

# İçindekiler

- Veri Ön İşlemeye Genel Bakış
- Aykırı Gözlem Analizi
  - Aykırı Gözlem
  - Kime Göre Neye Göre Aykırı?
  - Aykırı Değerleri Yakalamak
  - Aykırı Değer Problemini Çözmek
  - Çok Değişkenli Aykırı Gözlem Analizi
  - Baskılama
- Eksik Gözlem Analizi
  - Eksik Gözlem
  - Eksik Veri Hızlı Çözüm
  - Eksik Veri Yapısının Görselleştirilmesi
  - Silme Yöntemleri
  - Basit Değer Atama Yöntemleri
  - Kategorik Değişken Kırılımında Değer Atama
  - Kategorik Değişkenlerde Değer Atama
  - Tahmine Dayalı Değer Atama Yöntemleri
- Değişken Dönüşümleri
  - Değişken Standardizasyonu (Veri Standardizasyonu)
  - Değişken Dönüşümleri
  - One Hot Dönüşümü Ve Dummy Değişken Tuzağı

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
```

## Veri Ön İşlemeye Genel Bakış

Veri mi? Model mi?

Makine öğrenmesi modelinin amacı genellenebilir yapılar ortaya koymaktır.

Belirli olaylar gözlemediğinde belirli tahmin sonuçları vermektedir.

# Eğer veriniz kötü ise, makine öğrenmesi araçlarınız kullanıssız olacaktır\*

by Thomas C. Redman

APRIL 02, 2018

[SUMMARY](#) [SAVE](#) [SHARE](#) [COMMENT 0](#) [TEXT SIZE](#) [PRINT](#) [\\$8.95 BUY COPIES](#)



Veri Ön İşleme Genel Bakış

# Veri Ön İşleme Genel Bakış

- Veri Temizleme (data cleaning / cleasing)
  - Gürültülü Veri
  - Eksik Veri Analizi
  - Aykırı Gözlem Analizi
- Veri Standardizasyonu
  - 0-1 Dönüşümü
  - z-skoruna Dönüşüm
  - Logaritmik Dönüşüm
- Veri İndirgeme
  - Gözlem Sayısının Azaltılması
  - Değişken Sayısının Azaltılması
- Değişken Dönüşümleri
  - Sürekli değişkenlerde dönüşümler
  - Kategorik değişkenlerde dönüşümler

## Gürültülü Veri Nedir?

Kişiler	Cinsiyet	Hamilelik Durumu
Kişi1	Erkek	1 (Evet)

## Eksik Veri Nedir?

Araç Fiyatı	KM	Vites Türü	Hasar Durumu	Marka	Model
10000	300000	Manuel	Evet	A	A1
54000	40000	Manuel	Hayır	A	A2
46999	90000	Manuel	Evet	B	B1
89000	1000000	Otomatik	Evet	C	C1
70000	78000	Otomatik	Evet	B	B2
50000	30000	Manuel	Hayır	C	C2
NA	600000	Manuel	Hayır	C	C3
68900	50000	Otomatik	Hayır	B	B2
12000	200000	Manuel	Hayır	A	A1

## Aykırı Gözlem Nedir?

Araç Fiyatı	KM	Vites Türü	Hasar Durumu	Marka	Model
10000	300000	Manuel	Evet	A	A1
54000	40000	Manuel	Hayır	A	A2
46999	90000	Manuel	Evet	B	B1
89000	1000000	Otomatik	Evet	C	C1
70000	78000	Otomatik	Evet	B	B2
50000	30000	Manuel	Hayır	C	C2
NA	600000	Manuel	Hayır	C	C3
68900	50000	Otomatik	Hayır	B	B2
12000	200000	Manuel	Hayır	A	A1

# Aykırı Gözlem Analizi

## Aykırı Gözlem

Veride genel eğilimin oldukça dışına çıkan ya da diğer gözlemlerden oldukça farklı olan gözlemlere **aykırı gözlem** denir.

Aykırılığı ifade eden nümerik değere **aykırı değer** denir.

Aykırı değeri barındıran gözlem birimine **aykırı gözlem** denir.

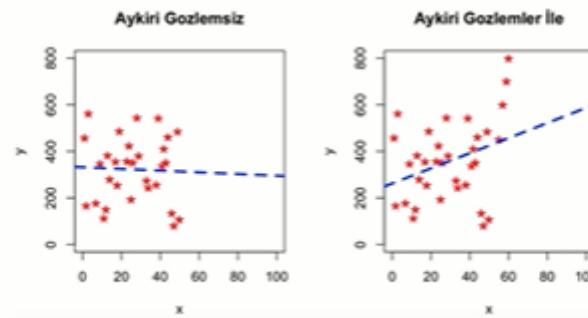
**Not:** Arada çok fark olmamakla beraber bilinmesinde fayda vardır. Biri diğerinin yerine kullanılabilir.

**EN ÖNEMLİ PROBLEMLERDEN BİRİSİDİR.**

**Aykırı Gözlem Neye Sebep Olur?**

**Aykırı Değer Neye Sebep Olur?**

*Genellenebilirlik kaygısı ile oluşturulan kural setlerini ya da fonksiyonları yanlıtır. Yanlılığa sebep olur.*



