datacamp

Conda Cheat Sheet

Learn data science online at www.DataCamp.com

> Definitions

Conda is an application for data science package management and environment management. It is primarily used for managing Python packages, though it also supports R, Ruby, Lua, Scala, Java, JavaScript, C, C++, and Fortran packages.

A **package** is a pre-built software package, including code, libraries needed to run the code, and metadata. Conda packages are .conda files or .tar.bz2 files. Popular packages include pandas, seaborn, and r-dplyr.

A **channel** is a location that hosts packages. Popular channels include conda-forge, bioconda, and intel.

An **environment** is a directory containing installed packages. Having multiple environments lets you use different versions of packages for different projects.

anaconda is a meta-package that allows you to install hundreds of data science packages with a single conda command.

Mamba is an open source reimplementation of Conda with the same syntax and some performance improvements. The majority of the code in this cheat sheet will also work with Mamba.

Conda vs. Pip

Feature	Pip	Conda
Package management	✅	✅
Environment management	❌	✅
Language support	Python only	Many data languages
Installation	Included with Python	Separate install
License	MIT	BSD

> Getting help

```
# Display Conda version with conda --version
conda --version

# Display Conda system info with conda info
conda info

# Get help on Conda with conda --help
docker --help

# Get help on Conda command usage with conda {command} --help
docker build --help
```

> Listing packages

```
# List all installed packages with conda list
conda list

# List installed packages matching a regular expression with conda list {regex}
conda list ^z # lists packages with names starting with z

# List all versions of all packages in all conda channels with conda search
conda search

# List all versions of a package (all channels) with conda search {pkg}
conda search scikit-learn

# List versions of a package (all channels) with conda search '{pkg}{version}'
conda search 'scikit-learn>=1'

# List package versions for a conda channel with conda search {channel}::{pkg}
conda search conda-forge::scikit-learn
```

> Installing & managing packages

```
# Install packages with conda install {pkg1} {pkg2} ...
conda install numpy pandas

# Install specific version of package with conda install {pkg}={version}
conda install scipy=1.10.1

# Update all packages with conda update --all
conda update --all

# Uninstall packages with conda uninstall {pkg}
conda uninstall pycaret
```

> Working with channels

```
# List channels with conda config --show channels
conda config --show channels

# Add a channel (highest priority) with conda config --prepend channels {channel}
conda config --prepend channels conda-forge

# Add a channel (lowest priority) with conda config --append channels {channel}
conda config --append channels bioconda
```

> Working with environments

```
# List environments with conda env list
conda env list

# Restrict command scope to an environment by appending --name {envname}
conda list --name base
conda install scikit-learn --name myenv
```

> Managing environments

```
# Create an environment for technology with conda create -n {env}
conda create --name my_python_env

# Clone existing environment with conda create --clone {old_env} --name {new_env}
conda create --clone template_env --name project_env

# Create environment, auto-accepting prompts with conda create --yes --name {env}
# For non-interactive usage
conda create --yes --name my_env

# Make environment the default environment with conda activate {env}
# This prepends the environment directory to the system PATH environment variable
conda activate my_env

# Make the base environment the default with conda deactivate {env}
# This removes the environment directory from system PATH environment variable
conda deactivate my_env
```

> Sharing environments

```
# Export active environment to a YAML file with conda env export > environment.yml
# Export every package including dependencies (maximum reproducibility)
conda env export > environment.yml
# Export only packages explicitly asked for (increased portability)
conda env export --from-history > environment.yml

# Import environment from YAML with conda env create --name {env} --file {yaml_file}
conda env create --name my_env2 --file environment.yml

# Export list of packages to TEXT file with conda list --export > requirements.txt
# Usually requires manual editing; you can also use pip freeze
conda list --export > requirements.txt

# Import environment from TEXT file with conda create --name {env} --file {yaml_file}
conda create --name my_env --file  requirements.txt
```

datacamp

Python For Data Science

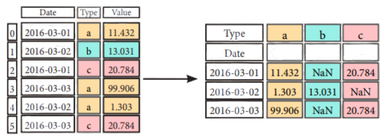
Data Wrangling in Pandas Cheat Sheet

Learn Data Wrangling online at www.DataCamp.com

> Reshaping Data

Pivot

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

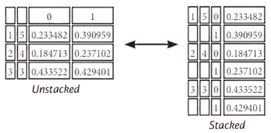


Pivot Table

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

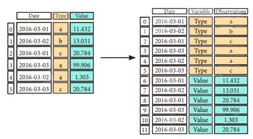
Stack / Unstack

```
>>> stacked = df5.stack() #Pivot a level of column labels
>>> stacked.unstack() #Pivot a level of index labels
```



Melt

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



> Iteration

```
>>> df.iteritems() #(Column-index, Series) pairs
>>> df.iterrows() #(Row-index, Series) pairs
```

> Missing Data

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

> Advanced Indexing

Also see NumPy Arrays

Selecting

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

Indexing With `isin()`

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

Where

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

Query

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

Setting/Resetting Index

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

Reindexing

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

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

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

Backward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

```
0    3
1    3
2    3
3    3
4    3
```

