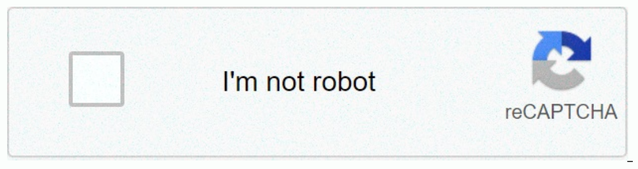

Continue

Datacamp pandas cheat sheet

The Pandas Library is one of the most preferred tools for data scientists to perform manipulation and analysis of data, besides matplotlib for data visualization and NumPy, the basic library for scientific computing in Python for which Pandas were built. Pandas data structures are fast, flexible, and expressly designed to make real-world data analytics significantly easier, but this may not be the immediate case for those just getting started with it. Precisely because there are so many built-in functions in this package that the options are overwhelming. That's where this Pandas cheat sheet can come in handy. It's a quick guide through the basics of Pandas that you'll need to start wrangling your data with Python. As such, you can use it as a handy reference if you're just starting their data science journey with Pandas or, for those of you who have not started yet, you can just use it as a guide to make it easier to learn about and use it. Pandas cheat sheet will walk you through the basics of the Pandas library, go from data structure to I/O, select, drop ratings or columns, sort and rank, get basic information about the data structure you're working on to apply the functions and align the data. In short, everything that you need to kickstart your data science with Python! (Click above to download the printable or read version online below.) An array with one-way labels is capable of keeping any type of data >>> s = pd. Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd']) Data frame A two-dimensional labeled data structure with columns with different capabilities >>> data = {'Country': ['Belgium', 'India', 'Brazil'], 'Capital': ['Brussels', 'New Delhi', 'Brasilia'], 'Population': [11190846, 1303171035, 207847528]} >>> df = pd. DataFrame(data,columns=['Country', 'Capital', 'Population']) Country Capital Population 1 Belgium Brussels 11190846 2 India New Delhi 1303171035 3 Brazil Brasilia 207847528 Please note that the first column 1.2.3 is index and country, Capital, Population are Columns. Ask for help >>> help (pd. Series.loc) I/O Read and Write to CSV >>> pd.read_csv('file.csv', header=None, nrows=5) >>> df.to_csv('myDataFrame.csv') Read multiple worksheets from the same file >>> xlsx = pd. ExcelFile('file.xls') >>> df = pd.read_excel(xlsx, 'Sheet1') Read and Write to Excel >>> pd.read_excel('file.xlsx') & gt;>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1') Read and Write to SQL Query or Database Table (read_sql()is a convenient wrapper around read_sql_table() and read_sql_query()) >>> from sqlalchemy import create_engine >>> engine = create_engine('sqlite:///:memory:') >>> pd.read_sql(SELECT * FROM my_table:, engine) >>> engine) >>> pd.read_sql_query(SELECT * FROM my_table;', engine) >>> df.to_sql('myDf', engine) Selection Getting Get one element >>> s['b'] -5 Get subset of a DataFrame >>> df[1:] Country Capital Population 1 độn co) co) New Delhi 1303171035 2 Brazil Brasilia 207847528 Pick', Index boolean and set by location Select a value by row and column >>> df.iloc([0], [0]) 'Belgium' >>> df.iat([0], [0]) 'Belgium' By Single Value Select Label by row and column >>> df.loc([0], ['Country']) 'Belgium' >>> df.at([0], ['Country']) 'Belgium' By Label /Place Select a child row of rows >>> df.ix[2] Country Brazil Capital Brasilia Population 207847528 Select a single column of the set of columns >>> df.ix[:, 'Capital'] 0 Brussels 1 New Delhi 2 Brasilia Select rows and columns >>> df.ix[1 , 'Capital'] 'New Delhi' Boolean Indexing Series s where the value is not >1 >>>> s [~(s > 1)] s where the value is <-1 or >2 >>>s[(s < -1) | (s > 2)] Use filters to adjust DataFrame >>> df[df['Population']>120000000] Set series a index to 6 >>> s['a'] = 6 Drop Values from rows (axis=0) >>>> s.drop(['a' , 'c']) Drop values from columns(axis=1) >>> df.drop('Country', axis=1) Sort and Rank by labels along axis >>> df.sort_index() Sort by values along axis >>> df.sort_values(by='Country') Assign ratings to entries >>> df.rank() Retrieve Line/Frame Data Basic Information (rows, columns) >>> df.shape Index description >>> df.index Describing Frame columns Data >>> df.columns Data Frame Information >>> df.info() Non-NA value number >>> df.count() Value summary >>> df.sum() Total value accumulation >>> df.cumsum() Minimum/maximum value >>gt;> df.min()/df.max() Minimum/maximum index value >>>> df.idxmin()/df.idxmax() Summary stats >>> df.describe() Average of values >>> df.mean() Median of values >>gt; df.median() Apply Function >>> f = lambda x: x*2 Apply function >>> df.apply(f) Apply internal data-aligned NA values >>> df.applymap(f) introduced in non-duplicate indicators: >&gt>gt; 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 align internal data yourself with the help of 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(s 3, fill_value = 4) >>> s.mul (s3, fill_value = 3) Originally published in: now, you'll know the Pandas library is one of the most preferred tools for manipulating and analyzing data , and you'll discover Pandas data structures quickly, flexibly and expressily, maybe with the help of DataCamp's Pandas Basics cheat sheet. However, there are still many functions built into this package to explore, especially when you practice data: you'll need to re-shape or rearrange your data, the current time on the Data Frame, your data visualization, and more. And this can be even more difficult than just mastering the basics. That's why today's post introduces a new, more advanced Pandas cheat sheet. It's a quick guide through functions Pandas can give you when you get into more advanced data wrangling with Python.Pandas