Grafiği inceleyeceğiz olursak normalde X-Y arasındaki ilişki, doğruya bakıldığından negatif yönlü iken aykırı gözlemler yüzünden doğrunun eğimi pozitif yönlü olmuş. Yani aykırı gözlemler değişkenler arasındaki ilişkinin **genel yönünü** bozmuştur. Bu yüzden aykırı gözlemler bu gibi sorunlara sebep olabilmektedir.

## Kime Göre Neye Göre Aykırı?

Aykırı gözlem, veride genel eğilimin oldukça dışına çıkan gözlemlerdir.

**Peki veri setinin genel eğiliminin dışına çıkmayı nasıl tanımlarız?**

**1. Sektör Bilgisi**

## **1. Sektör Bilgisi**

Örneğin bir ev fiyat tahmin modelinde 1000 metrekarelük evleri modellemeye almamak.

Eğer kurulan modelin bir genelleme kaygısı varsa;  
zaten çok seyrek olan senoryalar ve genele  
uymayan yapılar çalışmanın dışında bırakılmalıdır.

Aykırı gözlem tespiti için uygulanacak formülleri kullanmak yerine sektör kapsamında uzman kişi yardımıyla veya kendimiz hangi gözlemlerin aykırı olduğunu bulabiliriz.

## **2. Standart Sapma Yaklaşımı**

## 2. Standart Sapma Yaklaşımı

Bir değişkenin ortalamasının üzerine aynı değişkenin standart sapması hesaplanarak eklenir. 1,2 ya da 3 standart sapma değeri ortalama üzerine eklenerek ortaya çıkan bu değer eşik değer olarak düşünülür ve bu değerden yukarıda ya da aşağıda olan değerler aykırı değer olarak tanımlanır.

$$\text{Eşik Değer} = \text{Ortalama} + 1 \times \text{Standart Sapma}$$

$$\text{Eşik Değer} = \text{Ortalama} + 2 \times \text{Standart Sapma}$$

$$\text{Eşik Değer} = \text{Ortalama} + 3 \times \text{Standart Sapma}$$



**Not:** 1,2 ve 3 standart sapma değeri örnekten örneğe değişebilir.

**Örnek**

$$\text{Ortalama} = 100,000$$

$$\text{Standart Sapma} = 20,000$$

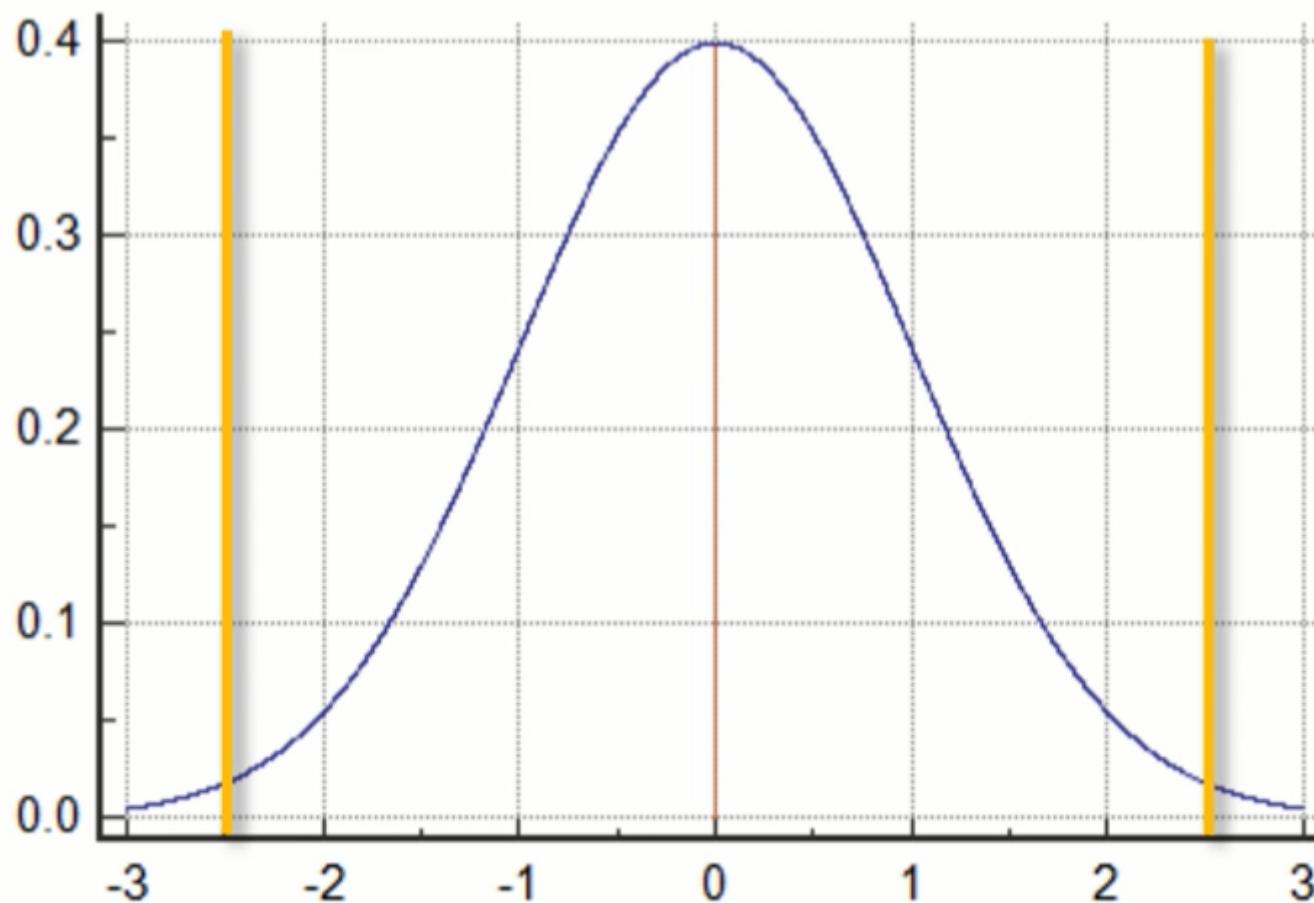
$$\text{Eşik Değer} = 100,000 + 2 \times 20,000$$

$$\text{Eşik Değer} = 140,000$$

## 3. Z-Skoru Yaklaşımı

### 3. Z-Skoru Yaklaşımı

Standart sapma yöntemine benzer şekilde çalışır. Değişken standart normal dağılıma uyarlanır, yani standartlaştırılır. Sonrasında -örneğin- dağılımin sağından ve solundan  $-2,5$  değerine göre bir eşik değer konulur ve bu değerin üzerinde ya da altında olan değerler aykırı değer olarak işaretlenir.

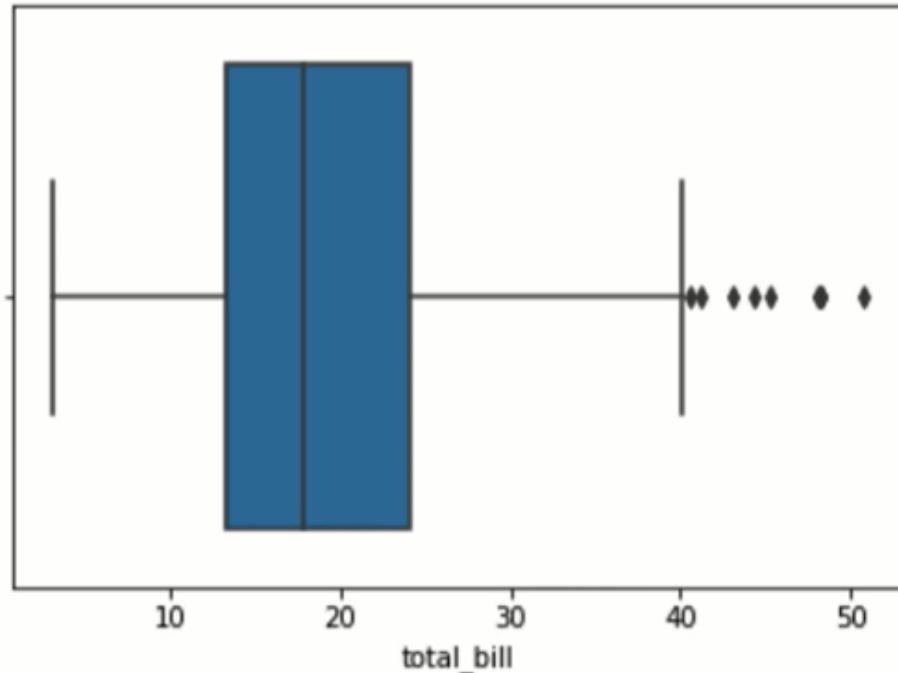


$$\pm 2,5$$

#### 4. Boxplot(Interquartile Range - IQR) Yöntemi

#### 4. Boxplot(interquartile range - IQR) Yöntemi

En sık kullanılan yöntemlerden birisidir. Değişkenin değerleri küçükten büyüğe sıralanır. Çeyrekliklerine (yüzdekliliklerine) yani Q1,Q3 değerlerine karşılık değerler üzerinden bir eşik değer hesaplanır ve bu eşik değere göre aykırı değer tanımı yapılır.



$$IQR = 1.5 \times (Q3 - Q1)$$

$$\text{Alt Eşik Değer} = Q1 - IQR$$

$$\text{Üst Eşik Değer} = Q3 + IQR$$

Bu yöntemlerden genellikle 1. ve 4. yöntemleri kullanmaya çalışacağız. Sektör bilgimiz varsa 1. yöntemi, yoksa ve tek değişkenli bir aykırı gözlem incelemesi yapacaksak 4. yöntemi kullanacağız.

## Aykırı Değerleri Yakalamak

Tek değişkenli olarak ele alınacaktır.

In [2]:

```
df = sns.load_dataset("diamonds")
df = df.select_dtypes(include = ["float64", "int64"]) #Sayısal değişkenler seçildi.
df = df.dropna() #Eksik değerler silindi. (Aykırı değer tespitinin kolay yapılması için silindi. İleriki derste detaylı ele alınacaktır.)
df.head()
```

Out[2]:

	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

Sayısal değişkenler geldi. Burada **table** değişkeninin aykırı değerlerini tespit edeceğiz.

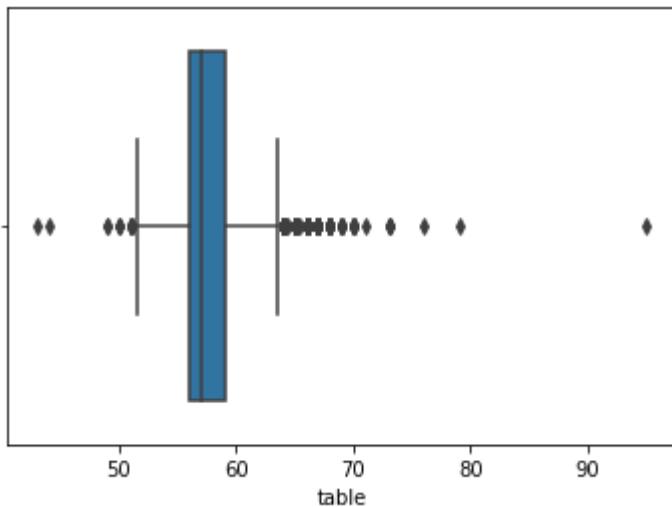
```
In [3]: df_table = df["table"]
```

```
In [4]: df_table.head()
```

```
Out[4]: 0    55.0  
1    61.0  
2    65.0  
3    58.0  
4    58.0  
Name: table, dtype: float64
```

Boxplot yöntemini kullanarak aykırı değerleri tespit edelim. Öncelikle grafiğini çizdirerek aykırı değerleri gözlemleyelim:

```
In [5]: sns.boxplot(x = df_table);
```



Göründüğü üzere grafikteki noktalar bizim aykırı değerlerimizdir. Şimdi bu yöntemden bir eşik değer hesapyalım:

```
In [6]: Q1 = df_table.quantile(0.25) #Çeyreklik hesaplama fonksiyonu. (Varsayılan: 0.5)  
Q3 = df_table.quantile(0.75)  
IQR = Q3 - Q1
```

```
In [7]: Q1
```

```
Out[7]: 56.0
```

In [8]:

Q3

Out[8]: 59.0

In [9]:

IQR

Out[9]: 3.0

In [10]:

```
alt_sinir = Q1 - 1.5 * IQR
ust_sinir = Q3 + 1.5 * IQR
#Konu anlatımdan farklı olarak bu formül de uygulanabilir.
```

In [11]:

alt\_sinir

Out[11]: 51.5

In [12]:

ust\_sinir

Out[12]: 63.5

Alt sınırları ve üst sınırları belirledik. Şimdi bunu `df_table` üzerinden bu şartları koyarak değerleri tespit edelim:

In [13]:

```
(df_table < alt_sinir) | (df_table > ust_sinir)
```

Out[13]:

Index	Value
0	False
1	False
2	True
3	False
4	False
...	
53935	False
53936	False
53937	False
53938	False
53939	False

Name: table, Length: 53940, dtype: bool

Örnek olması açısından alt sınırda olanları alalım:

In [14]:

```
aykiri_tf_alt = (df_table < alt_sinir)
```

```
In [15]:
```

```
aykiri_tf_alt.head()
```

```
Out[15]: 0    False  
1    False  
2    False  
3    False  
4    False  
Name: table, dtype: bool
```

Göründüğü üzere True-False şeklinde değerler döndürdü. Şimdi **True** olanların kendi değerlerine erişelim:

```
In [16]:
```

```
df_table[aykiri_tf_alt]
```

```
Out[16]: 1515    51.0  
3238    50.1  
3979    51.0  
4150    51.0  
5979    49.0  
7418    50.0  
8853    51.0  
11368   43.0  
22701   49.0  
25179   50.0  
26387   51.0  
33586   51.0  
35633   44.0  
45798   51.0  
46040   51.0  
47630   51.0  
Name: table, dtype: float64
```

Göründüğü üzere [] içerisine yazarak **True** değerlerine eristik. Şimdi bunların indexlerini çekelim:

```
In [17]:
```

```
df_table[aykiri_tf_alt].index
```

```
Out[17]: Int64Index([ 1515,  3238,  3979,  4150,  5979,  7418,  8853, 11368, 22701,  
                     25179, 26387, 33586, 35633, 45798, 46040, 47630],  
                     dtype='int64')
```

Bu indexlere erişmemizin sebebi bu aykırı değerlerin indexlerini kullanarak silme işlemi, ortalamaya çekme işlemi ya da baskılama işlemini gerçekleştirebiliriz.

## Aykırı Değer Problemini Çözmek

```
In [18]:
```

```
df_table[aykiri_tf_alt]
```

```
Out[18]: 1515    51.0
3238    50.1
3979    51.0
4150    51.0
5979    49.0
7418    50.0
8853    51.0
11368   43.0
22701   49.0
25179   50.0
26387   51.0
33586   51.0
35633   44.0
45798   51.0
46040   51.0
47630   51.0
Name: table, dtype: float64
```

## 1. Silme

```
In [19]: type(df_table)
```

```
Out[19]: pandas.core.series.Series
```

```
In [20]: df_table = pd.DataFrame(df_table)
```

```
In [21]: df_table.shape
```

```
Out[21]: (53940, 1)
```

```
In [22]: t_df = df_table[~((df_table < (alt_sinir)) | (df_table > (ust_sinir))).any(axis = 1)]
t_df #temiz_df
```

#Buradaki "~" işaretini bu şartı SAĞLAMAYANLARI getir demektir. Yani temizlenmesi için tüm aykırı değerleri almamış olduk.  
#any(axis = 1) komutunun anlamı da sütunda koşulu sağlayan her veriyi getir demektir.

```
Out[22]: table
          0    55.0
          1    61.0
          3    58.0
          4    58.0
```

table	
5	57.0
...	...
<b>53935</b>	57.0
<b>53936</b>	55.0
<b>53937</b>	60.0
<b>53938</b>	58.0
<b>53939</b>	55.0

53335 rows × 1 columns

```
In [23]: t_df.shape
```

```
Out[23]: (53335, 1)
```

## 2. Ortalama ile Doldurma

```
In [24]: df = sns.load_dataset("diamonds")
df = df.select_dtypes(include = ["float64", "int64"])
df = df.dropna()
df.head()
```

	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

```
In [25]: df_table = df["table"]
```

```
In [26]: aykiri_tf_alt.head() #Alt sınırın altındaki aykırı değerler
```

```
Out[26]: 0    False
1    False
2    False
3    False
4    False
Name: table, dtype: bool
```

```
In [27]: df_table[aykiri_tf_alt]
```

```
Out[27]: 1515      51.0
3238      50.1
3979      51.0
4150      51.0
5979      49.0
7418      50.0
8853      51.0
11368     43.0
22701     49.0
25179     50.0
26387     51.0
33586     51.0
35633     44.0
45798     51.0
46040     51.0
47630     51.0
Name: table, dtype: float64
```

```
In [28]: df_table.mean()
```

```
Out[28]: 57.45718390804598
```

Bu ortalamayı aykırı değerlere eşitleyelim:

```
In [29]: df_table[aykiri_tf_alt] = df_table.mean()
```

```
<ipython-input-29-2af50f7fa2cd>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_table[aykiri_tf_alt] = df_table.mean()
```

