Cheatography

Python Pandas top 25 Cheat Sheet

by Ianh via cheatography.com/85121/cs/20026/

1. Show installed versions

pd.__version__	Show Python version
pd.show_versions()	Show dependency version

2. Create an example DataFrame

df = pd.DataFrame({'col one':[100, 200], 'col two':[300, 400]})	Pass a dictionary to the DataFrame constructor, keys are the column names and the dictionary are the values
pd.DataFrame(np.random.rand(4, 8))	Use the rand function to create a larger data frame
pd.DataFrame(np.random.rand(4, 8), columns=list('abcdefgh'))	If you want non-numeric column names

3. Rename columns

df = df.rename({'col one':'col_one', 'col two':'col_two'}, axis='columns')	Pass a dictionary; keys are the old names and the values are the new names, and you also specify the axis
df.columns = ['col_one', 'col_two']	if you're going to rename all of the columns at once, a simpler method is just to overwrite the columns attribute of the DataFrame
df.columns = df.columns.str.replace(' ', '_')	To replace spaces with underscores, use the str.replace() method
df.add_prefix('X_')	Add a prefix
df.add_suffix('_Y')	Add a suffix

4. Reverse row order

drinks.loc[::-1]	The most straightforward method is to use the loc accessor
drinks.loc[::-1].reset_index(drop=True)	Reset the index; use reset_index() to drop the old index entirely

5. Reverse column order

drinks.loc[:, ::-1]	Use loc to reverse the left-to-right order of your columns

6. Select columns by data type

drinks.select_dtypes(include='number')	To select only the numeric columns
drinks.select_dtypes(include=['number', 'object', 'category', 'datetime'])	To include multiple data types by passing a list
drinks.select_dtypes(exclude='number')	To exclude certain data types

7. Convert strings to numbers

df.astype({'col_one':'float', 'col_two':'float'}).dtypes	o convert the data types to numeric. You can use the astype() method
pd.to_numeric(df.col_three, errors='coerce').fillna(0)	use the to_numeric() function on. If you know that the NaN values actually represent zeros, you can fill them with zeros using the fillna() method the third column and tell it to convert any invalid input into NaN values
df = df.apply(pd.to_numeric, errors='coerce').fillna(0)	To apply this function to the entire DataFrame all at once by using the apply() method

8. Reduce DataFrame size

drinks.info(memory_usage='deep')	Review the memory usage of a dataframe
cols = ['beer_servings', 'continent'] small_drinks = pd.read_csv('http://bit.ly/drinksbycountry', usecols=cols)	The first step is to only read in the columns that you actually need, which we specify with the "usecols" parameter
dtypes = {'continent':'category'} smaller_drinks = pd.read_csv('http://bit.ly/drinksbycountry', usecols=cols, dtype=dtypes)	The second step is to convert any object columns containing categorical data to the category data type, which we specify with the "dtype" parameter

9. Build a DataFrame from multiple files (row)

from glob import glob	Use the glog module
stock_files = sorted(glob('data/-stocks*.csv'))	In this case, glob is looking in the "data" subdir-ectory for all CSV files that start with the word "-stocks"; ['data/stocks1.csv', 'data/stocks2.csv', 'data/stocks3.csv']
pd.concat((pd.r-ead_csv(file) for file in stock_files))	use a generator expression to read each of the files using read_csv() and pass the results to the concat() function, which will concatenate the rows into a single DataFrame
pd.concat((pd.r-ead_csv(file) for file in stock_files), ignore_index-=True)	There are now duplicate values in the index. To avoid that, we can tell the concat() function to ignore the index and instead use the default integer index

10. Build a DF from multiple files (columns)¶

drink_files = sorted(glob('data/drinks*.csv'))

pd.concat((pd.read_csv(file) for file in drink_files), axis='columns')	Tell the concat() function to concatenate along the columns axis

11. Create a DataFrame from the clipboard

df = pd.rea-d_clip-board()	Just select the data and copy it to the clipboard. Then, you can use the read_clipboard() function to read it into a DataFrame

22. Create a pivot table

titanic.pivot_tabl-e(index='Sex', columns='Pclass', values='Survived', aggfunc='mean')	If you often create DataFrames, you might find it more convenient to use the pivot_table() method instead. With a pivot table, you directly specify the index, the columns, the values, and the aggregation function.

22. Create a pivot table (cont)

titanic.pivot_table(index='Sex', columns='Pclass', values='S-urvived', aggfunc='mean', margins=True)	An added benefit of a pivot table is that you can easily add row and column totals by setting margin-s=True
titanic.pivot_table(index='Sex', columns='Pclass', values='S-urvived', aggfunc='count', margins=True)	Create a cross-tabulation just by changing the aggregation function from "mean" to "count"

23. Convert continuous data into categorical data

pd.cut(titan-ic.Age, bins=[0, 18, 25, 99], labels=['-child', 'young adult', 'adult']).he-ad(10)	Label the age ranges, such as "child", "young adult", and "adult". The best way to do this is by using the cut() function. This assigned each value to a bin with a label. Ages 0 to 18 were assigned the label "child", ages 18 to 25 were assigned the label "young adult", and ages 25 to 99 were assigned the label "adult".

