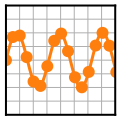
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



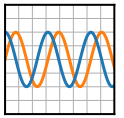
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



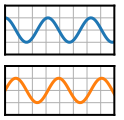
Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named *Axes*):

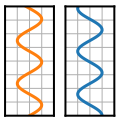
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots(2,1)
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

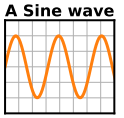


```
fig, (ax1, ax2) = plt.subplots(1,2)
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

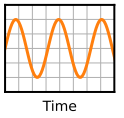


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Cheatography

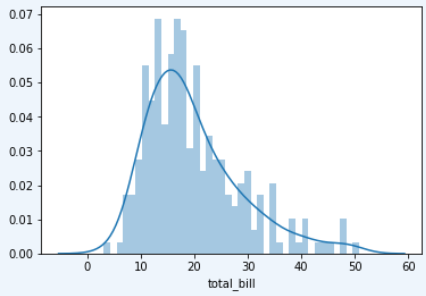
Python - Seaborn IMGs Cheat Sheet

by DarioPittera (aggialavura) via cheatography.com/83764/cs/19859/

TO START

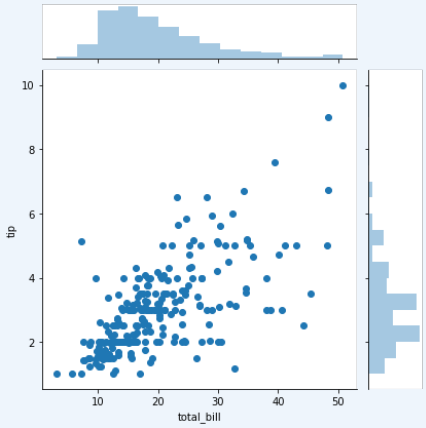
```
import seaborn as sns
%matplotlib inline
```

distplot()



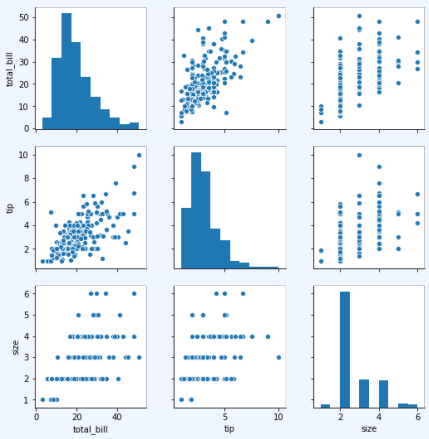
sns.distplot(tips['total_bill'], bins = 40)

jointplot()



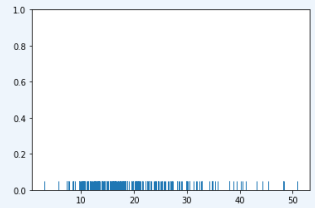
sns.jointplot(x='total_bill',y='tip',data=tips)

pairplot()



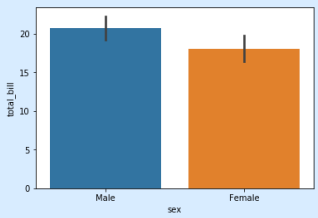
sns.pairplot(tips)

rugplot()



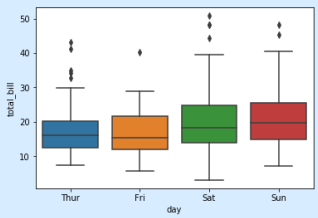
sns.rugplot(tips['total_bill'])

barplot()



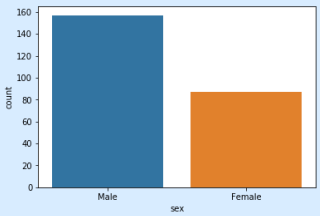
sns.barplot(x='sex',y='total_bill',data=tips)

boxplot()



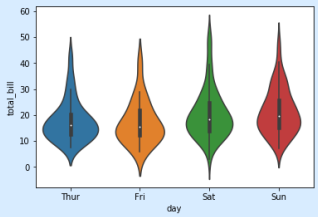
sns.boxplot(x='day',y='total_bill',data = tips)