```
In [30]: df_table[aykiri_tf_alt]
```

```
Out[30]: 1515      57.457184
3238      57.457184
3979      57.457184
```

```
4150    57.457184
5979    57.457184
7418    57.457184
8853    57.457184
11368   57.457184
22701   57.457184
25179   57.457184
26387   57.457184
33586   57.457184
35633   57.457184
45798   57.457184
46040   57.457184
47630   57.457184
Name: table, dtype: float64
```

Gördüğü üzere **alt sınırın altındaki tüm aykırı değerler** ortalamaya çevrilmiş oldu.

### 3. Baskılama

Bu yöntem, aykırılar yakalandıktan sonra üst sınırın üstündekiler üst sınıra eşitlenir, alt sınırın altındakiler ise alt sınıra eşitlenir.

Eğer ortalama ile sınırlar arasındaki mesafe çok uzaksa sınırlara eşitlemek mantıklı olacaktır.

In [31]:

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

Out[31]:

	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

In [32]:

```
df_table = df["table"]
```

In [33]:

```
df_table[aykiri_tf_alt]
```

Out[33]:

1515	51.0
3238	50.1
3979	51.0
4150	51.0

```
5979    49.0
7418    50.0
8853    51.0
11368   43.0
22701   49.0
25179   50.0
26387   51.0
33586   51.0
35633   44.0
45798   51.0
46040   51.0
47630   51.0
Name: table, dtype: float64
```

```
In [34]: alt_sinir
```

```
Out[34]: 51.5
```

Şimdi bu alttaki aykırı değerleri alt sınıra eşitleyelim:

```
In [35]: df_table[aykiri_tf_alt] = alt_sinir
```

```
<ipython-input-35-87e9b5d159e7>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_table[aykiri_tf_alt] = alt_sinir
```

```
In [36]: df_table[aykiri_tf_alt]
```

```
Out[36]: 1515    51.5
3238    51.5
3979    51.5
4150    51.5
5979    51.5
7418    51.5
8853    51.5
11368   51.5
22701   51.5
25179   51.5
26387   51.5
33586   51.5
35633   51.5
45798   51.5
46040   51.5
47630   51.5
Name: table, dtype: float64
```

Görüldüğü üzere alttaki aykırı değerler için alt sınıra eşitlenmiş oldu.

Örnekten dolayı sadece alt sınır için yapmak yeterlidir.

## Çok Değişkenli Aykırı Gözlem Analizi

### Local Outlier Factor

Gözlemleri bulundukları konumda yoğunluk tabanlı skorlayarak buna göre aykırı değer olabilecek değerleri tanımlayabilmemize imkan sağlar.

Bir noktanın local yoğunluğu bu noktanın komşuları ile karşılaştırılıyor. Eğer bir nokta, komşularının yoğunluğundan anlamlı şekilde düşük ise "**bu nokta komşularından daha seyrek bir bölgede bulunuyordur**" yorumu yapılabiliyor. Dolayısıyla burada bir komşuluk yapısı söz konusudur. "**Bir değerin çevresi yoğun değilse demek ki bu değer aykırı değerdir**" şeklinde değerlendiriliyor.

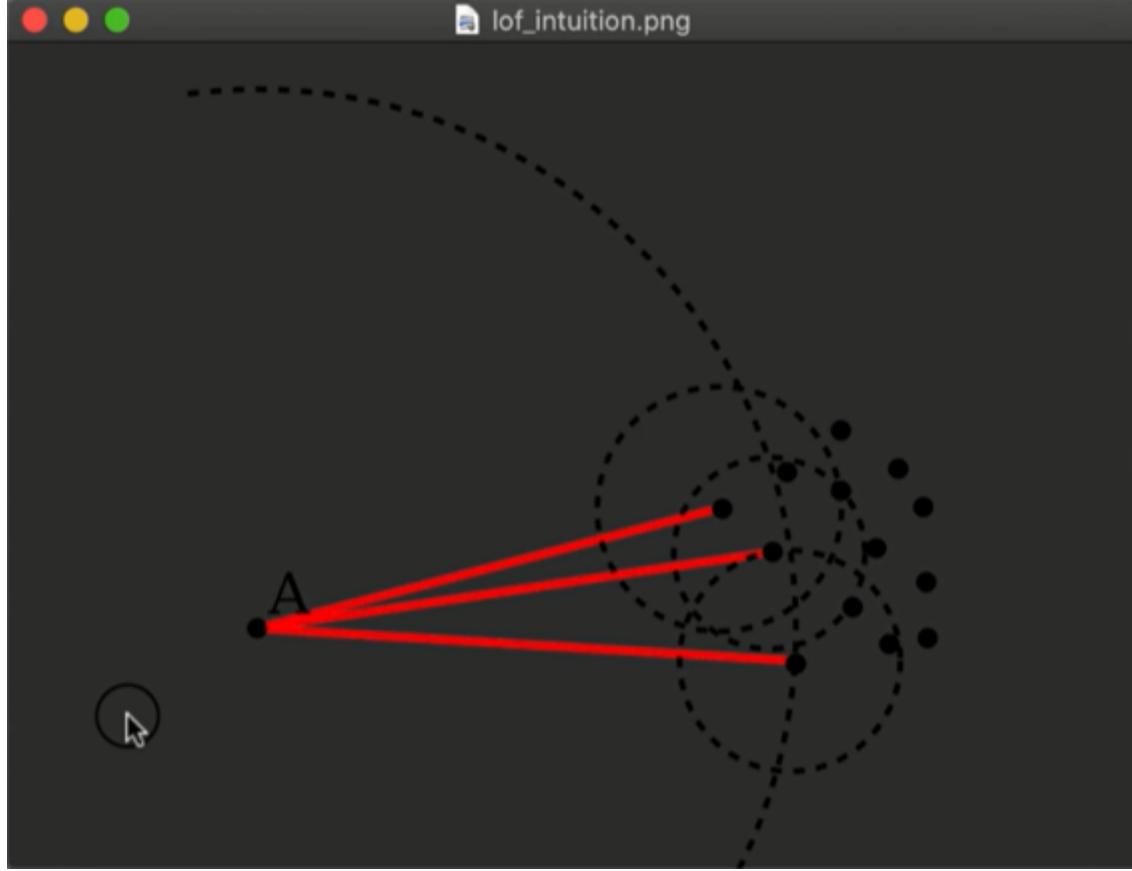
Bu yöntemin ortaya çıkış sebebi tek değişken için ilgili değişkenler aykırı değer olmazken çok değişkenli şeklinde bakıldığından aykırı değer olabilir. En çok kullanılan yöntemlerden birisidir.

**Örnek:**

Yaş	Evlilik Sayısı
17	3
18	2
70	3
80	1

**Evlilik Sayısı** değişkenine tekil olarak bakıldığından 3 değeri aykırı bir değer olmazken **Yaş** değişkeniyle beraber bakıldığından 17 yaşındaki birisinin 3 evlilik yapması aykırı bir durumdur. Gerçekte olsa bile genel yapının dışında olduğundan aykırı gözlem olarak algılanır.

Resim üzerinden bunu inceleyelim:



En yakın üç komşuya bakılacak olursa A noktasının diğerlerine göre komşusu daha uzak olduğundan aykırı değer olarak algılanır.

In [37]:

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

Out[37]:

	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

```
In [38]: from sklearn.neighbors import LocalOutlierFactor
```

```
In [39]: clf = LocalOutlierFactor(n_neighbors=20, contamination=0.1)