Data Wrangling Cheat SheetDownload here. Pandas cheat sheet will guide you through some more advanced indexing techniques, DataFrame iteration, processing missing values or duplicate data, grouping and combining data, data functionality and data visualization. In short, everything you need to complete your data manipulation with Python! Do you want to learn more? Start our Pandas Foundations course for free now or try our Pandas DataFrame guide! Don't miss out on our other cheat sheets for data science including Matplotlib, SciPy, Numpy, and the basics of Python. Originally published www.datacamp.com. April 25, 2017The world of your own neural network with this Keras cheating sheet to learn deep in Python for beginners, with models! On March 21, 2017This PySpark cheat sheet with code samples covers the basics like initializing Spark in Python, loading data, sorting, and repartitioning. March 8, 2017A Pandas cheat sheet, focusing on more advanced data with this popular Python data manipulation library. February 21, 2017This Matplotlib cheat sheet introduces you to the basics that you need to draw your data beautifully with Python. On February 7, 2017This Python cheat sheet is a handy reference to making linear numbers with SciPy and interacting with NumPy! By Karlijn Willems, Data Science Journalist & DataCamp Collaborator. Data Wrangling With Python A very important component in the data science work process is data wrangling. And just as matplotlib is one of the preferred tools for data visualization in data science, pandas library is the library to use if you want to perform manipulation and data analysis in Python. This library was originally built on NumPy, the basic library for scientific computers in Python. The data structures that the Pandas library provides are fast, flexible and expressly designed specifically to make real-world data analysis significantly easier. However, this flexibility can come at a cost to beginners; When you're just starting out, the Pandas library seems very complex and it can be hard to find a single entry point into the document: with other learning materials focusing on different aspects of this library, you can definitely use a reference board to get you its hang. That's where DataCamp's Pandas tutorials and cheat sheets come in. Pandas Cheat Sheet One of the first things that you need to do to use this library is to import it. What may come unnaturally for those who are just starting out with Python and/or programming is imported conventions. However, if you've seen the first cheat sheet, you'll have some ideas; In this case, the import convention says that you should import structure like pd. You can then use this abbreviation whenever you use the Pandas module in your code. That's very handy! Pandas Data Structure You will immediately see how this import works you are starting with the Pandas data structure. But what exactly is the data structure when you talk about it? The easiest way to think of them is a structured container for your data. Or, if you've worked with NumPy, these data structures are basically arrays with indicators. Watch this video if you want to know more about how pandas data structures are connected to NumPy arrays. In any case, the basic structures that the Pandas library uses are Series and DataFrames. A Series is basically a 1D array with indicators. To make a Series simple, you use pd. Series(). Move the list to this function and, if you want, a specific number of indicators. The result can be seen in the picture on the left side. Note that Strings can contain any type of data. To create DataFrames, two-dimensional structures with columns of different data types, you can use pd. Data frame(). In this case, you pass a dictionary to this function and some additional indicators to identify the columns. I/O When you are using the Pandas library for wrangling data, one of the first things that you will not do is invented a DataFrame yourself; Instead, you'll import data from an external source and you'll place it in the data frame so it becomes easier to process. As you've read in the introduction, pandas' ultimate goal is to make real-world data analytics significantly easier. Since most of your data won't necessarily come from text files alone, the fraudulent worksheet includes three ways to import and export your data to a Data Frame or file, namely CSV, Excel, and SQL Queries/Database Table. These three ways are considered the three most important ways for your data to come to you. You'll see how pandas have specific functions for dragging and pushing data in and out of these files: pd.read_csv(), pd.read_excel(), and pd.read_sql(). But you can also use pd.read_sql_table() or pd.read_sql_query(), depending on whether you're reading from a table or query. Note that to_sql() is a convenient wrapper around the two later functions. That means that it is a function that is nothing more than calling another function. Similarly, if you want to export your data to files, you use pd.to_csv(), pd.to_excel() and pd.to_sql(). Do you want to know more about entering your data with Pandas and using it to explore your data? Consider participating in DataCamp's Pandas Foundations course. Help! One of the things that just never catches the handy, is the help() function. What you should not forget when you are using this function is always as complete as you can be: if you want to get more information about a function or concept that is included in the Pandas library, such as series, call pd. Series. Select When you finally have all the information you need about Series and DataFrames, and you've entered data into these structures (or maybe you've done your own