MultiIndexing

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

> Duplicate Data

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

> Grouping Data

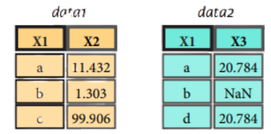
Aggregation

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

Transformation

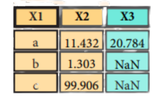
```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

> Combining Data

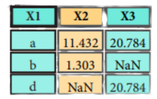


Merge

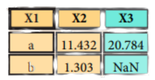
```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```



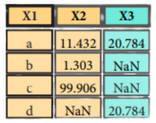

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```




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




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



Join

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

Concatenate

Vertical

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

Horizontal/Vertical

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

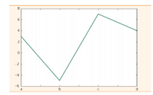
> Dates

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

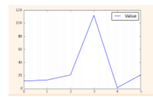
> Visualization

Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> s.plot()
>>> plt.show()
```



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



Learn Data Skills Online at www.DataCamp.com

Python For Data Science
Importing Data Cheat Sheet

Learn Python online at www.DataCamp.com

> Importing Data in Python

Most of the time, you'll use either NumPy or pandas to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

> Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

> Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Using the context manager with

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline()) #Read a single line
        print(file.readline())
        print(file.readline())
```

Table Data: Flat Files

Importing Flat Files with NumPy

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r') #Open the file for reading
>>> text = file.read() #Read a file's contents
>>> print(file.closed) #Check whether file is closed
>>> file.close() #Close file
>>> print(text)
```

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',', #String used to separate values
                      skiprows=2, #Skip the first 2 lines
                      usecols=[0,2], #Read the 1st and 3rd column
                      dtype=str) #The type of the resulting array
```

Files with mixed data type

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True, #Look for column header
                         dtype=None)
>>> data_array = np.recfromcsv(filename)
#The default dtype of the np.recfromcsv() function is None
```

Importing Flat Files with Pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5, #Number of rows of file to read
                       header=None, #Row number to use as col names
                       sep='\t', #Delimiter to use
                       comment='#', #Character to split comments
                       na_values=[""]) #String to recognize as NA/NaN
```

> Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype #Data type of array elements
>>> data_array.shape #Array dimensions
>>> len(data_array) #Length of array
```

Pandas DataFrames

```
>>> df.head() #Return first DataFrame rows
>>> df.tail() #Return last DataFrame rows
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> data_array = data.values #Convert a DataFrame to an a NumPy array
```

> SAS File

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

> Stata File

```
>>> data = pd.read_stata('urbanpop.dta')
```

> Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                           'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` **attribute:**

```
>>> data.sheet_names
```

> Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` **method to fetch a list of table names:**

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager with

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

> Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

> Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

> HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

> Exploring Dictionaries

Querying relational databases with pandas

```
>>> print(mat.keys()) #Print dictionary keys
>>> for key in data.keys(): #Print dictionary keys
        print(key)
meta
quality
strain
>>> pickled_data.values() #Return dictionary values
>>> print(mat.items()) #Returns items in list format of (key, value) tuple pairs
```

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys() #Explore the HDF5
structure
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
#Retrieve the value for a key
>>> print(data['meta']['Description'].value)
```

> Navigating Your FileSystem

Magic Commands

```
!ls  #List directory contents of files and directories
%cd .. #Change current working directory
%pwd #Return the current working directory path
```

OS Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd() #Store the name of current directory in a string
>>> os.listdir(wd) #Output contents of the directory in a list
>>> os.chdir(path) #Change current working directory
>>> os.rename("test1.txt", #Rename a file
              "test2.txt")
>>> os.remove("test1.txt") #Delete an existing file
>>> os.mkdir("newdir") #Create a new directory
```

Joining Data in SQL

Cheat Sheet

Learn SQL online at www.DataCamp.com

> Definitions used throughout this cheat sheet

Primary key:
A primary key is a field in a table that uniquely identifies each record in the table. In relational databases, primary keys can be used as fields to join tables on.

Foreign key:
A foreign key is a field in a table which references the primary key of another table. In a relational database, one way to join two tables is by connecting the foreign key from one table to the primary key of another.

One-to-one relationship:
Database relationships describe the relationships between records in different tables. When a one-to-one relationship exists between two tables, a given record in one table is uniquely related to exactly one record in the other table.

One-to-many relationship:
In a one-to-many relationship, a record in one table can be related to one or more records in a second table. However, a given record in the second table will only be related to one record in the first table.

Many-to-many relationship:
In a many-to-many relationship, records in a given table 'A' can be related to one or more records in another table 'B', and records in table B can also be related to many records in table A.

> Sample Data

Artist Table

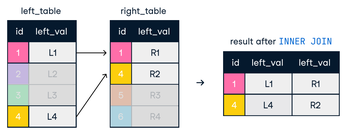
artist_id	name
1	AC/DC
2	Aerosmith
3	Alanis Morissette

Album Table

album_id	title	artist_id
1	For those who rock	1
2	Dream on	2
3	Restless and wild	2
4	Let there be rock	1
5	Rumours	6

INNER JOIN

An inner join between two tables will return only records where a joining field, such as a key, finds a match in both tables.



INNER JOIN join ON one field

