Groupby yöntemi, aşağıdaki gibi toplama işlevleriyle birlikte kullanılır:

- `mean`,
- `standard deviation`,
- `max` and `min`,
- `count`.
- `sum`.

Groupby

Groupby işlemi, orijinal nesne üzerinde aşağıdaki işlemlerden bazılarını içerir.

Splitting Nesneyi Bölme

Applying Bir fonksiyonun uygulanması

Combining Sonuçların birleştirilmesi

```
df=sns.load_dataset("iris")
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Groupby yöntemi, aşağıdaki gibi toplama işlevleriyle birlikte kullanılır:

- `mean`,
- `standard deviation`,
- `max` and `min`,
- `count`.
- `sum`.

```
df.groupby("species")[["petal_width"]].sum()
```

species	petal_width
setosa	12.3
versicolor	66.3
virginica	101.3

Bir fonksiyon uygulanırken aşağıdaki işlemlerden biri kullanılır. Aşağıdaki örneklerde new df kullanılmıştır.

```
d = {'batch': ['A', 'B', 'C', 'A', 'B', 'C'], 'var1': [10, 23, 33, 22, 11, 99], 'var2': [100, 253, 333, 262, 111, 969]}
df=pd.DataFrame(data=d)
df
```

	batch	var1	var2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

Aggregation(Toplama) - Bir özet istatistiği hesaplar. Bir sütuna veya birçok sütuna birden fazla fonksiyon uygulayın. Sonunda, uzunluğu groupby anahtarlarının benzersiz değerlerinin sayısı olan farklı bir veri çerçevesi çıkarılır.

```
d = {'batch': ['A', 'B', 'C', 'A', 'B', 'C'], 'var1': [10, 23, 33, 22, 11, 99], 'var2': [100, 253, 333, 262, 111, 969]}
df=pd.DataFrame(data=d)
df
```

	batch	var1	var2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

Transformation Gruba özgü bazı işlemleri gerçekleştirir. Alt veri çerçevelerinin sonuçlarını orijinal veri çerçevesine yayınlar. Her zaman orijinal veri çerçevesi ile aynı uzunlukta bir seri döndürecektir

```
df.groupby("batch").transform(lambda x: x.mean())
```

	var1	var2
0	16.0	181.0
1	17.0	182.0
2	66.0	651.0
3	16.0	181.0
4	17.0	182.0
5	66.0	651.0

Filtering Bazı koşullara sahip verileri atar. Bu alt veri çerçevelerinden elde edilen sonuçlara bir filtre uygular. Filtrelenen sonuçlar daha sonra orijinal veri çerçevesindeki eşleşen koşullara yayınlanır. Bu durumda, tam veri çerçevesinin yoğunlaştırılmış bir sürümünü elde edersiniz.

```
df.groupby("batch").filter(lambda x: x["var1"].mean()<17)
```

	batch	var1	var2
0	A	10	100
3	A	22	262

Apply Yalnızca bir işleve izin verir. Toplu sonuçlar üretir.

```
df.apply(np.sum)
```

```
batch    ABCABC
var1        198
var2       2028
dtype: object
```

Pivot Table & Stack

Özellikler arasındaki ilişki iki değişkenli analiz yoluyla elde edilebilir. Kategorik özelliklerin sayısal ve diğer kategorik özelliklerle olan ilişkilerini çıkarmak için groupby ve apply fonksiyonları kullanılır. Pivot tablolar ve Stack/Unstack fonksiyonları da bu bağlamda son derece kullanışlıdır.

Pivot Tablo

DataFrame olarak elektronik tablo tarzı bir pivot tablo oluşturun. Üç bağımsız değişken kabul eder; dizin, sütunlar ve değerler. DataFrame'in kategorik özellikleri indeks ve sütunlarda geçirilebilir. Yeni tablonun hücre değerleri, values parametresi tarafından belirtilen bir sütundan alınır.

