بسمی تعالی

موضوع پروژه: اموزش کتابخانه Pandas در پایتون

Pandas از کتابخانه‌های متن بازی است که برای کار با داده‌هایی با ساختار رابطه‌ای (rational) یا برچسب‌گذاری شده

ایجاد شده‌ است. این کتابخانه ساختار داده‌های متنوعی به همراه امکان اعمال عملیات عددی روی این داده‌ها را فراهم

می‌کند وبه خوبی می‌تواند با سری‌های زمانی کار کند. Pandas بر مبنای کتابخانه‌ی NumPy ساخته شده ‌است و بسیاری

از ساختار‌های NumPy در این کتابخانه استفاده شده و گسترش یافته‌اند. در ادامه به آموزش کتابخانه Pandas

می‌پردازیم. مزایای این کتابخانه شامل موارد زیر است:

* سرعت و کارایی بالا در کار با داده‌ها.
* امکان بارگذاری داده‌ها از فایل‌های متفاوت.
* کنترل راحت ویژگی‌هایی که مقداردهی نشده‌اند. (به عبارتی این مقادیر با NaN مقداردهی شده‌اند.)
* قابلیت تغییر اندازه: ستون‌هایی می‌توانند به داده‌ها اضافه شوند و یا از آن‌ها حذف شوند.
* ادغام (merge) و اتصال (join) مجموعه داده ها.
* تغییر شکل داده به طور منعطف.
* فراهم کردن امکان کار با سری‌های زمانی.
* امکان گروه‌بندی داده‌ها با توجه به اهداف کاربردی.

محیطی که در این مقاله برای کار انتخاب کرده‌ایم JuputerNotebook است که از توزیع آناکوندا استفاده شده ‌است.

**ساختار داده‌ها در کتابخانه Pandas**

به طور کلی کتابخانه Pandas  دارای دو دسته ساختار داده است:

* سری یا Series
* دیتافریم یا DataFrame

البته نوع سومی تحت عنوان Panel هم بود که در نسخه‌های جدید منسوخ شده ‌است.

**نوع داده‌ی Series**

سری یک آرایه‌ی یک بُعدی و برچسب‌گذاری شده ‌است (دارای ایندکس برای سطرهاست) که می‌تواند انواع مختلف داده

(عدد صحیح، اعشاری، رشته، آبجکت‌های پایتون و...) را در خود نگه دارد.

به محور برچسب‌ها اندیس یا index می‌گویند. سری‌های کتابخانه Pandas مثل یک ستون از صفحه‌ی اکسل است.

اندیس‌ها باید منحصر به فرد باشند و هر اندیس فقط به یک عنصر اشاره کند. منبع داده برای سری‌های کتابخانه

Pandas می‌تواند پایگاه داده‌ی SQL، فایل‌های CSV و اکسل باشد. نوع داده‌ی ورودی سری نیز می‌تواند به شکل لیست،

تاپل و دیکشنری و عدد اسکالر باشد.

**ایجاد یک سری:** یک سری در پانداس به شکل زیر تعریف می‌شود:

pandas.Series( data, index, dtype, copy)

که در آن:

* :data داده‌ی ورودی است که می‌تواند به شکل آرایه، لیست، دیکشنری یا عدد باشد.
* :index مقدار اندیس یا برچسب اختصاصی به هر سطر است که به طور پیش‌فرض از اعداد 0 تا n-1 استn:) تعداد داده‌هاست( و البته می‌تواند هر مقدار عددی یا رشته‌ای دلخواه را بگیرد.
* :dtype نوع داده‌ی اختصاصی به ستون داده است که اگر مقدار ندهیم، خودش از داده‌ها  نوع‌شان را استنباط می‌کند.
* :copy کپی از سری ایجاد می‌کند و مقدار پیش‌فرض False دارد.