countplot()



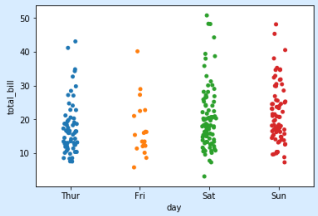
sns.countplot(x='sex',data=tips)

violinplot()



sns.violinplot(x="day", y="total_bill", data=tips)

stripplot()



sns.stripplot(x="day", y="total_bill", data=tips)

By **DarioPittera** (aggialavura)

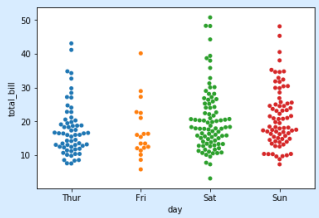
cheatography.com/aggialavura/

www.dariopittera.com

Not published yet.

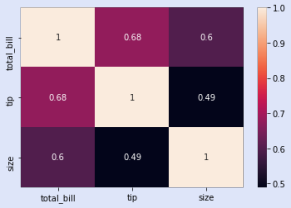
Last updated 24th June, 2019.

swarmplot()



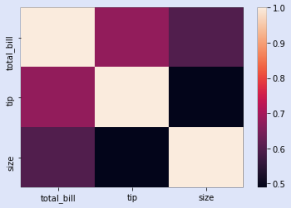
sns.swarmplot(x="day", y="total_bill", data=tips)

heatmap() with values



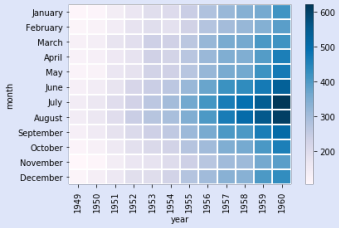
sns.heatmap(tips.corr(),**annot=True**)

heatmap()



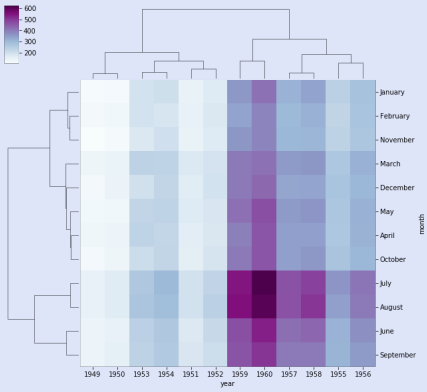
sns.heatmap(tips.corr())

heatmap(linecolor=', linewidth=x')



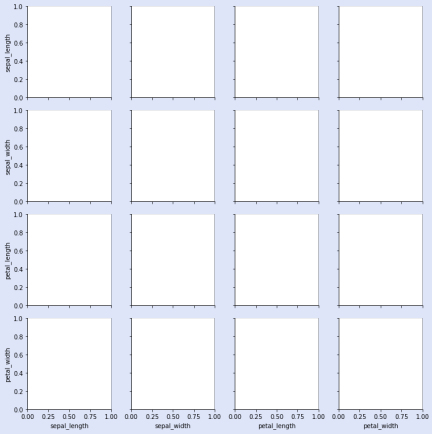
sns.heatmap(matrix,cmap='PuBu',linecolor='white',linewidths=1)

clustermap()



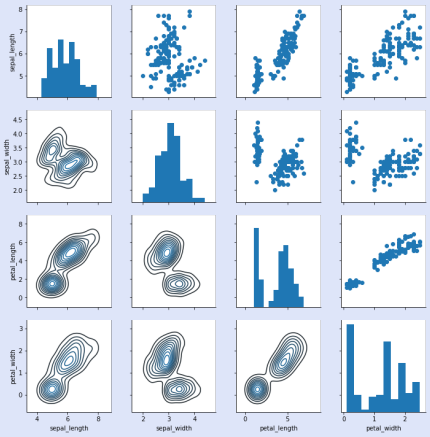
sns.clustermap(matrix, cmap='BuPu')

PairGrid()



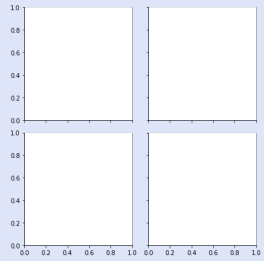
sns.PairGrid(df)

PairGrid() with options



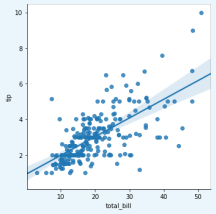
g = sns.PairGrid(df)
g.map_diag(plt.hist)
g.map_upper(plt.scatter)
g.map_lower(sns.kdeplot)

FacetGrid()



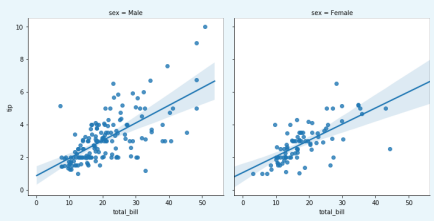
g = sns.FacetGrid(data=tips, col='time', row='smoker')

lmplot()



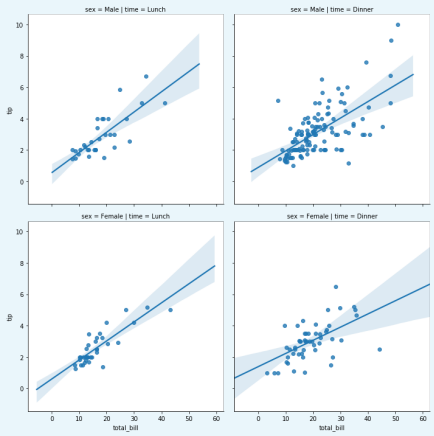
sns.lmplot(x='total_bill',y='tip',data=tips)

lmplot(col='')



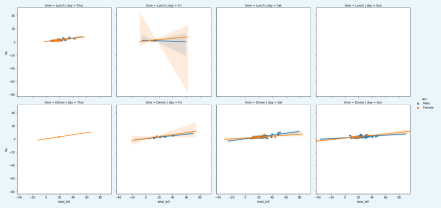