```
df = sns.load_dataset("titanic")
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

```
df.pivot_table(values="age", index = "sex", columns = "class")
```

class	First	Second	Third
sex			
female	34.611765	28.722973	21.750000
male	41.281386	30.740707	26.507589

Stack/Unstack

Bir DataFrame'i istiflediğinizde, en içteki sütun indeksi en içteki satır indeksi olur. İstiflemeyi kaldırmak ters işlemdir.

```
d={"A":["foo", "foo", "foo", "bar", "bar", "bar"],
  "B":["one", "one", "two", "two", "one", "one"],
  "C":["x", "y", "x", "y", "x", "y"],
  "D":[1, 3, 2, 5, 4, 1]}

new_df = pd.DataFrame(d)
new_df
```

	A	B	C	D
0	foo	one	x	1
1	foo	one	y	3
2	foo	two	x	2
3	bar	two	y	5
4	bar	one	x	4
5	bar	one	y	1

```
d={"var1":[1, 2, 3, 4, 5],
  "var2":[111, 222, 333, 444, 555]}


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

	var1	var2
0	1	111
1	2	222
2	3	333
3	4	444
4	5	555

apply() **DataFrame'in bir ekseni boyunca bir fonksiyon uygulayın. Verilen eksendeki seriler fonksiyona aktarılır.**

Satır veya Sütun Bilge Fonksiyon Uygulaması: apply()

```
df.apply(np.mean)
```

```
var1      3.0
var2    333.0
dtype: float64
```

Element wise Function Application: **applymap()**

Element Fonksiyon Uygulaması: applymap()

```
df.applymap(lambda x:x*5)
```

	var1	var2
0	5	555
1	10	1110
2	15	1665
3	20	2220
4	25	2775

Function application on Series : map()

(Seriler üzerinde fonksiyon uygulaması : map())

```
df.var1.map({3:"A"})
```

```
0    NaN
1    NaN
2      A
3    NaN
4    NaN
Name: var1, dtype: object
```

Dealing with unique values in a column using;

```
d={"sirket":["AMZ", "GOOG", "META","GOOG", "META", "AMZ"],
  "personel":["Kemal", "Elif", "Gökhan", "Enes", "Hacer", "Betül"],
  "satıs":[200, 120, 340, 124, 243, 350]}


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

	sirket	personel	satıs
0	AMZ	Kemal	200
1	GOOG	Elif	120
2	META	Gökhan	340
3	GOOG	Enes	124
4	META	Hacer	243
5	AMZ	Betül	350

- `unique()` : **Compute array of unique values in a Series, returned in the order observed**

```
df.sirket.unique()
```

```
array(['AMZ', 'GOOG', 'META'], dtype=object)
```

- `nunique()` : **gives the number of unique values**

```
df.sirket.nunique()
```

```
3
```

- `value_counts()` **:Return a Series containing unique values as its index and frequencies as its values, ordered count in descending order**

```
df.sirket.value_counts()
```

```
GOOG    2
META    2
AMZ     2
Name: sirket, dtype: int64
```

- `sort_values()` **method Sorting a DataFrame by a column. It accepts a 'by' argument with column name.**

```
df.sirket.sort_values()
```

```
0     AMZ
5     AMZ
1    GOOG
3    GOOG
2    META
4    META
Name: sirket, dtype: object
```

Missing Data

Column 0	age	years_seniority	income	parking_space	attending_party	entree	pets	emergency_contact
Tony	48	27		1	5	shrimp		Pepper
Donald	67	25	86	10	2	beef		Jane
Henry	69	21	95	6	1	chicken	62	Janet
Janet	62	21	110	3	1	beef		Henry
Nick		17		4				
Bruce	37	14	63		1	veggie		NA
Steve	83		77	7	1	chicken		n/a
Clint	27	9	118	9		shrimp	3	None
Wanda	19	7	52	2	2	shrimp		empty
Natasha	26	4	162	5	3			_
Carol		3	127	11	1	veggie	1	""""
Mandy	44	2	68	8	1	chicken		null

Gerçek dünya koşullarında, eksik veriler her zaman bir sorundur. Makine öğrenimi ve veri madenciliği, eksik değerlerin neden olduğu düşük veri kalitesi nedeniyle model tahmin doğruluğu açısından önemli zorluklarla karşı karşıyadır. Kayıp değer tedavisi, modellerinin doğruluğunu ve geçerliliğini artırmak için bu alanlarda birincil odak noktasıdır.

