$\boxed{28.09.2021}$

import numpy as np

Subject :   Date :....../....../......

⊛ z = np.random.randint(2, 15, size=5,3)
↓ 5 satır, 3 sütun

⊛ z.shape
→ (5,3)

⊛ z.size
→ 15

⊛ z = z.reshape(3,5) ⟷ ⊛ z = z.reshape(-1,5)
↓ 3 satır, 5 sütun   ↓ 3 satır, 5 sütun

⊛ z = z.ravel() # z'yi tek boyut yapar.

a = np.array([[1,2,3],[6,7,8]])

⊛ a[1,1] = 77 → 77'ye dönüşür

⊛ a[:,1] = 99
~ 1. column dakileri 99 yap.

⊛ np.empty((3,3)) ⇒ 3'e 3'lük 0 yazdırır

⊛ np.random.rand(3,3) # 0-1 arası değerler alır

⊛ np.random.randn(3,3) # mean=0 olan değerler alır
varyans=1 (Normal distrubution'ı tuttura bilmek için)

⊛ np.random.randint(5,10, size=(3,3))
⇓ 5-10 arası random değerler. 3'e 3'lük

Subject : CONCATENATE Date : ....../....../......

my_arr = np.array([0,1,2,3,4,5,6,7,8,9]).reshape(5,2)

my_arr_1 = np.array([3,4,5,6,7,8,9,10,11,12]).reshape(5,2)

* my_con = np.concatenate([my_arr, my_arr_1])

Alt alta yazdı (axis=0) — İkisini birleştirdi ama altına yazdı.

* my_con1 = np.concatenate([my_arr, my_arr_1], axis=1)

Yan yana yazdı (axis=1) — İkisini birleştirdi ama satırlara ekledi.

* my_con[1,[2,3]] # 1. satırdan 2 ve 3. elemanı al

* my_con[[3,1],2] # 3. ve 1. satırdan 2. satırdakileri getir.

→ ([9,5])

* my_con[0:2, 1:3]

→ array([[1,3],
[3,5]])

} Slicing ile

* my_con[:, [0,3]]

→ array([[0,4],
[2,6],
[4,8],
[6,10],
[8,12]])

} Fancy indexing ile

a = np. array([[1, 2], [3, 4]])
b = np. array([[5, 6], [7, 8]])

Subject : Date : ....../....../......

✱ X = np. stack((a, b)) # 2'ye 2'lik matrisi 3 boyutlu yaptık
X

✱ X.ndim
→ 3 (3 boyutlu)

✱ X.shape
→ (2, 2, 2)

✱ y = np. hstack((a, b)) # Horizontal = yatay demek. Yatayda ekleme yapar.
([[1, 2, 5, 6],
[3, 4, 7, 8]])

✱ y = np. vstack((a, b)) # Vertical = dikeyde ekleme yapar.
([[1, 2],
[3, 4],
[5, 6],
[7, 8]])

FILTERING

Çok kullanacağız. Data setini bununla temizleriz.

Subject : Date : ....../....../......

```
array_1 = np.array([[1,2,3,4],
                    [5,6,7,8],
                    [9,10,11,12]])
```

```
bigger_than_5 = array_1 > 5   # Boolean çıktı olur.
```

```
array_1[bigger_than_5]   # 5'ten büyük olanlar
→ array([6,7,8,9,10,11,12])
```

```
array_1[array_1 < 5]   # 5'ten küçük olanlar
→ array([1,2,3,4])
```

```
array_1[(array_1 < 10) & (array_1 > 3)]
                       ve
→ array([4,5,6,7,8,9])   # Çoklu filtreleme
```

```
array_1[(array_1 < 10) | (array_1 > 3)]
                      veya
→ array([1,2,3,4,5,6,7,8,9,10,11,12])
```

Subject :   Date :....../....../......

X = np.array([[1,2],[3,4]])
y= np.array([[5,6],[7,8]])

AYNISI

⊛ np.substract(x,y)   ⊛ x+y   ⊛ x-y

([[6,8],
[10,12]])

([[-4,-4],
[-4,-4]])

⊛ np.multiply(x,y)
(Çarp)
veya
x*y

⊛ np.divide(x,y) veya x/y

⊛ x/y

⊛ x**2

⊛ np.power(x,2)

⊛ np.sqrt(np.power(x,2))

⊛ np.log(x) # Sağa meyilli data sette kullanılır

⊛ np.exp(x) # Sola meyilli data sette kullanılır

Subject :   Date : ....../....../......

```
import pandas as pd

labels=['x', 'y', 'z', 'h']

my_list=[5,10,15,20]

arr=np.array([25,50,75,100])

d={'a':10, 'b':20, 'c':30, 'd':40}
```

(*) pd.Series(my_list, index=labels)
↳ Kafana göre index atama
labels'ları index olarak kullan.

(*) pd.Series(arr)

```
→0   25
 1   50
 2   75
 3   100
```

(*) pd.Series(d)

```
→ a   10
  b   20
  c   30
  d   40
```

Serilerde Concatenate

```
ser=pd.Series([5,10,15,100,25,30])
index=['USA', 'Germany', 'France', 'UK', 'Turkey', 'Greece'])
```

(*) ser.sort_values() ⇒ Değerleri sıralar.
Burda 100'ü sona atar

(*) ser.sort_index() ⇒ Indexlere göre sıralar.
Burda alfabetik sıralama yapar.

Label'lar üzerinden filtreleme

Subject : Date : ....../....../......

(*) 'Turkey' in ser
→ True (Çünkü seride var.)

(*) 'Holland' in ser
→ False

(*) ser.isin([55]) ⇒ Value üzerinden filtreleme

(*) ser.keys() ⇒ Key'ler yani ülke isimlerini listeler.

(*) ser.items
→ USA 5
Germany 15
⋮ ⋮

for döngüsüyle her bir item'ın içine girebilirim.

⇕ AYNISI

(*) for key, value in ser.items():
print(key, value)

DATA FRAMES

m=np.arange(1,10,2).reshape((3,5))

5 tane

df=pd.Dataframe(m, columns=['col1', 'col2', 'col3', 'col4', 'col5'

Değerler ne olacak?

Column'lar neler olacak

Data frame'e vermemiz gerekenler

(*) df.head(5)

(*) df.tail(2)

(*) df.sample(2)
→ Karışık 2 tane getir.

08.10.2021

Subject: Date: ....../....../......

```
import seaborn as sns

df_sns.load_dataset("planets")
df.head()
```

⊛ df.method.value_counts()

⊛ df.groupby('method')[['distance']].mean()
                                     .sum()

⊛ df.year.value_counts()
↳ yıl bilgilerine baktık

⊛ df.groupby('year')[['orbital_period']].count()
↓
Yıllara göre gruplayıp orbital_period'u saydırdık.

Subject : Date : ....../....../......

data = {'Company' : ['Google', 'Google', 'Mic", 'Mic', 'face", "Face'
'Person': ['PB', 'Chandler', 'Ross', 'Joey', 'Rachel', 'Monica']
'Sales': [100, 140, 150, 99, 102, 110]}

(*) df4=pd.Dataframe(data)

(*) my-map={"Google":"Goo", "face":fb"}

df4. Company = df4. Company. map(my_map)

İsimleri beğenmedim değiştirdik

! Map dataframe'e uygulanmaz.

Replace " uygulanabilir.
(Map sadece serilere, replace tüm dataya)

(*) df4. Sales = df4. Sales. map (lambda X: x*100)
df4

Sales değerlerini 100 ile çarptık.

REPLACE ile eski haline getirelim:

(*) df4. Company. replace (to_replace='Goo', value='Google', inplace=True)
df4

İsim tekrar

(*) df4.replace({"Goo":"Google", "fb":"face"})

Tüm dataya uygulayabiliyoruz. Çünkü replace.

Subject :   Date : ....../....../......

⊛ dup_df = pd.DataFrame([5,5,6,7,887],
['a','b','c','d','e','f','g'])

dup_df

⊛ dup_df.drop_duplicates()

▼ Aynı olan değerleri döndürmüyor.
• Data büyük olduğunda hata vermesin diye kullanıyoruz. Daha az değer getiriyor.

⊛ df.groupby('groups').sum()

groups column'ına göre grupla, numeric columnları topla yazdır.

⊛ df.groupby('groups').aggregate(['mean','max','min'])

Birden fazla değer yazacaksak liste içinde olmalı

from scipy import stats

⊛ df.groupby('groups').agg(['mean', min, max, 'count',
stats.mode])

Subject :   Date : ...../...../.....

(*) df.groupby("groups").agg({'var1' : ('min', 'max'),
'var2': 'median'})

var1'e min-max,
var2'ye median uygula

(*) df[df.var1 > 50]

var1'in 50'den büyük olanları getir.

(*) df.groupby('groups').filter(lambda x: x['var1'].sum()>80)

var1'i 80'den büyük olan satır
ları şart koştuk

(*) df.groupby('groups').transform('mean')

Mean'e dönüştür ve o gruba
o mean değerini yaz dedik

(*) df.groupby('groups').transform(np.log)

mean'deki gibi aynı grup ismine
aynı değeri yazmaz. Çünkü her
sayının kendi log'unu alır.

(*) df.groupby('groups').apply(np.mean)

→ transform'daki
gibi yazamıyoruz

Subject : Date : ..../..../.....

⊛ df.apply(np.sum)

! Apply içine str almaz, function ister.

⊛ df2 = df.iloc[:, 1:3]

↓ Yeni bir dataframe df'in bir kısmını aldık

⊛ df2.apply(np.sum, axis=1) → Apply'ı satır bazlı uyguladık.

! Satır bazlı işlem yapmak için:
Yukarıdan aşağı 0, soldan sağa 1 olmalı

\boxed{Concat}

⊛ df3 = pd.concat([df1, df2], ignore_index=True)

⊛ df3.groupby('grp').apply(sum)

↓ A'ları topla A'ya yaz
B'leri " " " .....

⊛ df3.groupby('grp').transform(sum) → farklı

Apply → A sumA B sumB C sumC	Transform → A sumA B sumB C sumC A sumA B sumB C sum

Subject :   Date : ....../....../......

! Genelde apply kullanacağız.

Ama missing value doldurmada transform da kullanılır.

MERGE

* pd.merge(col1, col2, on='id')

default => inner. Bu yüzden yazmadık.
id her iki data frame'de olunca oradan merge
etti ve sadece bir kere id sütunu yazdı.
inner => kesişimleri alır. Aynı column isimlerini
kendisi otomatik değiştiriyor. (Sonuna harf ekliyor.)

* pd.merge(col1, col2, how='outer')

Ortak olmayan satırları da getirir boş
kalan yerleri NaN yapar

* pd.merge(col1, col2, how='left')

Soldakinin kaç satırı varsa o kadar satır
döner. Sağdakini baz almaz. (soldakinin indexine göre)
Sağda boş kalan yere NaN yazar.

Subject :   Date : ....../....../......

(*) pd.merge(col1, col2, on=['id', 'subject_id'])

id ve subject_id üzerinden birleştirme yaptı ve sadece ortak olan varsa onu getirir.

[JOIN]

(*) col1.join(col2) → col1'in satırlarını baz aldı col2'de değer yoksa NaN yazdı.

! Joinin default'u => left

(*) col2.join(col1, how="inner")

How'ı özellikle belirttik çünkü default'u left.

Subject : Date :....../....../......

PRE-CLASS Check Yourself-4

(*) df.dropna(thresh=3)

↓ En az 3 NaN değere sahip olmayan sütunları tut, gerisini sil.

(*) df.dropna(axis=1, inplace=True)

↓ Drop all columns that have missing values.

(*) df.groupby('column_name').std()

↓ Bu column'a göre grupla std'ini al

(*) df.groupby('column_name').describe()

(*) pd.merge(df1, df2, on='key')

↳ Key sütunu üzerinden merge yap

(*) df1.join(df2)

↳ df2'yi df1'e ekle.

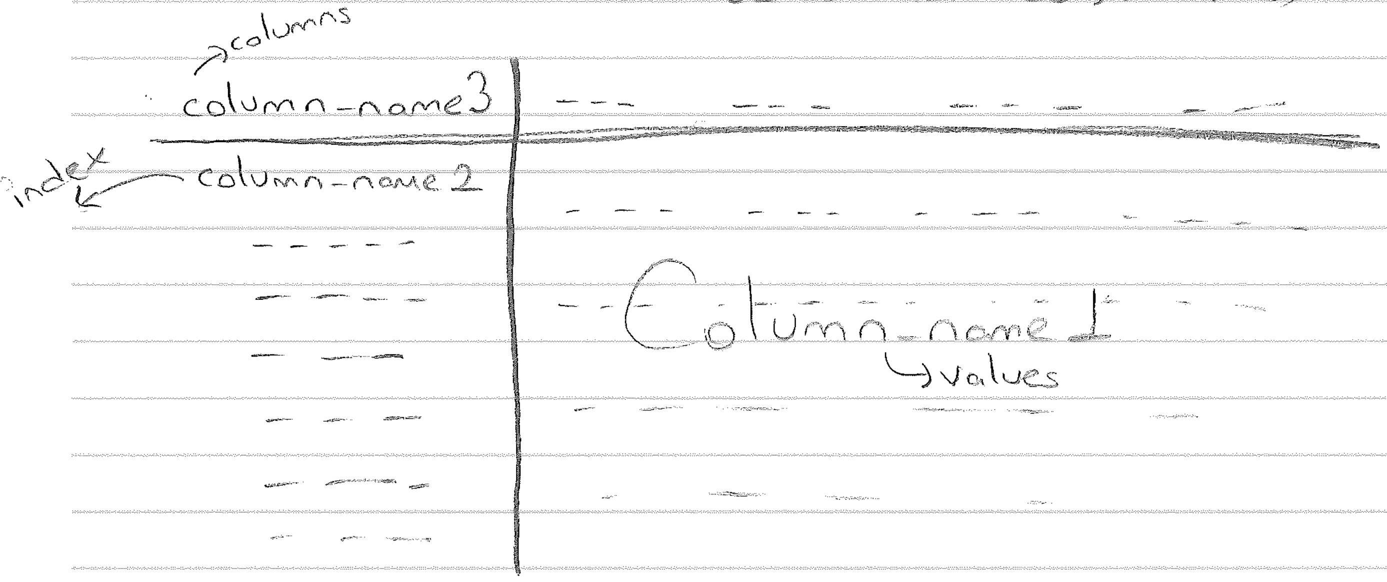
(*)
```
def times3(x):
    return x*3

df[['w', 'x']].apply(lambda x:x*3)
```

	W	X
A	—	—
B	—	—
C	—	—
D	—	—
E	—	—

Subject : Date : ....../....../......

(*) 'df.pivot_table(values='column_name1',
index='column_name2',
columns='column_name3').head()



\d → Digitleri bulur ('7' gibi)

\D → Digit olmayanları bulur

\w → Harf, digit, underscore karakterleri bulur ('A7' gibi)

\W → Harf, digit, underscore olmayanları bulur

\s → Boşluk, tab, newline 'ları bulur

\S → Boşluk, tab, newline olmayanları bulur

12.10.2021

Subject : Date : ....../....../......

REGEX

\d ➡ Any digit from 0 to 9.

\D ➡ Matches any character which is not a decimal digit (Opposite of \d)

\w ➡ Any letter, numeric digit or the underscore character.

