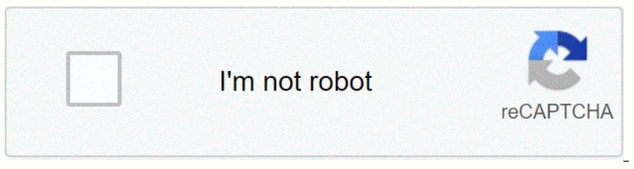

Continue

Pandas apply function to series

Applying the function to a series of pandas or dataFrame¶ B [1]: In [4]: URL = ' train = pd.read_csv(url) train.head(3) Out[4]: map() function as a series method Is mainly used to display categorical data to numeric data In [8]: # create a new train columns['Sex_num'] = train. Sex.map({'female':0, 'male':1}) In [9]: # let's compare sex and Sex_num columns # here we can see that we map men under 1 and women to 0 train.loc[0:4, ['Sex', 'Sex_num']] Out[9]: Apply() function as a series method Applies function to each element in series B [10]: # let's say we want to calculate the length of the row in each row in the Column Name # create a new column # we apply python's len function train ['Name_length'] = train. Name.apply(len) In [12]: # apply() applies the function to each train.loc element[0:4, ['Name', 'Name_length']] Out[12]: In [16]: import numpy as np # say we're looking at the Fare column and we want to round it up # we'll use numpy's ceil function to round up train numbers['Fare_ceil'] = train. Fare.apply(np.ceil) In [17]: train.loc[0:4, ['Fare', 'Fare_ceil']] Out[17]: In [19]: # let's extract each person's last name # we will use the str # method now the series is a list of rows # each cell has 2 rows in the list as you can see below the train. Name.str.split(',').head() Out[19]: 0 [Braud, Mr. Owen Harris] 1 [Cumings, Mrs. John Bradley (Florence Briggs ... 2 [Heikkinen, Miss Laina] 3 [Futrelle, Mrs. Jacques Heath (Lily May Peel)] 4 [Allen, Mr. William Henry] Name: Name, Type: Object U [22]: # we just want the first line from the #list we create a function to get def get_element(my_list, position): return my_list[position] In [23]: # use our created function get_element # we pass position = 0 train. Name.str.split(',').apply(get_element, position=0).head() Out[23]: 0 Braund 1 Cumings 2 Heikkinen 3 Futrelle 4 Allen Name: Name, Type: Object In [25]: # instead of above, we can use the lambda #input x function (list in this case) # output x[0] (first line of the list in this case) train. Name.str.split(',').apply(lambda x: x[0]).head() Out[25]: 0 Braund 1 Cumings 2 Heikkinen 3 Futrelle 4 Allen Name: Name, Type: Object U [27]: # getting the second line of the train. Name.str.split(',').apply(lambda x: x[1]).head() Out[27]: 0 Mr. Owen Harris 1 Mrs. John Bradley (Florence Briggs Teyer) 2 Miss Laine 3 Mrs. Jacques Heath (Lily May Peel) 4 Mr. William Henry's Name: Name, dtype: object apply() function as a Method DataFrame Applies function on any axis of DataFrame In [30]: URL = drinks = pd.read_csv(url) drinks.head() Out[30]: In [32]: drinks.loc[:, beer_servings:'wine_servings'].head() Out[32]: In [33]: # do you want to apply() method for travel axis = 0 (down, column) # apply Python's max() function beer_servings:'wine_servings'].apply(макс., beer_servings:'wine_servings'].apply(макс., Out[33]: beer_servings 376 spirit_servings 438 wine_servings 370 dtype: int64 У [34]: # ви хочете застосувати () метод для подорожі осі = 1 (праворуч, рядок) # застосувати Python's max() функція drinks.loc[:, 'beer_servings':'wine_servings']apply(макс., вісь=1) Вихід[34]: 0 0 1 132 2 25 3 312 4 217 5 128 6 221 7 179 8 261 9 279 10 46 1 176 12 63 13 0 14 173 15 373 16 295 17 263 18 34 19 23 20 167 21 173 22 173 23 245 24 31 25 252 26 25 27 88 28 37 29 144 ... 163 178 164 90 165 186 166 280 167 35 168 15 169 258 170 106 171 4 172 36 173 36 174 197 175 51 176 51 177 71 178 41 179 45 180 237 181 135 182 219 183 36 184 249 185 220 186 101 187 21 188 333 189 111 190 6 191 32 192 64 dtype: int64 In [35]: # finding which column is the maximum's category name drinks.loc[:, 'beer_servings':'wine_servings'].apply(np.argmax, axis=1) Out[35]: 0 beer_servings 1 spirit_servings 2 beer_servings 3 wine_servings 4 beer_servings 5 spirit_servings 6 wine_servings 7 spirit_servings 8 beer_servings 9 beer_servings 10 spirit_servings 11 spirit_servings 12 spirit_servings 13 beer_servings 14 spirit_servings 15 spirit_servings 16 beer_servings 17 beer_servings 18 beer_servings 19 beer_servings 20 beer_servings 21 spirit_servings 22 beer_servings 23 beer_servings 24 beer_servings 25 spirit_ обслуговує 26 beer_servings 27 beer_servings 28 beer_servings 29 beer_servings ... 163 spirit_servings 164 beer_servings 165 wine_servings 166 wine_servings 167 spirit_servings 168 spirit_servings 169 spirit_servings 170 beer_servings 171 wine_servings 172 beer_servings 173 beer_servings 174 beer_servings 175 beer_servings 176 beer_servings 177 spirit_servings 178 spirit_servings 179 beer_servings 180 spirit_servings 181 spirit_servings 182 beer_servings 183 beer_servings 184 beer_servings 185 wine_servings 186 spirit_servings 187 beer_servings 188 beer_servings 189 beer_servings 190 beer_servings 191 beer_servings 192 beer_servings dtype: object applymap () як метод DataFrame Застосовується функція до кожного елемента В [37]: drinks.loc[:, 'beer_servings': 'wine_servings'].applymap(float).head() Out[37]: У [41]: # перезаписати існуючі напої столу.loc[:, «beer_servings» : 'wine_servings'] = drinks.loc[:, 'beer_servings': 'wine_servings'].applymap(float) drinks.head() Out[41]: Сторінку переміщено на цю сторінку Застосувати функцію вздовж осі DataFrame. Об'єкти, передані функції, є об'єктами ряду, індекс яких є індексом DataFrame (вісь = 0) або стовпцями DataFrame (вісь = 1). За промовчанням (result_type =None) остаточний тип повернення виходить з типу повернення застосованої функції. В іншому випадку це залежить від result_type аргументу. Параметри funcfunctionФункціональність наносити на кожен стовпець або рядок. axis{0 або index, 1 або стовпці}, за замовчуванням 0Axis, по якому застосовується функція: 0 або index: застосувати функцію до кожного стовпця. 