در مثال زیر نحوه‌ی ایجاد یک سری با استفاده از آرایه را می‌بینیم.

import pandas as pd

import numpy as np

*# Creating empty series*

ser = pd.Series()

print(ser)

*# simple array*

data = np.array(['g', 'e', 'e', 'k', 's'])

ser = pd.Series(data)

print(ser)

Series([], dtype: float64) 0    g 1    e 2    e 3    k 4    s dtype: object

اگر مقدار ستون اندیس را مقداردهی نکنیم مانند مثال بالا به طور پیش‌فرض از صفر شروع به مقداردهی اندیس‌ها

می‌کند. می‌توانیم ستون اندیس‌ها را  با عبارت  index=[ ]نام‌گذاری کنیم:

pd.Series([1., 2., 3.], index=['a', 'b', 'c'])

a 1.0 b 2.0 c 3.0 dtype: float64

برای مشاهده‌ی داده‌های یک سری از دستور values استفاده می‌کنیم و با index اندیس‌ها را استخراج می‌کنیم:

my\_series=pd.Series([1., 2., 3.], index=['a', 'b', 'c'])

print(my\_series.values)

print(my\_series.index)

[1. 2. 3.] Index(['a', 'b', 'c'], dtype='object')

داده‌ی ورودی یک سری می‌تواند به شکل دیکشنری نیز باشد. کلید‌های دیکشنری به طور پیش‌فرض در نقش ستون

اندیس‌ها ظاهر می‌شوند و اگر مقدار index به عنوان ورودی به کد داده شود، داده‌های دیکشنری در مقابل اندیس‌هایی

ظاهر می‌شوند که آن اندیس با مقدار کلید داده در دیکشنری برابر باشد.

import pandas as pd

import numpy as np

data = {'a' : 0., 'b' : 1., 'c' : 2.}

s = pd.Series(data)

print (s)

سری خروجی به شکل زیر است:

a 0.0 b 1.0 c 2.0 dtype: float64

می‌توانیم ستون اندیس را خودمان مقدار دهی کنیم:

import pandas as pd

import numpy as np

data = {'a' : 0., 'b' : 1., 'c' : 2.}

s = pd.Series(data,index=['b','c','d','a'])

print (s)

b 1.0 c 2.0 d NaN a 0.0 dtype: float64

**انتخاب و دستیابی به عناصر سری با مکان داده**

می‌توان به عناصر با شماره‌ی اندیس آن‌ها و یا نام اندیس آن‌ها در داخل براکت [ ] دسترسی داشت. در مثال زیر پنج

عنصر اول را انتخاب می‌کنیم.

*# import pandas and numpy*

import pandas as pd

import numpy as np

*# creating simple array*

data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])

ser = pd.Series(data)

*#retrieve the first element*

print(ser[:5])

0 g

1 e

2 e

3 k

4 s

dtype: object

در مثال زیر هم به کمک برچسب اندیس به داده دسترسی داریم:

*# import pandas and numpy*

import pandas as pd

import numpy as np

*# creating simple array*

data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])

ser = pd.Series(data,index=[10,11,12,13,14,15,16,17,18,19,20,21,22])

*# accessing a element using index element*

print(ser[16])

o

اندیس‌دهی به داده می‌تواند به معنای انتخاب زیرمجموعه‌ای از داده باشد. در مثال زیر ستونی با نام Name از یک فایل

csv را انتخاب می‌کنیم و با تابع( head(10 به ده عنصر ابتدایی سری، دست می‌یابیم:

*# importing pandas module*

import pandas as pd

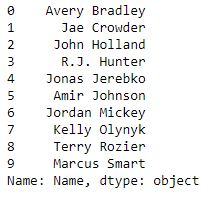
*# making data frame*

df = pd.read\_csv("nba.csv")

ser = pd.Series(df['Name'])

data = ser.head(10)

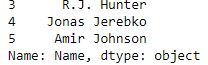
data



برای دستیابی به عناصر با اندیس 3 و 4 و 5 دستور زیر را می‌نویسیم:

*# using indexing operator*

data[3:6]

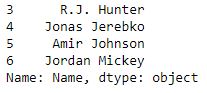


**انتخاب عناصر سری با دستور loc**

می‌توان عدد یا برچسب اندیس‌های مورد نظر را داخل براکت مقابل loc[a:b] قرار داد. توجه کنید که در اندیس‌دهی با

loc، عدد آخر یعنی b  درنظر گرفته می‌شود. به این مثال توجه کنید:

data.loc[3:6]