\W ➡ Any character that is not a letter, numeric digit or the underscore characte

\s ➡ Any space, tab or newline character

\S ➡ Any character that is not a space, tab or newline (Think of this as matching white-space characters.

import re (Regex kütüphanesi)
import pandas as pd

SEARCH

text="A78L4K"

num=re.search('\d', text) → digitleri getir demek

num
→ <re.Match object; span=(1,2), match='7'>

Digit istedik ilk digit 7 diyor ve 1. indexte diyor.

* num1=re.search('\d\d', text)

* num1.group(0)
→ 78
→ Bulduğu ilk digit grubu getirdi

▼ Search fonk. gruplar halinde pattern'leri bulur.

Subject : Date : ....../....../......

text = "8PM19MIN" → text'in içinden ilk digit olanı getir.

(*) nondigi = re.search('\D', text)

print(nondigi)

→ P (Açıklamalı bir çıktı verdi.

print(nondigi.group())
→ P ⟶ Groupla açıklamasız olarak ne istiyorsam onu verir.

(*) text = "My phone number is 5556667777"

telno = re.search('\d\d\d\d\d\d\d\d\d\d', text)

print(telno.group()) Tüm noları almak istiyorum.

→ 5556667777

! Aralıklı yazmak istersem:

telno = re.search('\d\d\d \d\d\d \d\d\d\d')

print(telno.group())

→ 555 666 7777

Subject : Date : ....../....../......

(*) text = 'My phone number is 415-555-1212'

AYNISI

telno = re.search('\d\d\d-\d\d\d-\d\d\d\d', text)

print(telno.group(0))
→ 415-555-1212

group(1)
→ 415

group(2)
→ 555

group(3)
→ 1212

(*) ("\d"*3 + "-" + "\d"*3 + "-" + "\d"*4, text)

→ 415-555-1212

with open('text.txt', 'w') as file : (YAZDIRDIK → 'w' ile)
    file.write(text)

with open('text.txt', 'r') as file : (OKUTTUK → 'r' ile)
    txt = file.read
print(txt)

→ My number is 415-555-1212

İKİ FARKLI GRUPLAMA

Paranteze alınca gruplara bölüyor.

(*) output = re.search(("(\d\d\d)-(\d\d\d-\d\d\d\d)", txt)

print(output.group(1))
→ 415

print(output.group(2))
→ 555-1212

2 Parantez yapıp group(1), group(2) diye yazdırdık

Subject :   Date : ....../....../......

value = "0 1 , t 10 , a 100. 100000"

(*) output = re. findall ('\d{1}', value)
                          value'nın içindeki tüm sayıları
print (output)   bul.

→ ['1', '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '0']

findall ('\d{1}', value)
            süslü parantez içine yazılır
            1'li olan digitleri bul

(*) output = re. findall ('\d{1,6}', value)

print (output)

→ ['1', '10', '100', '100000']
       ↳ 1'li 2'li 3'lü .... 6'lı olanları getir
                                    dedik

! sub → değişiklik yapar. (replace gibi) (*)

Subject : Date : ....../....../......

```
phone = "2004-959-559 # This is Phone Number"

(*)output = re.sub('\D', '*', phone)
print(output)
```

\D ↓ neyi

'*' ↓ ne ile değiştireceksin

→ 2004*959*559

↘ Digit olmayanları al aralara çarpı koy

```
(*)output = re.sub('\d', '+', phone)
print(output)
```

→ + +

Special Characters

"[]" ➡ A set of characters "[a-m]" → Bu aradaki karakterlerden herhangi biri olur.

"\\" ➡ Özel bir sequence varsa onu gösterir.

"." ➡ Newline character dışında hepsi (Mesela digit ve letter isteğim sonrası önemli değil ne gelirse gelsin

"^" ➡ Bununla başlasın demek. "^hello"

"$" ➡ Bununla bitsin demek "word$"

"*" ➡ Hiç olmayadabilir, birden fazla da olabilir.

"+" ➡ Aradığım pattern 1 veya daha fazla olabilir.

