PANDAS

▼ Intro

- sayı= [10,20,30,40]
  pd.Series(sayı)
  Pandas serisi oluşturur
- .ndim : dimension verir
  .dtype : Data tipini verir
  .shape : kaç x kaç olduğunu verir
  .sum : toplamı verir
  .max() : maksimum değeri verir

▼ DataFrame

- pd.DataFrame(data) : Data yapısını data frame'e çevirir

```
s1 = pd.Series([3,2,0,1])
s2 = pd.Series([4,9,2,7])
data = dict(apples = s1, oranges = s2)

df = pd.DataFrame(data)
df
```

- df = pd.DataFrame(data, columns=["isim","not"])
  kolon başlıklarını değiştirir
  index = [1,2,3,4} : index numarasını değiştirir

```
1  data = ([["Batu",10],["Duygu",100],["Sado",22],["Meto",32]])
2  df = pd.DataFrame(data, columns=["isim","not"], index = [1,2,3,4]
3  df
```

✓ 0.1s Python

	isim	not
1	Batu	10
2	Duygu	100
3	Sado	22
4	Meto	32

- Dictionary üzerinden de data frame oluşturulabilir.

```
1  dict = {"Name":["Batu","Duygu","Sado","Meto"],
2  "Not":[10,91,22,32]}
3  df = pd.DataFrame(dict, index =[1001,1002,1003,1004])
4  df
```

✓ 0.4s

	Name	Not
1001	Batu	10
1002	Duygu	91
1003	Sado	22
1004	Meto	32

- Böyle de yapılabilir

```python
dict = {"Name":["Batu","Duygu","Sado","Meto"],
"Not":[10,91,22,32]}
dict_list=[
            {"isim":"Batu","Not":10},
            {"isim":"Duygu","Not":91},
            {"isim":"Sado","Not":22},
            {"isim":"Meto","Not":32}
          ]
df = pd.DataFrame(dict, index =[1001,1002,1003,1004])
df
```

✓ 0.3s

	Name	Not
1001	Batu	10
1002	Duygu	91
1003	Sado	22
1004	Meto	32

- .loc[”column”,”row”] : satır ve sürundaki değeri(leri) verir

```python
from numpy.random import randn
df = pd.DataFrame(randn(3,3), index = ["A","B","C"], columns=[1,2,3])
print(df)
print(df.loc["A",1])
```

✓ 0.1s

```
          1         2         3
A  0.029504 -3.375407  0.057888
B  1.312267  1.526281  0.422580
C  1.270968  1.576435  0.655204
0.029503904720519416
```

- .iloc[index num] : o indeksdeki değerleri verir

```
1  print(df.iloc[2])
```
✓ 0.4s

```
1  -1.977663
2  -2.207510
3   0.684401
Name: C, dtype: float64
```

- yeni kolon eklemek için

```
1  df[4] = pd.Series(randn(3), ["A","B","C"])
2  df[5] = df[1] + df[3]
3  df
```
✓ 0.1s

	1	2	3	4	5
A	-0.006735	-0.385505	0.465763	0.385019	0.459027
B	0.410237	0.327974	0.345538	0.298850	0.755775
C	-0.341451	-1.926654	0.283900	-0.257412	-0.057550

- .drop(değer, axis = 0 veya 1, inplace= True)
  değeri siler, axis = 1 dikey, axis = 0 yatay
  inplace = True : orijinali siler, False : orijinali tutar kopyadan siler

```
1  df.drop(5, axis = 1)
```
✓ 0.4s

	1	2	3	4
A	-0.006735	-0.385505	0.465763	0.385019
B	0.410237	0.327974	0.345538	0.298850
C	-0.341451	-1.926654	0.283900	-0.257412

- df.columns : kolon başlıkları

```
1  df.columns
✓ 0.4s
Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

- df[kolon başlığı]head(3) : seçili kolonun ilk 3 elemanı

```
1  df[1].head(3)
✓ 0.7s
0  19
1  38
2  92
Name: 1, dtype: int32
```

- 3. kolondaki 50den büyük olanların olduğu satırı 1 2 3 kolonlarla göster

```
1  df[df[3] > 50][[1,2,3]]
✓ 0.5s
```

	1	2	3
2	92	50	98
4	80	64	79
6	35	37	51
12	98	17	71

- 3. kolon 50den küçük ve 1. kolon 25’ten büyük eşit olanları getirir
  & yerine | eklenirse “or” komunu olur

```
1  df[(df[3] < 50) & (df[1] >= 25)]
```

✓ 0.1s

	1	2	3	4	5
1	38	56	19	71	61
3	94	78	45	44	10
5	29	96	14	79	29
7	49	45	49	95	89
9	41	26	18	22	98
10	70	93	34	22	66
13	26	16	10	85	57
14	50	65	16	53	93

- Gruplandırma sağlar
  df.groupby([”key1”,”key1]).groups : şeklinde yazılabilir