24. Change display options

pd.set_option('display.float_f-ormat', '{:.2f}'.format)	To standardise the display to use 2 decimal places
pd.reset_option('display.float-_format')	Reset any option back to its default

25. Style a DataFrame

format_dict = {'Date':'{:%m/%-d/%y}', 'Close':'${:.2f}', 'Volum-e':'{:,}'}	Create a dictionary of format strings that specifies how each column should be formatted
stocks.style.format(format_dict)	Pass it to the DataFrame's style.format() method
(stocks.style.format(format_di-ct).hide_index().highlight_min-('Close', color='red').highligh-t_max('Close', color='lightgreen'))	We've now hidden the index, highlighted the minimum Close value in red, and highlighted the maximum Close value in green
(stocks.style.format(format_di-ct).hide_index().background_gr-adient(subset='Volume', cmap='Blues'))	Highlight the minimum Close value in red, and highlighted the maximum Close value in green

25. Style a DataFrame (cont)

(stocks.style.format(format_dict) .hide_-index() .bar('Volume', color='lightblue', align='zero') .set_caption('Stock Prices from October 2016'))	There's now a bar chart within the Volume column and a caption above the DataFrame.

21. Reshape a MultiIndexed Series

titanic.groupby-('Sex').Survive-d.mean()	If you wanted to calculate the survival rate by a single category such as "Sex", you would use a groupby()
titanic.groupby-(['Sex', 'Pclas-s']).Survived.m-ean()	If you wanted to calculate the survival rate across two different categories at once, you would groupby() both of those categories
titanic.groupby-(['Sex', 'Pclas-s']).Survived.m-ean().unstack()	It can be hard to read and interact with data in this format, so it's often more convenient to reshape a MultiIndexed Series into a DataFrame by using the unstack() method

20. Select a slice of rows and columns

titanic.desc-ribe()	If you wanted a numerical summary of the dataset, you would use the describe() method
titanic.describ-e().loc['min':'-max']	If you wanted to filter it to only show the "five-n-umber summary", you can use the loc accessor and pass it a slice of the "min" through the "max" row labels
titanic.describ-e().loc['min':'-max', 'Pclas-s':'Parch']	And if you're not interested in all of the columns, you can also pass it a slice of column labels

18. Aggregate by multiple functions

orders[order-s.order_id == 1].ite-m_pric-e.sum()	Each order has an order_id and consists of one or more rows. To figure out the total price of an order, you sum the item_price for that order_id. For example, here's the total price of order number 1
orders.gr-oupby('or-der_id').ite-m_pric-e.sum()	If you wanted to calculate the total price of every order, you would groupby() order_id and then take the sum of item_price for each group
orders.gr-oupby('or-der_id').ite-m_price.a-gg(['sum', 'count'])	However, you're not actually limited to aggregating by a single function such as sum(). To aggregate by multiple functions, you use the agg() method and pass it a list of functions such as sum() and count()

19. Combine the output of an aggregation with a DF

orders.groupby('order-_id').item_price.sum()	What if we wanted to create a new column listing the total price of each order? Recall that we calculated the total price using the sum() method
len(orders.groupby('o-rder_id').item_price.s-um())	In other words, the output of the sum() function
len(orders.item_price)	...is smaller than the input to the function
total_price = orders.gr-oupby('order_id').ite-m_price.transfo-rm('sum') len(total-_price)	The solution is to use the transform() method, which performs the same calcul-ation but returns output data that is the same shape as the input data

This needs more work!

16. Split a string into multiple columns

df.name.str.split(' ', expand=True)	What if we wanted to split the "name" column into three separate columns, for first, middle, and last name? We would use the str.split() method and tell it to split on a space character and expand the results into a DataFrame
df[['first', 'middle', 'last']] = df.name.str.split(' ', expand=True) df	These three columns can actually be saved to the original DataFrame in a single assignment statement
df.location.str.split(', ', expand=True)	What if we wanted to split a string, but only keep one of the resulting columns? For example, let's split the location column on "comma space"
df['city'] = df.location.str.split(', ', expand=True)[0]	If we only cared about saving the city name in column 0, we can just select that column and save it to the DataFrame

13. Filter a DataFrame by multiple categories

movies[(movies.genre == 'Action') | (movies.genre == 'Drama') | (movies.genre == 'Western')]	If we wanted to filter the DataFrame to only show movies with the genre Action or Drama or Western, we could use multiple conditions separated by the "or" operator
movies[movies.genre.isin(['Action', 'Drama', 'Western'])]	However, you can actually rewrite this code more clearly by using the isin() method and passing it a list of genres
movies[~movies.genre.isin(['Action', 'Drama', 'Western'])]	And if you want to reverse this filter, so that you are excluding (rather than including) those three genres, you can put a tilde in front of the condition

14. Filter a DataFrame by largest categories

counts.nlargest(3)	The Series method nlargest() makes it easy to select the 3 largest values in this Series
counts.nlargest(3).index	And all we actually need from this Series is the index
movies[movies.genre.isin(counts.nlargest(3).index)]	Finally, we can pass the index object to isin(), and it will be treated like a list of genres

15. Handle missing values

ufo.isna().sum()	To find out how many values are missing in each column, you can use the isna() method and then take the sum()
ufo.isna().mean()	Similarly, you can find out the percentage of values that are missing by taking the mean() of isna()
ufo.dropna(axis='columns')	If you want to drop the columns that have any missing values, you can use the dropna() method
ufo.dropna(thresh=len(ufo)*0.9, axis='columns')	Or if you want to drop columns in which more than 10% of the values are missing, you can set a threshold for dropna()