datacamp

Python Basics

Python Cheat Sheet for Beginners

Learn Python online at www.DataCamp.com

> How to use this cheat sheet

Python is the most popular programming language in data science. It is easy to learn and comes with a wide array of powerful libraries for data analysis. This cheat sheet provides beginners and intermediate users a guide to starting using python. Use it to jump-start your journey with python. If you want more detailed Python cheat sheets, check out the following cheat sheets below:

Importing data in python

Data wrangling in pandas

> Accessing help and getting object types

```
1 + 1 # Everything after the hash symbol is ignored by Python
help(max) # Display the documentation for the max function
type('a') # Get the type of an object — this returns str
```

> Importing packages

Python packages are a collection of useful tools developed by the open-source community. They extend the capabilities of the python language. To install a new package (for example, pandas), you can go to your command prompt and type in `pip install pandas`. Once a package is installed, you can import it as follows.

```
import pandas # Import a package without an alias
import pandas as pd # Import a package with an alias
from pandas import DataFrame # Import an object from a package
```

> The working directory

The working directory is the default file path that python reads or saves files into. An example of the working directory is "C://file/path". The os library is needed to set and get the working directory.

```
import os # Import the operating system package
os.getcwd() # Get the current directory
os.setcwd("new/working/directory") # Set the working directory to a new file path
```

> Operators

Arithmetic operators

```
102 + 37 # Add two numbers with +
102 - 37 # Subtract a number with -
4 * 6 # Multiply two numbers with *
22 / 7 # Divide a number by another with /
22 // 7 # Integer divide a number with //
3 ** 4 # Raise to the power with **
22 % 7 # Returns 1 # Get the remainder after
division with %
```

Assignment operators

```
a = 5 # Assign a value to a
x[0] = 1 # Change the value of an item in a list
```

Numeric comparison operators

```
3 == 3 # Test for equality with ==
3 != 3 # Test for inequality with !=
3 > 1  # Test greater than with >
3 >= 3 # Test greater than or equal to with >=
3 < 4  # Test less than with <
3 <= 4 # Test less than or equal to with <=
```

Logical operators

```
~(2 == 2)          # Logical NOT with ~
(1 != 1) & (1 < 1) # Logical AND with &
(1 >= 1) | (1 < 1) # Logical OR with |
(1 != 1) ^ (1 < 1) # Logical XOR with ^
```

> Getting started with lists

A list is an ordered and changeable sequence of elements. It can hold integers, characters, floats, strings, and even objects.

Creating lists

```
# Create lists with [], elements separated by commas
x = [1, 3, 2]
```

List functions and methods

```
x.sorted(x) # Return a sorted copy of the list e.g., [1,2,3]
x.sort() # Sorts the list in-place (replaces x)
reversed(x) # Reverse the order of elements in x e.g., [2,3,1]
x.reversed() # Reverse the list in-place
x.count(2) # Count the number of element 2 in the list
```

Selecting list elements

Python lists are zero-indexed (the first element has index 0). For ranges, the first element is included but the last is not.

```
# Define the list
x = ['a', 'b', 'c', 'd', 'e']
x[0] # Select the 0th element in the list
x[-1] # Select the last element in the list
x[1:3] # Select 1st (inclusive) to 3rd (exclusive)
x[2:] # Select the 2nd to the end
x[:3] # Select 0th to 3rd (exclusive)
```

Concatenating lists

```
# Define the x and y lists
x = [1, 3, 6]
y = [10, 15, 21]
x + y # Returns [1, 3, 6, 10, 15, 21]
3 * x # Returns [1, 3, 6, 1, 3, 6, 1, 3, 6]
```

> Getting started with dictionaries

A dictionary stores data values in key-value pairs. That is, unlike lists which are indexed by position, dictionaries are indexed by their keys, the names of which must be unique.

Creating dictionaries

```
# Create a dictionary with {}
{'a': 1, 'b': 4, 'c': 9}
```

Dictionary functions and methods

```
x = {'a': 1, 'b': 2, 'c': 3} # Define the x ditionary
x.keys() # Get the keys of a dictionary, returns dict_keys(['a', 'b', 'c'])
x.values() # Get the values of a dictionary, returns dict_values([1, 2, 3])
```

Selecting dictionary elements

```
x['a'] # 1 # Get a value from a dictionary by specifying the key
```

> NumPy arrays

NumPy is a python package for scientific computing. It provides multidimensional array objects and efficient operations on them. To import NumPy, you can run this Python code `import numpy as np`

Creating arrays

```
# Convert a python list to a NumPy array
np.array([1, 2, 3]) # Returns array([1, 2, 3])
# Return a sequence from start (inclusive) to end (exclusive)
np.arange(1,5) # Returns array([1, 2, 3, 4])
# Return a stepped sequence from start (inclusive) to end (exclusive)
np.arange(1,5,2) # Returns array([1, 3])
# Repeat values n times
np.repeat([1, 3, 6], 3) # Returns array([1, 1, 1, 3, 3, 3, 6, 6, 6])
# Repeat values n times
np.tile([1, 3, 6], 3) # Returns array([1, 3, 6, 1, 3, 6, 1, 3, 6])
```

> Math functions and methods

All functions take an array as the input.

```
np.log(x) # Calculate logarithm
np.exp(x) # Calculate exponential
np.max(x) # Get maximum value
np.min(x) # Get minimum value
np.sum(x) # Calculate sum
np.mean(x) # Calculate mean
np.quantile(x, q) # Calculate q-th quantile
np.round(x, n) # Round to n decimal places
np.var(x) # Calculate variance
np.std(x) # Calculate standard deviation
```

> Getting started with characters and strings

```
# Create a string with double or single quotes
"DataCamp"

# Embed a quote in string with the escape character \
"He said, \"DataCamp\""

# Create multi-line strings with triple quotes
"""
A Frame of Data
Tidy, Mine, Analyze It
Now You Have Meaning
Citation: https://mdsr-book.github.io/haikus.html
"""

str[0] # Get the character at a specific position
str[0:2] # Get a substring from starting to ending index (exclusive)
```

Combining and splitting strings

```
"Data" + "Framed" # Concatenate strings with +, this returns 'DataFramed'
3 * "data " # Repeat strings with *, this returns 'data data data '
"beekeepers".split("e") # Split a string on a delimiter, returns ['b', '', 'k', '', 'p', 'rs']
```

Mutate strings

```
str = "Jack and Jill" # Define str
str.upper() # Convert a string to uppercase, returns 'JACK AND JILL'
str.lower() # Convert a string to lowercase, returns 'jack and jill'
str.title() # Convert a string to title case, returns 'Jack And Jill'
str.replace("J", "P") # Replaces matches of a substring with another, returns 'Pack and Pill'
```

> Getting started with DataFrames

Pandas is a fast and powerful package for data analysis and manipulation in python. To import the package, you can use `import pandas as pd`. A pandas DataFrame is a structure that contains two-dimensional data stored as rows and columns. A pandas series is a structure that contains one-dimensional data.

Creating DataFrames

```
# Create a dataframe from a dictionary
pd.DataFrame({
 'a': [1, 2, 3],
 'b': np.array([4, 4, 6]),
 'c': ['x', 'x', 'y']
})
# Create a dataframe from a list of dictionaries
pd.DataFrame([
 {'a': 1, 'b': 4, 'c': 'x'},
 {'a': 1, 'b': 4, 'c': 'x'},
 {'a': 3, 'b': 6, 'c': 'y'}
])
```

Selecting DataFrame Elements

Select a row, column or element from a dataframe. Remember: all positions are counted from zero, not one.

```
# Select the 3rd row
df.iloc[3]
# Select one column by name
df['col']
# Select multiple columns by names
df[['col1', 'col2']]
# Select 2nd column
df.iloc[:, 2]
# Select the element in the 3rd row, 2nd column
df.iloc[3, 2]
```

Manipulating DataFrames

```
# Concatenate DataFrames vertically
pd.concat([df, df])
# Concatenate DataFrames horizontally
pd.concat([df,df],axis="columns")
# Get rows matching a condition
df.query('logical_condition')
# Drop columns by name
df.drop(columns=['col_name'])
# Rename columns
df.rename(columns={"oldname": "newname"})
# Add a new column
df.assign(temp_f=9 / 5 * df['temp_c'] + 32)
# Calculate the mean of each column
df.mean()
# Get summary statistics by column
df.agg(aggregation_function)
# Get unique rows
df.drop_duplicates()
# Sort by values in a column
df.sort_values(by='col_name')
# Get rows with largest values in a column
df.nlargest(n, 'col_name')
```

Python For Data Science *Cheat Sheet*

Python Basics

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

Variables and Data Types

Variable Assignment

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

Calculations With Variables

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

Types and Type Conversion

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

Asking For Help

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

Lists

Also see NumPy Arrays

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

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]` `>>> my_list2[1][:2]`	my_list[list][itemOfList]

List Operations

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

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

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

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

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

Numpy Arrays

Also see Lists

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

Selecting Numpy Array Elements

Index starts at 0

Subset	
`>>> my_array[1]` `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

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

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Strings

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

String Operations

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

String Operations

Index starts at 0

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

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Python For Data Science *Cheat Sheet*

Jupyter Notebook

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

Saving/Loading Notebooks

- Create new notebook
- Open an existing notebook
- Make a copy of the current notebook
- Rename notebook
- Save current notebook and record checkpoint
- Revert notebook to a previous checkpoint
- Preview of the printed notebook
- Download notebook as
  - IPython notebook
  - Python
  - HTML
  - Markdown
  - reST
  - LaTeX
  - PDF
- Close notebook & stop running any scripts

Writing Code And Text

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

Edit Cells

- Cut currently selected cells to clipboard
- Copy cells from clipboard to current cursor position
- Paste cells from clipboard above current cell
- Paste cells from clipboard below current cell
- Paste cells from clipboard on top of current cel
- Delete current cells
- Revert "*Delete Cells*" invocation
- Split up a cell from current cursor position
- Merge current cell with the one above
- Merge current cell with the one below
- Move current cell up
- Move current cell down
- Adjust metadata underlying the current notebook
- Find and replace in selected cells
- Remove cell attachments
- Copy attachments of current cell
- Paste attachments of current cell
- Insert image in selected cells

Insert Cells

- Add new cell above the current one
- Add new cell below the current one

Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IP[y]: IPython | IRkernel | IJulia

Installing Jupyter Notebook will automatically install the IPython kernel.

- Restart kernel
- Interrupt kernel
- Restart kernel & run all cells
- Interrupt kernel & clear all output
- Restart kernel & run all cells
- Connect back to a remote notebook
- Run other installed kernels

Widgets

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

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

- Download serialized state of all widget models in use
- Save notebook with interactive widgets
- Embed current widgets

Command Mode:

Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

Executing Cells

- Run selected cell(s)
- Run current cells down and create a new one below
- Run current cells down and create a new one above
- Run all cells
- Run all cells above the current cell
- Run all cells below the current cell
- Change the cell type of current cell
- toggle, toggle scrolling and clear current outputs
- toggle, toggle scrolling and clear all output

View Cells

- Toggle display of Jupyter logo and filename
- Toggle display of toolbar
- Toggle display of cell action icons:
  - None
  - Edit metadata
  - Raw cell format
  - Slideshow
  - Attachments
  - Tags
- Toggle line numbers in cells

Asking For Help

- Walk through a UI tour
- List of built-in keyboard shortcuts
- Edit the built-in keyboard shortcuts
- Notebook help topics
- Description of markdown available in notebook
- Information on unofficial Jupyter Notebook extensions
- Python help topics
- IPython help topics
- NumPy help topics
- SciPy help topics
- Matplotlib help topics
- SymPy help topics
- Pandas help topics
- About Jupyter Notebook

Python For Data Science *Cheat Sheet*

NumPy Basics

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

NumPy

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

Use the following import convention:

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

NumPy Arrays

1D array

1	2	3

2D array

axis 1 →, axis 0 →

1.5	2	3
4	5	6

3D array

axis 2, axis 1, axis 0

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`
`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
`>>> 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 0 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])]`	Split the array horizontally at the 3rd index

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

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

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

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

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

Index Tricks

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

Shape Manipulation

`>>> np.transpose(b)`	Permute array dimensions
`>>> b.flatten()`	Flatten the array
`>>> np.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(c)`	Return the real part of the array elements
`>>> np.imag(c)`	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 argument
`>>> 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
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
`>>> np.trace(A)`	Trace
Norm	
`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	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(D,E)`	Least-squares solution to linear matrix equation
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 Sparse 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

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

Matrix Functions

Addition	
`>>> np.add(A,D)`	Addition
Subtraction	
`>>> np.subtract(A,D)`	Subtraction
Division	
`>>> np.divide(A,D)`	Division
Multiplication	
`>>> 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 Tunctions	
`>>> 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.sigm(A)`	Matrix sign function
Matrix Square Root	
`>>> linalg.sqrtm(A)`	Matrix square root
Arbitrary Functions	
`>>> linalg.funm(A, lambda x: x*x)`	Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors	
`>>> la, v = linalg.eig(A)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues
Singular Value Decomposition	
`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> 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)
```

DataCamp
Learn Python for Data Science Interactively

Python For Data Science *Cheat Sheet*

Pandas Basics

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

Pandas

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

Use the following import convention:

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

Pandas Data Structures

Series

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

Index

a	3
b	-5
c	7
d	4

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

DataFrame

Columns

Index

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

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

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

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

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

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

Setting

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

Set index a of Series s to 6

Dropping

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

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

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

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

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

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

Summary

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

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

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

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

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

Arithmetic Operations with Fill Methods

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

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

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)
```

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

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

A Basic Example

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

Loading The Data

Also see NumPy & Pandas

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

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

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	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)
```

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_)
```

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

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

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

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

2D Data or Images

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

2 Create Plot

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

Figure

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

Axes

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

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

Plot Anatomy & Workflow

Plot Anatomy

Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

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

3 Plotting Routines

1D Data

`>>> fig, ax = plt.subplots()`	
`>>> 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 o

Vector Fields

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

Data Distributions

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

2D Data or Images

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

`>>> axes2[0].pcolor(data2)`	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

4 Customize Plot

Colors, Color Bars & Color Maps

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

Markers

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

Linestyles

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

Text & Annotations

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

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

`>>> ax.tick_params(axis='y',`
`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)`
	Adjust the spacing between subplots

`>>> fig.tight_layout()`	Fit subplot(s) in to the figure area
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

5 Save Plot

Save figures

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

Save transparent figures

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

6 Show Plot

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

Close & Clear

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear the entire figure
`>>> plt.close()`	Close a window

DataCamp
Learn Python for Data Science Interactively

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

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

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

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

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")          Step 1
>>> sns.set_style("whitegrid")               Step 2
>>> g = sns.lmplot(x="tip",                  Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))               Step 4
>>> plt.title("title")
>>> plt.show(g)                              Step 5
```

1 Data

Also see Lists, NumPy & Pandas

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

Seaborn also offers built-in data sets:

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

2 Figure Aesthetics

Also see Matplotlib

`>>> f, ax = plt.subplots(figsize=(5,6))`	Create a figure and one subplot

Seaborn styles

`>>> sns.set()`	(Re)set the seaborn default
`>>> sns.set_style("whitegrid")`	Set the matplotlib parameters
`>>> sns.set_style("ticks", {"xtick.major.size":8, "ytick.major.size":8})`	Set the matplotlib parameters
`>>> sns.axes_style("whitegrid")`	Return a dict of params or use with `with` to temporarily set the style

Context Functions

`>>> sns.set_context("talk")`	Set context to `"talk"`
`>>> sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth":2.5})`	Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

`>>> sns.set_palette("husl",3)`	Define the color palette
`>>> sns.color_palette("husl")`	Use with `with` to temporarily set palette
`>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]`	
`>>> sns.set_palette(flatui)`	Set your own color palette

3 Plotting With Seaborn

Axis Grids

`>>> g = sns.FacetGrid(titanic, col="survived", row="sex")`	Subplot grid for plotting conditional relationships
`>>> g = g.map(plt.hist,"age")`	
`>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)`	Draw a categorical plot onto a Facetgrid
`>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)`	Plot data and regression model fits across a FacetGrid
`>>> h = sns.PairGrid(iris)`	Subplot grid for plotting pairwise relationships
`>>> h = h.map(plt.scatter)`	
`>>> sns.pairplot(iris)`	Plot pairwise bivariate distributions
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	Grid for bivariate plot with marginal univariate plots
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')`	Plot bivariate distribution

Categorical Plots

Scatterplot	
`>>> sns.stripplot(x="species", y="petal_length", data=iris)`	Scatterplot with one categorical variable
`>>> sns.swarmplot(x="species", y="petal_length", data=iris)`	Categorical scatterplot with non-overlapping points
Bar Chart	
`>>> sns.barplot(x="sex", y="survived", hue="class", data=titanic)`	Show point estimates and confidence intervals with scatterplot glyphs
Count Plot	
`>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")`	Show count of observations
Point Plot	
`>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male":"g", "female":"m"}, markers=["^","o"], linestyles=["-","--"])`	Show point estimates and confidence intervals as rectangular bars
Boxplot	
`>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)`	Boxplot
`>>> sns.boxplot(data=iris,orient="h")`	Boxplot with wide-form data
Violinplot	
`>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)`	Violin plot

Regression Plots

`>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)`	Plot data and a linear regression model fit

Distribution Plots

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

Matrix Plots

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

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

`>>> g.despine(left=True)`	Remove left spine
`>>> g.set_ylabels("Survived")`	Set the labels of the y-axis
`>>> g.set_xticklabels(rotation=45)`	Set the tick labels for x
`>>> g.set_axis_labels("Survived", "Sex")`	Set the axis labels
`>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])`	Set the limit and ticks of the x-and y-axis

Plot

`>>> plt.title("A Title")`	Add plot title
`>>> plt.ylabel("Survived")`	Adjust the label of the y-axis
`>>> plt.xlabel("Sex")`	Adjust the label of the x-axis
`>>> plt.ylim(0,100)`	Adjust the limits of the y-axis
`>>> plt.xlim(0,10)`	Adjust the limits of the x-axis
`>>> plt.setp(ax,yticks=[0,5])`	Adjust a plot property
`>>> plt.tight_layout()`	Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

`>>> plt.show()`	Show the plot
`>>> plt.savefig("foo.png")`	Save the plot as a figure
`>>> plt.savefig("foo.png", transparent=True)`	Save transparent figure

Close & Clear

Also see Matplotlib

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear an entire figure
`>>> plt.close()`	Close a window

Python For Data Science *Cheat Sheet*

Bokeh

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

Plotting With Bokeh

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

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

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

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

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

1 Data

Also see Lists, NumPy & Pandas

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

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

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

2 Plotting

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

3 Renderers & Visual Customizations

Glyphs

Scatter Markers

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

Line Glyphs

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

Customized Glyphs

Also see Data

Selection and Non-Selection Glyphs

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

Hover Glyphs

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

Colormapping

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

Legend Location

Inside Plot Area

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

Outside Plot Area

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

Legend Orientation

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

Legend Background & Border

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

Rows & Columns Layout

Rows

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

Columns

```
>>> from bokeh.layouts import 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]])
```

Tabbed Layout

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

Linked Plots

Linked Axes

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

Linked Brushing

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

4 Output & Export

Notebook

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

HTML

Standalone HTML

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

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

Components

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

PNG

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

SVG

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

5 Show or Save Your Plots

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

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

datacamp

Python For Data Science
Keras Cheat Sheet

Learn Keras online at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library forTheano 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 tensorflow.keras.models import Sequential
>>> from tensorflow.keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

> Data

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

Other

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

> Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from tensorflow.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 tensorflow.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 tensorflow.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 tensorflow.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'))
```

> Prediction

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

> Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from tensorflow.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 tensorflow.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)
```

> 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
```

> Compile Model

MLP: Binary Classification

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

MLP: Multi-Class Classification

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

MLP: Regression

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

Recurrent Neural Network

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

> Model Training

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

> Evaluate Your Model's Performance

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

> Save/ Reload Models

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

> Model Fine-tuning

Optimization Parameters

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

Early Stopping

```
>>> from tensorflow.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])
```

Learn Data Skills Online at www.DataCamp.com

TensorFlow Cheat Sheet

About

TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

Skflow

Scikit Flow provides a set of high level model classes that you can use to easily integrate with your existing Scikit-learn pipeline code. Scikit Flow is a simplified interface for TensorFlow, to get people started on predictive analytics and data mining. Scikit Flow has been merged into TensorFlow since version 0.8 and now called TensorFlow Learn.