```
1  df.groupby("key").groups
```

```
1  df.groupby("key1").get_group("key2")
```

- .reindex([”index1”,”index2”]) : indekslemeyi değiştirir

```
1  df = df.reindex(["A","B","C","D","E","F","G","H"])
2  print(df)
```
✓ 0.4s

```
    1   2   3
A  63.0  65.0  29.0
B   NaN   NaN   NaN
C  83.0  79.0  35.0
D   NaN   NaN   NaN
E  94.0  56.0  51.0
F  46.0  95.0  12.0
G   NaN   NaN   NaN
H  34.0  40.0  52.0
```

- .drop(”A”) : seçili olanı atar
  .drop(”A”, axis = 1) : axis = 0 satır, axis = 1 sütun seçer onu siler

```
1  result = df.drop("A")
2  print(result)
```
✓ 0.1s

```
    1   2   3
B   NaN   NaN   NaN
C  83.0  79.0  35.0
D   NaN   NaN   NaN
E  94.0  56.0  51.0
F  46.0  95.0  12.0
G   NaN   NaN   NaN
H  34.0  40.0  52.0
```

- .isnull() : null NaN değerleri True gösterir
  .notnull() : tam tersi

```
1  result = df.isnull()
2  print(result)
```
✓ 0.3s

```
       1      2      3
A  False  False  False
B   True   True   True
C  False  False  False
D   True   True   True
E  False  False  False
F  False  False  False
G   True   True   True
H  False  False  False
```

- np.nan : NaN değer ekler

```
1  newColumn = [np.nan,30,np.nan,51,np.nan,84,48,95]
2  df[4] = newColumn
3  print(df)
```
✓ 0.3s

```
      1     2     3     4
A  63.0  65.0  29.0   NaN
B   NaN   NaN   NaN  30.0
C  83.0  79.0  35.0   NaN
D   NaN   NaN   NaN  51.0
E  94.0  56.0  51.0   NaN
F  46.0  95.0  12.0  84.0
G   NaN   NaN   NaN  48.0
H  34.0  40.0  52.0  95.0
```

- seçili indekslerdeki null değerler gelir

```
1  result = df[df[1].isnull()]
2  print(result)
```
✓ 0.6s

```
   1  2  3   4
B NaN NaN NaN  30.0
D NaN NaN NaN  51.0
G NaN NaN NaN  48.0
```

```
1  result = df[df[1].isnull()][1]
2  print(result)
```
✓ 0.5s

```
B   NaN
D   NaN
G   NaN
Name: 1, dtype: float64
```

- .dropna() : NaN değerleri siler

```
1  result = df.dropna()
2  # default axis = 0
3  print(result)
```
✓ 0.7s

```
    1     2     3     4
F  46.0  95.0  12.0  84.0
H  34.0  40.0  52.0  95.0
```

- .dropna(how = “any”) : NaN varsa satırı siler
  .dropna(how = “all”) : sün satırda NaN varsa satırı siler

```
1  result = df.dropna(how = "any")
2  print(result)
```
✓ 0.2s

```
   1   2   3   4
F 46.0 95.0 12.0 84.0
H 34.0 40.0 52.0 95.0
```

```
1  result = df.dropna(how = "all")
2  print(result)
```
✓ 0.3s

```
   1   2   3   4
A 63.0 65.0 29.0  NaN
B  NaN  NaN  NaN 30.0
C 83.0 79.0 35.0  NaN
D  NaN  NaN  NaN 51.0
E 94.0 56.0 51.0  NaN
F 46.0 95.0 12.0 84.0
H 34.0 40.0 52.0 95.0
```

- subset = [1,2] : sadece o kolondaki değerlere bakar

```
1  result = df.dropna(subset = [1,2], how="all")
2  print(result)
```
✓ 0.3s

```
   1   2   3   4
A 63.0 65.0 29.0  NaN
C 83.0 79.0 35.0  NaN
E 94.0 56.0 51.0  NaN
F 46.0 95.0 12.0 84.0
H 34.0 40.0 52.0 95.0
```

- .fillna() : NaN değerleri verilen değerle değiştirir

```
1  result = df.fillna( value = "no input")
2  print(result)
```
✓ 0.3s

```
          1         2         3         4
A      63.0      65.0      29.0  no input
B  no input  no input  no input      30.0
C      83.0      79.0      35.0  no input
D  no input  no input  no input      51.0
E      94.0      56.0      51.0  no input
F      46.0      95.0      12.0      84.0
G  no input  no input  no input  no input
H      34.0      40.0      52.0      95.0
```

- .sum() : 1. kolonları 2. tamamını toplar

```
1  result = df.isnull().sum().sum()
2  print(result)
```
✓ 0.5s

```
13
```

- Değerlerin ortalamasını NaN değerlerin içine yazma (ORT : 57.58)