**انتخاب عناصر سری با دستور iloc**

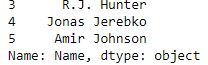
دستور iloc مشابه loc است با این تفاوت که در iloc[a:b] اندیس‌ها فقط باید عددی باشند و داده‌ی مربوط به اندیس عدد

b نیز جزو عناصر انتخابی نخواهد بود، همان‌طور که در کد زیر مشاهده می‌کنید، عنصر با اندیس 6 جزو خروجی

نیست:

# using .iloc[] function

data.iloc[3:6]



**اجرای عملیات باینری روی سری‌ها**

عملیات باینری مثل جمع و تفریق و ... را می‌توان روی سری‌ها انجام داد:

*# importing pandas module*

import pandas as pd

*# creating a series*

data = pd.Series([5, 2, 3,7], index=['a', 'b', 'c', 'd'])

print(data)

*# creating a series*

data1 = pd.Series([1, 6, 4, 9], index=['a', 'b', 'd', 'e'])

print(data1)

دو سری data و data1 را داریم:

a 5

b 2

c 3

d 7

dtype: int64

a 1

b 6

d 4

e 9

dtype: int64

حال برای جمع data و data1 از دستور add استفاده می‌کنیم:

data.add(data1, fill\_value=0)

a 6.0

b 8.0

c 3.0

d 11.0

e 9.0

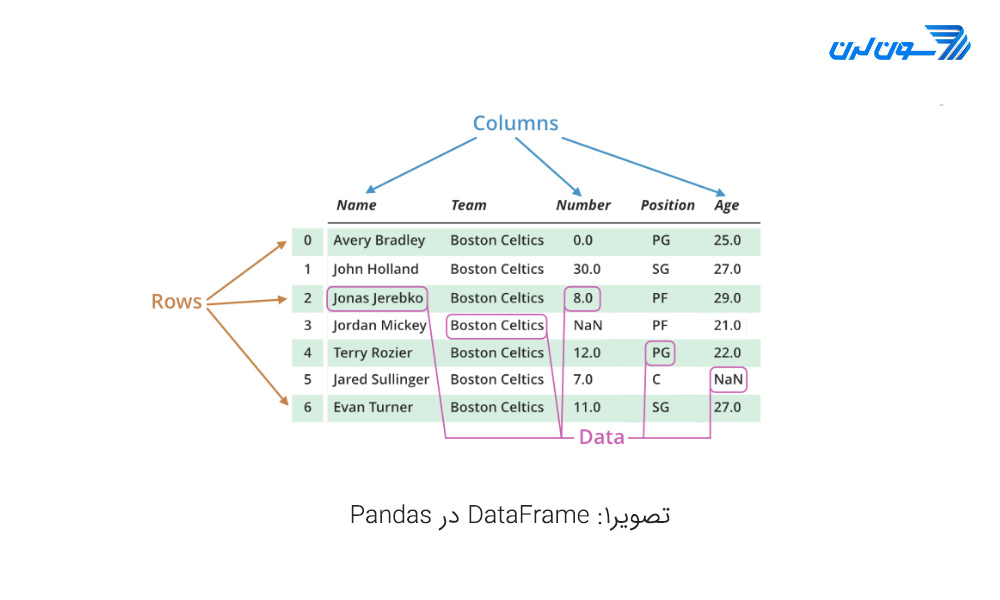
dtype: float64

**دیتافریم (Dataframe) در کتابخانه Pandas**

دیتافریم (Dataframe) یک ساختار داده‌ی دو بعُدی است و داده‌ها شامل سطر(یا همان index) و ستون (یا همان

columns) است و ساختاری رابطه‌ای است و هر تعداد داده می‌توانیم در آن ذخیره کنیم و انواع عملیات محاسباتی و

رابطه‌ای را همچون انتخاب، اتصال و گروه‌بندی را روی آن انجام دهیم.



یک دیتافریم (Dataframe) در کتابخانه Pandas (پانداس) به شکل زیر تعریف می‌شود:

pandas.DataFrame( data, index, columns, dtype, copy)

که در آن:

* :data داده‌ی ورودی است که می‌تواند لیست، آرایه، مقدار ثابت، دیکشنری و یا یک دیتافریم (Dataframe) باشد.
* :index برای برچسب‌دهی به سطرها استفاده می‌شود و اگر مقداری برایش تعریف نشود به طور خودکار از عدد 0 تا n-1 را که n تعداد داده‌هاست را، می‌گیرد.
* :columns برای برچسب‌دهی به ستون‌ها استفاده می‌شود و اگر مقداری برایش تعریف نشود به طور خودکار از عدد 0 تا n-1 را که n تعداد ویژگی یا ستون‌هاست را، می‌گیرد.
* :dtype نوع ستون‌ها را برمی‌گرداند.
* :copy اگر مقدارش True باشد از داده‌ها کپی ایجاد می‌کند، اما مقدار پیش‌فرض آن False است.