Keras

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano

Installation

How to install new package in Python:

`pip install <package-name>`

Example: `pip install requests`

How to install tensorflow?

device = **cpu/gpu**

python_version = **cp27/cp34**

```
sudo pip install
https://storage.googleapis.com/
tensorflow/linux/$device/tensorflow-
0.8.0-$python_version-none-linux_x86
_64.whl
```

How to install Skflow

`pip install sklearn`

How to install Keras

`pip install keras`

update ~/.keras/keras.json - replace "theano" by "tensorflow"

Helpers

Python helper

Important functions

`type(object)`

Get object type

`help(object)`

Get help for object (list of available methods, attributes, signatures and so on)

`dir(object)`

Get list of object attributes (fields, functions)

`str(object)`

Transform an object to string

`object?`

Shows documentations about the object

`globals()`

Return the dictionary containing the current scope's global variables.

`locals()`

Update and return a dictionary containing the current scope's local variables.

`id(object)`

Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects.

`import __builtin__`

`dir(__builtin__)`

Other built-in functions

TensorFlow

Main classes

`tf.Graph()`
`tf.Operation()`
`tf.Tensor()`
`tf.Session()`

Some useful functions

`tf.get_default_session()`
`tf.get_default_graph()`
`tf.reset_default_graph()`
`ops.reset_default_graph()`
`tf.device("/cpu:0")`
`tf.name_scope(value)`
`tf.convert_to_tensor(value)`

TensorFlow Optimizers

`GradientDescentOptimizer`
`AdadeltaOptimizer`
`AdagradOptimizer`
`MomentumOptimizer`
`AdamOptimizer`
`FtrlOptimizer`
`RMSPropOptimizer`

Reduction

`reduce_sum`
`reduce_prod`
`reduce_min`
`reduce_max`
`reduce_mean`
`reduce_all`
`reduce_any`
`accumulate_n`

Activation functions

`tf.nn?`

relu
relu6
elu
softplus
softsign
dropout
bias_add
sigmoid
tanh
sigmoid_cross_entropy_with_logits
softmax
log_softmax
softmax_cross_entropy_with_logits
sparse_softmax_cross_entropy_with_logits
weighted_cross_entropy_with_logits
etc.

Skflow

Main classes

TensorFlowClassifier
TensorFlowRegressor
TensorFlowDNNClassifier
TensorFlowDNNRegressor
TensorFlowLinearClassifier
TensorFlowLinearRegressor
TensorFlowRNNClassifier
TensorFlowRNNRegressor
TensorFlowEstimator

Each classifier and regressor have following fields

n_classes=0 (Regressor), n_classes are expected to be input (Classifiers)

batch_size=32,
steps=200, // except TensorFlowRNNClassifier - there is 50
optimizer='Adagrad',
learning_rate=0.1,
class_weight=None,
clip_gradients=5.0,
continue_training=False,
config=None,
verbose=1.

Each class has a method fit

`fit(X, y, monitor=None, logdir=None)`

X: matrix or tensor of shape [n_samples, n_features...]. Can be iterator that returns arrays of features. The training input samples for fitting the model.

Y: vector or matrix [n_samples] or [n_samples, n_outputs]. Can be iterator that returns array of targets. The training target values (class labels in classification, real numbers in regression).

monitor: Monitor object to print training progress and invoke early stopping

logdir: the directory to save the log file that can be used for optional visualization.

`predict (X, axis=1, batch_size=None)`

Args:

X: array-like matrix, [n_samples, n_features...] or iterator.

axis: Which axis to argmax for classification.
By default axis 1 (next after batch) is used. Use 2 for sequence predictions.

batch_size: If test set is too big, use batch size to split it into mini batches. By default the batch_size member variable is used.

Returns:

y: array of shape [n_samples]. The predicted classes or predicted value.

Useful links

TensorFlow API
https://www.tensorflow.org/versions/r0.9/api_docs/index.html
TensorFlow How-Tos
https://www.tensorflow.org/versions/r0.9/how_tos/index.html
Skflow github
https://github.com/tensorflow/skflow
Skflow API
http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster
Keras
http://keras.io/
Our github
https://github.com/Altoros/TensorFlow-training
TensorFlow github
https://github.com/tensorflow/tensorflow

Python For Data Science *Cheat Sheet*

PySpark - SQL Basics

Learn Python for data science Interactively at www.DataCamp.com

PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.some.config.option", "some-value") \
        .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
```

Infer Schema

```
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

Specify Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
                                     age=int(p[1].strip())))
>>> schemaString = "name age"
>>> fields = [StructField(field_name, StringType(), True) for
field_name in schemaString.split()]
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
  +--------+---+
  |    name|age|
  +--------+---+
  |    Mine| 28|
  |   Filip| 29|
  |Jonathan| 30|
  +--------+---+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
 +--------------------+---+---------+--------+--------------------+
 |             address|age|firstName |lastName|          phoneNumber|
 +--------------------+---+---------+--------+--------------------+
 |[New York,10021,N...| 25|     John|   Smith|[[212 555-1234,ho...|
 |[New York,10021,N...| 21|     Jane|     Doe|[[322 888-1234,ho...|
 +--------------------+---+---------+--------+--------------------+
>>> df2 = spark.read.load("people.json", format="json")
```

Parquet files

```
>>> df3 = spark.read.load("users.parquet")
```

TXT files

```
>>> df4 = spark.read.text("people.txt")
```

Duplicate Values

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

Queries

```
>>> from pyspark.sql import functions as F
```

Select

```
>>> df.select("firstName").show()
>>> df.select("firstName","lastName") \
        .show()
>>> df.select("firstName",
              "age",
              explode("phoneNumber") \
              .alias("contactInfo")) \
        .select("contactInfo.type",
              "firstName",
              "age") \
        .show()
>>> df.select(df["firstName"],df["age"]+ 1)
        .show()
>>> df.select(df['age'] > 24).show()
```

Show all entries in `firstName` column

Show all entries in `firstName`, `age` and `type`

Show all entries in `firstName` and `age`, add `1` to the entries of `age`

Show all entries where `age` >24

When

```
>>> df.select("firstName",
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
        .show()
>>> df[df.firstName.isin("Jane","Boris")]
              .collect()
```

Show `firstName` and 0 or 1 depending on `age` >30

Show `firstName` if in the given options

Like

```
>>> df.select("firstName",
              df.lastName.like("Smith")) \
        .show()
```

Show `firstName`, and `lastName` is `TRUE` if `lastName` is like `Smith`

Startswith - Endswith

```
>>> df.select("firstName",
              df.lastName \
              .startswith("Sm")) \
        .show()
>>> df.select(df.lastName.endswith("th")) \
        .show()
```

Show `firstName`, and `TRUE` if `lastName` starts with `Sm`

Show last names ending in `th`

Substring

```
>>> df.select(df.firstName.substr(1, 3) \
              .alias("name")) \
        .collect()
```

Return substrings of `firstName`

Between

```
>>> df.select(df.age.between(22, 24)) \
        .show()
```

Show `age`: values are `TRUE` if between `22` and `24`

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
            .withColumn('postalCode',df.address.postalCode) \
            .withColumn('state',df.address.state) \
            .withColumn('streetAddress',df.address.streetAddress) \
            .withColumn('telePhoneNumber',
                        explode(df.phoneNumber.number)) \
            .withColumn('telePhoneType',
                        explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telePhoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
```

GroupBy

```
>>> df.groupBy("age")\
        .count() \
        .show()
```

Group by `age`, count the members in the groups

Filter

```
>>> df.filter(df["age"]>24).show()
```

Filter entries of `age`, only keep those records of which the values are >24

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age", ascending=False).collect()
>>> df.orderBy(["age","city"],ascending=[0,1])\
        .collect()
```

Missing & Replacing Values

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

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

Repartitioning

```
>>> df.repartition(10)\
        .rdd \
        .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions()
```

`df` with 10 partitions

`df` with 1 partition

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people")\
                .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
>>> df.toPandas()
```

Convert `df` into an RDD
Convert `df` into a RDD of string
Return the contents of `df` as Pandas `DataFrame`

Write & Save to Files

```
>>> df.select("firstName", "city")\
        .write \
        .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
        .write \
        .save("namesAndAges.json",format="json")
```

Stopping SparkSession

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

Inspect Data

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

`>>> df.describe().show()`	Compute summary statistics
`>>> df.columns`	Return the columns of `df`
`>>> df.count()`	Count the number of rows in `df`
`>>> df.distinct().count()`	Count the number of distinct rows in `df`
`>>> df.printSchema()`	Print the schema of `df`
`>>> df.explain()`	Print the (logical and physical) plans

SQL Basics Cheat Sheet

LearnSQL.com

SQL, or *Structured Query Language*, is a language to talk to databases. It allows you to select specific data and to build complex reports. Today, SQL is a universal language of data. It is used in practically all technologies that process data.

SAMPLE DATA

COUNTRY

id	name	population	area
1	France	66600000	640680
2	Germany	80700000	357000
...	...	...	...

CITY

id	name	country_id	population	rating
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...	...	...	...	...

QUERYING SINGLE TABLE

Fetch all columns from the country table:

```
SELECT *
FROM country;
```

Fetch id and name columns from the city table:

```
SELECT id, name
FROM city;
```

Fetch city names sorted by the rating column in the default ASCending order:

```
SELECT name
FROM city
ORDER BY rating [ASC];
```

Fetch city names sorted by the rating column in the DESCending order:

```
SELECT name
FROM city
ORDER BY rating DESC;
```

ALIASES

COLUMNS

```
SELECT name AS city_name
FROM city;
```

TABLES

```
SELECT co.name, ci.name
FROM city AS ci
JOIN country AS co
  ON ci.country_id = co.id;
```

FILTERING THE OUTPUT

COMPARISON OPERATORS

Fetch names of cities that have a rating above 3:

```
SELECT name
FROM city
WHERE rating > 3;
```

Fetch names of cities that are neither Berlin nor Madrid:

```
SELECT name
FROM city
WHERE name != 'Berlin'
  AND name != 'Madrid';
```

TEXT OPERATORS

Fetch names of cities that start with a 'P' or end with an 's':

```
SELECT name
FROM city
WHERE name LIKE 'P%'
  OR name LIKE '%s';
```

Fetch names of cities that start with any letter followed by 'ublin' (like Dublin in Ireland or Lublin in Poland):

```
SELECT name
FROM city
WHERE name LIKE '_ublin';
```

OTHER OPERATORS

Fetch names of cities that have a population between 500K and 5M:

```
SELECT name
FROM city
WHERE population BETWEEN 500000 AND 5000000;
```

Fetch names of cities that don't miss a rating value:

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

Fetch names of cities that are in countries with IDs 1, 4, 7, or 8:

```
SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);
```

QUERYING MULTIPLE TABLES

INNER JOIN

JOIN (or explicitly **INNER JOIN**) returns rows that have matching values in both tables.

```
SELECT city.name, country.name
FROM city
[INNER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

LEFT JOIN

LEFT JOIN returns all rows from the left table with corresponding rows from the right table. If there's no matching row, **NULL**s are returned as values from the second table.

```
SELECT city.name, country.name
FROM city
LEFT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

RIGHT JOIN

RIGHT JOIN returns all rows from the right table with corresponding rows from the left table. If there's no matching row, **NULL**s are returned as values from the left table.

```
SELECT city.name, country.name
FROM city
RIGHT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

FULL JOIN

FULL JOIN (or explicitly **FULL OUTER JOIN**) returns all rows from both tables – if there's no matching row in the second table, **NULL**s are returned.

```
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

CROSS JOIN

CROSS JOIN returns all possible combinations of rows from both tables. There are two syntaxes available.

```
SELECT city.name, country.name
FROM city
CROSS JOIN country;

SELECT city.name, country.name
FROM city, country;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

NATURAL JOIN

NATURAL JOIN will join tables by all columns with the same name.

```
SELECT city.name, country.name
FROM city
NATURAL JOIN country;
```

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

NATURAL JOIN used these columns to match rows: **city.id**, **city.name**, **country.id**, **country.name**.
NATURAL JOIN is very rarely used in practice.

Try out the interactive **SQL Basics** course at **LearnSQL.com**, and check out our other SQL courses.

LearnSQL.com is owned by Vertabelo SA
vertabelo.com | CC BY-NC-ND Vertabelo SA

SQL Basics Cheat Sheet

LearnSQL.com

AGGREGATION AND GROUPING

GROUP BY **groups** together rows that have the same values in specified columns. It computes summaries (aggregates) for each unique combination of values.

CITY		
id	name	country_id
1	Paris	1
101	Marseille	1
102	Lyon	1
2	Berlin	2
103	Hamburg	2
104	Munich	2
3	Warsaw	4
105	Cracow	4

→

CITY	
country_id	count
1	3
2	3
4	2

AGGREGATE FUNCTIONS

- `avg(expr)` – average value for rows within the group
- `count(expr)` – count of values for rows within the group
- `max(expr)` – maximum value within the group
- `min(expr)` – minimum value within the group
- `sum(expr)` – sum of values within the group

EXAMPLE QUERIES

Find out the number of cities:

```
SELECT COUNT(*)
FROM city;
```

Find out the number of cities with non-null ratings:

```
SELECT COUNT(rating)
FROM city;
```

Find out the number of distinctive country values:

```
SELECT COUNT(DISTINCT country_id)
FROM city;
```

Find out the smallest and the greatest country populations:

```
SELECT MIN(population), MAX(population)
FROM country;
```

Find out the total population of cities in respective countries:

```
SELECT country_id, SUM(population)
FROM city
GROUP BY country_id;
```

Find out the average rating for cities in respective countries if the average is above 3.0:

```
SELECT country_id, AVG(rating)
FROM city
GROUP BY country_id
HAVING AVG(rating) > 3.0;
```

SUBQUERIES

A subquery is a query that is nested inside another query, or inside another subquery. There are different types of subqueries.

SINGLE VALUE

The simplest subquery returns exactly one column and exactly one row. It can be used with comparison operators =, <, <=, >, or >=.
This query finds cities with the same rating as Paris:

```
SELECT name
FROM city
WHERE rating = (
  SELECT rating
  FROM city
  WHERE name = 'Paris'
);
```

MULTIPLE VALUES

A subquery can also return multiple columns or multiple rows. Such subqueries can be used with operators `IN`, `EXISTS`, `ALL`, or `ANY`.
This query finds cities in countries that have a population above 20M:

```
SELECT name
FROM city
WHERE country_id IN (
  SELECT country_id
  FROM country
  WHERE population > 20000000
);
```

CORRELATED

A correlated subquery refers to the tables introduced in the outer query. A correlated subquery depends on the outer query. It cannot be run independently from the outer query.
This query finds cities with a population greater than the average population in the country:

```
SELECT *
FROM city main_city
WHERE population > (
  SELECT AVG(population)
  FROM city average_city
  WHERE average_city.country_id = main_city.country_id
);
```

This query finds countries that have at least one city:

```
SELECT name
FROM country
WHERE EXISTS (
  SELECT *
  FROM city
  WHERE country_id = country.id
);
```

SET OPERATIONS

Set operations are used to combine the results of two or more queries into a single result. The combined queries must return the same number of columns and compatible data types. The names of the corresponding columns can be different.

CYCLING		
id	name	country
1	YK	DE
2	ZG	DE
3	WT	PL
...	...	...

SKATING		
id	name	country
1	YK	DE
2	DF	DE
3	AK	PL
...	...	...

UNION

`UNION` combines the results of two result sets and removes duplicates. **`UNION ALL`** doesn't remove duplicate rows.

This query displays German cyclists together with German skaters:

```
SELECT name
FROM cycling
WHERE country = 'DE'
UNION / UNION ALL
SELECT name
FROM skating
WHERE country = 'DE';
```

INTERSECT

`INTERSECT` returns only rows that appear in both result sets.

This query displays German cyclists who are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
INTERSECT
SELECT name
FROM skating
WHERE country = 'DE';
```

EXCEPT

`EXCEPT` returns only the rows that appear in the first result set but do not appear in the second result set.

This query displays German cyclists unless they are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
EXCEPT / MINUS
SELECT name
FROM skating
WHERE country = 'DE';
```

Try out the interactive **SQL Basics** course at **LearnSQL.com**, and check out our other SQL courses.

LearnSQL.com is owned by Vertabelo SA
vertabelo.com | CC BY-NC-ND Vertabelo SA

datacamp

MySQL Cheat Sheet

Learn SQL online at www.DataCamp.com

What is MySQL?

MySQL is an open-source relational database management system (RDBMS) known for its fast performance and reliability. Developed by Oracle Corporation, it's widely used for web applications and online publishing.

> Sample Data

The dataset contains details of the world's highest valued media franchises by gross revenue. Each row contains one franchise, and the table is named franchises.

Franchise	inception_year	total_revenue_busd	original_medium	owner	n_movies
Star Wars	1977	46.7	movie	The Walt Disney Company	12
Mickey Mouse and Friends	1928	52.2	cartoon	The Walt Disney Company	
Anpanman	1973	38.4	book	Froebel-kan	33
Winnie the Pooh	1924	48.5	book	The Walt Disney Company	6
Pokémon	1996	88	video game	The Pokémon Company	24
Disney Princess	2000	45.4	movie	The Walt Disney Company	

> Querying tables

Get all the columns from a table using `SELECT *`

```
SELECT *
  FROM franchises
```

Get a column from a table by name using `SELECT col`

```
SELECT franchise
  FROM franchises
```

Get multiple columns from a table by name using `SELECT col1, col2`

```
SELECT franchise, inception_year
  FROM franchises
```

Override column names with `SELECT col AS new_name`

```
SELECT franchise, inception_year AS creation_year
  FROM franchises
```

Arrange the rows in ascending order of values in a column with `ORDER BY col`

```
SELECT franchise, inception_year
  FROM franchises
  ORDER BY inception_year
```

Arrange the rows in descending order of values in a column with `ORDER BY col DESC`

```
SELECT franchise, total_revenue_busd
  FROM franchises
  ORDER BY total_revenue_busd DESC
```

Limit the number of rows returned with `LIMIT n`

```
SELECT *
  FROM franchises
  LIMIT 2
```

Get unique values with `SELECT DISTINCT`

```
SELECT DISTINCT owner
  FROM franchises
```

> Filtering Data

Filtering on numeric columns

Get rows where a number is greater than a value with `WHERE col > n`

```
SELECT franchise, inception_year
  FROM franchises
  WHERE inception_year > 1928
```

Get rows where a number is greater than or equal to a value with `WHERE col >= n`

```
SELECT franchise, inception_year
  FROM franchises
  WHERE inception_year >= 1928
```

Get rows where a number is less than a value with `WHERE col < n`