```python
def ortalama(df):
    toplam = df.sum().sum()
    adet = df.size - df.isnull().sum().sum()
    return toplam / adet

result = df.fillna(value = ortalama(df))
print(result)
```

✓ 0.1s

	1	2	3	4
A	63.000000	65.000000	29.000000	57.578947
B	57.578947	57.578947	57.578947	30.000000
C	83.000000	79.000000	35.000000	57.578947
D	57.578947	57.578947	57.578947	51.000000
E	94.000000	56.000000	51.000000	57.578947
F	46.000000	95.000000	12.000000	84.000000
G	57.578947	57.578947	57.578947	57.578947
H	34.000000	40.000000	52.000000	95.000000

- .apply() : fonksiyonu data frame uygular

```python
def kareal(x):
    return x*x
result = df["column2"].apply(kareal)
print(result)
```

✓ 0.3s

```
0    100
1    484
2   1089
3   1600
4   2916
Name: column2, dtype: int64
```

- Conditional filtering
  .isin() verilen liste dataFrame'de mi kontrolünü yapar

```
1 options =["Sat","Sun"]
```
✓ 0.1s Python

```
1 df[df["day"].isin(options)]
```
✓ 0.1s Python

	total_bill	tip	sex	smoker	day	time	size	price_per_person
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15

- .set_index("index") : seçilen kolona göre indeksler

```
1 df.set_index("Payment ID")
```
✓ 0.4s Python

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC I
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	356032516
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	447807137
Sun4458	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181211
Sun5260	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	467613764

- .reset_index() : indeksi resetler

```
1  df.reset_index()
```

✓ 0.1s

	Payment ID	total_bill	tip	sex	smoker	day	time	size	pri
0	Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	
1	Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	
2	Sun4458	21.01	3.50	Male	No	Sun	Dinner	3	
3	Sun5260	23.68	3.31	Male	No	Sun	Dinner	2	
4	Sun2251	24.59	3.61	Female	No	Sun	Dinner	4	
...	...	...	...	...	...	...	...	...	...

- .apply() : 1den fazla sütunda işlem yaparken

```python
def quality(total_bill, tip):
    if tip/total_bill > 0.25:
        return "Generous"
    else:
        return "Other"
```

0.3s Python

```python
df["Quality"] = df[["total_bill","tip"]].apply(lambda df: quality
(df["total_bill"],df["tip"]),axis = 1)
df
```

0.2s Python

size	price_per_person	Payer Name	CC Number	Payment ID	yelp	tip_percent	Quality
2	8.49	Christy Cunningham	3560325168603410	Sun2959	$$	5.94	Other
3	3.45	Douglas Tucker	4478071379779230	Sun4608	$$	16.05	Other
3	7.00	Travis Walters	6011812112971322	Sun4458	$$	16.66	Other
2	11.84	Nathaniel Harris	4676137647685994	Sun5260	$$	13.98	Other

- np.vectorize(func name) : üsttekinin aynısı ama daha kolay yazılıyor ve daha hızlı

```python
df["Quality"] = np.vectorize(quality)(df["total_bill"], df["tip"])
df
```

0.1s Python

ne	size	price_per_person	Payer Name	CC Number	Payment ID	yelp	tip_percent	Quality
ner	2	8.49	Christy Cunningham	3560325168603410	Sun2959	$$	5.94	Other
ner	3	3.45	Douglas Tucker	4478071379779230	Sun4608	$$	16.05	Other
ner	3	7.00	Travis Walters	6011812112971322	Sun4458	$$	16.66	Other
ner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260	$$	13.98	Other

- .sort_value(”columns”, ascending = True) : değere göre artan şekilde sıralar
  ascending = False : azalan şekilde sıralar

```
1 df.sort_values("tip",ascending=True)
```
✓ 0.4s

	total_bill	tip	sex	smoker	day	time	size
67	3.07	1.00	Female	Yes	Sat	Dinner	1
236	12.60	1.00	Male	Yes	Sat	Dinner	2
92	5.75	1.00	Female	Yes	Fri	Dinner	2
111	7.25	1.00	Female	No	Sat	Dinner	1
0	16.99	1.01	Female	No	Sun	Dinner	2
...	...	...	...	...	...	...	...
141	34.30	6.70	Male	No	Thur	Lunch	6
59	48.27	6.73	Male	No	Sat	Dinner	4
23	39.42	7.58	Male	No	Sat	Dinner	4
212	48.33	9.00	Male	No	Sat	Dinner	4

- .idxmax() : maks değerin indeksi, aynısı min için de geçerli