```
sns.lmplot(x='total_bill',y='tip',data=tips,c-
ol='sex')
```

lmplot(col='', row='')




```
sns.lmplot(x="total_bill", y="tip", row="sex",
col="time",data=tips)
```

lmplot(col='', row='', hue='')



```
sns.lmplot(x='total_bill',y='tip',data=tips,c-
ol='day', row='time', hue='sex')
```

datacamp

Python For Data Science
Scikit-Learn Cheat Sheet

Learn Scikit-Learn online at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

> Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

> Training And Test Data

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

> Model Fitting

Supervised learning

```
>>> lr.fit(X, y) #Fit the model to the data
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Unsupervised Learning

```
>>> k_means.fit(X_train) #Fit the model to the data
>>> pca_model = pca.fit_transform(X_train) #Fit to data, then transform it
```

> Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5))) #Predict labels
>>> y_pred = lr.predict(X_test) #Predict labels
>>> y_pred = knn.predict_proba(X_test) #Estimate probability of a label
```

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test) #Predict labels in clustering algos
```

> Preprocessing The Data

Standardization

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

Normalization

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

Binarization

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

Encoding Categorical Features

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

Imputing Missing Values

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

Generating Polynomial Features

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

> 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)
```

> Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test) #Estimator score method
>>> from sklearn.metrics import accuracy_score #Metric scoring functions
>>> accuracy_score(y_test, y_pred)
```