```
SELECT franchise, inception_year
  FROM franchises
  WHERE inception_year <= 1977
```

Get rows where a number is equal to a value with `WHERE col = n`

```
SELECT franchise, inception_year
  FROM franchises
  WHERE inception_year = 1996
```

Get rows where a number is not equal to a value with `WHERE col <> n` **or** `WHERE col != n`

```
SELECT franchise, inception_year
  FROM franchises
  WHERE inception_year <> 1996
```

Get rows where a number is between two values (inclusive) with `WHERE col BETWEEN m AND n`

```
SELECT franchise, inception_year
  FROM franchises
  WHERE inception_year BETWEEN 1928 AND 1977
```

Filtering on text columns

Get rows where text is equal to a value with `WHERE col = 'x'`

```
SELECT franchise, original_medium
  FROM franchises
  WHERE original_medium = 'book'
```

Get rows where text is one of several values with `WHERE col IN ('x', 'y')`

```
SELECT franchise, original_medium
  FROM franchises
  WHERE original_medium IN ('movie', 'video game')
```

Get rows where text contains specific letters with `WHERE col LIKE '%abc%'`
(% represents any characters)

```
SELECT franchise, original_medium
  FROM franchises
  WHERE original_medium LIKE '%oo%'
```

Filtering on multiple columns

Get the rows where one condition and another condition holds with `WHERE condn1 AND condn2`

```
SELECT franchise, inception_year, total_revenue_busd
  FROM franchises
  WHERE inception_year < 1950 AND total_revenue_busd > 50
```

Get the rows where one condition or another condition holds with `WHERE condn1 OR condn2`

```
SELECT franchise, inception_year, total_revenue_busd
  FROM franchises
  WHERE inception_year < 1950 OR total_revenue_busd > 50
```

Filtering on missing data

Get rows where values are missing with `WHERE col IS NULL`

```
SELECT franchise, n_movies
  FROM franchises
  WHERE n_movies IS NULL
```

Get rows where values are not missing with `WHERE col IS NOT NULL`

```
SELECT franchise, n_movies
  FROM franchises
  WHERE n_movies IS NOT NULL
```

> Aggregating Data

Simple aggregations

Get the total number of rows `SELECT COUNT(*)`

```
SELECT COUNT(*)
  FROM franchises
```

Get the total value of a column with `SELECT SUM(col)`

```
SELECT SUM(total_revenue_busd)
  FROM franchises
```

Get the mean value of a column with `SELECT AVG(col)`

```
SELECT AVG(total_revenue_busd)
  FROM franchises
```

Get the minimum value of a column with `SELECT MIN(col)`

```
SELECT MIN(total_revenue_busd)
  FROM franchises
```

Get the maximum value of a column with `SELECT MAX(col)`

```
SELECT MAX(total_revenue_busd)
  FROM franchises
```

Grouping, filtering, and sorting

Get summaries grouped by values with `GROUP BY col`

```
SELECT owner, COUNT(*)
  FROM franchises
  GROUP BY owner
```

Get summaries grouped by values, in order of summaries with `GROUP BY col ORDER BY smmry DESC`

```
SELECT original_medium, SUM(n_movies) AS total_movies
  FROM franchises
  GROUP BY original_medium
  ORDER BY total_movies DESC
```

Get rows where values in a group meet a criterion with `GROUP BY col HAVING condn`

```
SELECT original_medium, SUM(n_movies) AS total_movies
  FROM franchises
  GROUP BY original_medium
  ORDER BY total_movies DESC
  HAVING total_movies > 10
```

Filter before and after grouping with `WHERE condn_before GROUP BY col HAVING condn_after`

```
SELECT original_medium, SUM(n_movies) AS total_movies
  FROM franchises
  WHERE owner = 'The Walt Disney Company'
  GROUP BY original_medium
  ORDER BY total_movies DESC
  HAVING total_movies > 10
```

> MySQL-Specific Syntax

Not all code works in every dialect of SQL. The following examples work in MySQL, but are not guaranteed to work in other dialects.

Limit the number of rows returned, offset from the top with `LIMIT m, n`

```
SELECT *
  FROM franchises
  LIMIT 2, 3
```

By default, MySQL uses case insensitive matching in `WHERE clauses.`

```
SELECT *
  FROM franchises
  WHERE owner = 'THE WALT DISNEY COMPANY'
```

To get case sensitive matching, use `WHERE BINARY condn`

```
SELECT *
  FROM franchises
  WHERE BINARY owner = 'THE WALT DISNEY COMPANY'
```

Get the current date with `CURDATE()` **and the current** `datetime` **with** `NOW()` **or** `CURTIME()`

```
SELECT CURDATE(), NOW(), CURTIME()
```

List available tables with `show tables`

```
show tables
```

datacamp

PostgreSQL Basics

Learn SQL online at www.DataCamp.com

What is PostgreSQL?

PostgreSQL is an open-source relational database management system (RDBMS) known for its extensibility and standards compliance. Developed and maintained by a group of volunteers known as The PostgreSQL Global Development Group, it is popular across a wide range of organizations from enterprises to government departments. It has powerful data analysis capabilities.

> Sample Data

The dataset contains details of the world's fastest production cars by 0 to 60 mph acceleration time. Each row contains one car model, and the table is named cars.

make	model	year	propulsion_type	time_to_60_mph_s	limited_production_count
Lamborghini	Huracán Performante	2018	ICE	2.2	
Ferrari	SF90 Stradale	2021	Hybrid	2	
Tesla	Model S Plaid	2021	Electric	1.98	
Porsche	918 Spyder	2015	Hybrid	2.1	918
Rimac	Nevera	2021	Electric	1.74	150
Porsche	911 Turbo S (992)	2020	ICE	2.1	

> Querying tables

Get all the columns from a table using `SELECT *`

```
SELECT *
 FROM cars
```

Get a column from a table by name using `SELECT col`

```
SELECT model
 FROM cars
```

Get multiple columns from a table by name using `SELECT col1, col2`

```
SELECT make, model
 FROM cars
```

Override column names with `SELECT col AS new_name`

```
SELECT make, model, propulsion_type AS engine_type
 FROM cars
```

Override column names with `SELECT col AS new_name`

```
SELECT make, model, propulsion_type
 AS engine_type
 FROM cars
```

Arrange the rows in ascending order of values in a column with `ORDER BY col`

```
SELECT  make, model, time_to_60_mph_s
 FROM cars
 ORDER BY time_to_60_mph_s
```

Arrange the rows in descending order of values in a column with ORDER BY col DESC

```
SELECT make, model, model_year
 FROM cars
 ORDER BY model_year DESC
```

Limit the number of rows returned with `LIMIT n`

```
SELECT *
 FROM cars
LIMIT 2
```

Get unique values with `SELECT DISTINCT`

```
SELECT DISTINCT propulsion_type
 FROM cars
```

> Filtering Data

Filtering on numeric columns

Get rows where a number is greater than a value with `WHERE col > n`

```
SELECT make, model, time_to_60_mph_s
  FROM cars
  WHERE time_to_60_mph_s > 2.1
```

Get rows where a number is greater than or equal to a value with `WHERE col >= n`

```
SELECT make, model, time_to_60_mph_s
  FROM cars
  WHERE time_to_60_mph_s >= 2.1
```

Get rows where a number is less than a value with `WHERE col < n`

```
SELECT make, model, time_to_60_mph_s
  FROM cars
  WHERE time_to_60_mph_s < 2.1
```

Get rows where a number is less than or equal to a value with `WHERE col <= n`

```
SELECT make, model, time_to_60_mph_s
  FROM cars
  WHERE time_to_60_mph_s <= 2.1
```

Get rows where a number is equal to a value with `WHERE col = n`

```
SELECT make, model, time_to_60_mph_s
 FROM cars
 WHERE time_to_60_mph_s = 2.1
```

Get rows where a number is not equal to a value with `WHERE col <> n` **or** `WHERE col != n`

```
SELECT make, model, time_to_60_mph_s
  FROM cars
  WHERE time_to_60_mph_s <> 2.1
```

Get rows where a number is between two values (inclusive) with `WHERE col BETWEEN m AND n`

```
SELECT make, model, time_to_60_mph_s
 FROM cars
 WHERE time_to_60_mph_s BETWEEN 1.9 AND 2.1
```

Filtering on text columns

Get rows where text is equal to a value with `WHERE col = 'x'`

```
SELECT make, model, propulsion_type
 FROM cars
 WHERE propulsion_type = 'Hybrid'
```

Get rows where text is one of several values with `WHERE col IN ('x', 'y')`

```
SELECT make, model, propulsion_type
 FROM cars
 WHERE propulsion_type IN ('Electric',  'Hybrid')
```

Get rows where text contains specific letters with `WHERE col LIKE '%abc%'` (% represents any characters)

```
SELECT make, model, propulsion_type
 FROM cars
 WHERE propulsion_type LIKE '%ic%'
```

For case insensitive matching, use `WHERE col ILIKE '%abc%'`

```
SELECT make, model, propulsion_type
 FROM cars
 WHERE propulsion_type ILIKE '%ic%'
```

Filtering on multiple columns

Get the rows where one condition and another condition holds with `WHERE condn1 AND condn2`

```
SELECT make, model, propulsion_type, model_year
 FROM cars
 WHERE propulsion_type = 'Hybrid' AND model_year < 2020
```

Get the rows where one condition or another condition holds with `WHERE condn1 OR condn2`

```
SELECT make, model, propulsion_type, model_year
 FROM cars
 WHERE propulsion_type = 'Hybrid' OR model_year < 2020
```

Filtering on missing data

Get rows where values are missing with `WHERE col IS NULL`

```
SELECT make, model, limited_production_count
 FROM cars
 WHERE limited_production_count IS NULL
```

Get rows where values are not missing with `WHERE col IS NOT NULL`

```
SELECT make, model, limited_production_count
 FROM cars
 WHERE limited_production_count IS NOT NULL
```

> Aggregating Data

Simple aggregations

Get the total number of rows `SELECT COUNT(*)`