```
1 df["total_bill"].idxmax()
```
✓ 0.1s

```
170
```

- .corr() : korelasyon verir

```
1 df.corr()
```
✓ 0.1s

	total_bill	tip	size	price_per_person	CC Number	tip_percent
total_bill	1.000000	0.675734	0.598315	0.647554	0.104576	-0.338629
tip	0.675734	1.000000	0.489299	0.347405	0.110857	0.342361
size	0.598315	0.489299	1.000000	-0.175359	-0.030239	-0.142844
price_per_person	0.647554	0.347405	-0.175359	1.000000	0.135240	-0.314254
CC Number	0.104576	0.110857	-0.030239	0.135240	1.000000	-0.032349
tip_percent	-0.338629	0.342361	-0.142844	-0.314254	-0.032349	1.000000

- .value_counts() : değerleri sayar

```
1 df["sex"].value_counts()
```
✓ 0.4s

```
Male      157
Female     87
Name: sex, dtype: int64
```

- .unique() : eşsiz değerleri gösterir
  .nunique() : eşsiz değerlerin sayısını gösterir

```
1 df["day"].unique()
```
✓ 0.3s

```
array(['Sun', 'Sat', 'Thur', 'Fri'], dtype=object)
```

- .replace(”önceki”,”sonraki”) : değerleri değiştirir

```
1  df["sex"].replace(["Female","Male"],["F","M"])
✓ 0.5s

0   F
1   M
2   M
3   M
4   F
```

- .map() : üsttekinin aynısı

```
1  mymap = {"Female" : "F", "Male" : "M"}
2  df["sex"].map(mymap)
✓ 0.2s

0   F
1   M
2   M
3   M
4   F
```

- .duplicated() : aynı satır var mı kontrol eder

```
1  df.duplicated()
✓ 0.1s

0   False
1   False
2   False
3   False
4   False
```

- .drop_duplicates() : tekrar edenleri kaldırır

```
1 df.drop_duplicates()
```
✓ 0.8s

	total_bill	tip	sex	smoker	day	time	size	price_
0	16.99	1.01	Female	No	Sun	Dinner	2	
1	10.34	1.66	Male	No	Sun	Dinner	3	
2	21.01	3.50	Male	No	Sun	Dinner	3	

- .between(a,b, inclusive=True) : a,b değerleri arasındakileri verir.
  inclusive = True : sınırlar dahil
  inclusive = False : sınırlar dahil değil

```
1 df["total_bill"].between(10,18, inclusive = True)
```
✓ 0.7s

```
0    True
1    True
2   False
3   False
4   False
```

- .nlargest(sayı, kolon) : seçilen kolonun o sayıda büyükten küçüğe sıralı halini getirir

```
1 df.nlargest(10,"tip")
```
✓ 0.5s

	total_bill	tip	sex	smoker	day	time	size	p
170	50.81	10.00	Male	Yes	Sat	Dinner	3	
212	48.33	9.00	Male	No	Sat	Dinner	4	
23	39.42	7.58	Male	No	Sat	Dinner	4	
59	48.27	6.73	Male	No	Sat	Dinner	4	
141	34.30	6.70	Male	No	Thur	Lunch	6	

- .sample(frac = 0.1) : restgele %10unu getirir

```
1 df.sample(frac = 0.1)
```
✓ 0.9s

	total_bill	tip	sex	smoker	day	time	size	pr
169	10.63	2.00	Female	Yes	Sat	Dinner	2	
161	12.66	2.50	Male	No	Sun	Dinner	2	
69	15.01	2.09	Male	Yes	Sat	Dinner	2	
222	8.58	1.92	Male	Yes	Fri	Lunch	1	

▼ Missing Data Operations

- isnull() : null var mı yok mu kontrol eder bool değer verir.
  .sum() : kaç null olduğunu gösterir

```
1  hotels.isnull().sum()
✓ 0.3s

Output exceeds the size limit. Open the full outpu
hotel                           0
is_canceled                     0
lead_time                       0
arrival_date_year               0
arrival_date_month              0
arrival_date_week_number        0
arrival_date_day_of_month       0
stays_in_weekend_nights         0
stays_in_week_nights            0
adults                          0
children                        4
babies                          0
meal                            0
country                       488
```

- .drop(”column name”, axis=1) : kolonu siler

```
1  hotels = hotels.drop("company",axis= 1)
2  hotels
✓ 0.3s
```

- .dropna(threst = 2) : 2 den fazla NaN varsa siler yoksa tutar

```
1  df.dropna(thresh = 2)
```
✓ 0.1s

	first_name	last_name	age	sex	pre_movie_score	pos
0	Tom	Hanks	63.0	m	8.0	
2	Hugh	Jackman	51.0	m	NaN	
3	Oprah	Winfrey	66.0	f	6.0	
4	Emma	Stone	31.0	f	7.0	

- .dropna(subset = [”column name”]) : o kolonda eksik değer varsa atar