Example Series and like in cheat sheets), you may want to check the data structures more closely. One of the ways to This is by selecting the elements. As a suggestion, the cheat sheet indicates that you can also check how you do this with NumPy arrays. If you already know about NumPy, you clearly have an advantage here! If you need to get started with this Python library for scientific calculations, consider this NumPy guide. And indeed, the procedure to choose from is very similar if you have worked with NumPy. Don't worry if you haven't been there yet, because it's easy: to get an item, you use [brackets] and put the desired index on it. In this case, you put 'b' in brackets and you get back to -5. If you look back at the panda data structure, you will find that this is correct. The same thing keeps on when you're working with the 2D DataFrame structure: you use brackets combined with a colon. What you put in front of the colon, is the number to index the row; In the second example df[1:], you request all rows, starting from index 1.This means that rows with Belgian entries will not appear in the results. To select a value based on its location in DataFrame or Series, you not only use brackets associated with indicators, but you also use iloc() or iat(). However, you can also select components by row and column labels. As you've seen before in the introduction of Pandas data structures, the columns are labeled: Country, Capital, and Population. With the help of loc() and at(), you can actually select elements based on these labels. Besides these four functions, there is also an indexer that works on labels or locations: ix is primarily based on labels, but when no labels are provided, it accepts in in insomeric numbers to address locations in DataFrame or Series from where you want to retrieve a value. Boolean indexing is also included in the cheat sheet; It's an important mechanism to select only values that meet a certain condition from your original DataFrame or Series. Conditions can easily be specified with reasonable operators & or | combined with <, >, == or combinations, such as < = or > =. Finally, there is also the option to set the value: in this case, you change the index to a value string of 6 instead of the original value 3. Are you interested in learning more about how you can manipulate your DataFrames to get the most out of them? Consider datacamp manipulating DataFrames with Pandas of course. Reduce values Besides receiving, selecting, indexing, and setting your DataFrame or Series values, you'll also need the flexibility to drop values if you no longer need them. Use drop() to drop values from columns or rows. The default axis affected by this function is the 0-axis or rows. That means if you want to remove the values from the columns, you should not forget to add the 1 = 1 axis to your code! Sort & Rank Another way to manipulate your DataFrame or Series is to sort and/or rank values are included in the data structures. Use sort_index() to sort by label along the axis or sort_values() to sort by value along the axis. As you would have expected, rank() allows you to rank your DataFrame or Series entries. Retrieve DataFrame/Series Information When you finally hold data for your data science project, it can be very convenient to know some basic information about your DataFrame or Series, especially when it can tell you more about non-NA shapes, indicators, columns, and number of non-NA values. Also any other information that you may receive information() will be more welcome when you are working with unfamiliar or new data for you. Besides the properties and functionality you see on the left side, you can also use the agrily functions to learn your data. You'll find that most of them will look familiar to you as functions commonly used in statistics, such as mean() or median(). Apply Functions In some cases, you'll also want to apply functions to your Data Frame or Series. Besides lambda functions, which you may already know from DataCamp's Python Data Science Toolbox course, you can also use apply() to apply a function to your entire data set or applymap(), in case you want to apply the function-wise elements. In other words, you'll apply functionality on each of your DataFrame or Series elements. Do you want to see some examples of how it actually works? Try DataCamp's Pandas DataFrame guide. Data alignment The last thing you need to know to get started with Pandas is how to handle your data when your indicatorss are not synchronized. In the example that the cheat sheet gives, you find that the indicators of s3 are not equal to the indicators that your Series has. This can happen very often! What Pandas do for you in such cases is to introduce NA values in non-duplicate indicators. Whenever this happens, you can pass an

fill_value to the angology function that you make use of, so that any NA value is replaced by a meaningful alternative value. Now that you've seen the separate components that make up the basics of Pandas, click on the image below to access the full cheat sheet. DataCamp is an online interactive education platform focused on building the best learning experience dedicated to Data Science. Our courses on R, Python, and Data Science are built around a certain topic and combine video tutorials with coding challenges in the browser so you can learn by doing. You can start every course for free, whenever you want, wherever you want. Biography: Karlijn Willems is a data science journalist and writer for the DataCamp community, focusing on data science education, the latest news and hot trends She has a degree in literature and linguistics and information management. Related: Related: