Pandas Cheat Sheet — Python for Data Science

dataquest.io/blog/pandas-cheat-sheet/

Josh Devlin

March 4, 2020

If you're interested in working with data in Python, you're almost certainly going to be using the pandas library. But even when you've learned pandas — perhaps in our interactive pandas course — it's easy to forget the specific syntax for doing something. That's why we've created a pandas cheat sheet to help you easily reference the most common pandas tasks.

Before we dive into the cheat sheet, it's worth mentioning that you shouldn't rely on just this. If you haven't learned any pandas yet, we'd strongly recommend working through our pandas course. This cheat sheet will help you quickly find and recall things you've already learned about pandas; it isn't designed to teach you pandas from scratch!

It's also a good idea to check to the official pandas documentation from time to time, even if you can find what you need in the cheat sheet. Reading documentation is a skill every data professional needs, and the documentation goes into a lot more detail than we can fit in a single sheet anyway!

If you're looking to use pandas for a specific task, we also recommend checking out the full list of **our free Python tutorials**; many of them make use of pandas in addition to other Python libraries. In our Python datetime tutorial, for example, you'll also learn how to work with dates and times in pandas.

Pandas Cheat Sheet: Guide

First, it may be a good idea to bookmark this page, which will be easy to search with Ctrl+F when you're looking for something specific. However, we've also created a PDF version of this cheat sheet that you can download from here in case you'd like to print it out.

In this cheat sheet, **we'll use the following shorthand**:

`df` | Any pandas DataFrame object
`s` | Any pandas Series object

As you scroll down, you'll see we've organized related commands using subheadings so that you can quickly search for and find the correct syntax based on the task you're trying to complete.

Also, a quick reminder — to make use of the commands listed below, you'll need to first import the relevant libraries like so:

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

Importing Data

Use these commands to import data from a variety of different sources and formats.

`pd.read_csv(filename)` | From a CSV file
`pd.read_table(filename)` | From a delimited text file (like TSV)
`pd.read_excel(filename)` | From an Excel file
`pd.read_sql(query, connection_object)` | Read from a SQL table/database
`pd.read_json(json_string)` | Read from a JSON formatted string, URL or file.
`pd.read_html(url)` | Parses an html URL, string or file and extracts tables to a list of dataframes
`pd.read_clipboard()` | Takes the contents of your clipboard and passes it to `read_table()`
`pd.DataFrame(dict)` | From a dict, keys for columns names, values for data as lists

Exporting Data

Use these commands to export a DataFrame to CSV, .xlsx, SQL, or JSON.

`df.to_csv(filename)` | Write to a CSV file
`df.to_excel(filename)` | Write to an Excel file
`df.to_sql(table_name, connection_object)` | Write to a SQL table
`df.to_json(filename)` | Write to a file in JSON format

Create Test Objects

These commands can be useful for creating test segments.

`pd.DataFrame(np.random.rand(20,5))` | 5 columns and 20 rows of random floats
`pd.Series(my_list)` | Create a series from an iterable `my_list`
`df.index = pd.date_range('1900/1/30', periods=df.shape[0])` | Add a date index

Viewing/Inspecting Data

Use these commands to take a look at specific sections of your pandas DataFrame or Series.

`df.head(n)` | First n rows of the DataFrame
`df.tail(n)` | Last n rows of the DataFrame
`df.shape` | Number of rows and columns
`df.info()` | Index, Datatype and Memory information
`df.describe()` | Summary statistics for numerical columns
`s.value_counts(dropna=False)` | View unique values and counts
`df.apply(pd.Series.value_counts)` | Unique values and counts for all columns

Selection

Use these commands to select a specific subset of your data.

`df[col]` | Returns column with label col as Series
`df[[col1, col2]]` | Returns columns as a new DataFrame
`s.iloc[0]` | Selection by position
`s.loc['index_one']` | Selection by index
`df.iloc[0,:]` | First row
`df.iloc[0,0]` | First element of first column

Data Cleaning

Use these commands to perform a variety of data cleaning tasks.

`df.columns = ['a','b','c']` | Rename columns
`pd.isnull()` | Checks for null Values, Returns Boolean Arrray
`pd.notnull()` | Opposite of `pd.isnull()`
`df.dropna()` | Drop all rows that contain null values
`df.dropna(axis=1)` | Drop all columns that contain null values
`df.dropna(axis=1,thresh=n)` | Drop all rows have have less than n non null values
`df.fillna(x)` | Replace all null values with x
`s.fillna(s.mean())` | Replace all null values with the mean (mean can be replaced with almost any function from the statistics module)
`s.astype(float)` | Convert the datatype of the series to float
`s.replace(1,'one')` | Replace all values equal to `1` with `'one'`
`s.replace([1,3],['one','three'])` | Replace all 1 with `'one'` and `3` with `'three'`
`df.rename(columns=lambda x: x + 1)` | Mass renaming of columns

`df.rename(columns={'old_name': 'new_ name'})` | Selective renaming
`df.set_index('column_one')` | Change the index
`df.rename(index=lambda x: x + 1)` | Mass renaming of index

Filter, Sort, and Groupby

Use these commands to filter, sort, and group your data.

`df[df[col] > 0.5]` | Rows where the column `col` is greater than `0.5`
`df[(df[col] > 0.5) & (df[col] < 0.7)]` | Rows where `0.7 > col > 0.5`
`df.sort_values(col1)` | Sort values by col1 in ascending order
`df.sort_values(col2,ascending=False)` | Sort values by `col2` in descending order
`df.sort_values([col1,col2],ascending=[True,False])` | Sort values by `col1` in ascending order then `col2` in descending order
`df.groupby(col)` | Returns a groupby object for values from one column
`df.groupby([col1,col2])` | Returns groupby object for values from multiple columns
`df.groupby(col1)[col2]` | Returns the mean of the values in `col2`, grouped by the values in `col1` (mean can be replaced with almost any function from the statistics module)
`df.pivot_table(index=col1,values=[col2,col3],aggfunc=mean)` | Create a pivot table that groups by `col1` and calculates the mean of `col2` and `col3`
`df.groupby(col1).agg(np.mean)` | Find the average across all columns for every unique `col1` group
`df.apply(np.mean)` | Apply the function `np.mean()` across each column
`nf.apply(np.max,axis=1)` | Apply the function `np.max()` across each row

Join/Combine

Use these commands to combine multiple dataframes into a single one.

`df1.append(df2)` | Add the rows in `df1` to the end of `df2` (columns should be identical)
`pd.concat([df1, df2],axis=1)` | Add the columns in `df1` to the end of `df2` (rows should be identical)
`df1.join(df2,on=col1,how='inner')` | SQL-style join the columns in `df1` with the columns on `df2` where the rows for `col` have identical values. `'how'` can be one of `'left'`, `'right'`, `'outer'`, `'inner'`

Statistics

Use these commands to perform various statistical tests. (These can all be applied to a series as well.)

`df.describe()` | Summary statistics for numerical columns
`df.mean()` | Returns the mean of all columns
`df.corr()` | Returns the correlation between columns in a DataFrame

`df.count()` | Returns the number of non-null values in each DataFrame column
`df.max()` | Returns the highest value in each column
`df.min()` | Returns the lowest value in each column
`df.median()` | Returns the median of each column
`df.std()` | Returns the standard deviation of each column

Download a printable version of this cheat sheet

If you'd like to download a printable version of this cheat sheet you can do so here.