```
1  df.dropna(subset = ["last_name"])
```
✓ 0.6s

	first_name	last_name	age	sex	pre_movi
0	Tom	Hanks	63.0	m	
2	Hugh	Jackman	51.0	m	
3	Oprah	Winfrey	66.0	f	
4	Emma	Stone	31.0	f	

- .fillna(”value”) : kayıp değerleri value olarak doldurur
  df["pre_movie_score"].fillna(0) : seçilen kolondaki değerleri o olarak doldurur

```
1 df.fillna("VALUE")
```

✓ 0.4s

	first_name	last_name	age	sex	pre_movie_score
0	Tom	Hanks	63.0	m	8.0
1	VALUE	VALUE	VALUE	VALUE	VALUE
2	Hugh	Jackman	51.0	m	VALUE
3	Oprah	Winfrey	66.0	f	6.0
4	Emma	Stone	31.0	f	7.0

```
1 df["pre_movie_score"].fillna(0)
```

✓ 0.1s

```
0 8.0
1 0.0
2 0.0
3 6.0
4 7.0
Name: pre_movie_score, dtype: float64
```

- seçili kolonları diğer değerlerin ortalaması ile doldurur
- df.fillna(df.mean()) : bütün null değerleri ortalama ile doldurur

```
1 df["pre_movie_score"].fillna(df["pre_movie_score"].mean())
```

✓ 0.1s

```
0 8.0
1 7.0
2 7.0
3 6.0
4 7.0
Name: pre_movie_score, dtype: float64
```

- .interpolate() : NaN değerleri üst ve alttaki değerlere göre interpolasyonla doldurur

```
1  airline_tix = {
2      "first" : 100,
3      "business" : np.nan,
4      "economy +" : 50,
5      "economy" : 30
6  }
7  series = pd.Series(airline_tix)
8  series
```

✓ 0.4s

```
first       100.0
business      NaN
economy +    50.0
economy      30.0
dtype: float64
```

```
1  series.interpolate()
```

✓ 0.4s

```
first       100.0
business     75.0
economy +    50.0
economy      30.0
dtype: float64
```

▼ Groupby

- .groupby(”column name”) : seçili kolona göre gruplandırır

```
1  df.groupby("model_year")
✓ 0.2s
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001B6B239D9A0>
```

- df.groupby("model_year").mean() : değerlerin ortalamasına göre gruplandırılmış

```
1  df.groupby("model_year").mean()
✓ 0.1s                                                                    Python
```

model_year	mpg	cylinders	displacement	weight	acceleration	origin
70	17.689655	6.758621	281.413793	3372.793103	12.948276	1.310345
71	21.250000	5.571429	209.750000	2995.428571	15.142857	1.428571
72	18.714286	5.821429	218.375000	3237.714286	15.125000	1.535714
73	17.100000	6.375000	256.875000	3419.025000	14.312500	1.375000
74	22.703704	5.259259	171.740741	2877.925926	16.203704	1.666667

- df.groupby(["model_year","cylinders"]).mean() : model yılı ve silindşir sayısınıa göre gruplandırıp ortalama değerleri verir
  .columns : sonuna eklenirse kolon başlıklarını da verir
  [”column name”] : sonuna eklenirse sadec o kolondakideğerleri verir

```
1 df.groupby(["model_year","cylinders"]).mean()
```

✓ 0.1s Python

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	4	25.285714	107.000000	2292.571429	16.000000	2.285714
	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	4	27.461538	101.846154	2056.384615	16.961538	1.923077
	6	18.000000	243.375000	3171.875000	14.750000	1.000000
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
72	3	19.000000	70.000000	2330.000000	13.500000	3.000000
	4	23.428571	111.535714	2382.642857	17.214286	1.928571
	8	13.615385	344.846154	4228.384615	13.000000	1.000000
73	3	18.000000	70.000000	2124.000000	13.500000	3.000000
	4	22.727273	109.272727	2338.090909	17.136364	2.000000
	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	8	13.200000	365.250000	4279.050000	12.250000	1.000000

- year_cyl.index.names : index başlıkları
  year_cyl.index.levels : değerler

```
1 year_cyl.index.names
```

✓ 0.1s

```
FrozenList(['model_year', 'cylinders'])
```

```
1 year_cyl.index.levels
```

✓ 0.5s

```
FrozenList([[70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82], [3, 4, 5, 6, 8]])
```

- year_cyl.loc[[(70, 6),(73, 4),(81, 8), (82, 6)]] : seçili yuple'daki değerleri verir

```
1 year_cyl.loc[[(70, 6),(73, 4),(81, 8), (82, 6)]]
```

✓ 0.2s

		mpg	displacement	weight	acceleration	origin
model_year	cylinders					
70	6	20.500000	199.000000	2710.500000	15.500000	1.0
73	4	22.727273	109.272727	2338.090909	17.136364	2.0
81	8	26.600000	350.000000	3725.000000	19.000000	1.0
82	6	28.333333	225.000000	2931.666667	16.033333	1.0

- .xs(key=4, level= "cylinders") : verilen key ve level'e (aranacak alan) göre sonuç çıkarır

```
1 year_cyl.xs(key=4, level= "cylinders")
```

✓ 0.8s

	mpg	displacement	weight	acceleration	origin
model_year					
70	25.285714	107.000000	2292.571429	16.000000	2.285714
71	27.461538	101.846154	2056.384615	16.961538	1.923077
72	23.428571	111.535714	2382.642857	17.214286	1.928571
73	22.727273	109.272727	2338.090909	17.136364	2.000000
74	27.800000	96.533333	2151.466667	16.400000	2.200000
75	25.250000	114.833333	2489.250000	15.833333	2.166667
76	26.766667	106.333333	2306.600000	16.866667	1.866667
77	29.107143	106.500000	2205.071429	16.064286	1.857143
78	29.576471	112.117647	2296.764706	16.282353	2.117647
79	31.525000	113.583333	2357.583333	15.991667	1.583333
80	34.612000	111.000000	2360.080000	17.144000	2.200000

- df[df["cylinders"].isin([6,8])].groupby(["model_year","cylinders"]).mean() : 6,8 silindirlileri model yılı ve silindir sayısına göre gruplar

```
df[df["cylinders"].isin([6,8])].groupby(["model_year","cylinders"]).mean()
```

0.1s Python

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	6	18.000000	243.375000	3171.875000	14.750000	1.000000
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
72	8	13.615385	344.846154	4228.384615	13.000000	1.000000
73	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	8	13.200000	365.250000	4279.050000	12.250000	1.000000
74	6	17.857143	230.428571	3320.000000	16.857143	1.000000
	8	14.200000	315.200000	4438.400000	14.700000	1.000000
75	6	17.583333	233.750000	3398.333333	17.708333	1.000000
	8	15.666667	330.500000	4108.833333	13.166667	1.000000
76	6	20.000000	221.400000	3349.600000	17.000000	1.300000
	8	14.666667	324.000000	4064.666667	13.222222	1.000000

- year_cyl.sort_index(level= "model_year", ascending=False) : artan şekilde model yılına göre sıralar

```
year_cyl.sort_index(level= "model_year", ascending=False)
```

0.1s Python

model_year	cylinders	mpg	displacement	weight	acceleration	origin
82	6	28.333333	225.000000	2931.666667	16.033333	1.000000
	4	32.071429	118.571429	2402.321429	16.703571	1.714286
81	8	26.600000	350.000000	3725.000000	19.000000	1.000000
	6	23.428571	184.000000	3093.571429	15.442857	1.714286
	4	32.814286	108.857143	2275.476190	16.466667	2.095238
80	6	25.900000	196.500000	3145.500000	15.050000	2.000000
	5	36.400000	121.000000	2950.000000	19.900000	2.000000
	4	34.612000	111.000000	2360.080000	17.144000	2.200000
	3	23.700000	70.000000	2420.000000	12.500000	3.000000
79	8	18.630000	321.400000	3862.900000	15.400000	1.000000

- .agg() : aggregation: describe içinden istenen değerleri getirir

```
1  df.agg(["std","mean"])["mpg"]
```

✓ 0.1s

```
std     7.815984
mean   23.514573
Name: mpg, dtype: float64
```

+ Code + Markdown

```
1  df.agg({
2      "mpg" : ["max","mean","min"],
3      "weight" : ["mean","std"]
4  })
```

✓ 0.1s

	mpg	weight
max	46.600000	NaN
mean	23.514573	2970.424623
min	9.000000	NaN
std	NaN	846.841774

▼ Combining

- DataFrame'ler

```python
data_one = {'A': ['A0', 'A1', 'A2', 'A3'],'B': ['B0', 'B1', 'B2', 'B3']}
data_two = {'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']}
one = pd.DataFrame(data_one)
two = pd.DataFrame(data_two)
print(one)
print(two)
```

✓ 0.7s Python

```
  A  B
0 A0 B0
1 A1 B1
2 A2 B2
3 A3 B3
  C  D
0 C0 D0
1 C1 D1
2 C2 D2
3 C3 D3
```

- .concat([one,two], axis= 1) : sütunlara göre birleştirir (Sütunlar yanyana)

```python
pd.concat([one,two], axis= 1)
```

✓ 0.2s

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

- .concat([one,two], axis= 0) : satırlara göre birleştirir (Satırlar yanyana)

```
1  pd.concat([one,two], axis= 0)
```

0.1s

	A	B	C	D
0	A0	B0	NaN	NaN
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN
3	A3	B3	NaN	NaN
0	NaN	NaN	C0	D0
1	NaN	NaN	C1	D1
2	NaN	NaN	C2	D2
3	NaN	NaN	C3	D3

- two.columns = one.columns : kolonları birbirine eşitleyerek alt alta yazdırabiliriz

```
1  two.columns = one.columns
2  pd.concat([one,two], axis= 0)
```

0.4s

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3
0	C0	D0
1	C1	D1
2	C2	D2
3	C3	D3

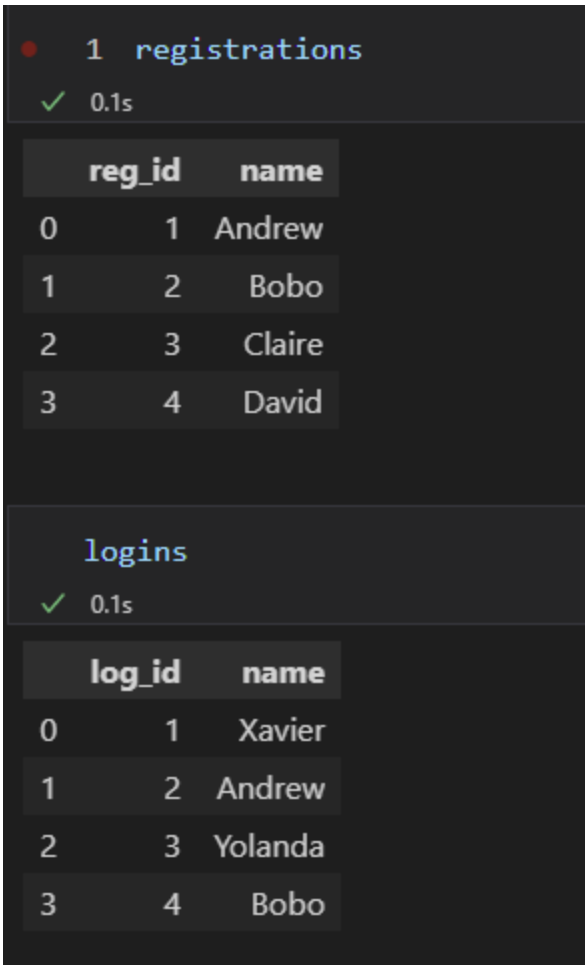
- mydf.index = range(len(mydf)) : indexi yen,den düzenlemek için

```
1  mydf = pd.concat([one,two], axis= 0)
```
✓ 0.4s

```
1  mydf.index = range(len(mydf))
2  mydf
```
✓ 0.3s

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3
4	C0	D0
5	C1	D1
6	C2	D2
7	C3	D3

- birleştirilecek dataframe’ler



- .merge(registrations,logins,how="inner", on="name")
  .merge(tablo1,tablo2, how=” “, on= “ “) : how metoduna göre tablo 1 ve 2’yi birleştirir
  how=”inner” : sadece her ikisinde de olanları birleştirir

```
1 pd.merge(registrations,logins,how="inner", on="name")
```

✓ 0.5s

	reg_id	name	log_id
0	1	Andrew	2
1	2	Bobo	4

```
1 pd.merge(logins,registrations,how="inner", on="name")
```

✓ 0.2s

	log_id	name	reg_id
0	2	Andrew	1
1	4	Bobo	2

- pd.merge(left=registrations,right=logins,how= "left",on="name") : soldan birleştirir olmayan değerlere NaN atar

```
1 pd.merge(left=registrations,right=logins,how= "left",on="name")
```

✓ 0.6s

	reg_id	name	log_id
0	1	Andrew	2.0
1	2	Bobo	4.0
2	3	Claire	NaN
3	4	David	NaN

- sağdan birleştirilmiş dataframe

```
1 pd.merge(left=registrations,right=logins,how= "right",on="name")
```
✓ 0.1s

	reg_id	name	log_id
0	NaN	Xavier	1
1	1.0	Andrew	2
2	NaN	Yolanda	3
3	2.0	Bobo	4

- other join : bütün değerleri birleştirir olmayanlara NaN atar

```
1 pd.merge(registrations,logins,how= "outer", on= "name")
```
✓ 0.9s

	reg_id	name	log_id
0	1.0	Andrew	2.0
1	2.0	Bobo	4.0
2	3.0	Claire	NaN
3	4.0	David	NaN
4	NaN	Xavier	1.0
5	NaN	Yolanda	3.0

- kolon başlıklarını değiştirme

```
1 registrations.columns = ["regname","reg_id"]
```

- suffixes=(”ek1”,”ek2”) : kolon başlıkları aynı olsa da ek1 ve ek2 ile ayrılabilir. aksi halde _x ve _y olarak adlandırılır.