```
SELECT *
FROM artist AS art
INNER JOIN album AS alb
ON art.artist_id = alb.artist_id;
```

Result after INNER JOIN:

album_id	name	artist_id
1	AC/DC	1
1	AC/DC	4
2	Aerosmith	2
2	Aerosmith	3

INNER JOIN with USING

```
SELECT *
FROM artist AS art
INNER JOIN album AS alb
USING (artist_id);
```

SELF JOIN

Self-joins are used to compare values in a table to other values of the same table by joining different parts of a table together.

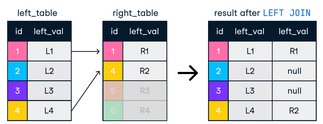
```
SELECT
  art1.artist_id,
  art1.title AS art1_title,
  art2.title AS art2_title
FROM artist as art1
INNER JOIN artist as art2
ON art1.artist_id = art2.album_id;
```

Result after Self join:

artist_id	art1_title	art2_title
1	For those who rock	For those who rock
2	Dream on	Dream on
2	Restless and wild	Dream on
1	Let there be rock	For those who rock

LEFT JOIN

A left join keeps all of the original records in the left table and returns missing values for any columns from the right table where the joining field did not find a match.



LEFT JOIN on one field

```
SELECT *
FROM artist AS art
LEFT JOIN album AS alb
ON art.artist_id = alb.album_id;
```

Result after LEFT JOIN:

artist_id	name	album_id	title	name
1	AC/DC	1	For those who rock	1
1	AC/DC	4	Let there be rock	1
2	Aerosmith	2	Dream on	2
2	Aerosmith	3	Restless and wild	2
3	Alanis Morissette	null	null	null

RIGHT JOIN

A right join keeps all of the original records in the right table and returns missing values for any columns from the left table where the joining field did not find a match. Right joins are far less common than left joins, because right joins can always be re-written as left joins.

RIGHT JOIN on one field

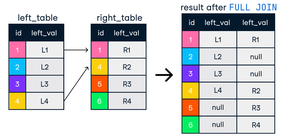
```
SELECT*
FROM artist as art
RIGHT JOIN album AS alb
ON art.artist_id = alb.album_id;
```

Result after RIGHT JOIN:

artist_id	name	album_id	title	name
1	AC/DC	1	For those who rock	1
1	Aerosmith	2	Dream on	2
2	Aerosmith	3	Restless and wild	2
2	AC/DC	4	Let there be rock	1
3	null	5	Rumours	6

FULL JOIN

A full join combines a left join and right join. A full join will return all records from a table, irrespective of whether there is a match on the joining field in the other table, returning null values accordingly.



FULL JOIN on one field

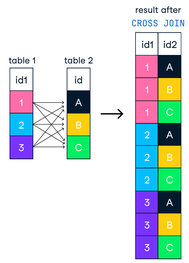
```
SELECT *
FROM artist as art
FULL OUTER JOIN album AS alb
ON art.artist_id = alb.album_id;
```

Result after FULL JOIN:

artist_id	name	album_id	title	name
1	AC/DC	1	For those who rock	1
1	AC/DC	4	Let there be rock	1
2	Aerosmith	2	Balls to the wall	2
2	Aerosmith	3	Restless and wild	2
3	Alanis Morissette	null	null	null
null	null	5	Rumours	6

CROSS JOIN

`CROSS JOIN` creates all possible combinations of two tables. `CROSS JOIN` does not require a field to join `ON`.




```
SELECT name, title
FROM artist
CROSS JOIN album;
```

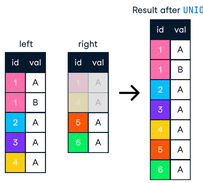
Result after CROSS JOIN:

name	title
AC/DC	For those who rock
AC/DC	Dream on
AC/DC	Restless and wild
AC/DC	Let there be rock
AC/DC	Rumours
Aerosmith	For those who rock
Aerosmith	Dream on
Aerosmith	Restless and wild
Aerosmith	Let there be rock
Aerosmith	Rumours
Alanis Morissette	For those who rock
Alanis Morissette	Dream on
Alanis Morissette	Restless and wild
Alanis Morissette	Let there be rock
Alanis Morissette	Rumours

Set Theory Operators in SQL

UNION

The `UNION` operator is used to vertically combine the results of two `SELECT` statements. For `UNION` to work without errors, all `SELECT` statements must have the same number of columns and corresponding columns must have the same data type. `UNION` does not return duplicates.



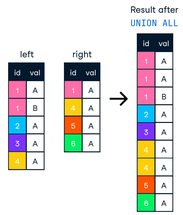

```
SELECT artist_id
FROM artist
UNION
SELECT artist_id
FROM album;
```

Result after UNION:

artist_id
1
2
3
6

UNION ALL

The `UNION ALL` operator works just like `UNION`, but it returns duplicate values. The same restrictions of `UNION` hold true for `UNION ALL`



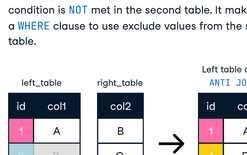

```
SELECT artist_id
FROM artist
UNION ALL
SELECT artist_id
FROM album;
```

Result after UNION ALL:

artist_id
1
2
3
1
2
2
1
6

INTERSECT

The `INTERSECT` operator returns only identical rows from two tables.




```
SELECT artist_id
FROM artist
INTERSECT
SELECT artist_id
FROM album;
```

Result after INTERSECT:

artist_id
1
2

EXCEPT

The `EXCEPT` operator returns only those rows from the left table that are not present in the right table.

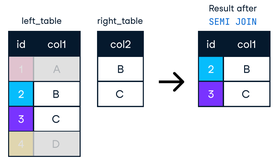

```
SELECT artist_id
FROM artist
EXCEPT
SELECT artist_id
FROM album;
```

Result after EXCEPT:

artist_id
1
2
3

SEMI JOIN

A semi join chooses records in the first table where a condition is met in the second table. A semi join makes use of a `WHERE` clause to use the second table as a filter for the first.




```
SELECT *
FROM album
WHERE artist_id IN
(SELECT artist_id
FROM artist);
```

Result after Semi join:

album_id	title	artist_id
1	For those who rock	1
2	Dream on	2
3	Restless and wild	2

ANTI JOIN

The anti join chooses records in the first table where a condition is `NOT` met in the second table. It makes use of a `WHERE` clause to use exclude values from the second table.


```
SELECT *
FROM album
WHERE artist_id NOT IN
(SELECT artist_id
FROM artist);
```

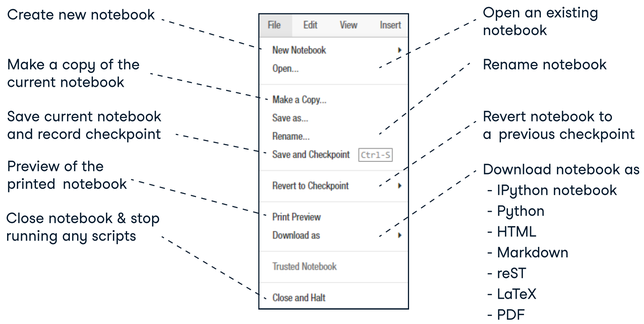
Result after Anti join:

album_id	title	artist_id
5	Rumours	6

Python For Data Science
Jupyter Cheat Sheet

Learn Jupyter online at www.DataCamp.com

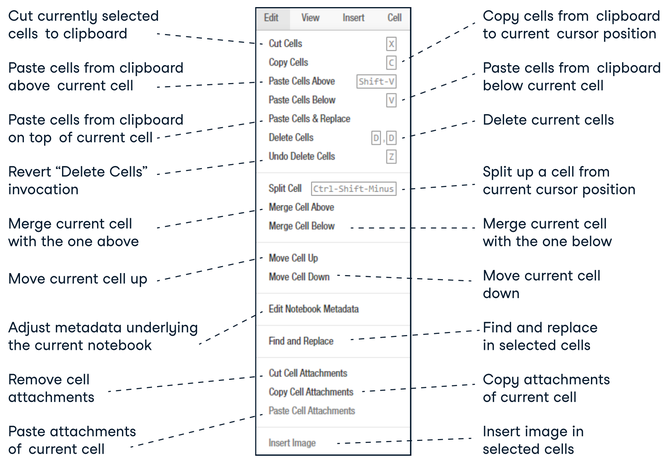
> Saving/Loading Notebooks



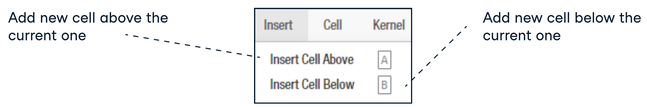
> Writing Code And Text

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

Edit Cells



Insert Cells

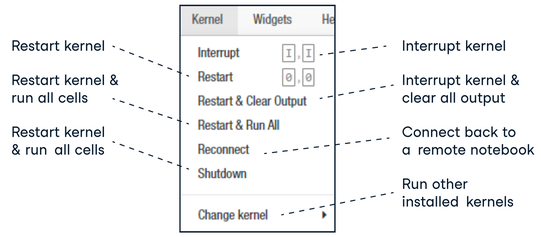


> Working with Different Programming Languages

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

IPython IRkernel IJulia

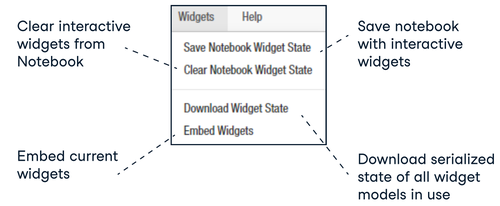
Installing Jupyter Notebook will automatically install the IPython kernel.



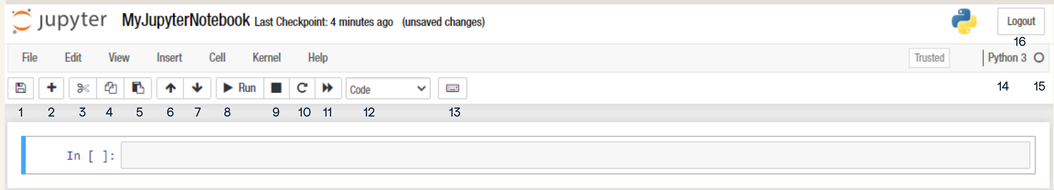
> Widgets

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.

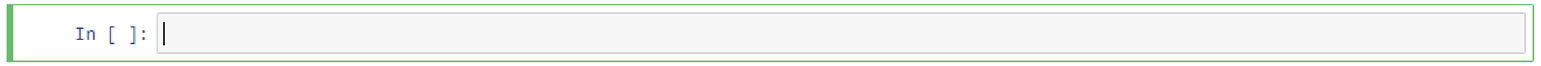


Command Mode:

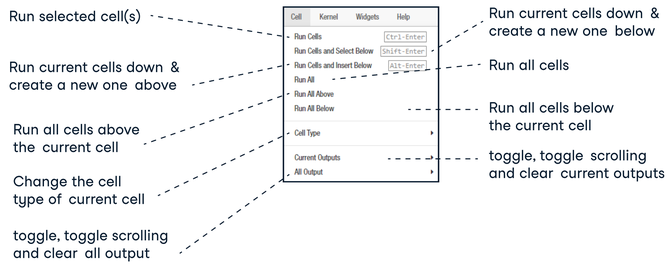


1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Restart kernel and re-run notebook
12. Display characteristics
13. Open command palette
14. Current kernel
15.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

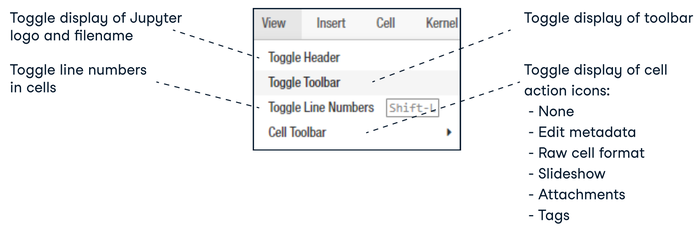
Edit Mode:



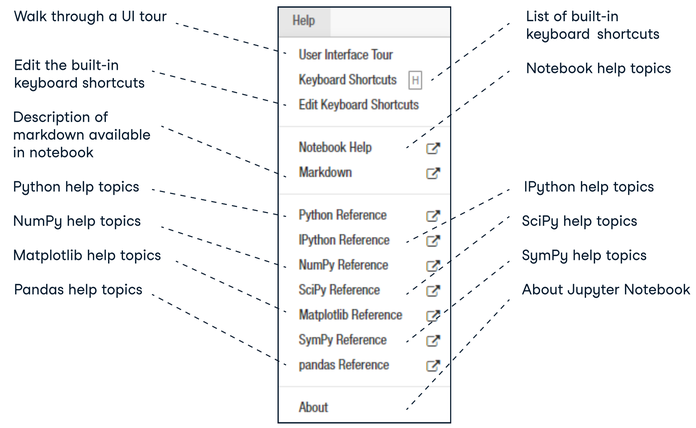
Executing Cells



View Cells



Asking For Help



Python For Data Science
Pandas Basics Cheat Sheet

Learn Pandas Basics online 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