#n_neighbors: Komşu sayısı
#contamination: Yoğunluk
```

```
In [40]: clf.fit_predict(df)
```

```
Out[40]: array([-1, -1, -1, ..., 1, 1])
```

```
In [41]: df_scores = clf.negative_outlier_factor_
```

```
In [42]: df_scores[0:10]
```

```
Out[42]: array([-1.58352526, -1.59732899, -1.62278873, -1.33002541, -1.30712521,
 -1.28408436, -1.28428162, -1.26458706, -1.28422952, -1.27351342])
```

Her bir değer için skorları hesapladı. Bunları sıralayacak olursak:

```
In [43]: np.sort(df_scores)[0:20]
```

```
Out[43]: array([-8.60430658, -8.20889984, -5.86084355, -4.98415175, -4.81502092,
 -4.81502092, -4.61522833, -4.37081214, -4.29842288, -4.10492387,
 -4.0566648 , -4.01831733, -3.94882806, -3.82378797, -3.80135297,
 -3.75680919, -3.65947378, -3.59249261, -3.55564138, -3.47157375])
```

Gördüğü üzere skorlar sıralandı. Bundan sonra belirli kriterlere (proje hassaslığı, gözlem çokluğu vs.) ve yönteme göre bir eşik değer belirlememiz gerekiyor fakat örnek yaptığımızdan dolayı gözlem yaparak bulabiliriz.

Örneğin 13. skoru eşik değer olarak belirleyelim:

```
In [44]: esik_deger = np.sort(df_scores)[13]
esik_deger
```

```
Out[44]: -3.823787967755565
```

Sonra eşik değerden büyük olanları alalım. Böylelikle aykırı değerleri almamış oluruz.

```
In [45]: aykiri_tf = df_scores > esik_deger
```

In [46]:

```
aykiri_tf
```

Out[46]: array([ True, True, True, ..., True, True, True])

True şeklinde geldi. Şimdi bu değerleri getirelim:

### Silme Yöntemi

In [47]:

```
yeni_df = df[aykiri_tf]
```

In [48]:

```
yeni_df
```

Out[48]:

	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

53926 rows × 7 columns

Aykırı değerleri getirelim:

In [49]:

```
df[df_scores < esik_deger]
```

Out[49]:

	carat	depth	table	price	x	y	z
6341	1.00	44.0	53.0	4032	6.31	6.24	4.12

	carat	depth	table	price	x	y	z
<b>10377</b>	1.09	43.0	54.0	4778	6.53	6.55	4.12
<b>24067</b>	2.00	58.9	57.0	12210	8.09	58.90	8.06
<b>35633</b>	0.29	62.8	44.0	474	4.20	4.24	2.65
<b>36503</b>	0.30	51.0	67.0	945	4.67	4.62	2.37
<b>38840</b>	0.73	70.8	55.0	1049	5.51	5.34	3.84
<b>41918</b>	1.03	78.2	54.0	1262	5.72	5.59	4.42
<b>45688</b>	0.70	71.6	55.0	1696	5.47	5.28	3.85
<b>48410</b>	0.51	61.8	54.7	1970	5.12	5.15	31.80
<b>49189</b>	0.51	61.8	55.0	2075	5.15	31.80	5.12
<b>50773</b>	0.81	68.8	79.0	2301	5.26	5.20	3.58
<b>52860</b>	0.50	79.0	73.0	2579	5.21	5.18	4.09
<b>52861</b>	0.50	79.0	73.0	2579	5.21	5.18	4.09

Önceki derste tek değişkenli olarak "**table**" değişkenini incelediğimizde ona kıyasla çok daha az aykırı gözlemler geldi.

## Baskılama

Eşik değere ilişkin gözlemi getirelim:

In [50]:

```
df[df_scores == esik_deger]
```

Out[50]:

	carat	depth	table	price	x	y	z
<b>31230</b>	0.45	68.6	57.0	756	4.73	4.5	3.19

In [51]:

```
baski_deger = df[df_scores == esik_deger]
```

In [52]:

```
aykirlar = df[~aykiri_tf]
#"aykiri_tf = df_scores > esik_deger" idi. Bunu sağlamayanları (eşik değer de dahil) getir demektir.
#Çünkü bunun zıt koşulu: df_scores <= esik_deger şeklindedir.
#Yani burada eşik değerin kendisi de alındı.
```

aykirlar

Out[52]:

	<b>carat</b>	<b>depth</b>	<b>table</b>	<b>price</b>	<b>x</b>	<b>y</b>	<b>z</b>
<b>6341</b>	1.00	44.0	53.0	4032	6.31	6.24	4.12
<b>10377</b>	1.09	43.0	54.0	4778	6.53	6.55	4.12
<b>24067</b>	2.00	58.9	57.0	12210	8.09	58.90	8.06
<b>31230</b>	0.45	68.6	57.0	756	4.73	4.50	3.19
<b>35633</b>	0.29	62.8	44.0	474	4.20	4.24	2.65
<b>36503</b>	0.30	51.0	67.0	945	4.67	4.62	2.37
<b>38840</b>	0.73	70.8	55.0	1049	5.51	5.34	3.84
<b>41918</b>	1.03	78.2	54.0	1262	5.72	5.59	4.42
<b>45688</b>	0.70	71.6	55.0	1696	5.47	5.28	3.85
<b>48410</b>	0.51	61.8	54.7	1970	5.12	5.15	31.80
<b>49189</b>	0.51	61.8	55.0	2075	5.15	31.80	5.12
<b>50773</b>	0.81	68.8	79.0	2301	5.26	5.20	3.58
<b>52860</b>	0.50	79.0	73.0	2579	5.21	5.18	4.09
<b>52861</b>	0.50	79.0	73.0	2579	5.21	5.18	4.09

Eşik değeri ve aykırıları değişkenlere kaydettik. Bu aykırıları baskı değerine (eşik değeri) eşitlememiz için bunların indexlerini kaldırıp değerleri numpy array'ine çevirmemiz gerekiyor. Aksi takdirde index sorunlarına yol açabiliyor.

In [53]:

```
res = aykirlar.to_records(index = False)
res
```

Out[53]:

```
rec.array([(1. , 44. , 53. , 4032, 6.31, 6.24, 4.12),
(1.09, 43. , 54. , 4778, 6.53, 6.55, 4.12),
(2. , 58.9, 57. , 12210, 8.09, 58.9 , 8.06),
(0.45, 68.6, 57. , 756, 4.73, 4.5 , 3.19),
(0.29, 62.8, 44. , 474, 4.2 , 4.24, 2.65),
(0.3 , 51. , 67. , 945, 4.67, 4.62, 2.37),
(0.73, 70.8, 55. , 1049, 5.51, 5.34, 3.84),
(1.03, 78.2, 54. , 1262, 5.72, 5.59, 4.42),
(0.7 , 71.6, 55. , 1696, 5.47, 5.28, 3.85),
(0.51, 61.8, 54.7, 1970, 5.12, 5.15, 31.8 ),
(0.51, 61.8, 55. , 2075, 5.15, 31.8 , 5.12),
(0.81, 68.8, 79. , 2301, 5.26, 5.2 , 3.58),
(0.5 , 79. , 73. , 2579, 5.21, 5.18, 4.09),
```

```
(0.5 , 79. , 73. , 2579, 5.21, 5.18, 4.09)],  
dtype=[('carat', '<f8'), ('depth', '<f8'), ('table', '<f8'), ('price', '<i8'), ('x', '<f8'), ('y', '<f8'), ('z', '<f8')])
```

Şimdi eşitlememizi yapalım:

In [54]:

```
res[:] = baski_deger.to_records(index = False)
```

#Kod pratikliği açısından baskı değerindeki gözlemin indexini direkt kaldırıp eşitledik.

In [55]:

```
res
```

```
Out[55]: rec.array([(0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),  
                   (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19)],  
                  dtype=[('carat', '<f8'), ('depth', '<f8'), ('table', '<f8'), ('price', '<i8'), ('x', '<f8'), ('y', '<f8'), ('z', '<f8')])
```

Göründüğü üzere aykırı gözlemler eşik değerdeki gözlemle aynı olmuş oldu.

Şimdi ise bu **res** değişkenini DataFrame'e çevirip aykırı gözlemlerin olduğu DataFrame'e eşitlememiz gerekiyor.

In [56]:

```
df[~aykiri_tf]
```

Out[56]:

	carat	depth	table	price	x	y	z
<b>6341</b>	1.00	44.0	53.0	4032	6.31	6.24	4.12
<b>10377</b>	1.09	43.0	54.0	4778	6.53	6.55	4.12
<b>24067</b>	2.00	58.9	57.0	12210	8.09	58.90	8.06
<b>31230</b>	0.45	68.6	57.0	756	4.73	4.50	3.19
<b>35633</b>	0.29	62.8	44.0	474	4.20	4.24	2.65
<b>36503</b>	0.30	51.0	67.0	945	4.67	4.62	2.37
<b>38840</b>	0.73	70.8	55.0	1049	5.51	5.34	3.84

	carat	depth	table	price	x	y	z
<b>41918</b>	1.03	78.2	54.0	1262	5.72	5.59	4.42
<b>45688</b>	0.70	71.6	55.0	1696	5.47	5.28	3.85
<b>48410</b>	0.51	61.8	54.7	1970	5.12	5.15	31.80
<b>49189</b>	0.51	61.8	55.0	2075	5.15	31.80	5.12
<b>50773</b>	0.81	68.8	79.0	2301	5.26	5.20	3.58
<b>52860</b>	0.50	79.0	73.0	2579	5.21	5.18	4.09
<b>52861</b>	0.50	79.0	73.0	2579	5.21	5.18	4.09

Yani değişkenimiz bu DataFrame'e eşitlenmelidir.

```
In [57]: df[~aykiri_tf] = pd.DataFrame(res, index = df[~aykiri_tf].index)
```

```
In [58]: df[~aykiri_tf]
```

Out[58]:

	carat	depth	table	price	x	y	z
<b>6341</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>10377</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>24067</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>31230</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>35633</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>36503</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>38840</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>41918</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>45688</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>48410</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>49189</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>50773</b>	0.45	68.6	57.0	756	4.73	4.5	3.19
<b>52860</b>	0.45	68.6	57.0	756	4.73	4.5	3.19

	carat	depth	table	price	x	y	z
52861	0.45	68.6	57.0	756	4.73	4.5	3.19

**Not:** Bu yapılan işlem aslında ileri seviye bir işlemidir. Udemy, Coursera vs. gibi eğitim platformlarında %99 oranda böyle bir yaklaşım anlatılmamaktadır. Kaggle'da da genelde tek değişkenli yaklaşımla aykırı gözlemler tespit edilir.

Bayağı havalı 😊. Buradan Vahit hocamıza çok teşekkür ederim. 😊

## Eksik Gözlem Analizi

### Eksik Gözlem

Eksik veri, incelenen veri setindeki gözlemlerin eksiklik olması durumunu ifade etmektedir.

Araç Fiyatı	KM	Vites Türü	Hasar Durumu	Marka	Model
10000	300000	Manuel	Evet	A	A1
54000	40000	Manuel	Hayır	A	A2
46999	90000	Manuel	Evet	B	B1
89000	1000000	Otomatik	Evet	C	C1
70000	78000	Otomatik	Evet	B	B2
50000	30000	Manuel	Hayır	C	C2
NA	600000	Manuel	Hayır	C	C3
68900	50000	Otomatik	Hayır	B	B2
12000	200000	Manuel	Hayır	A	A1

Böyle bir durumla karşılaşıldığında eksik veriyi ya silebiliriz, ya da doldurabiliriz.

### Eksik Veriyi Direkt Silmenin Zararları

Eksik değere sahip gözlemlerin veri setinden direk  
çıkarılması ve rassallığının incelenmemesi yapılacak  
istatistiksel çıkarımların, modelleme çalışmalarının  
güvenilirliğini düşürecektir. (Alpar, 2011)

Eksik gözlemlerin veri setinden direk çıkarılabilmesi için veri setindeki eksikliğin bazı durumlarda kısmen bazı durumlarda tamamen rastlantısal olarak olmuş olması gerekmektedir. Eğer eksiklikler değişkenler ile ilişkili olarak ortaya çıkan yapısal problemler ile meydana gelmiş ise bu durumda yapılacak silme işlemleri ciddi yanlılıklara sebep olabilecektir.

(Tabachnick ve Fidell, 1996)

### Eksik Verilerde Bilinmesi Gerekenler

- 1. Veri setindeki eksikliğin yapısal bir eksilik olup olmadığı bilinmesi gereklidir!

Müşteriler	Kredi Kartı Harcaması	Kredi Kartı Sahip Olma Durumu
Müşteri1	NA	0 (Sahip değil)

Tek değişkenli olarak bakıldığından eksik veri olarak gözükse de başka bir değişkenle bakıldığından aslında eksik olmasının mantıklı olduğunu görürüz. **Eksik veri incelemesinde bu çok dikkat edilmelidir.** Yani açıklamada dendiği gibi bu **eksik veriler başka değişkenlerle bağımlı olmamalı, rastgele oluşmalı.** Böyle bir durumda veriyi silmek mantıksız olacaktır.

- 2. NA her zaman eksiklik anlamına gelmez!

Müşteriler	Kredi Kartı Harcaması	Kredi Kartı Sahip Olma Durumu
Müşteri1	NA	1 (Kredi Kartına Sahip )

Örneğin bu örnekte kişinin kredi kartı var fakat harcaması **NA** olarak gözükmüyor. Bu durum o verinin eksik veya boş olduğu anlamına gelmez, kişinin o ay hiç harcama yapmadığı anlamına da gelebilir. Bu yüzden o veri 0 olarak kabul edilmelidir. Yani **NA** her zaman eksiklik değildir.

- 3. Bilgi kaybı!

	Değişken1	Değişken2	.	.	.	Değişken100
Müşteri1	.	.	.	.	.	NA
Müşteri2	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
Müşteri-n	.	.	.	.	.	.

100 değişkenli bir veri seti için sadece 1 tane eksik verinin olması o satırı komple silmemiz gerektiği anlamına gelmez. Eğer bu yapılrsa çok büyük oranda bilgi kaybına sebep olur. Bu durumda doldurma yapmak mantıklı olacaktır.

#### Eksik Veri Türleri

### Eksik Veri Türleri Nelerdir?

- **Tümüyle Raslantısal Kayıp:** Diğer değişkenlerden ya da yapısal bir problemden kaynaklanmayan tamamen rastgele oluşan gözlemler.
- **Raslantısal Kayıp:** Diğer değişkenlere bağlı olarak oluşabilen eksiklik türü.
- **Raslantısal Olmayan Kayıp:** Göz ardı edilemeyecek olan ve yapısal problemler ile ortaya çıkan eksiklik türü.

Bu türlerin bilinmesi çok önemlidir. Çünkü bu türlere göre gerekli silme veya doldurma yöntemleri uygulanır. En çok sevilen tür **tümüyle rastlantısal kayıptır**. Çünkü bu durumda eksik veri gönül rahatlığıyla silinebilir veya ortalama ile doldurulabilir.

#### Eksik Veri Rassallığının Testi

- Görsel Teknikler
- Bağımsız iki örneklem t testi
- Korelasyon testi
- Little'nin MCAR testi

En çok kullanılan testler **görsel teknikler** ve **Little'in MCAR testidir**. Bu eğitimde daha çok görsel teknikler ile test edeceğiz.

"Atama fikri hem çekici hem de tehlikelidir." -R.J.A. Little & D.B. Rubin

### Eksik Veri Problemi Nasıl Giderilir?

- Silme Yöntemleri
  - Gözlem ya da değişken silme yöntemi
  - Liste bazında silme yöntemi (Listwise Method)
  - Çiftler bazında silme yöntemi (Pairwise Method)
- Değer Atama Yöntemleri
  - Ortanca, ortalama, medyan
  - En Benzer Birime Atama (hot deck)
  - Dış Kaynaklı Atama
- Tahmine Dayalı Yöntemler
  - Makine Öğrenmesi
  - EM
  - Çoklu Atama Yöntemi

### Eksik Veri Hızlı Çözüm

Eksik veri analizi uzun bir analiz olduğundan bu analize ihtiyaç duymayan veri setleri ve hızlı bir çözüm isteyenler için bu çözüm yardımcı olacaktır. Fakat bu şekilde yapmanın sorunlara yol açacağını söylemekte fayda var.