```
1  registrations.columns = ["name","id"]
2  logins.columns = ["id","name"]
```
✓ 0.8s

+ Code + Markdown

```
1  pd.merge(registrations, logins, how="inner", on="name",
2  suffixes=("_reg","_log"))
```
✓ 0.2s

	name	id_reg	id_log
0	Andrew	1	2
1	Bobo	2	4

▼ Text Methods

- .split(”bölünen”) : metni seçilenden parçalarına ayırır

```
1  email = "kumay@email.com"
```
✓ 0.1s

```
1  email.split("@")
```
✓ 0.1s

['kumay', 'email.com']

- .str.upper() : büyük harf yapar

```
1  pokemon = pd.Series(["lugia","pikachu","entei","ryquaza","9"])
```
✓ 0.7s

```
1  pokemon.str.upper()
```
✓ 0.1s

```
0     LUGIA
1    PIKACHU
2     ENTEI
3    RYQUAZA
4          9
dtype: object
```

- .str.isdigit() : sayı olup olmadığına bakar

```
1  pokemon.str.isdigit()
```
✓ 0.6s

```
0    False
1    False
2    False
3    False
4     True
dtype: bool
```

- tickers.str.split(",").str[0] : 1. sütunu verir

```
1  tech_finance = ["GOOG,APPL,AMZN", "JPM,BAC,GS"]
2  tickers = pd.Series(tech_finance)
3  tickers
```

✓ 0.1s

```
0   GOOG,APPL,AMZN
1      JPM,BAC,GS
dtype: object
```

```
1  tickers.str.split(",").str[0]
```

✓ 0.1s

```
0   GOOG
1    JPM
dtype: object
```

- .str.replace(”değişen”,”yerine gelen”) : stringdeki değerleri değiştirir
  str.strip() : başta ve sonraki boşlukları siler

```
1  messy_names = pd.Series([" andrew  ","bo;bo","     claire    "])
2  messy_names
```

✓ 0.2s

```
0         andrew
1          bo;bo
2       claire
dtype: object
```

```
1  messy_names.str.replace(";","").str.strip()
```

✓ 0.2s

```
0   andrew
1    bobo
2   claire
dtype: object
```

- .str.capitalize() : ilk harfleri büyük yapar

```
1  messy_names.str.capitalize()
✓ 0.5s

0   Andrew
1    Bobo
2   Claire
dtype: object
```

- fonksiyon olarak da yazılabilir

```
1  def cleanup(name):
2      name = name.replace(";","")
3      name = name.strip()
4      name = name.capitalize()
5      return name
6  messy_names.apply(cleanup)
✓ 0.6s

0   Andrew
1    Bobo
2   Claire
dtype: object
```

▼ Input & Output

- df.to_csv("filename.csv", index= True) : df dosyasını csv olarak kaydeder. index= True : yeni index kalır, False : orijinal halindeki indeksle kaydeder

```
1  df.to_csv("filename.csv", index= True)
```

▼ Pivot Table

- Groupby ile neredeyse aynı şekilde kullanılabilir.
- .pivot(data=tablo, index='Company', columns='Product',values='Licenses')

```
1 pd.pivot(data=licenses,index='Company',
2   columns='Product',values='Licenses')
```

✓ 0.2s

Product	Analytics	GPS Positioning	Prediction	Tracking
Company				
Google	150.0	NaN	150.0	300.0
ATT	NaN	NaN	150.0	150.0
Apple	300.0	NaN	NaN	NaN
BOBO	150.0	NaN	NaN	NaN
CVS Health	NaN	NaN	NaN	450.0
Cisco	300.0	300.0	NaN	NaN
Exxon Mobile	150.0	NaN	NaN	NaN
IKEA	300.0	NaN	NaN	NaN
Microsoft	NaN	NaN	NaN	300.0
Salesforce	750.0	NaN	NaN	NaN
Tesla Inc.	300.0	NaN	150.0	NaN
Walmart	150.0	NaN	NaN	NaN

- Pivot ve Groupby ayni işlemi yapar

```
1  pd.pivot_table(df,index="Company", aggfunc="sum")
```

✓ 0.1s

Company	Account Number	Licenses	Sale Price
Google	6370194	600	3150000
ATT	1396064	300	1050000
Apple	405886	300	4550000
BOBO	2192650	150	2450000
CVS Health	902797	450	490000
Cisco	4338998	600	4900000
Exxon Mobile	470248	150	2100000
IKEA	420496	300	4550000
Microsoft	1216870	300	350000
Salesforce	2046943	750	7000000
Tesla Inc.	1273370	450	3500000
Walmart	2200450	150	2450000

```
1  df.groupby("Company").sum()
```

✓ 0.1s

Company	Account Number	Licenses	Sale Price
Google	6370194	600	3150000
ATT	1396064	300	1050000
Apple	405886	300	4550000
BOBO	2192650	150	2450000
CVS Health	902797	450	490000
Cisco	4338998	600	4900000
Exxon Mobile	470248	150	2100000
IKEA	420496	300	4550000