Data Science Cheat Sheet

A reference guide to common tasks

Contents

[About This Document 2](#_Toc60771502)

[Dictionary of Terms 3](#_Toc60771503)

[General Python 4](#_Toc60771504)

[Numpy 5](#_Toc60771505)

[Pandas 6](#_Toc60771506)

[Reading Files 6](#_Toc60771507)

[Selection 6](#_Toc60771508)

[Cleaning 6](#_Toc60771509)

[Filtering 6](#_Toc60771510)

[Analysis 6](#_Toc60771511)

[Joining 7](#_Toc60771512)

[Matplotlib 9](#_Toc60771513)

[Seaborn 10](#_Toc60771514)

[Appendix A 11](#_Toc60771515)

# About This Document

I began creating this document with the intent to use it as a reference to common programming tasks knowing early on that I’d want to re-use bits and pieces of code. Thus far I’ve found package documentation to be exceptional but inefficiently slow. Hence, the scope of this document isn’t to replace documentation, but rather provide structure to approaches for solving common problems.

# Dictionary of Terms

|  |  |
| --- | --- |
| **Term** | **Description** |
| **Mutating Joins** | Combines data from two tables based on matching criteria from both |
| **Filtering Joins** | Filters observations in one table based on matching criteria with another table |

# General Python

[Python 3.9.1 Documentation (python.org)](https://docs.python.org/3/)

Line Continuation

# Use of \ at the end of a line allows code to continue on the next line

List Comprehension

Used to build lists using an iterator in a condensed format

New\_list = [ x \*\* 2 for x in range(10)]

Dictionary Comprehension

Used to build dictionaries with an iterator in a condensed format

# building a dictionary from two lists of the same length using dict comprehension

New\_dictionary = {key\_list[i] : value\_list[i] for i in range(len(key\_list))}

-- [end of section] --

# Numpy

[NumPy v1.19 Documentation (numpy.org)](https://numpy.org/doc/stable/index.html)

-- [end of section] --

# Pandas

[Pandas 1.2.0 Documentation (pydata.org)](https://pandas.pydata.org/docs/reference/index.html)

## Reading Files

Reading CSV

Import pandas as pd

dataframe = pd.read\_csv(‘file.csv’, index\_col=[‘column’])

## Selection

Checking for null in multiple columns

# Note the use of ~ as the not operator

returned\_dataframe = data[(~data['column'].isna()) & (~data['column'].isna())]

Filtering & column selection

# Returns a values within a column that match the filtering criteria

returned\_dataframe = data[data[‘column’] == ‘value’, [‘column’]]

## Cleaning

## Filtering

Counting values by category

returned\_dataframe = data['column'].sort\_values(ascending=False).value\_counts()

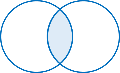
## Analysis

Grouping and applying custom functions to individual columns

# This code is frequently used to create plots by category

data.groupby('column').agg({'column':'median', 'column1':'count'}))

## Joining



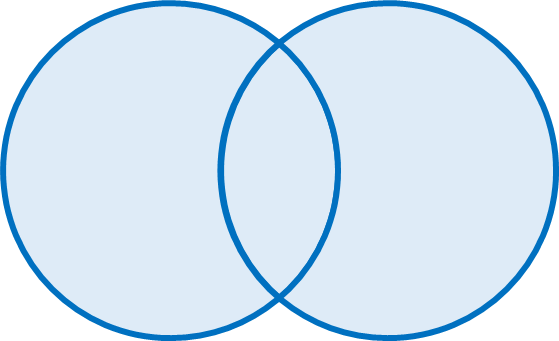
Inner Join

Returns only rows with overlap on the join columns for both datasets

# Merged data set will use suffixes to indicate the original dataset if there is overlap

# Use of the suffixes = (‘\_suffix1’, ‘\_suffix2’) can override the x/y default suffixes

returned\_dataframe = data.merge(data2, on=[’column1’, ‘column2’])

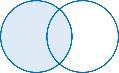


Outer Join

Outer joins return rows from both tables regardless of a match. If there is no match for a row a new row is added with the corresponding data from the table to be merged.

# Non-matching row data will be filled in with null values

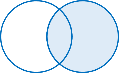
returned\_dataframe = data.merge(data2, on=’column’, how=’outer’, suffixes=(‘\_data’, ‘\_data2’))



Left Join

Returns all rows from the left data set and overlapping data from the right set join criteria. Number of rows will always be equal to the number in the left table. This type of join is typically used to add data to a dataset.

returned\_dataframe = data.merge(data2, on=[’column1’, ‘column2’], how=’left’)



Right Join

The opposite of the left join. Included is some additional arguments that apply to the merge function for specifying columns to merge on with different names in each data set.

returned\_dataframe = data.merge(data2, how=’right’, left\_on=’left\_column \

right\_on=’right\_column’)

Self Join

Used to merge a table with itself to add columns for hierarchical or sequential data. To self join use the same data set for both the left and right tables. Setting the merge method can produce filtering effects as well.

returned\_dataframe = data.merge(data, on=’column’, suffixes=(‘\_level1’, ‘\_level2’))

Merging Left/Right on Indexes

# under matching indexes only the on argument is used, but when indexes for left and right

# have different names the left\_on, right\_on, as well as left\_index=True, and right\_index=True

# are required

returned\_dataframe = data.merge(data2, left\_on=’index\_left’, left\_index=True, \

right\_on=’index\_right’, right\_index=True)

Semi-Join

Filters table A by records in table B that match

-- [end of section] --

# Matplotlib

[Matplotlib 3.3.3 Documentation](https://matplotlib.org/contents.html) (matplotlib.org)

-- [end of section] --

# Seaborn

[Seaborn 0.11.1 Documentation (pydata.org)](http://seaborn.pydata.org/api.html)

-- [end of section] --

# Appendix A