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

Columns →

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

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

> Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

> Asking For Help

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

> Sort & Rank

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

> 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')
>>> df.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()

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

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element
 -5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
 1 India New Delhi 1303171035
 2 Brazil Brasília 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
 Country Brazil
 Capital Brasília
 Population 207847528
>>> df.ix[:,'Capital'] #Select a single column of subset of columns
 0 Brussels
 1 New Delhi
 2 Brasília
>>> df.ix[1,'Capital'] #Select rows and columns
 'New Delhi'
```

Boolean Indexing

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

Setting

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

> Retrieving Series/DataFrame Information

Basic Information

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

Summary

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

> Applying Functions

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

> Data Alignment

Internal Data Alignment

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

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

Arithmetic Operations with Fill Methods

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

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

datacamp

Python For Data Science

PySpark RDD Cheat Sheet

Learn PySpark RDD online at www.DataCamp.com

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

> Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version #Retrieve SparkContext version
>>> sc.pythonVer #Retrieve Python version
>>> sc.master #Master URL to connect to
>>> str(sc.sparkHome) #Path where Spark is installed on worker nodes
>>> str(sc.sparkUser()) #Retrieve name of the Spark User running SparkContext
>>> sc.appName #Return application name
>>> sc.applicationId #Retrieve application ID
>>> sc.defaultParallelism #Return default level of parallelism
>>> sc.defaultMinPartitions #Default minimum number of partitions for RDDs
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