دیتافریم (Dataframe) در کتابخانه Pandas (پانداس)، به چند شکل مختلف می‌تواند ایجاد شود: 1.ایجاد دیتافریم

(Dataframe) خالی یک دیتافریم (Dataframe) اولیه می‌تواند شامل هیچ مقداری نباشد و تنها با متد DataFrame آن

را ایجاد کنیم:

*# import pandas as pd*

import pandas as pd

*# Calling DataFrame constructor*

df = pd.DataFrame()

print(df)

Empty DataFrame Columns: [] Index: []

2. ایجاد دیتافریم (Dataframe) با لیست داده‌ی دیتافریم (Dataframe) می‌تواند یک لیست یا لیست‌های تودرتو باشد:

*# import pandas as pd*

import pandas as pd

*# list of strings*

lst = ['this', 'is', 'a', 'practial', 'article', 'for', 'Pandas']

*# Calling DataFrame constructor on list*

df = pd.DataFrame(lst)

print(df)

0

0 this

1 is

2 a

3 practial

4 article

5 for

6 Pandas

**نکته**: پیش از ادامه‌ی بقیه‌ی مطالب، یادآور می‌شویم که در دیتافریم (Dataframe) هم مانند سری‌ها می‌توان ویژگی

مرتبط با سطرها و ستون‌ها را مقداردهی کرد).مقادیر عددی یا رشته‌ای به index و columns داد.( اما اگر مقداردهی

نکنیم به طور پیش‌فرض عددی و با شروع از صفر مقداردهی می‌شوند((numpy.arrange(n :از 0 تا n-1). مثال زیر

را ببینید:

*# import pandas as pd*

import pandas as pd

*# list of strings*

lst = ['this', 'is', 'a', 'practical',

'article', 'for', 'Pandas']

*# Calling DataFrame constructor on list*

df = pd.DataFrame(lst, index=['row1','row2','row3','row4','row5','row6','row7'], columns=['col1'])

print(df)

col1

row1 this

row2 is

row3 a

row4 practical

row5 article

row6 for

row7 Pandas

**3- ایجاد دیتافریم (Dataframe) با دیکشنری**

برای اینکه بتوانیم یک دیتافریم (Dataframe) با داده‌ای از  جنس دیکشنری‌ ایجاد کنیم که مقدار اختصاصی به هر کلید

از نوع آرایه یا لیست است، باید طول این لیست یا آرایه‌ها با هم برابر باشند. )اگر index را خودمان مقداردهی کنیم این

مقدار باید برابر طول آرایه یا لیست باشد.(. کلیدهای دیکشنری برای ستون‌های دیتافریم (Dataframe) در نظر گرفته

می‌شوند.

*# Python code demonstrate creating*

*# DataFrame from dict narray / lists*

*# By default addresses.*

import pandas as pd

*# intialise data of lists.*

data = {'Name':['Tom', 'nick', 'krish', 'jack'], 'Age':[20, 21, 19, 18]}

*# Create DataFrame*

df = pd.DataFrame(data)

*# Print the output.*

print(df)

خروجی این کد دیتافریمی (Dataframe) به شکل زیر است:

Name Age

0 Tom 20

1 nick 21

2 krish 19

3 jack 18

**کار با سطرها و ستون‌ها در دیتافریم (Dataframe)**

می‌توان بر روی دیتافریم (Dataframe)‌ها عملیاتی چون انتخاب، حذف، افزودن و تغییر نام سطرها و ستون‌ها را انجام داد.

انتخاب ستون‌ها: برای انتخاب ستون‌ها می‌توان از نام آن‌ها استفاده کرد. در کد زیر دیتافریمی (Dataframe) با نام df ایجاد می‌کنیم و سپس ستون‌های Name و Qualification را انتخاب می‌کنیم.