"{}" ➡ {1} → 1'li digitleri bul gibi
{3} → 3'lü digitleri bul gibi

"|" ➡ or demek

"()" ➡ Sadece görmek istediğini getirir. (Sonlarda kullanacağız)

Subject : Date : ...../...../......

```
txt="1 person against 100 people"
```

⊛
```
output=re.findall('\d+', txt)
print(output)
→ ['1', '100']
```

```
        ----.findall('\d*', txt)
        print(output)
```

⊛
```
txt="hello world"
  output=re.findall('^he', txt)
  print(output)
→ ['hello']
```

↳ he ile başlayanları getir.

⊛
```
output=re.findall('world$', txt)
  print(output)
→ ['world']
```

↳ world ile bitenleri getir.

•str.contains
•str. extract

Subject :   Date :....../....../......

s = pd.Series(['a3', 'b4', 'c5', 'd'])

(*) s.str.contains("(\d)") → Digit içeriyor mu diye sorduk True-False ~~döndürür~~ döndürür.

s.apply(lambda x: True if re.search("\d", x) else False)

s

```
→0   a3
 1   b4
 2   c5
 3   d
```

(*) s.str.extract('(\d)') → Sadece digit olanları getir

	0
0	3
1	4
2	5
3	NaN

(*) s.str.extract('(\w)')

→ Harf, digit, underscore ları getir
(Neden rakamları getirmedi?)
↓
ANLAMADIM

	0
0	a
1	b
2	c
3	d

▼ Serilerde kullanılan regex methodları ⇒ •str.contains
●   •str.extract

Subject :   Date : ....../....../......

```
s=pd.Series(['a3aa', 'b4aa', 'c5aa'])

s.str.extract('(\w)(\w)(\w)')
     0  1  2
0    a  3  a
1    b  4  a
2    c  5  a
```

3 tane getir dedik
'\w' digitleri de alır. Sayılardan nasıl kurtulurum?

```
s.str.extract('(\w)\d(\w)(\w)')
     0  1  2
0  | a  a  a
1  | b  a  a
2  | c  a  a
```

Digit olanı atla, diğerlerini al dedik bu yüzden \d 'yi parantez içine almadık.

❗ .extract ⇒ Dataframe döndürür.

❗ Regex ile temizlik yapıp veriden sadece istediğim kısımları çekiyorum.

(*) `result = s.str.extract('(\d\d|\d.\d)')`

İki digit 12 veya digit nokta digit 1.2 olanları getir.

(*) `result = s.str.extract('(\d\d|\d.\d|\d)')`

2'li digit veya noktalı 2'li digit veya tekli digitleri al!

Subject :   Date : ....../....../......

```
result= s.str.extract('(\d\d|\d.\d|\d).+(\d\d\d)')
   0    1
  6.4  100
```

(\d\d|\d.\d|\d) → Önceki kod

(\d\d\d) → sonraki 3'lüyü de getir.

6.4 100 → gibi bir çıktı.

AYNISI

```
result= s.str.extract('(^d*.\d*)\w*/(\d*)')
result
```

^d → digit ile başlasın

* → Birden fazla olsun

.\d* → Noktalı olanlar için

arada kalan → \w* → letter olanları atla

(\d*) → sonraki digitleri al

```
a=s.str.extract('(\S+)\s+')
a
   | 0
 0 | 40
 1 | 38
 2 | 6.4
 3 |
```

(\S+) → Boşluk olmayan karakter bir veya daha fazla karakter

\s+ → Bir veya daha fazla boşluk → Bunu yazmasak da aynı sonuç

```
result= s.str.extract('(\d+).(\d+)')
     0     1
 0 | 06    2020
 1 | 11    2020
 2 | 10    2019
 3 | 05    2022
```

AYNISI

```
result= s.str.extract('(\S+\S+)')
result
        0
 0 | 06/2020
 1 | 11/2020
 2 | 10/2019
```

```
result=s.str.extract('(\d{2})/(\d{4})')
```

Aynı sonuç

```
('(\d{2}).(\d{4})')
```

4. yol