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

> Loading Data

Parallelized Collections

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

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

> Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions() #List the number of partitions
>>> rdd.count() #Count RDD instances 3
>>> rdd.countByKey() #Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue() #Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap() #Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum() #Sum of RDD elements 4950
>>> sc.parallelize([]).isEmpty() #Check whether RDD is empty
 True
```

Summary

```
>>> rdd3.max() #Maximum value of RDD elements
 99
>>> rdd3.min() #Minimum value of RDD elements
 0
>>> rdd3.mean() #Mean value of RDD elements
 49.5
>>> rdd3.stdev() #Standard deviation of RDD elements
 28.866070047722118
>>> rdd3.variance() #Compute variance of RDD elements
 833.25
>>> rdd3.histogram(3) #Compute histogram by bins
 ([0,33,66,99],[33,33,34])
>>> rdd3.stats() #Summary statistics (count, mean, stdev, max & min)
```

> Applying Functions

```
#Apply a function to each RDD element
>>> rdd.map(lambda x: x+(x[1],x[0])).collect()
 [('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]
#Apply a function to each RDD element and flatten the result
>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))
>>> rdd5.collect()
 ['a',7,7,'a','a',2,2,'a','b',2,2,'b']
#Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
>>> rdd4.flatMapValues(lambda x: x).collect()
 [('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]
```

> Selecting Data

Getting

```
>>> rdd.collect() #Return a list with all RDD elements
 [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2) #Take first 2 RDD elements
 [('a', 7), ('a', 2)]
>>> rdd.first() #Take first RDD element
 ('a', 7)
>>> rdd.top(2) #Take top 2 RDD elements
 [('b', 2), ('a', 7)]
```

Sampling

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

Filtering

```
>>> rdd.filter(lambda x: "a" in x).collect()  #Filter the RDD
 [('a',7),('a',2)]
>>> rdd5.distinct().collect() #Return distinct RDD values
 ['a',2,'b',7]
>>> rdd.keys().collect() #Return (key,value) RDD's keys
 ['a', 'a', 'b']
```

> Iterating

```
>>> def g(x): print(x)
>>> rdd.foreach(g) #Apply a function to all RDD elements
 ('a', 7)
 ('b', 2)
 ('a', 2)
```

> Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y).collect() #Merge the rdd values for each key
 [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b) #Merge the rdd values
 ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2) #Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey() #Group rdd by key
       .mapValues(list)
       .collect()
 [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
#Aggregate RDD elements of each partition and then the results
>>> rdd3.aggregate((0,0),seqOp,combOp)
 (4950,100)
 #Aggregate values of each RDD key
>>> rdd.aggregateByKey((0,0),seqop,combop).collect()
 [('a',(9,2)), ('b',(2,1))]
#Aggregate the elements of each partition, and then the results
>>> rdd3.fold(0,add)
 4950
#Merge the values for each key
>>> rdd.foldByKey(0, add).collect()
 [('a',9),('b',2)]
#Create tuples of RDD elements by applying a function
>>> rdd3.keyBy(lambda x: x+x).collect()
```

> Mathematical Operations

```
>>> rdd.subtract(rdd2).collect() #Return each rdd value not contained in rdd2
 [('b',2),('a',7)]
#Return each (key,value) pair of rdd2  with no matching key in rdd
>>> rdd2.subtractByKey(rdd).collect()
 [('d', 1)]
>>> rdd.cartesian(rdd2).collect() #Return the Cartesian product of rdd and rdd2
```

> Sort

```
>>> rdd2.sortBy(lambda x: x[1]).collect() #Sort RDD by given function
 [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey().collect() #Sort (key, value) RDD by key
 [('a',2),('b',1),('d',1)]
```

> Repartitioning

```
>>> rdd.repartition(4) #New RDD with 4 partitions
>>> rdd.coalesce(1) #Decrease the number of partitions in the RDD to 1
```

> Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

> Stopping SparkContext

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

> Execution

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

datacamp

Python For Data Science

PySpark SQL Basics Cheat Sheet

Learn PySpark SQL online 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")
```

> Filter

```
#Filter entries of age, only keep those records of which the values are >24
>>> df.filter(df["age"]>24).show()
```

> Duplicate Values

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

> Queries

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

Select

```
>>> df.select("firstName").show() #Show all entries in firstName column
>>> df.select("firstName","lastName") \
      .show()
>>> df.select("firstName", #Show all entries in firstName, age and type
              "age",
              explode("phoneNumber") \
              .alias("contactInfo")) \
      .select("contactInfo.type",
              "firstName",
              "age") \
      .show()
>>> df.select(df["firstName"],df["age"]+ 1)  #Show all entries in firstName and age,
      .show()                                  add 1 to the entries of age
>>> df.select(df['age'] > 24).show() #Show all entries where age >24
```

When

```
>>> df.select("firstName", #Show firstName and 0 or 1 depending on age >30
              F.when(df.age > 30, 1) \
              .otherwise(0)) \
      .show()
>>> df[df.firstName.isin("Jane","Boris")] #Show firstName if in the given options
                 .collect()
```

Like