```
SELECT COUNT(*)
  FROM cars
```

Get the total value of a column with `SELECT SUM(col)`

```
SELECT SUM(limited_production_count)
  FROM cars
```

Get the mean value of a column with `SELECT AVG(col)`

```
SELECT AVG(time_to_60_mph_s)
  FROM cars
```

Get the minimum value of a column with `SELECT MIN(col)`

```
SELECT MIN(time_to_60_mph_s)
  FROM cars
```

Get the maximum value of a column with `SELECT MAX(col)`

```
SELECT MAX(time_to_60_mph_s)
 FROM cars
```

Grouping, filtering, and sorting

Get summaries grouped by values with `GROUP BY col`

```
SELECT propulsion_type, COUNT(*)
 FROM cars
 GROUP BY propulsion_type
```

Get summaries grouped by values, in order of summaries with `GROUP BY col ORDER BY smmry DESC`

```
SELECT propulsion_type, AVG(time_to_60_mph_s) AS mean_time_to_60_mph_s
 FROM cars
 GROUP BY propulsion_type
 ORDER BY mean_time_to_60_mph_s
```

Get rows where values in a group meet a criterion with `GROUP BY col HAVING condn`

```
SELECT propulsion_type, AVG(time_to_60_mph_s) AS mean_time_to_60_mph_s
 FROM cars
 GROUP BY propulsion_type
 HAVING mean_time_to_60_mph_s > 2
```

Filter before and after grouping with `WHERE condn_before GROUP BY col HAVING condn_after`

```
SELECT propulsion_type, AVG(time_to_60_mph_s) AS mean_time_to_60_mph_s
 FROM cars
 WHERE limited_production_count IS NOT NULL
 GROUP BY propulsion_type
 HAVING mean_time_to_60_mph_s > 2
```

> PostgreSQL-Specific Syntax

Not all code works in every dialect of SQL. The following examples work in PostgreSQL, but are not guaranteed to work in other dialects.

Limit the number of rows returned, offset from the top with `LIMIT m OFFSET n`

```
SELECT *
 FROM cars
 LIMIT 2 OFFSET 3
```

PostgreSQL allows text concatenation with the `||` **operator**

```
SELECT make || ' ' || model AS make_and_model
  FROM cars
```

Get the current date with `CURRENT_DATE` **and the current datetime with** `NOW()` **or** `CURRENT_TIME`

```
SELECT NOW(), CURRENT_DATE, CURRENT_TIME
```

List available tables by selecting from `pg_catalog.pg_tables`