Eksik veriler, manuel veri giriş teknikleri, ekipman hataları ve yanlış ölçümler gibi çeşitli nedenlerden dolayı ortaya çıkar. Bir veri kümesinde NaN, Sayı Değil anlamına gelir. Pandas, eksik verileri tespit etmek için isnull() ve notnull() fonksiyonlarını sağlar.

Bir DataFrame'de eksik değerler nasıl işlenir,

```
df = sns.load_dataset("penguins")
df
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

344 rows × 7 columns

dropna yöntemi eksik değerleri eksen bağımsız değişkeniyle birlikte düşürür,

```
df.isnull().sum()
```

```
species              0
island               0
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
sex                 11
dtype: int64
```

```
df.body_mass_g.dropna().isnull().sum()
```

```
0
```

fillna yöntemi NA değerlerini null olmayan verilerle "doldurabilir"

```
df.body_mass_g
```

```
0      3750.0
1      3800.0
2      3250.0
3         NaN
4      3450.0
       ...
339       NaN
340    4850.0
341    5750.0
342    5200.0
343    5400.0
Name: body_mass_g, Length: 344, dtype: float64
```

```
df.body_mass_g.fillna(method = "ffill")
```

```
0      3750.0
1      3800.0
2      3250.0
3      3250.0
4      3450.0
       ...
339    4925.0
340    4850.0
341    5750.0
342    5200.0
343    5400.0
Name: body_mass_g, Length: 344, dtype: float64
```

Replace yontemi ---fillna'ya benzer replace yöntemi NA değerlerini null olmayan değerlerle "doldurur".

```
df.body_mass_g.head(10)
```

```
0    3750.0
1    3800.0
2    3250.0
3       NaN
4    3450.0
5    3650.0
6    3625.0
7    4675.0
8    3475.0
9    4250.0
Name: body_mass_g, dtype: float64
```

```
df.body_mass_g.head(10).replace([np.nan], method = "bfill")
```

```
0    3750.0
1    3800.0
2    3250.0
3    3450.0
4    3450.0
5    3650.0
6    3625.0
7    4675.0
8    3475.0
9    4250.0
Name: body_mass_g, dtype: float64
```

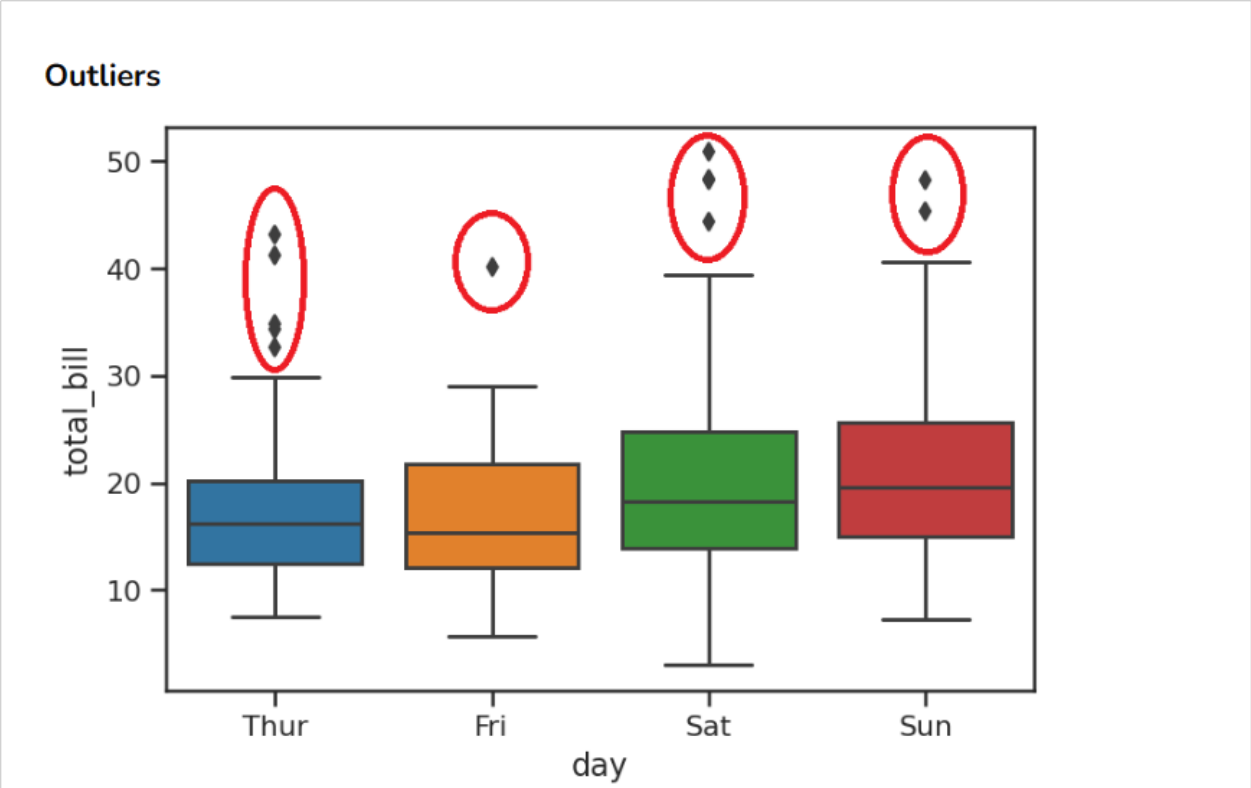
INTERPOLATE interpolate yöntemi eksik değerleri doldurmak için çeşitli interpolasyon teknikleri kullanır