```
>>> df.select("firstName", #Show firstName, and lastName is TRUE if lastName is like Smith
              df.lastName.like("Smith")) \
      .show()
```

Startswith - Endswith

```
>>> df.select("firstName", #Show firstName, and TRUE if lastName starts with Sm
              df.lastName \
                .startswith("Sm")) \
      .show()
>>> df.select(df.lastName.endswith("th"))\ #Show last names ending in th
      .show()
```

Substring

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

Between

```
>>> df.select(df.age.between(22, 24)) \ #Show age: values are TRUE if between 22 and 24
      .show()
```

> 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)
```

> Missing & Replacing Values

```
>>> df.na.fill(50).show() #Replace null values
>>> df.na.drop().show() #Return new df omitting rows with null values
>>> df.na \ #Return new df replacing one value with another
      .replace(10, 20) \
      .show()
```

> GroupBy

```
>>> df.groupBy("age")\ #Group by age, count the members in the groups
      .count() \
      .show()
```

> Sort

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

> Repartitioning

```
>>> df.repartition(10)\ #df with 10 partitions
      .rdd \
      .getNumPartitions()
>>> df.coalesce(1).rdd.getNumPartitions() #df with 1 partition
```

> Running 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()
```

> 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
```

> Output

Data Structures

```
>>> rdd1 = df.rdd #Convert df into an RDD
>>> df.toJSON().first() #Convert df into a RDD of string
>>> df.toPandas() #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()
```

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')
```

SELENIUM
CHEAT SHEET

Driver Initialization Basics

- Firefox — WebDriver driver = new FirefoxDriver();
- Chrome — WebDriver driver = new ChromeDriver();
- Internet Explorer — WebDriver driver = new SafariDriver();
- Safari Driver — WebDriver driver = new InternetExplorerDriver();

Selenium Locators

- Locating by ID
  driver.findElement(By.id("q")).sendKeys("Se lenium 3");
- Locating by Name
  driver.findElement(By.name("q")).sendKeys ("Selenium 3");
- Locating by Xpath
  driver.findElement(By.xpath("//input[@id='q']")).sendKeys ("Selenium 3");
- Locating Hyperlinks by Link Text
  driver.FindElement(By.LinkText("edit this page")).Click();
- Locating by DOM
  dom =document.getElementByIId('signinForm')
- Locating by CSS
  driver.FindElement(By.CssSelector("#rightb ar >.menu>li:nth-of-type(2)>h4"));
- Locating by ClassName
  driver.findElement(By.className("profileheader"));
- Locating by TagName
  driver.findElement(By.tagName("select")).C lick();
- Locating by LinkText
  driver.findElement(By.linkText("NextP age")).click();
- Locating by PartialLinkText
  driver.findElement(By.partialLinkText(" NextP")).click();

Selenium Navigators

- Navigate to url
  driver.get("http://newexample.com")
  driver.navigate().to("http://newexample.com")
- Refresh page
  driver.navigate().refresh()
- Navigate forwards in browser history
  driver.navigate().forward()
- Navigate backwards in browser history
  driver.navigate().back()

Test NG

@BeforeSuite @AfterSuite @BeforeTest @AfterTest @BeforeGroups @AfterGroups @BeforeClass @AfterClass @BeforeMethod @AfterMethod

JUNIT

@After @AfterClass @Before @BeforeClass @Ignore @Test

Windows

```
String handle=driver.getWindowHandle();
Set<String> handles = getWindowHandles();
driver.switchTo().window(handle);
```

- How to switch to newly created window
  String curWindow=driver.getWindowHandle();
- Get all window handles

```
Set<String> handles = getWindowHandles();
for(string handle: handles)
{
        if (!handle.equals(curWindow))
        {
        driver.switchTo().window(handle);
        }
}
```

Frames

- Using Frame Index: driver.switchTO().frame(1);
- Using Name of Frame: driver.switchTo().frame("name")
- Using web Element Object: driver.switchTO().frame(element);
- Get back to main document: driver.switchTO().defaultContent();

Alerts

```
driver.switchTO().alert.getText();
driver.switchTO().alert.accept();
driver.switchTO().alert.dismiss();
driver.switchTO().alert.sendKeys("Text");
```

Operations

- Launch Webpage — driver.get("www.webdriverinselenium.com");
- Click Button — driver.findElement(By.id("submit")).click();
- Handle Alert — Alert Alertpopup = driver.switchTo().alert();
- Disable a Field — driver.getElementsByName('') [0].setAttribute('disabled','');
- Enable a Field — driver.getElementsByName('') [0].removeAttribute('disabled'; Screenshot.File snapshot=((TakesScreenshot)driver) .getScreenshotAs(OutputTypeFILE); FileUtils.copyFile(snapshot, newFile("C:\\screenshot.jpg"));
- Print Title of Page — String pagetitle = driver.getTitle(); System.out.print(pagetitle);
- Implicit Wait — driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); Explicit Wait WebDriverWait wait=new WebDriverWait(driver, 20);
- Sleep — Thread.Sleep(10);

SeleniumGrid

- Start hub: java-jar selenium-server- standalone-x.xx.x.jar-role hub
- Start node: java-jar selenium-server- standalone-x.xx.x.jar-role node-hub
- Server: http://localhost:4444/grid/console

FURTHERMORE:

Selenium Testing Training

IntelliPaat

datacamp

python™

Working with text data in Python

Learn Python online at www.DataCamp.com

> Example data used throughout this cheat sheet

Throughout this cheat sheet, we'll be using two pandas series named `suits` and `rock_paper_scissors`.