*# Import pandas package*

import pandas as pd

*# Define a dictionary containing employee data*

data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Age':[27, 24, 22, 32],

'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

*# Convert the dictionary into DataFrame*

df = pd.DataFrame(data)

*# select two columns*

print(df[['Name', 'Qualification']])

نتیجه:

Name Qualification

0 Jai Msc

1 Princi MA

2 Gaurav MCA

3 Anuj Phd

**انتخاب سطرها**: سطرهای مورد نظر را می‌توان با دستور loc یا iloc  یا ix انتخاب کرد و البته همان‌طور که در قسمت Series گفته‌ شد برای انتخاب سطر با iloc تنها از شماره‌ی برچسب اندیس می‌توان استفاده کرد. داده‌های زیر را در نظر بگیرید:

*# Import pandas package*

import pandas as pd

*# Define a dictionary containing employee data*

data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

'Age':[27, 24, 22, 32],

'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],

'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

*# Convert the dictionary into DataFrame*

df = pd.DataFrame(data,index=('Person1','Person2','Person3','Person4'))

df

دیتافریم (Dataframe) ما در این کد به شکل زیر است:

|  | **Name** | **Age** | **Address** | **Qualification** |
| --- | --- | --- | --- | --- |
| **Person1** | Jai | 27 | Delhi | Msc |
| **Person2** | Princi | 24 | Kanpur | MA |
| **Person3** | Gaurav | 22 | Allahabad | MCA |
| **Person4** | Anuj | 32 | Kannauj | Phd |

حال اگر بخواهیم سطر اول را انتخاب کنیم، با یکی از دو دستور زیر این کار را می‌توانیم انجام دهیم:

*#with loc*

df.loc[['Person1']]

*#with iloc*

df.iloc[[0]]

که خروجی هر دو کد یکسان است:

|  | **Name** | **Age** | **Address** | **Qualification** |
| --- | --- | --- | --- | --- |
| **Person1** | Jai | 27 | Delhi | Msc |

اگر بخواهیم دو سطر اول و دوم را انتخاب کنیم مانند کد زیر عمل می‌کنیم:

*#with loc*

df.loc['Person1':'Person2']

*#or*

df.loc[['Person1','Person2']]

*#with iloc*

df.iloc[0:2]

|  | **Name** | **Age** | **Address** | **Qualification** |
| --- | --- | --- | --- | --- |
| **Person1** | Jai | 27 | Delhi | Msc |
| **Person2** | Princi | 24 | Kanpur | MA |

و برای انتخاب سطر اول و اول و ستون دوم و سوم:

#with loc

df.loc['Person1':'Person2','Age':'Address']

#or

df.loc[['Person1','Person2'],['Age','Address']]

#with iloc

df.iloc[0:2,1:3]

|  | **Age** | **Address** |
| --- | --- | --- |
| **Person1** | 27 | Delhi |
| **Person2** | 24 | Kanpur |

**دسترسی به دیتافریم (Dataframe) با اندیس‌دهی بولین**

برای دسترسی به دیتافریم (Dataframe) با اندیس بولین، باید دیتافریم‌ها (Dataframe) را با اندیس بولین ایجاد کنیم.

یعنی دیتافریمی (Dataframe) با اندیس‌هایی که مقادیر آن True یا False باشد. به طور مثال:

*# importing pandas as pd*

import pandas as pd

*# dictionary of lists*

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["MBA", "BCA", "M.Tech", "MBA"],

'score':[90, 40, 80, 98]}

df = pd.DataFrame(dict, index = [True, False, True, False])

print(df)

شکل این دیتافریم (Dataframe) به صورت زیر است:

name degree score

True aparna MBA 90

False pankaj BCA 40

True sudhir M.Tech 80

False Geeku MBA 98

دسترسی به عناصر دیتافریم (Dataframe) با اندیس بولین همچون سایر دیتافریم‌ها (Dataframe)‌ به یکی از

روش‌های loc و iloc است. برای دسترسی با loc مقدار True یا False را به عنوان پارامتر ورودی ارسال می‌کنیم.