```
In [59]: V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
df = pd.DataFrame({
    "V1" : V1,
    "V2" : V2,
    "V3" : V3
})
df
```

```
Out[59]:   V1    V2    V3
0    1.0   7.0  NaN
1    3.0  NaN  12.0
2    6.0   5.0   5.0
3    NaN   8.0   6.0
4    7.0  12.0  14.0
5    1.0  NaN   7.0
6    NaN  NaN  NaN
7    9.0   2.0   2.0
8   15.0   3.0  31.0
```

Eksik değerlerin değişken bazında toplam sayısına bakalım:

```
In [60]: df.isnull().sum()
```

```
Out[60]: V1    2
          V2    3
          V3    2
          dtype: int64
```

Eksik değer olmayanların sayısına bakalım:

```
In [61]: df.notnull().sum()
```

```
Out[61]: V1    7
          V2    6
          V3    7
          dtype: int64
```

Veri setindeki **toplam eksik değerlere** bakalım:

In [62]: `df.isnull().sum().sum()`

Out[62]: 7

In [63]: `df.isnull()`

Out[63]: **V1 V2 V3**

<b>0</b>	False	False	True
<b>1</b>	False	True	False
<b>2</b>	False	False	False
<b>3</b>	True	False	False
<b>4</b>	False	False	False
<b>5</b>	False	True	False
<b>6</b>	True	True	True
<b>7</b>	False	False	False
<b>8</b>	False	False	False

En az 1 tane **NaN** olan satırları getirelim. Bunun için köşeli parantez içerisinde `any()` fonksiyonunu da kullanmamız gerekiyor:

In [64]: `df[df.isnull().any(axis = 1)] #axis = 1, sütun için bak demektir.`

Out[64]: **V1 V2 V3**

<b>0</b>	1.0	7.0	NaN
<b>1</b>	3.0	NaN	12.0
<b>3</b>	NaN	8.0	6.0
<b>5</b>	1.0	NaN	7.0
<b>6</b>	NaN	NaN	NaN

Şimdi hiç **NaN** değeri olmayan satırları getirelim:

```
In [65]: df[df.notnull().all(axis = 1)]
```

```
Out[65]:    V1    V2    V3
```

<b>2</b>	6.0	5.0	5.0
<b>4</b>	7.0	12.0	14.0
<b>7</b>	9.0	2.0	2.0
<b>8</b>	15.0	3.0	31.0

Başa bir yöntemle yapmak istersek:

```
In [66]: df[df["V1"].notnull() & df["V2"].notnull() & df["V3"].notnull()] #all() fonksiyonunun kod hali
```

```
Out[66]:    V1    V2    V3
```

<b>2</b>	6.0	5.0	5.0
<b>4</b>	7.0	12.0	14.0
<b>7</b>	9.0	2.0	2.0
<b>8</b>	15.0	3.0	31.0

Şimdi bu problemi giderelim:

### Eksik Gözlemlerin Direkt Silinmesi

```
In [67]: df.dropna()
```

```
Out[67]:    V1    V2    V3
```

<b>2</b>	6.0	5.0	5.0
<b>4</b>	7.0	12.0	14.0
<b>7</b>	9.0	2.0	2.0
<b>8</b>	15.0	3.0	31.0

Göründüğü üzere `dropna()` fonksiyonu bir tane bile **NaN** değeri olan bütün satırları sildi. Fakat bunu kalıcı olarak yapmadı. `df` değişkenimiz hala normal.

```
In [68]: df
```

Out[68]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Kalıcı silme yapmak için:

In [69]:

```
df.dropna(inplace = True)
```

In [70]:

```
df
```

Out[70]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Göründüğü üzere kalıcı olarak silindi.

Başa örnek için df'yi tekrar normal hale getirelim:

In [71]:

```
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
df = pd.DataFrame({
    "V1" : V1,
    "V2" : V2,
    "V3" : V3})
```

```
)
```

```
df
```

```
Out[71]:
```

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

## Basit Değer Atama

Eksik değerleri her bir değişkenin ortalamasını atayalım:

```
In [72]: df["V1"]
```

```
Out[72]: 0    1.0
1    3.0
2    6.0
3    NaN
4    7.0
5    1.0
6    NaN
7    9.0
8   15.0
Name: V1, dtype: float64
```

**V1** değişkeninin eksik değerlerine ortalama atayalım:

```
In [73]: df["V1"].mean()
```

```
Out[73]: 6.0
```

```
In [74]: df["V1"].fillna(df["V1"].mean())
```

```
Out[74]: 0    1.0
1    3.0
2    6.0
3    6.0
4    7.0
5    1.0
6    6.0
7    9.0
8   15.0
Name: V1, dtype: float64
```

**V2** değişkeninin eksik değerlerine 0 atayalım:

```
In [75]: df["V2"].fillna(0)
```

```
Out[75]: 0    7.0
1    0.0
2    5.0
3    8.0
4   12.0
5    0.0
6    0.0
7    2.0
8    3.0
Name: V2, dtype: float64
```

Göründüğü üzere eksik değerler dolduruldu.

Birden fazla değişken için bu işlemleri tek tek yapmak zor olacaktır. Bunu otomatik gerçekleştirmek için `apply()` ve `lambda` mimarilerini kullanmalıyız. Çünkü `apply()` fonksiyonu DataFrame'in tüm sütunlardaki değişkenlere tek seferde değişiklik yapabilmemizi sağlar.

`df`'yi yeniden eski haline getirelim:

```
In [76]: V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
df = pd.DataFrame({
    "V1" : V1,
    "V2" : V2,
    "V3" : V3
})
df
```

```
Out[76]:   V1    V2    V3
0    1.0    7.0   NaN
1    3.0   NaN   12.0
```

	V1	V2	V3
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

```
In [77]: df.apply(lambda x: x.fillna(x.mean()), axis = 0)
```

```
Out[77]:
```

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

Göründüğü üzere `lambda` fonksiyonunu kullanarak eksik değerleri ortalama ile doldurmuş olduk.

## Eksik Veri Yapısının Görselleştirilmesi

Eksik verilerin rastgele oluşup olmadığını incelemenin en iyi yolu onları görselleştirmektir. Bunu yapmak için `missingno` kütüphanesini indirmemiz gerekiyor.