```
df.body_mass_g.head(10)
```

```
0    3750.0
1    3800.0
2    3250.0
3       NaN
4    3450.0
5    3650.0
6    3625.0
7    4675.0
8    3475.0
9    4250.0
Name: body_mass_g, dtype: float64
```

```
df.body_mass_g.head(10).interpolate()
```

```
0    3750.0
1    3800.0
2    3250.0
3    3350.0
4    3450.0
5    3650.0
6    3625.0
7    4675.0
8    3475.0
9    4250.0
Name: body_mass_g, dtype: float64
```



EDA aşamasında, gelecekte Makine Öğrenimi modelleme sürecinin etkilenmemesi için verilere bazı önlemler uygulanmalıdır. Eksik verilerin ele alınmasının yanı sıra bir sonraki işlem verilerimizdeki aykırı değerlerin ele alınması olmalıdır.

Aykırı değer, Wikipedia'ya göre diğer gözlemlerden uzak olan bir gözlem noktasıdır.Veri kümemizdeki bazı değerlerin aykırı değer haline gelmesine neden olan veri toplama hataları veya veri kümesindeki varyans gibi birçok farklı neden olabilir. Yukarıdaki resim, bir veri çerçevesi tablosu sütunundaki aykırı değerleri göstermektedir. Kutu grafiğinin her iki tarafındaki siyah noktalar aykırı değerleri göstermektedir.

Aykırı değerleri bulabilmek için IQR yöntemi kullanılır. Wikipedia'dan;

Orta yayılım veya orta %50 veya teknik olarak H-yayılımı olarak da adlandırılan çeyrekler arası aralık (IQR), 75. ve 25. yüzdelik dilimler veya üst ve alt çeyrek dilimler arasındaki farka eşit olan bir istatistiksel dağılım ölçüsüdür, IQR = Q3 - Q1.