1 або 'стовпці': застосувати to each line. Line. By default FalseDetermines, if a row or column is passed as a series object or ndarray: False : transmits each row or column as a series of functions. True: The transferred function will receive ndarray objects instead. If you just apply the NumPy mitigation feature, it will achieve much better performance. result_type{'expand', 'reduce', 'broadcast', None}, default NoneThen only act when the branch = 1 (columns): expand: results similar to a list will be converted to columns. Shrink: Returns a series if possible instead of expanding results similar to a list. It's the opposite of expanding. 'broadcast': The results will be broadcast to the original DataFrame form, the source index and columns will be saved. The default behavior (None) depends on the value returned by the function applied: results similar to a list will be returned as a series of them. However, if the apply function returns a series, they are expanded to columns. argstuplePosition arguments to convey funk in addition to array / series. ** kwdsAdditary keyword arguments to pass as keyword arguments func. Returns a series or dataFrameResult application of funk along this dataFrame axis. Examples >>> df = pd. DataFrame([[4, 9]] * 3, columns=['A', 'B']) >>> df A B 0 4 9 1 4 9 2 4 9 Using a numpy universal function (in this case the same as np.sqrt(df)): >>> df.apply(np.sqrt) A B 0 2.0 3.0 1 2.0 3.0 2 2.0 3.0 Using a reducing function on either axis >>> df.apply(np.sum, axis=0) A 12 B 27 dtype: int64 >>> df.apply(np.sum, axis=1) 0 13 1 13 2 13 dtype: int64 Returning a list-like will result in a Series >>> df.apply(lambda x: [1, 2], axis=1) 0 [1, 2] 1 [1, 2] 2 [1, 2] dtype: object Passing result_type='expand' will expand list-like results to columns of a Dataframe >>> df.apply(lambda x : [1, 2], axis=1, result_type='expand') 0 1 0 1 2 1 1 2 2 1 2 Returning a Series inside the function is similar to passing result_type='expand'. The resulting column names will be the series index. >>> df.apply(lambda x: pd. Series([1, 2], index=['foo', 'bar']), axis=1) foo bar 0 1 2 1 1 2 2 1 2 Passing result_type='broadcast' will provide the same form result, regardless of whether the function is returned or the function is returned and broadcast along the axis. The resulting column names will be the originals. >>> df.apply(lambda x: [1, 2], axis=1, result_type='broadcast') A B 0 1 2 1 2 2 1 2 Series.apply(func, convert_dtype=True, args=(), **kwds)[source]¶ Call series values function. There may be ufunc (the NumPy function that applies to the entire series) or a Python function that works for only one value. FuncfunctionPython or ufunc NumPy settings for application. convert_dtypebool, the default TrueTry is to find the best type for elementwise function results. If false, leave the dtype=object. argstuplePosition arguments moved to funk after the value of the series. **kwdsAddable keyword moved on to funk. Returns series or DataFrameIf the series object returns the result of DataFrame. Examples Create a series with typical summer temperatures for each city. >>> s = pd. Series([20, 21, 12], ... index=['London', 'New York', 'Helsinki']) >>> s London 20 New York 21 Helsinki 12 dtype: int64 Square value by defining the function and passing it on as an argument for application(). >>> def square (x): ... return x** 2 >>> s.apply(square) London 400 New York 441 Helsinki 144 dtype: int64 Square value by passing anonymous function as argument apply(). >>> s.apply(lambda x: x** 2) London 400 New York 441 Helsinki 144 dtype: int64 Define a special function that requires additional positional arguments and pass these additional arguments using the keyword args. >>> def subtract_custom_value(x, custom_value): ... return x - custom_value >>> s.apply(subtract_custom_value, args=(5,)) London 15 New York 16 Helsinki 7 dtype: int64 Define a special function that accepts keyword arguments and passes these arguments for application. >>> def add_custom_values(x, **kwargs): ... within a month in kwargs: ... x += kwargs[month] ... return x >>> s.apply(add_custom_values, june=30, july=20, august=25) London 95 New York 96 Helsinki 87 dtype: int64 Use the function from the Numpy library. >>> s.apply(np.log) London 2.995732 New York 3.044522 Helsinki 2.484907 dtype: float64 float64