Classification Report

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

Confusion Matrix

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

Regression Metrics

Mean Absolute Error

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

Mean Squared Error

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

R^2 Score

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

Clustering Metrics

Adjusted Rand Index

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

Homogeneity

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

V-measure

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

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

> Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5), "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn, param_distributions=params,
                                 cv=4, n_iter=8, random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Cheatography

Network Analysis with Python and NetworkX Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15946/

Basic graph manipulation

`import networkx as nx`	
`G=nx.G raph()`	
`G=nx.M ult iGr aph()`	Create a graph allowing parallel edges
`G.add_ edg es_ fro m([(0, 1),(0, 2),(1, 3),(2, 4)]`	Create graph from edges
`nx.dra w_n etw orkx(G)`	Draw the graph
`G.add_ nod e(' A', rol e=' man ager')`	Add a node
`G.add_ edg e(' A', 'B' ,re lation = 'friend')`	Add an edge
`G.node ['A '][ 'role'] = 'team member'`	Set attribute of a node
`G.node ['A'],G.edge [(' A', 'B')]`	View attributes of node, edge
`G.edges(),G.nodes()`	Show edges, nodes
`list(G.ed ges())`	Return as list instead of EdgeView class
`G.node s(d ata =True),G.edge s(d ata =True)`	Include node/edge attributes
`G.node s(d ata ='r ela tion)`	Return specific attribute

Creating graphs from data

Bipartite graphs (cont)

`bipart ite.se ts(B)`	Get each set of nodes of bipartite graph
`bipart ite.pr oje cte d_g raph(B, X)`	Bipartite projected graph - nodes with bipartite friends in common
`P=bipa rti te.w ei ght ed_ pro jec ted _gr aph(B, X)`	projected graph with weights (number of friends in common)

Network Connectivity

`G=nx.r ead _ad jli st( 'G_ adj lis t.txt', nodety pe=int)`	Create from adjacency list
`G=nx.G rap h(G _mat)`	Create from matrix (np.array)
`G=nx.r ead _ed gel ist ('G _ed gel ist.txt', data=[ ('W eight', int)])`	Create from edgelist
`G=nx.f rom _pa nda s_d ata fra me( G_df, 'n1', 'n2', edge_a ttr ='w eight')`	Create from df

Adjacency list format
0 1 2 3 5
1 3 6 ...

Edgelist format:
0 1 14
0 2 17

Bipartite graphs

`from networ kx.a lg orithms import bipartite`

`bipart ite.is _bi par tite(B)`	Check if graph B is bipartite
`bipart ite.is _bi par tit e_n ode _se - t(B ,set)`	Check if set of nodes is bipartition of graph

`nx.clu ste rin g(G, node)`	Local clustering coefficient
`nx.ave rag e_c lus ter ing(G)`	Global clustering coefficient
`nx.tra nsi tiv ity(G)`	Transitivity (% of open triads)
`nx.sho rte st_ pat h(G ,n1,n2)`	Outputs the path itself
`nx.sho rte st_ pat h_l eng th( G,n 1,n2)`	
`T=nx.b fs_ tree(G, n1)`	Create breadth-first search tree from node n1
`nx.ave rag e_s hor tes t_p ath _le ngth(G)`	Average distance between all pairs of nodes
`nx.dia met er(G)`	Maximum distance between any pair of nodes
`nx.ecc ent ric ity(G)`	Returns each node's distance to furthest node
`nx.rad ius(G)`	Minimum eccentricity in the graph
`nx.per iph ery(G)`	Set of nodes where eccentricity=diameter
`nx.cen ter(G)`	Set of nodes where eccentricity=radius

Connectivity: Network Robustness

`nx.nod e_c onn ect ivi ty(G)`	Min nodes removed to disconnect a network
`nx.min imu m_n ode _cut()`	Which nodes?
`nx.edg e_c onn ect ivi ty(G)`	Min edges removed to disconnect a network
`nx.min imu m_e dge _cut(G)`	Which edges?
`nx.all _si mpl e_p ath s(G , n1,n2)`	Show all paths between two nodes

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 4th June, 2018.
Last updated 4th June, 2018.

Cheatography

Network Analysis with Python and NetworkX Cheat Sheet

by RJ Murray (murenei) via cheatography.com/58736/cs/15946/

Network Connectivity: Connected Components

`nx.is_ con nec ted(G)`	Is there a path between every pair of nodes?
`nx.num ber _co nne cte d_c omp on e nts(G)`	# separate components
`nx.nod e_c onn ect ed_ com pon en t(G, N)`	Which connected component does *N* belong to?
`nx.is_ str ong ly_ con nec ted(G)`	Is the network connected direction-ally?
`nx.is_ wea kly _co nne cted(G)`	Is the directed network connected if assumed undire-cted?

Common Graphs

`G=nx.k ara te_ clu b_g raph ()`	Karate club graph (social network)
`G=nx.p ath _gr aph(n)`	Path graph with n nodes
`G=nx.c omp let e_g raph(n)`	Complete graph on n nodes
`G=rand om_ reg ula r_g rap - h(d,n)`	Random d-regular graph on n-nodes

See NetworkX Graph Generators reference for more.
Also see “An Atlas of Graphs” by Read and Wilson (1998).

Influence Measures and Network Centralization

Influence Measures and Network Centralization (cont)

`nx.edg e_b etw een nes s_c ent ral ity(G)`	B e
`nx.edg e_b etw een nes s_c ent ral ity _su bse t( G ,{s ubset})`	B s o e

Normalization: Divide by number of pairs of nodes.

PageRank and Hubs & Authorities Algorithms

`nx.pag era nk(G, alpha=0.8)`	Scaled PageRank of G with dampening parameter
`h,a=nx.hi ts(G)`	HITS algorithm - outputs 2 dictio-naries (hubs, authorities)
`h,a=nx.hi ts( G,m ax_ ite r=1 0,n orm ali - zed =True)`	Constr-ained HITS and normalized by sum at each stage

Centrality measures make different assumptions about what it means to be a “central” node. Thus, they produce different rankings.

Network Evolution - Real-world Applications

`G.degree(),G.in_d egree(),G.out_ deg ree()`	Distributio of node degrees
Preferential Attachment Model	Results in power law -> many nodes with low degrees; few with high degrees

`dc=nx.d eg ree _ce ntr ali ty(G)`	Degree centrality for network
`dc[node]`	Degree centrality for a node
`nx.in_ deg ree _ce ntr ali ty(G),nx.out _de g re e_c ent ral ity(G)`	DC for directed networks
`cc=nx.c lo sen ess _ce ntr ali ty( G,n orm al i zed =True)`	Closeness centrality (normalised) for the network
`cc[node]`	Closeness centrality for an individual node
`bC=nx.b et wee nne ss_ cen tra lity(G)`	Betweenness centrality
`..., normal ize d=T rue ,...)`	Normalized betweenness centrality
`..., endpoi nts =False, ...)`	BC excluding endpoints
`..., K=10,...)`	BC approximated using random sample of K nodes
`nx.bet wee nne ss_ cen tra lit y_s ubs et( G, { sub set})`	BC calculated on subset

`G=bara bas i_a lbe rt_ gra ph(n,m)`	Preferential Attachment Model with *n* nodes and each new node attaching to *m* existing nodes
Small World model	High average degree (global clustering) and low average shortest path
`G=watt s_s tro gat z_g rap h(n ,k,p)`	Small World network of *n* nodes, connected to its *k* nearest neighbours, with chance *p* of rewiring
`G=conn ect ed_ wat ts_ str oga tz_ gra ph( n,k,p, t)`	*t* = max iterations to try to ensure connected graph
`G=newm an_ wat ts_ str oga tz_ gra ph( n,k ,p)`	*p* = probability of adding (no rewiring)
Link Prediction measures	How likely are 2 nodes to connect, given an existing network

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 4th June, 2018.
Last updated 4th June, 2018.