```
df = sns.load_dataset("diamonds")
df = df.select_dtypes(include = ["float64", "int64"])
df = df.dropna()
df
```

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63
4	0.31	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...
53935	0.72	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 7 columns

```
df_table = df.table
df_table
```

```
0         55.0
1         61.0
2         65.0
3         58.0
4         58.0
          ...
53935     57.0
53936     55.0
53937     60.0
53938     58.0
53939     55.0
Name: table, Length: 53940, dtype: float64
```

Bu aykırı değerlerden kurtulmanın temelde iki yolu vardır.

IQR yöntemi ile aykırı değerlerin kaldırılması

```
Q1 = df_table.quantile(0.25)
Q3 = df_table.quantile(0.75)
IQR = Q3-Q1
IQR
```

```
3.0
```

```
low_lim = Q1 - 1.5*IQR
low_lim
```

```
51.5
```

```
up_lim = Q3 - 1.5*IQR
up_lim
```

```
54.5
```

```
df[(df.table >= low_lim) & (df.table < up_lim)]
```

	carat	depth	table	price	x	y	z
13	0.31	62.2	54.0	344	4.35	4.37	2.71
16	0.30	62.0	54.0	348	4.31	4.34	2.68
17	0.30	63.4	54.0	351	4.23	4.29	2.70
37	0.31	64.0	54.0	402	4.29	4.31	2.75
51	0.23	61.9	54.0	404	3.93	3.95	2.44
...	...	...	...	...	...	...	...
53792	0.51	62.5	54.0	2730	5.12	5.16	3.21
53833	0.72	61.7	54.0	2737	5.77	5.80	3.57
53878	0.51	61.9	54.0	2745	5.17	5.11	3.18
53881	0.72	62.6	53.0	2746	5.76	5.78	3.61
53896	0.83	62.4	54.0	2751	6.01	6.08	3.77

3459 rows × 7 columns

Winsorize yöntemi ile aykırı olmayan değerlere dönüştürme

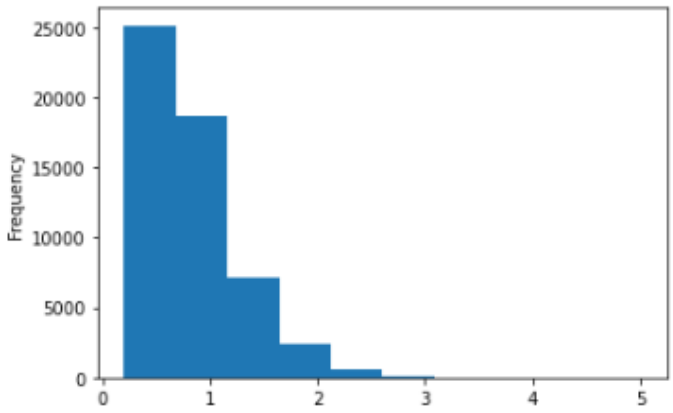
2. Transforming them into non-outlier values with Winsorize method

```
from scipy.stats.mstats import winsorize
df_w = winsorize(df.table, (0.001, 0.019))
df_w
```

```
masked_array(data=[55., 61., 63., ..., 60., 58., 55.],
             mask=False,
       fill_value=1e+20)
```

Log transformation method

```
df.carat.plot(kind="hist");
```

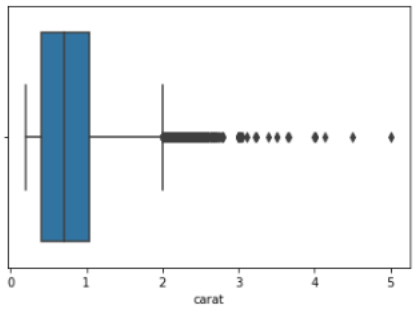


```
df["log_carat"] = np.log(df.carat)
df
```

	carat	depth	table	price	x	y	z	log_carat
0	0.23	61.5	55.0	326	3.95	3.98	2.43	-1.469676
1	0.21	59.8	61.0	326	3.89	3.84	2.31	-1.560648
2	0.23	56.9	65.0	327	4.05	4.07	2.31	-1.469676
3	0.29	62.4	58.0	334	4.20	4.23	2.63	-1.237874
4	0.31	63.3	58.0	335	4.34	4.35	2.75	-1.171183
...	...	...	...	...	...	...	...	...
53935	0.72	60.8	57.0	2757	5.75	5.76	3.50	-0.328504
53936	0.72	63.1	55.0	2757	5.69	5.75	3.61	-0.328504
53937	0.70	62.8	60.0	2757	5.66	5.68	3.56	-0.356675
53938	0.86	61.0	58.0	2757	6.15	6.12	3.74	-0.150823
53939	0.75	62.2	55.0	2757	5.83	5.87	3.64	-0.287682

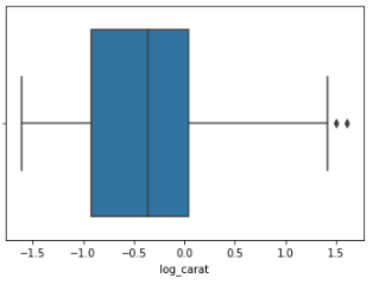
53940 rows × 8 columns