*# importing pandas as pd*

import pandas as pd

*# dictionary of lists*

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["MBA", "BCA", "M.Tech", "MBA"],

'score':[90, 40, 80, 98]}

*# creating a dataframe with boolean index*

df = pd.DataFrame(dict, index = [True, False, True, False])

*# accessing a dataframe using .loc[] function*

print(df.loc[True])

نتیجه:

name degree score

True aparna MBA 90

True sudhir M.Tech 80

همان‌طور که قبلاً گفته شد، iloc تنها مقادیر عددی را به عنوان پارامتر ورودی می‌پذیرد، پس در نتیجه مقدار

Boolean را نمی‌توان به آن داد.

**دسترسی به عناصر دیتافریم (Dataframe) با اپراتورهای منطقی روی ستون‌ها وسطرها**

می‌توان برای دسترسی به عناصر برمبنای مقادیرشان به کمک اپراتورهای منطقی همچون ==، <=، >= و != فیلترهایی

تعریف کرد. خروجی چنین فیلترهایی مجموعه‌ای بولین با مقادیر  True و False است.

*# importing pandas as pd*

import pandas as pd

*# dictionary of lists*

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["BCA", "BCA", "M.Tech", "BCA"],

'score':[90, 40, 80, 98]}

*# creating a dataframe*

df = pd.DataFrame(dict)

*# using a comparsion operator for filtering of data*

print(df['degree'] == 'BCA')

نتنیجه:

0 True

1 True

2 False

3 True

Name: degree, dtype: bool

همین فیلترها را بر روی index یا همان برچسب سطرها نیز می‌توان اعمال کرد. مثال زیر را در نظر بگیرید:

*# importing pandas as pd*

import pandas as pd

*# dictionary of lists*

dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],

'degree': ["BCA", "BCA", "M.Tech", "BCA"],

'score':[90, 40, 80, 98]}

df = pd.DataFrame(dict, index = [0, 1, 2, 3])

print(df[df.index>=2])

نتیجه:

name degree score

2 sudhir M.Tech 80

3 Geeku BCA 98

**مرتب‌سازی عناصر دیتافریم (Dataframe) در کتابخانه Pandas**

برای مرتب‌سازی صعودی یا نزولی داده‌های ستون‌های دیتافریم (Dataframe) از تابع sort\_values() استفاده می‌کنیم.

قاعده‌ی کلی به شکل زیر است:

***DataFrame.sort\_values(by, axis=0, ascending=True, inplace=False, kind=’quicksort’, na\_position=’last’)***

تمامی پارامترها به جز by دارای مقادیر پیش‌فرض‌اند.

:by نام یک یا مجموعه‌ای از ستون‌ها برای مرتب‌سازی داده‌ها بر مبنای آن ستون‌ها است.

:axis عدد 0 یا عبارت 'index' برای سطر  و 1 یا 'columns' برای ستون.

:ascending یک مقدار بولین که اگر True بود صعودی، و اگر False بود نزولی مرتب می‌کند.

:inplace یک مقدار بولین است. اگر True بود در دیتافریم (Dataframe) انتقالی به تابع، تغییراتی ایجاد می‌کند.

:kind یک مقدار رشته‌ای است که می‌تواند سه مقدار 'quicksort'، 'mergesort' و یا 'heapsort' را بگیرد و الگوریتم مرتب‌سازی بر اساس آن عمل کند.

|  | **name** | **Age** |
| --- | --- | --- |
| **0** | John | 25 |
| **1** | Smith | 30 |
| **2** | Paul | 50 |
| **3** | Adam | 26 |
| **4** | Smith | 11 |

:na\_position

می‌تواند یکی از دو مقدار رشته‌ای 'last' یا 'first' را بگیرد و بر اساس آن تعیین کند که مکان مقادیر NAN یا همان تهی، پس از مرتب‌سازی در ابتدای جدول قرار گیرد یا در انتهای آن.

دیتافریم (Dataframe) برگشت داده شده از تابع مرتب‌سازی، همان ابعاد و shape دیتافریم (Dataframe) اولیه را دارد. فرض کنید دیتافریمی (Dataframe) با نام data به شکل زیر داریم و می‌خواهیم براساس ستون name سطرها را مرتب کنیم.