```
import pandas as pd

suits = pd.Series(["clubs", "Diamonds", "hearts", "Spades"])
rock_paper_scissors = pd.Series(["rock ", " paper", "scissors"])
```

> String lengths and substrings

```
# Get the number of characters with .str.len()
suits.str.len() # Returns 5 8 6 6

# Get substrings by position with .str[]
suits.str[2:5] # Returns "ubs" "amo" "art" "ade"

# Get substrings by negative position with .str[]
suits.str[:-3] # "cl" "Diamo" "hea" "Spa

# Remove whitespace from the start/end with .str.strip()
rock_paper_scissors.str.strip() # "rock" "paper" "scissors"

# Pad strings to a given length with .str.pad()
suits.str.pad(8, fillchar="_") # "___clubs" "Diamonds" "__hearts" "__Spades"
```

> Changing case

```
# Convert to lowercase with .str.lower()
suits.str.lower() # "clubs" "diamonds" "hearts" "spades"

# Convert to uppercase with .str.upper()
suits.str.upper() # "CLUBS" "DIAMONDS" "HEARTS" "SPADES"

# Convert to title case with .str.title()
pd.Series("hello, world!").str.title() # "Hello, World!"

# Convert to sentence case with .str.capitalize()
pd.Series("hello, world!").str.capitalize() # "Hello, world!"
```

> Formatting settings

```
# Generate an example DataFramed named df
df = pd.DataFrame({"x": [0.123, 4.567, 8.901]})
#     x
#  0 0.123
#  1 4.567
#  2 8.901
```

```
# Visualize and format table output
df.style.format(precision = 1)
```

-	x
0	0.1
1	4.5
2	8.9

The output of `style.format` is an HTML table

> Splitting strings

```
# Split strings into list of characters with .str.split(pat="")
suits.str.split(pat="")

# [, "c" "l" "u" "b" "s", ]
# [, "D" "i" "a" "m" "o" "n" "d" "s", ]
# [, "h" "e" "a" "r" "t" "s", ]
# [, "S" "p" "a" "d" "e" "s", ]

# Split strings by a separator with .str.split()
suits.str.split(pat = "a")

# ["clubs"]
# ["Di", "monds"]
# ["he", "rts"]
# ["Sp", "des"]

# Split strings and return DataFrame with .str.split(expand=True)
suits.str.split(pat = "a", expand=True)

#        0      1
# 0  clubs   None
# 1     Di  monds
# 2     he    rts
# 3     Sp    des
```

> Joining or concatenating strings

```
# Combine two strings with +
suits + "5" # "clubs5" "Diamonds5" "hearts5" "Spades5"

# Collapse character vector to string with .str.cat()
suits.str.cat(sep=", ") # "clubs, Diamonds, hearts, Spades"

# Duplicate and concatenate strings with *
suits * 2 # "clubsclubs" "DiamondsDiamonds" "heartshearts" "SpadesSpades"
```

> Detecting Matches

```
# Detect if a regex pattern is present in strings with .str.contains()
suits.str.contains("[ae]") # False True True True

# Count the number of matches with .str.count()
suits.str.count("[ae]") # 0 1 2 2

# Locate the position of substrings with str.find()
suits.str.find("e") # -1 -1 1 4
```

> Extracting matches

```
# Extract matches from strings with str.findall()
suits.str.findall(".[ae]") # [] ["ia"] ["he"[ ["pa", "de"]

# Extract capture groups with .str.extractall()
suits.str.extractall("([ae])(.)")
#           0 1
#   match
# 1 0       a m
# 2 0       e a
# 3 0       a d
#   1       e s

# Get subset of strings that match with x[x.str.contains()]
suits[suits.str.contains("d")] # "Diamonds" "Spades"
```

> Replacing matches

```
# Replace a regex match with another string with .str.replace()
suits.str.replace("a", "4") # "clubs" "Di4monds" "he4rts" "Sp4des"

# Remove a suffix with .str.removesuffix()
suits.str.removesuffix # "club" "Diamond" "heart" "Spade"

# Replace a substring with .str.slice_replace()
rhymes = pd.Series(["vein", "gain", "deign"])
rhymes.str.slice_replace(0, 1, "r") # "rein" "rain" "reign"
```

Learn Python Online at
www.DataCamp.com

datacamp