```
sns.boxplot(x=df.carat);
```



carat

```
sns.boxplot(x=df.log_carat);
```



Bir DataFrame'de aykırı değerler nasıl işlenir,

log dönüşümü çarpık verilerde çarpıklığı azaltarak normal dağılımlı hale getirmek için kullanılır.

winsorize(from scipy.stats.mstats import winsorize) (limits[0])inci en düşük değerler (limits[0])inci yüzdelik dilime ve (limits[1])inci en yüksek değerler (1 - limits[1])inci yüzdelik dilime ayarlanır

Combining DataFrames (Merging-Joining and Concatenating)

DataFrame'leri Birleştirme (Birleştirme-Birleştirme ve Birleştirme)

Merge;DataFrame veya adlandırılmış Seri nesnelerini SQL gibi veritabanı tarzı bir birleştirme ile birleştirir. İki veya daha fazla veri çerçevesini ortak sütunların değerleri temelinde birleştirmek için kullanılır**.**

```
left = pd.DataFrame({'key': ['K0', 'K1', 'K2'],  'A': ['A0', 'A1', 'A2'], 'B': ['B0', 'B1', 'B2']})
```

```
right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'], 'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']})
```

```
left
```

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2

```
right
```

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

```
pd.merge(left, right, on = "key", how ="inner")
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2

Join; sütunları diğer DataFrame ile indeks veya bir anahtar sütun üzerinde birleştirir. İndeks temelinde 2 veri çerçevesini birleştirmek için kullanılır; merge(left_index=True) kullanmak yerine join() kullanabiliriz.

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'], 'B': ['B0', 'B1', 'B2']}, index = ['K0', 'K1', 'K2'])
right = pd.DataFrame({'C': ['C0', 'C2', 'C3'], 'D': ['D0', 'D2', 'D3']},index = ['K0', 'K2', 'K3'])
```

```
left
```

	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2

```
right
```

	C	D
K0	C0	D0
K2	C2	D2
K3	C3	D3

```
left.join(right, how = "inner")
```

	A	B	C	D
K0	A0	B0	C0	D0
K2	A2	B2	C2	D2

Concat; Pandas nesnelerini belirli bir eksen boyunca birleştirir. Bir tür veri çerçevelerini üst üste veya yan yana ekleme işlemidir.

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],  'B': ['B0', 'B1', 'B2', 'B3'], 'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']})

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],  'B': ['B4', 'B5', 'B6', 'B7'], 'C': ['C4', 'C5', 'C6', 'C7'], 'D': ['D4', 'D5', 'D6', 'D7']})

df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'], 'B': ['B8', 'B9', 'B10', 'B11'], 'C': ['C8', 'C9', 'C10', 'C11'], 'D': ['D8', 'D9', 'D10', 'D11']})
```

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

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
pd.concat([df1, df2, df3], axis=1)
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	A4	B4	C4	D4	A8	B8	C8	D8
1	A1	B1	C1	D1	A5	B5	C5	D5	A9	B9	C9	D9
2	A2	B2	C2	D2	A6	B6	C6	D6	A10	B10	C10	D10
3	A3	B3	C3	D3	A7	B7	C7	D7	A11	B11	C11	D11

Pandas'ta yatay kombinasyon için merge() ve join() kullanılırken, dikey kombinasyon için concat() ve append() kullanılır.