*# importing pandas package*

import pandas as pd

*# sorting data frame by name*

data.sort\_values("name", axis = 0, ascending = True,

inplace = True, na\_position ='last')

*# display*

data

|  | **name** | **Age** |
| --- | --- | --- |
| **3** | Adam | 26 |
| **0** | John | 25 |
| **2** | Paul | 50 |
| **1** | Smith | 30 |
| **4** | Smith | 11 |

**حذف مقادیر null با تابع dropna**

به منظور حذف مقادیر NAN از دیتافریم (Dataframe) از تابع dropna استفاده می‌کنیم. این تابع سطرها و ستون‌های NAN را با شیوه‌های مختلف

حذف می‌کند. دیتافریم (Dataframe) df را به شکل زیر داریم:

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, 40, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

df

|  | **First Score** | **Second Score** | **Third Score** | **Fourth Score** |
| --- | --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | 52 | NaN |
| **1** | 90.0 | NaN | 40 | NaN |
| **2** | NaN | 45.0 | 80 | NaN |
| **3** | 95.0 | 56.0 | 98 | 65.0 |

در کد زیر سطرهایی که حداقل یک مقدار NAN دارند را حذف می‌کند.

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, 40, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

# using dropna() function

df.dropna()

خروجی به شکل زیر خواهد بود:

|  | **First Score** | **Second Score** | **Third Score** | **Fourth Score** |
| --- | --- | --- | --- | --- |
| **3** | 95.0 | 56.0 | 98 | 65.0 |

حال کدی می‌نویسیم که با آن سطری را که همه‌ی عناصر آن صفر باشد را حذف کنیم. فرض کنید دیتافریم (Dataframe) زیر را داریم که

سطر دوم آن همگی NAN است:

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, np.nan, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, np.nan, 80, 98],

'Fourth Score':[np.nan, np.nan, np.nan, 65]}

df = pd.DataFrame(dict)

# using dropna() function

df.dropna(how = 'all')

خروجی حاصل به شکل زیر است:

| **First Score** | **Second Score** | **Third Score** | **Fourth Score** |
| --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | 52.0 | NaN |
| **2** | NaN | 45.0 | 80.0 | NaN |
| **3** | 95.0 | 56.0 | 98.0 | 65.0 |

**حذف ستونی با حداقل یک مقدار null**

مجموعه داده‌ی زیر را در نظر بگیرید:

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, np.nan, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, np.nan, 80, 98],

'Fourth Score':[60, 67, 68, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

df

|  | **First Score** | **Second Score** | **Third Score** | **Fourth Score** |
| --- | --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | 52.0 | 60 |
| **1** | NaN | NaN | NaN | 67 |
| **2** | NaN | 45.0 | 80.0 | 68 |
| **3** | 95.0 | 56.0 | 98.0 | 65 |

دستور( dropna(axis=1 منجر به حذف ستون‌های اول و دوم وسوم خواهد شد. (وقتی axis را برابر 1 قرار می‌دهیم یعنی متد روی ستون‌ها کار می‌کند و اگر صفر باشد روی سطرها):

# importing pandas as pd

import pandas as pd

# importing numpy as np

import numpy as np

# dictionary of lists

dict = {'First Score':[100, np.nan, np.nan, 95],

'Second Score': [30, np.nan, 45, 56],

'Third Score':[52, np.nan, 80, 98],

'Fourth Score':[60, 67, 68, 65]}

# creating a dataframe from dictionary

df = pd.DataFrame(dict)

# using dropna() function

df.dropna(axis = 1)

|  | **Fourth Score** |
| --- | --- |
| **0** | 60 |
| **1** | 67 |
| **2** | 68 |
| **3** | 65 |

نتیجه گیری: پانداس یک کتابخانه‌ی متن باز است که برای کار بر روی داده‌های برچسب‌دار و رابطه‌ای طراحی شده ‌است. چنین ساختار

داده‌ای باعث می‌شود تا به کمک عملیات تعریف شده بر روی داده‌های رابطه‌ای تحلیل سودمندی روی داده‌ها انجام داد. این کتابخانه به

خاطر قابلیت سرعت و قدرت خوب پردازشی در علم داده بسیار پرکاربرد است و توابع و متدهای فراوانی را فراهم کرده‌است.

پروژه کتابخانه pandas در پایتون

مهدی باغگلی

منبع:

https://7learn.com/blog/pandas-library-tutorial

23 صفحه