Network Evolution - Real-world Applications (cont)	
`nx.com mon _ne igh bor s(G ,n1,n2)`	Calc common neighbors of nodes *n1*, *n2*
`nx.jac car d_c oef fic ient(G)`	Normalised common neighbors measure
`nx.res our ce_ all oca tio n_i ndex (G)`	Calc RAI of all nodes not already connected by an edge
`nx.ada mic _ad ar_ ind ex(G)`	As per RAI but with log of degree of common neighbor
`nx.pre fer ent ial _at tac hme nt(G )`	Product of two nodes' degrees
Community Common Neighbors	Common neighbors but with bonus if they belong in same 'community'
`nx.cn_ sou nda raj an_ hop cro ft(n 1, n2)`	CCN score for *n1*, *n2*
`G.node ['A '][ 'co mmu nit y']=1`	Add community attribute to node
`nx.ra_ ind ex_ sou nda raj an_ hop - cro ft(G)`	Community Resource Allocation score

These scores give only an indication of whether 2 nodes are likely to connect.
To make a link prediction, you would use these scores as features in a classification ML model.

By **RJ Murray** (murenei)
cheatography.com/murenei/
tutify.com.au

Published 4th June, 2018.
Last updated 4th June, 2018.

Python For Data Science *Cheat Sheet*

NumPy Basics

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

NumPy

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

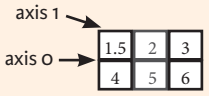
Use the following import convention:

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

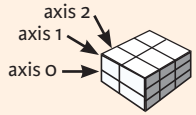
NumPy Arrays

1D array

2D array



3D array



Creating Arrays

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

Initial Placeholders

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

I/O

Saving & Loading On Disk

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

Saving & Loading Text Files

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

Data Types

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

Inspecting Your Array

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

Asking For Help

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

Array Mathematics

Arithmetic Operations

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

Comparison

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

Aggregate Functions

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

Copying Arrays

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

Sorting Arrays

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

Subsetting, Slicing, Indexing

Also see Lists

Subsetting


```
>>> a[2]
  3
```

Select the element at the 2nd index


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

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

Slicing


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

Select items at index 0 and 1


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

Select items at rows 0 and 1 in column 1


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

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

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

Same as `[1,:,:]`

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

Reversed array `a`

Boolean Indexing


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

Select elements from `a` less than 2

Fancy Indexing

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

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

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

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

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

Changing Array Shape

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

Adding/Removing Elements

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

Combining Arrays

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

Splitting Arrays

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

Cheatography

Essential Python Cheat Sheet

by sschaub via cheatography.com/1000/cs/374/

Data Types

Integer	-256, 15
Float	-253.23, 1.253e-10
String	"Hello", 'Goodbye', """Multiline"""
Boolean	True, False
List	[*value*, ...]
Tuple	(*value*, ...)[1]
Dictionary	{ *key*: *value*, ... }
Set	{ *value*, *value*, ... }[2]

[1] Parentheses usually optional

[2] Create an empty set with set()

Statements

If Statement

if *expression*:
  statements
elif *expression*:
  statements
else:
  statements

While Loop

while *expression*:
  statements

For Loop

for *var* in *collection*:
  statements

Counting For Loop

for i in range(*start*, *end* [, *step*]):
  statements

(*start* is included; *end* is not)

Arithmetic Operators

x + y	add	x - y	subtract
x * y	multiply	x / y	divide
x % y	modulus	x ** y	x^y

Assignment shortcuts: x *op*= y
Example: x += 1 increments x

Comparison Operators

x< y	Less	x <= y	Less or eq
x > y	Greater	x >= y	Greater or eq
x == y	Equal	x != y	Not equal