Text and Time Methods(Metin ve Zaman Yöntemleri)

Birçok veri parçası saf sayılardan ziyade metin biçimindedir. Bu, dizginin incelenmeden, algoritmalar tarafından işlenmeden veya herkese gösterilmeden önce temizlenmesi ve ön işlemden geçirilmesi gerektiği anlamına gelir. Neyse ki pandas kütüphanesi, dize verileriyle çalışmayı kolaylaştıran dize işlemeye ayrılmış bir bölüme sahiptir.

Series ve Index, dizinin her bir elemanı üzerinde işlem yapmayı kolaylaştıran bir dizi dize işleme yöntemiyle donatılmıştır. Belki de en önemlisi, bu yöntemler eksik/NA değerlerini otomatik olarak hariç tutar. Bunlara str niteliği aracılığıyla erişilir.

```
df=sns.load_dataset("titanic")
df.head(4)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False

lower(): Dizeleri küçük harfe dönüştürür

```
df.embark_town = df.embark_town.str.lower()
df.embark_town.head(4)
```

```
0    southampton
1      cherbourg
2    southampton
3    southampton
Name: embark_town, dtype: object
```

upper() : Dizeleri büyük harfe dönüştürür

```
df.who = df.who.str.upper()
df.who.head(4)
```

```
0      MAN
1    WOMAN
2    WOMAN
3    WOMAN
Name: who, dtype: object
```

islower() : Her dizgideki tüm karakterlerin küçük harf olup olmadığını kontrol eder. Boolean döndürür

```
df.who.head(3).str.islower()
```

```
0    False
1    False
2    False
Name: who, dtype: bool
```

isupper() : Her karakter dizisindeki tüm karakterlerin büyük harf olup olmadığını kontrol eder. Boolean döndürür

```
df.who.head(3).str.isupper()
```

```
0    True
1    True
2    True
Name: who, dtype: bool
```

isdigit() : Her dizideki tüm karakterlerin rakam olup olmadığını kontrol eder.

```
df.survived.astype("string").str.isdigit().head(3)
```

```
0    True
1    True
2    True
Name: survived, dtype: boolean
```

isnumeric(): Her dizedeki tüm karakterlerin sayısal olup olmadığını kontrol eder. Boolean döndürür.

```
df.survived.astype("string").str.isnumeric().head(3)
```

```
0    True
1    True
2    True
Name: survived, dtype: boolean
```

replace(): a değerini b değeri ile değiştirir

```
df["age"] = df["age"]. replace(np.nan, "UNKNOWN")
df.head(6)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	MAN	True	NaN	southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	WOMAN	False	C	cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	WOMAN	False	NaN	southampton	yes	True
3	1	1	female	UNKNOWN	1	0	53.1000	S	First	WOMAN	False	C	southampton	yes	False
4	0	3	male	UNKNOWN	0	0	8.0500	S	Third	MAN	True	NaN	southampton	no	True
5	0	3	male	UNKNOWN	0	0	8.4583	Q	Third	MAN	True	NaN	queenstown	no	True

contains(): Alt dize öğeyi içeriyorsa her öğe için True, aksi takdirde False Boolean değerini döndürür.

```
df[df.sibsp.astype("string").str.contains("1")].head(3)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	MAN	True	NaN	southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	WOMAN	False	C	cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	WOMAN	False	C	southampton	yes	False

split(): Her dizeyi verilen desenle böler

```
df.who.str.split("MAN").head(3)
```

```
0      [, ]
1    [WO, ]
2    [WO, ]
Name: who, dtype: object
```

strip(): Her dizeden boşlukları (satırsonu dahil) silmeye yardımcı olur

```
df.sex.str.strip("le").head(3)
```

```
0      ma
1    fema
2    fema
Name: sex, dtype: object
```

find() : Örüntünün ilk oluşumunun ilk konumunu döndürür

```
df.embark_town.str.find("southampton").head(3)
```

```
0    0.0
1   -1.0
2    0.0
Name: embark_town, dtype: float64
```

findall() : Kalıbın tüm oluşumlarının bir listesini döndürür.