```
SELECT *
 FROM pg_catalog.pg_tables
```

Azure

DEVELOPER'S SHEET

CREATED BY DR. MILAN MILANOVIĆ

Contact: milan_milanovic milanmilanovic

Fetched from: azure.microsoft.com/en-gb/products (August 2022.)

CC

AI + Machine Learning

Create the next generation of applications usingartificial intelligence capabilitiesfor any developer and any scenariot

ANOMALY DETECTOR
Easily add anomaly detection capabilities to your apps.

AZURE BOT SERVICE
Build conversational AI experiences for your customers

AZURE COGNITIVE SEARCH
Enterprise scale search for app development

AZURE DATABRICKS
Design AI with Apache Spark™-based analytics

AZURE MACHINE LEARNING
Use an enterprise-grade service for the end-to-end machine learning lifecycle

AZURE OPEN DATASETS
Cloud platform to host and share curated open datasets to accelerate development of machine learning models

AZURE COGNITIVE SERVICES
Deploy high-quality AI models as APIs

AZURE VIDEO INDEXER
Easily extract meaningful insights from audio and video files using media AI

COMPUTER VISION
Distill actionable information from images

CONTENT MODERATOR
Automated image, text and video moderation

CUSTOM VISION
Easily customise your own state-of-the-art computer vision models for your unique use case

DATA SCIENCE VIRTUAL MACHINES
Rich pre-configured environment for AI development

FACE API
Detect, identify, analyse, organise and tag faces in photos

AZURE FORM RECOGNISER
The AI-powered document extraction service that understands your forms

AZURE IMMERSIVE READER
Empower users of all ages and abilities to read and comprehend text

KINECT DK
Build computer vision and speech models using a developer kit with advanced AI sensors

LANGUAGE UNDERSTANDING (LUIS)
Teach your apps to understand commands from your users

MICROSOFT GENOMICS
Power genome sequencing & research insights

PERSONALISER
An AI service that delivers a personalised user experience

PROJECT BONSAI PREVIEW
Create intelligent industrial control systems using simulations

QNA MAKER
Distill information into conversational, easy-to-navigate answers

SPEAKER RECOGNITION
A Speech service feature that verifies and identifies speakers

SPEECH-TO-TEXT
A Speech service feature that accurately converts spoken audio to text

SPEECH TRANSLATION
Easily integrate real-time speech translation to your app

COGNITIVE SERVICE FOR LANGUAGE
Add natural language capabilities with a single API call

TEXT TO SPEECH
A Speech service feature that converts text to lifelike speech

TRANSLATOR
Easily conduct machine translation with a simple REST API call

AZURE METRICS ADVISOR
An AI service that monitors metrics and diagnoses issues

HEALTH BOT
A managed service purpose-built for development of virtual healthcare assistants

AZURE PERCEPT PREVIEW
Accelerate edge intelligence from silicon to service

AZURE APPLIED AI SERVICES
Specialist services that enable organisations to accelerate time to value n applying AI to solve common scenarios

AZURE OPENAI SERVICE PREVIEW
Apply advanced coding and language models to a variety of use cases

Containers

Develop and manage your containerised apps faster with integrated tools

APP CONFIGURATION
Fast, scalable parameter storage for app configuration

AZURE KUBERNETES SERVICE (AKS)
Build and scale with managed Kubernetes

AZURE RED HAT OPENSHIFT
Fully managed OpenShift service, jointly operated with Red Hat

AZURE CONTAINER APPS
Build and deploy modern apps and microservices using serverless containers

AZURE FUNCTIONS
Execute event-driven serverless code functions with an end-to-end development experience

WEB APP FOR CONTAINERS
Easily deploy and run containerised web apps on Windows and Linux

CONTAINER INSTANCES
Easily run containers on Azure without managing servers.

SERVICE FABRIC
Develop microservices and orchestrate containers on Windows or Linux

CONTAINER REGISTRY
Store and manage container images across all types of deployments

Databases

Support rapid growth and innovate faster with secure, enterprise-grade and fully managed database services

AZURE CACHE FOR REDIS
Accelerate apps with high-throughput, low-latency data caching

AZURE COSMOS DB
Fast NoSQL database with open APIs for any scale

AZURE DATA FACTORY
Hybrid data integration at enterprise scale, made easy

AZURE DATABASE FOR MARIADB
Managed MariaDB database service for app developers

AZURE DATABASE FOR MYSQL
Fully managed, scalable MySQL database

AZURE DATABASE FOR POSTGRESQL
Fully managed, intelligent, and scalable PostgreSQL

AZURE DATABASE MIGRATION SERVICE
Accelerate your data migration to Azure

AZURE SQL
Modern SQL family for migration and app modernisation

AZURE SQL DATABASE
Managed, intelligent SQL in the cloud

AZURE SQL EDGE
Consume Services privately on Azure Platform

AZURE SQL MANAGED INSTANCE
Managed, always up-to-date SQL instance in the cloud

SQL SERVER ON AZURE VIRTUAL MACHINES
Migrate SQL Server workloads to the cloud at lower total cost of ownership (TCO)

TABLE STORAGE
NoSQL key-value store using semi-structured datasets

AZURE MANAGED INSTANCE FOR APACHE CASSANDRA
Cloud Cassandra with flexibility, control and scale

AZURE CONFIDENTIAL LEDGER
Tamperproof, unstructured data store hosted in trusted execution environments (TEEs)and backed by cryptographically verifiable evidence

Identity

Manage user identities and access to protect against advanced threats across devices, data, apps and infrastructure

AZURE ACTIVE DIRECTORY (AZURE AD)
Synchronise on-premises directories and enable single sign-on

AZURE ACTIVE DIRECTORY DOMAIN SERVICES
Join Azure virtual machines to a domain without domain controllers

AZURE INFORMATION PROTECTION
Better protect your sensitive information—anytime, anywhere

AZURE ACTIVE DIRECTORY EXTERNAL IDENTITIES
Consumer identity and access management in the cloud

Integration

Seamlessly integrate on-premises and cloud-based applications, data and processes across your enterprise

API MANAGEMENT
Publish APIs to developers, partners, and employees securely and at scale

AZURE HEALTH DATA SERVICES
Unify and manage health data and protected health information (PHI) in the cloud

EVENT GRID
Reliable event delivery at massive scale

LOGIC APPS
Automate the access and use of data across clouds

SERVICE BUS
Connect across private and public cloud environments

AZURE WEB PUBSUB
Easily build real-time messaging web applications using WebSockets and the publish-subscribe pattern

Developer Tools

Build, manage and continuously deliver cloud applications – using any platform or language

APP CONFIGURATION
Fast, scalable parameter storage for app configuration

AZURE DEVOPS
Services for teams to share code, track work and ship software

AZURE DEVTEST LABS
Quickly create environments using reusable templates and artifacts

AZURE PIPELINES
Continuously build, test, and deploy to any platform and cloud

SDKS
Get the SDKs and command-line tools you need

VISUAL STUDIO
Powerful and flexible environment to develop apps in the cloud

VISUAL STUDIO CODE
A powerful, lightweight code editor for cloud development

AZURE LOAD TESTING PREVIEW
Optimise app performance with high-scale load testing

Compute

Access cloud compute capacity and scale on demand – and only pay for the resources you use

APP SERVICE
Quickly create powerful cloud apps for web and mobile

AZURE CYCLECLOUD
Create, manage, operate and optimise HPC and big compute clusters of any scale

AZURE QUANTUM
Jump in and explore a diverse selection of today's quantum hardware, software, and solutions

AZURE SPOT VIRTUAL MACHINES
Provision unused compute capacity at deep discounts to run interruptible workloads

AZURE SPRING APPS
Build and deploy Spring Boot applications with a fully managed service from Microsoft and VMware

AZURE VMWARE SOLUTION
Run your VMware workloads natively on Azure

BATCH
Cloud-scale job scheduling and compute management

CLOUD SERVICES
Create highly-available, infinitely-scalable cloud applications and APIs

LINUX VIRTUAL MACHINES
Provision virtual machines for Ubuntu, Red Hat, and more

SQL SERVER ON AZURE VIRTUAL MACHINES
Migrate SQL Server workloads to the cloud at lower total cost of ownership (TCO)

STATIC WEB APPS
A modern web app service that offers streamlined full-stack development from source code to global high availability

VIRTUAL MACHINE SCALE SETS
Manage and scale up to thousands of Linux and Windows VMs

VIRTUAL MACHINES
Provision Windows and Linux virtual machines in seconds

AZURE VIRTUAL DESKTOP
Enable a secure, remote desktop experience from anywhere

AZURE DEDICATED HOST
A dedicated physical server to host your Azure VMs for Windows and Linux

AZURE VM IMAGE BUILDER
Simplify your image building process with easy to use tool

AZURE KUBERNETES SERVICE (AKS)
Build and scale with managed Kubernetes

AZURE FUNCTIONS
Execute event-driven serverless code functions with an end-to-end development experience

CONTAINER INSTANCES
Easily run containers on Azure without managing servers.

SERVICE FABRIC
Develop microservices and orchestrate containers on Windows or Linux

DevOps

Deliver innovation faster with simple, reliable tools for continuous delivery

AZURE ARTIFACTS
Create, host, and share packages with your team

AZURE BOARDS
Plan, track, and discuss work across your teams

AZURE DEVOPS
Services for teams to share code, track work and ship software

AZURE DEVTEST LABS
Quickly create environments using reusable templates and artifacts

AZURE MONITOR
Full observability into your apps, infrastructure and network

AZURE PIPELINES
Continuously build, test, and deploy to any platform and cloud

AZURE REPOS
Get unlimited, cloud-hosted private Git repos for your project

AZURE TEST PLANS
Test and ship confidently with an exploratory test toolkit

DEVOPS TOOL INTEGRATIONS
Use your favorite DevOps tools with Azure

AZURE LOAD TESTING PREVIEW
Optimise app performance with high-scale load testing

AZURE MANAGED GRAFANA PREVIEW
Deploy Grafana dashboards as a fully managed Azure service

Hybrid + multicloud

Get Azure innovation everywhere—bring the agility and innovation of cloud computing to your on-premises workloads

AZURE ACTIVE DIRECTORY (AZURE AD)
Synchronise on-premises directories and enable single sign-on

AZURE ARC
Secure, develop, and operate infrastructure, apps, and Azure services anywhere

AZURE DATABASE FOR POSTGRESQL
Fully managed, intelligent, and scalable PostgreSQL

AZURE DEVOPS
Services for teams to share code, track work and ship software

AZURE EXPRESSROUTE
Dedicated private-network fibre connections to Azure

AZURE IOT EDGE
Extend cloud intelligence and analytics to edge devices

AZURE SENTINEL
Put cloud-native SIEM and intelligent security analytics to work to help protect your enterprise

AZURE SQL DATABASE
Managed, intelligent SQL in the cloud

AZURE SQL EDGE
Consume Services privately on Azure Platform

AZURE STACK
Build and run innovative hybrid apps across cloud boundaries

SECURITY CENTER
Unify security management and enable advanced threat protection across hybrid cloud workloads

AZURE STACK HCI
Run your production workloads anywhere on hybrid, familiar hyperconverged infrastructure

AZURE STACK HUB
Azure Stack Hub is sold as an integrated hardware system, with software pre-installed on validated hardware

AZURE STACK EDGE
An Azure managed device that brings the compute, storage, and intelligence of Azure to the edge

AZURE MODULAR DATACENTRE
A complete, rugged datacentre solution

Analytics

Gather, store, process, analyse and visualise data of any variety, volume or velocity

AZURE ANALYSIS SERVICES
Enterprise-grade analytics engine as a service

AZURE DATA EXPLORER
Fast and highly scalable data exploration service

AZURE DATA FACTORY
Hybrid data integration at enterprise scale, made easy

AZURE DATA LAKE STORAGE
Scalable, secure data lake for high-performance analytics

AZURE DATA LAKE STORAGE GEN1
Hyperscale repository for big data analytics workloads

AZURE DATA SHARE
A simple and safe service for sharing big data with external organisations

AZURE DATABRICKS
Design AI with Apache Spark™-based analytics

AZURE STREAM ANALYTICS
Real-time analytics on fast-moving streaming data

AZURE SYNAPSE ANALYTICS
Limitless analytics service with unmatched time to insight

DATA CATALOGUE
Get more value from your enterprise data assets

DATA LAKE ANALYTICS
Distributed analytics service that makes big data easy

EVENT HUBS
Receive telemetry from millions of devices

HDINSIGHT
Provision cloud Hadoop, Spark, R Server, HBase, and Storm clusters

POWER BI EMBEDDED
White label Power BI to quickly and easily provide exceptional customer facing dashboards and analytics in your own applications

R SERVER FOR HDINSIGHT
Predictive analytics, machine learning and statistical modelling for big data

MICROSOFT PURVIEW
Govern, protect, and manage your data estate

MICROSOFT GRAPH DATA CONNECT PREVIEW
A secure, high-throughput connector designed to copy select Microsoft 365 productivity datasets into your Azure tenant

AZURE CHAOS STUDIO PREVIEW
Improve application resilience by introducing faults and simulating outages

Internet of Things
Bring IoT to any device and any platform, without changing your infrastructure

API MANAGEMENT
Publish APIs to developers, partners, and employees securely and at scale

AZURE COSMOS DB
Fast NoSQL database with open APIs for any scale

AZURE DIGITAL TWINS
Use IoT spatial intelligence to create models of physical environments

AZURE IOT CENTRAL
Go from proof of concept to proof of value

AZURE IOT EDGE
Extend cloud intelligence and analytics to edge devices

AZURE IOT HUB
Connect, monitor and manage billions of IoT assets

AZURE MACHINE LEARNING
Use an enterprise-grade service for the end-to-end machine learning lifecycle

AZURE MAPS
Add location data and mapping visuals to business applications and solutions

AZURE RTOS
Making embedded IoT development and connectivity easy

AZURE SPHERE
Create, connect, and maintain secured intelligent IoT devices from the edge to the cloud

AZURE SQL EDGE
Consume Services privately on Azure Platform

AZURE STREAM ANALYTICS
Real-time analytics on fast-moving streaming data

AZURE TIME SERIES INSIGHTS
Visualise IoT data in real time

EVENT GRID
Reliable event delivery at massive scale

KINECT DK
Build computer vision and speech models using a developer kit with advanced AI sensors

LOGIC APPS
Automate the access and use of data across clouds

NOTIFICATION HUBS
Send push notifications to any platform from any back-end

WINDOWS 10 IOT CORE SERVICES
Long-term OS support and services to manage device updates and assess device health

AZURE DEFENDER FOR IOT
Unified threat protection for all your IoT/OT devices

AZURE PERCEPT PREVIEW
Accelerate edge intelligence from silicon to service

WINDOWS FOR IOT
Build intelligent edge solutions with world-class developer tools, long-term support and enterprise-grade security.

AZURE FUNCTIONS
Execute event-driven serverless code functions with an end-to-end development experience

Management and Governance
Simplify, automate and optimise the management and compliance of your cloud resources

AUTOMATION
Simplify cloud management with process automation

AZURE ADVISOR
Your personalised Azure best practices recommendation engine

AZURE BACKUP
Simplify data protection and protect against ransomware

AZURE BLUEPRINTS PREVIEW
Enabling quick, repeatable creation of governed environments

AZURE LIGHTHOUSE
Empowering service providers to manage customers at scale and with precision

AZURE MANAGED APPLICATIONS
Simplify management of cloud offerings

AZURE MIGRATE
Simplify migration and modernisation with a unified platform

AZURE MOBILE APP
Stay connected to your Azure resources—anytime, anywhere

AZURE MONITOR
Full observability into your apps, infrastructure and network

AZURE POLICY
Implement corporate governance and standards at scale

AZURE RESOURCE MANAGER
Simplify how you manage your app resources

AZURE RESOURCE MANAGER TEMPLATES
Deliver infrastructure as code for all your Azure resources using Resource Manager

AZURE SERVICE HEALTH
Personalised guidance and support for when issues in Azure services affect you

AZURE SITE RECOVERY
Keep your business running with built-in disaster recovery service

CLOUD SHELL
Streamline Azure administration with a browser-based shell

MICROSOFT COST MANAGEMENT
Monitor, allocate and optimise cloud costs with transparency, accuracy and efficiency using Microsoft Cost Management

MICROSOFT AZURE PORTAL
Build, manage, and monitor all Azure products in a single, unified console

NETWORK WATCHER
Network performance monitoring and diagnostics solution

TRAFFIC MANAGER
Route incoming traffic for high performance and availability

AZURE AUTOMANAGE PREVIEW
Simplify and optimise IT management with automated operations

AZURE RESOURCE MOVER
Simplify how you move multiple resources between Azure regions

MICROSOFT PURVIEW
Govern, protect, and manage your data estate

AZURE CHAOS STUDIO PREVIEW
Improve application resilience by introducing faults and simulating outages

AZURE MANAGED GRAFANA PREVIEW
Deploy Grafana dashboards as a fully managed Azure service

UPDATE MANAGEMENT CENTRE PREVIEW
Centrally manage updates and compliance at scale

Media
Deliver high-quality video content anywhere, at any time and on any device

AZURE MEDIA PLAYER
A single player for all your playback needs

CONTENT DELIVERY NETWORK
Fast, reliable content delivery network with global reach

CONTENT PROTECTION
Securely deliver content using AES, PlayReady, Widevine, and Fairplay

ENCODING
Studio grade encoding at cloud scale

LIVE AND ON-DEMAND STREAMING
Deliver content to virtually all devices with ability to scale

MEDIA SERVICES
Encode, store and stream video and audio at scale

Security
Protect your enterprise from advanced threats across hybrid cloud workloads

APP CONFIGURATION
Fast, scalable parameter storage for app configuration

APPLICATION GATEWAY
Build secure, scalable, highly available web front ends in Azure

AZURE ACTIVE DIRECTORY (AZURE AD)
Synchronise on-premises directories and enable single sign-on

AZURE ACTIVE DIRECTORY DOMAIN SERVICES
Join Azure virtual machines to a domain without domain controllers

MICROSOFT DEFENDER FOR CLOUD
Protect your multi-cloud and hybrid environments

AZURE BASTION
Fully managed service that helps secure remote access to your virtual machines

AZURE DDOS PROTECTION
Protect your Azure resources from distributed denial-of-service (DDoS) attacks

AZURE DEDICATED HSM
Manage hardware security modules that you use in the cloud

AZURE FIREWALL
Cloud-native and intelligent network firewall security

AZURE FIREWALL MANAGER
Central network security policy and route management for globally distributed, software-defined perimeters

AZURE FRONT DOOR
Modern cloud CDN that delivers optimised experiences to your users anywhere

AZURE INFORMATION PROTECTION
Better protect your sensitive information—anytime, anywhere

AZURE SENTINEL
Put cloud-native SIEM and intelligent security analytics to work to help protect your enterprise

KEY VAULT
Safeguard and maintain control of keys and other secrets

SECURITY CENTER
Unify security management and enable advanced threat protection across hybrid cloud workloads

VPN GATEWAY
Establish secure, cross-premises connectivity

WEB APPLICATION FIREWALL
A cloud-native web application firewall (WAF) service that provides powerful protection for web apps

AZURE DEFENDER FOR IOT
Unified threat protection for all your IoT/OT devices

MICROSOFT AZURE ATTESTATION
A unified solution for remotely verifying the trustworthiness of a platform and integrity of the binaries running inside it

AZURE CONFIDENTIAL LEDGER
Tamperproof, unstructured data store hosted in trusted execution environments (TEEs) and backed by cryptographically verifiable evidence

Mobile
Build and deploy cross-platform and native apps for any mobile device

API MANAGEMENT
Publish APIs to developers, partners, and employees securely and at scale

APP CONFIGURATION
Fast, scalable parameter storage for app configuration

APP SERVICE
Quickly create powerful cloud apps for web and mobile

AZURE COGNITIVE SEARCH
Enterprise scale search for app development

AZURE MAPS
Add location data and mapping visuals to business applications and solutions

AZURE COGNITIVE SERVICES
Deploy high-quality AI models as APIs

NOTIFICATION HUBS
Send push notifications to any platform from any back-end

SPATIAL ANCHORS
Create multi-user, spatially aware mixed reality experiences

VISUAL STUDIO APP CENTRE
Build, test, release, and monitor your mobile and desktop apps

XAMARIN
Create cloud-powered mobile apps faster

AZURE COMMUNICATION SERVICES
Build rich communication experiences with the same secure platform capabilities used by Microsoft Teams

Networking
Connect cloud and on-premises infrastructure and services, to provide your customers and users with the best possible experience

APPLICATION GATEWAY
Build secure, scalable, highly available web front ends in Azure

AZURE BASTION
Fully managed service that helps secure remote access to your virtual machines

AZURE DDOS PROTECTION
Protect your Azure resources from distributed denial-of-service (DDoS) attacks

AZURE DNS
Host your Domain Name System (DNS) domain in Azure

AZURE EXPRESSROUTE
Dedicated private-network fibre connections to Azure

AZURE FIREWALL
Cloud-native and intelligent network firewall security

LOAD BALANCER
Deliver high availability and network performance to your apps

AZURE FIREWALL MANAGER
Central network security policy and route management for globally distributed, software-defined perimeters

AZURE FRONT DOOR
Modern cloud CDN that delivers optimised experiences to your users anywhere

AZURE INTERNET ANALYSER PREVIEW
Test how networking infrastructure changes will impact your customers' performance.

AZURE PRIVATE LINK
Private access to services hosted on the Azure platform, keeping your data on the Microsoft network

CONTENT DELIVERY NETWORK
Fast, reliable content delivery network with global reach

NETWORK WATCHER
Network performance monitoring and diagnostics solution

TRAFFIC MANAGER
Route incoming traffic for high performance and availability

VIRTUAL NETWORK
Create your own private network infrastructure in the cloud

VIRTUAL WAN
Optimise and automate branch-to-branch connectivity through Azure

VPN GATEWAY
Establish secure, cross-premises connectivity

WEB APPLICATION FIREWALL
A cloud-native web application firewall (WAF) service that provides powerful protection for web apps

AZURE ORBITAL PREVIEW
Satellite ground station and scheduling services for fast downlinking of data

AZURE ROUTE SERVER
Enable network appliances to exchange routes dynamically with virtual networks in Azure

AZURE NETWORK FUNCTION MANAGER
Extend Azure management for deploying 5G and SD-WAN network functions on edge devices

AZURE VIRTUAL NETWORK MANAGER PREVIEW
Centrally manage virtual networks in Azure from a single pane of glass

AZURE PRIVATE 5G CORE PREVIEW
Rapidly deploy and manage private 5G networks at the enterprise edge

Migration
Simplify and accelerate your migration to the cloud with guidance, tools and resources

AZURE DATABASE MIGRATION SERVICE
Accelerate your data migration to Azure

AZURE MIGRATE
Simplify migration and modernisation with a unified platform

AZURE SITE RECOVERY
Keep your business running with built-in disaster recovery service

MICROSOFT COST MANAGEMENT
Monitor, allocate and optimise cloud costs with transparency, accuracy and efficiency using Microsoft Cost Management

AZURE DATA BOX
Appliances and solutions for data transfer to Azure and edge compute

Mixed Reality
Blend your physical and digital worlds to create immersive, collaborative experiences

AZURE DIGITAL TWINS
Use IoT spatial intelligence to create models of physical environments

KINECT DK
Build computer vision and speech models using a developer kit with advanced AI sensors

REMOTE RENDERING
Render high-quality, interactive 3D content with real-time streaming

SPATIAL ANCHORS
Create multi-user, spatially aware mixed reality experiences

OBJECT ANCHORS PREVIEW
Automatically align and anchor 3D content to objects in the physical world

Storage
Get secure, massively scalable cloud storage for your data, apps and workloads

ARCHIVE STORAGE
Industry-leading price point for storing rarely accessed data

AVERE VFXT FOR AZURE
Run high-performance, file-based workloads in the cloud

AZURE BACKUP
Simplify data protection and protect against ransomware

AZURE DATA LAKE STORAGE
Scalable, secure data lake for high-performance analytics

AZURE DATA LAKE STORAGE GEN1
Hyperscale repository for big data analytics workloads

AZURE DATA SHARE
A simple and safe service for sharing big data with external organisations

AZURE FILES
Simple, secure and serverless enterprise-grade cloud file shares

AZURE HPC CACHE
File caching for high-performance computing (HPC)

AZURE NETAPP FILES
Enterprise-grade Azure file shares, powered by NetApp

AZURE BLOB STORAGE
Massively scalable and secure object storage

AZURE DATA BOX
Appliances and solutions for data transfer to Azure and edge compute

AZURE DISK STORAGE
High-performance, highly durable block storage

QUEUE STORAGE
Effectively scale apps according to traffic

STORAGE ACCOUNTS
Durable, highly available, and massively scalable cloud storage

STORAGE EXPLORER
View and interact with Azure Storage resources

STORSIMPLE
Lower costs with an enterprise hybrid cloud storage solution

AZURE CONFIDENTIAL LEDGER
Tamperproof, unstructured data store hosted in trusted execution environments (TEEs) and backed by cryptographically verifiable evidence

Web
Build, deploy and scale powerful web applications quickly and efficiently

API MANAGEMENT
Publish APIs to developers, partners, and employees securely and at scale

APP CONFIGURATION
Fast, scalable parameter storage for app configuration

APP SERVICE
Quickly create powerful cloud apps for web and mobile

AZURE COGNITIVE SEARCH
Enterprise scale search for app development

AZURE MAPS
Add location data and mapping visuals to business applications and solutions

AZURE SIGNALR SERVICE
Add real-time web functionalities easily

CONTENT DELIVERY NETWORK
Fast, reliable content delivery network with global reach

NOTIFICATION HUBS
Send push notifications to any platform from any back-end

STATIC WEB APPS
A modern web app service that offers streamlined full-stack development from source code to global high availability

AZURE COMMUNICATION SERVICES
Build rich communication experiences with the same secure platform capabilities used by Microsoft Teams

AZURE WEB PUBSUB
Easily build real-time messaging web applications using WebSockets and the publish-subscribe pattern

AZURE FLUID RELAY PREVIEW
Easily add real-time collaborative experiences to your apps with Fluid Framework

WEB APP FOR CONTAINERS
Easily deploy and run containerised web apps on Windows and Linux

Windows Virtual Desktop
The best virtual desktop experience – delivered on Azure

AZURE LAB SERVICES
Set up labs for education, training, and other related scenarios

AZURE VIRTUAL DESKTOP
Enable a secure, remote desktop experience from anywhere

VMWARE HORIZON CLOUD ON MICROSOFT AZURE
Provision Windows desktops and apps with VMware and Azure Virtual Desktop

CITRIX VIRTUAL APPS AND DESKTOPS FOR AZURE
Provision Windows desktops and apps on Azure with Citrix and Azure Virtual Desktop

TechWorld WithMilan
simplifying complex topics