Boolean Operators

not x	x and y	x or y

Exception Handling

try:
  statements
except [*exception type* [as *var*]]:
  statements
finally:
  statements

Conversion Functions

int(*expr*)	Converts *expr* to integer
float(*expr*)	Converts *expr* to float
str(*expr*)	Converts *expr* to string
chr(*num*)	ASCII char *num*

String / List / Tuple Operations

len(*s*)	length of *s*
s[*i*]	*i*th item in *s* (0-based)
s[*start* : *end*]	slice of *s* from *start* (included) to *end* (excluded)
x in *s*	**True** if *x* is contained in *s*
x not in *s*	**True** if *x* is not contained in *s*
s + *t*	the concatenation of *s* with *t*
s * *n*	*n* copies of *s* concatenated
sorted(*s*)	a sorted copy of *s*
s.index(*item*)	position in *s* of *item*

More String Operations

s.lower()	lowercase copy of *s*
s.replace(*old*, *new*)	copy of *s* with *old* replaced with *new*
s.split(*delim*)	list of substrings delimited by *delim*

More String Operations (cont)

s.strip()	copy of *s* with whitespace trimmed
s.upper()	uppercase copy of *s*

See also http://docs.python.org/library/stdtypes.html#string-methods

Mutating List Operations

del *lst*[*i*]	Deletes *i*th item from *lst*
lst.append(*e*)	Appends *e* to *lst*
lst.insert(*i*, *e*)	Inserts *e* before *i*th item in *lst*
lst.sort()	Sorts *lst*

See also http://docs.python.org/library/stdtypes.html#typesseq-mutable

Dictionary Operations

len(*d*)	Number of items in *d*
del *d*[*key*]	Removes *key* from *d*
key in *d*	True if *d* contains *key*
d.keys()	Returns a list of keys in *d*

See also http://docs.python.org/library/stdtypes.html#mapping-types-dict

Function Definitions

def *name*(*arg1*, *arg2*, ...):
  statements
  return *expr*

Environment

sys.argv	List of command line arguments (argv[0] is executable)
os.environ	Dictionary of environment variables
os.curdir	String with path of current directory

import sys; print(sys.argv) *or*
from sys import argv; print(argv)

By **sschaub**
cheatography.com/sschaub/

Published 21st May, 2012.
Last updated 2nd June, 2014.

Cheatography

Essential Python Cheat Sheet

by sschaub via cheatography.com/1000/cs/374/

String Formatting

"Hello, {0} {1}".format("abe", "jones")

Hello, abe jones

"Hello, {fn} {ln}".format(fn="abe", ln="jones")

Hello, abe jones

"You owe me ${0:,.2f}".format(253422.3)

You owe me $253,422.30

now = datetime.now()

'{:%Y-%m-%d %H:%M:%S}'.format(now)

2012-05-16 15:04:33

See also http://docs.python.org/library/string.html#format-specification-mini-language

Useful Functions

exit(*code*)	Terminate program with exit *code*
raw_input("*prompt*")	Print *prompt* and readline() from stdin[1]

[1] Use input("*prompt*") in Python 3

Code Snippets

Loop Over Sequence

```
for index, value in enumerate(seq):
  print("{} : {}".format(index, value))
```

Loop Over Dictionary

```
for key in sorted(dict):
  print(dict[key])
```

Read a File

```
with open("filename", "r") as f:
  for line in f:
    line = line.rstrip("\n") # Strip newline
    print(line)
```

Other References

http://rgruet.free.fr/

Great Python 2.x Quick Reference

http://www.cheatography.com/davechild/cheat-sheets/python/

More Python Cheatsheet Goodness

By **sschaub**
cheatography.com/sschaub/

Published 21st May, 2012.
Last updated 2nd June, 2014.