```
df.embark_town.str.findall("southampton").head(3)
```

```
0    [southampton]
1               []
2    [southampton]
Name: embark_town, dtype: object
```

Time Methods(Zaman Yöntemleri)

Zaman serisi verileri şu anda çok çeşitli sektörlerde zaman serisi tahmini, mevsimsellik analizi, trend tespiti ve kritik iş ve araştırma seçimleri yapmak için kullanılmaktadır. Sonuç olarak, bir veri bilimcisi veya veri analisti için zaman serisi verilerini doğru bir şekilde anlamak kritik öneme sahiptir.

datetime modülü Python'da zaman serisi verileriyle çalışmak için temel nesneleri içerir

```
df = sns.load_dataset("flights")
df.head(3)
```

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132

to_datetime() yöntemi birçok farklı türde tarih gösterimini ayrıştırarak bir Timestamp nesnesi döndürür

```
df["year"] = pd.to_datetime(df["year"], format = "%Y")
df.year.head(4)
```

```
0   1949-01-01
1   1949-01-01
2   1949-01-01
3   1949-01-01
Name: year, dtype: datetime64[ns]
```

strftime - nesneyi belirli bir biçime göre bir dizeye dönüştürür

```
from datetime import datetime
now_date = datetime.now()
now_date
```

```
datetime.datetime(2022, 11, 25, 15, 54, 1, 724114)
```

```
date = now_date.strftime(("%d" + " ""%b" + " ""%Y"))
date
```

```
'25 Nov 2022'
```

strptime - bir dizeyi, karşılık gelen bir biçim verilen bir datetime nesnesine ayrıştırır

```
datetime.strptime(date, "%d %b %Y")
```

```
datetime.datetime(2022, 11, 25, 0, 0)
```

- `timedelta` - gives time difference

```
from datetime import timedelta
five_days_before = now_date.now()-timedelta(days = 5)
five_days_before
```

```
datetime.datetime(2022, 11, 20, 16, 7, 52, 116287)
```

t**imedelta**- zaman farkını verir


```
from datetime import timedelta
five_days_before = now_date.now()-timedelta(days = 5)
five_days_before
```

```
datetime.datetime(2022, 11, 20, 16, 7, 52, 116287)
```


Get_Dummies Yöntemi

Makine Öğrenimi modelleri kategorik/sayısal olmayan verilerle çalışmadığından, kategorik verileri bir modele beslemeden önce sayısal verilere dönüştürmemiz gerekir. Get_Dummies yöntemi temel olarak kategorik değişkeni kukla/gösterge değişkenlerine dönüştürür.

Data Input and Output(Veri Girişi ve Çıkışı)

- **read_csv() : Read a comma-separated values (csv) file into DataFrame.**

```
pd.read_csv("penguins_train.csv")
```

- **DataFrame.to_csv() :Write DataFrame to a comma-separated values (csv) file.**

```
df.to_csv("./example.csv", index = False)
```

- **read_excel(): Read an Excel file into a pandas DataFrame.**

```
df = pd.read_excel("excel_sample1.xlsx")
```

- **DataFrame.to_excel(): Write object to an Excel sheet.**

```
df = pd.read_excel("excel_sample1.xlsx", sheet_name = "Sheet1", index = False)
```

- **read_html()** : Read HTML tables into a `list` of `DataFrame` objects. Most of the time you don't need to save your file as HTML so we pass that.

```
df = pd.read_html("https://www.imdb.com/chart/top/")
```

SQL dosyaları

İlişkisel bir veritabanından read_sql() ile veri okumak mümkün değildir, çünkü tüm verileri bir kerede belleğe yüklemek belleğinizi aşırı yükler. Bunun yerine sqlalchemy kütüphanesi--> create_engine metodu yardımıyla önce bir veritabanı oluşturabilirsiniz. Daha sonra to_sql() ile bu veritabanına herhangi bir kaynaktan veri aktarabilirsiniz. Son olarak, SQL veritabanından veri kümesini okumak için read_sql() komutunu veya veritabanından benzersiz bir sorgu yapmak için read_sql_query() komutunu kullanabilirsiniz.