```
In [78]: !pip install missingno
```

```
Requirement already satisfied: missingno in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (0.5.0)
Requirement already satisfied: scipy in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from missingno) (1.7.1)
```

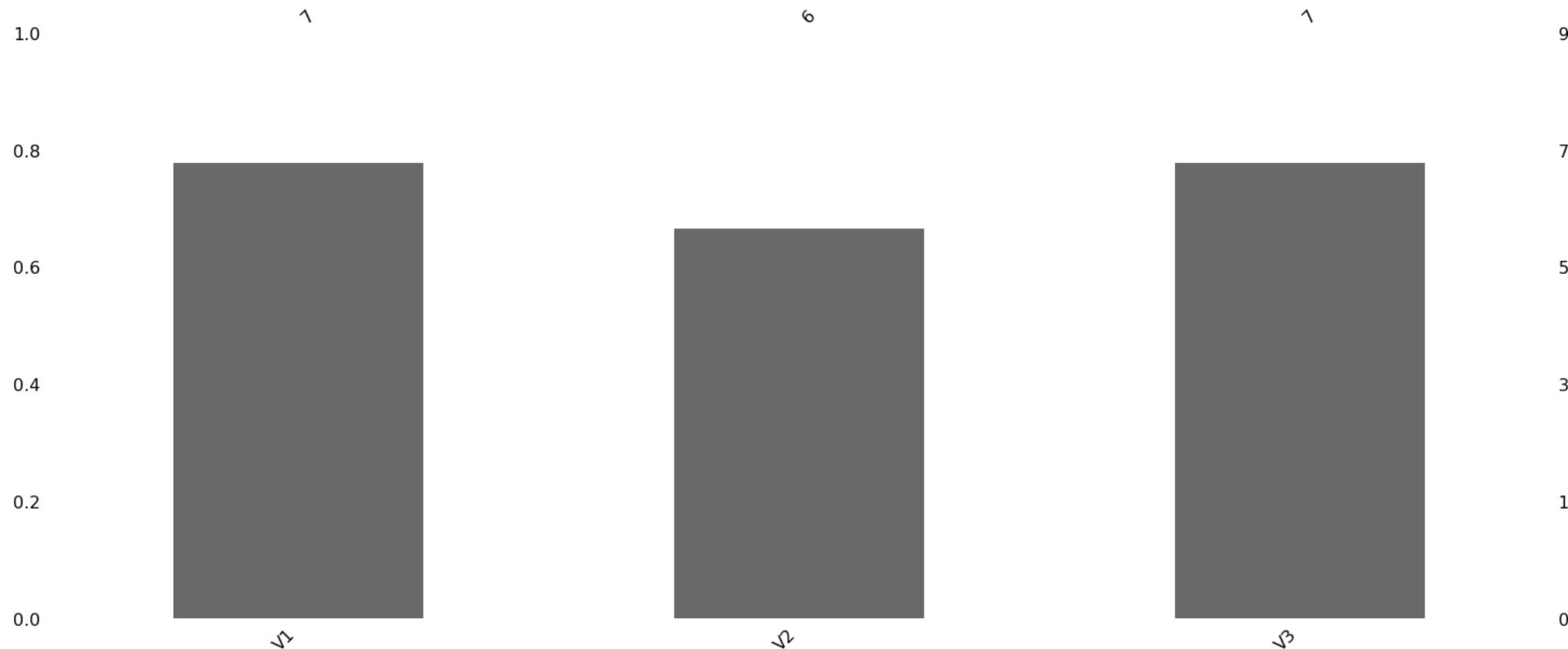
```
Requirement already satisfied: numpy in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from missingno) (1.19.2)
Requirement already satisfied: matplotlib in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from missingno) (3.3.2)
Requirement already satisfied: seaborn in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from missingno) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from matplotlib->missingno) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from matplotlib->missingno) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from matplotlib->missingno) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!!=2.1.2,!!=2.1.6,>=2.0.3 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from matplotlib->missingno) (8.0.1)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from matplotlib->missingno) (2021.1.0.8)
Requirement already satisfied: six in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from cycler>=0.10->matplotlib->missingno) (1.15.0)
Requirement already satisfied: pandas>=0.23 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from seaborn->missingno) (1.1.5)
Requirement already satisfied: pytz>=2017.2 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from pandas>=0.23->seaborn->missingno) (2020.4)
```

In [79]:

```
import missingno as msno
```

In [80]:

```
msno.bar(df);
```

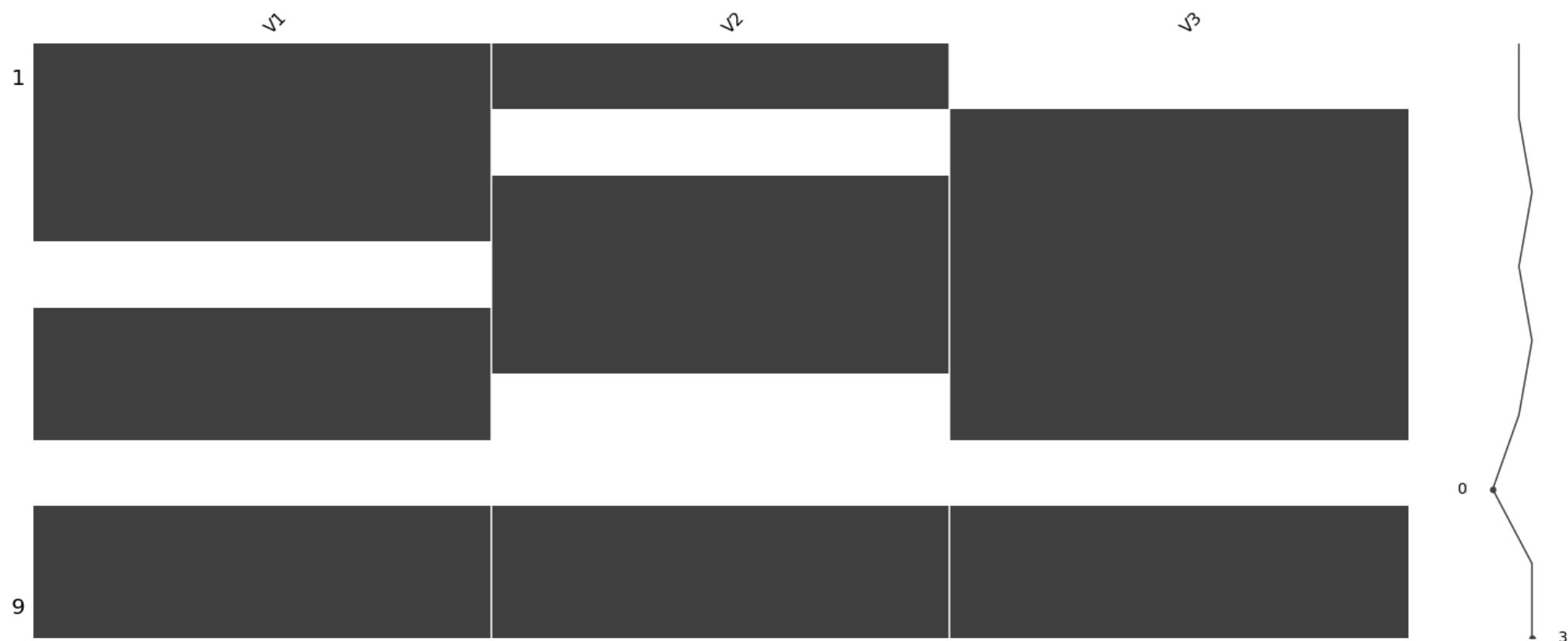


Grafiği inceleyeceğimizde eksik olan verilerin sayısını yukarıda bizlere vermiş oldu. Sağ taraftaki sayılar veri sayısını ifade ederken (max = 9 tane gözlem var), sol taraf yüzdeliği ifade etmektedir.

Başa bir yöntemle inceleyelim:

In [81]:

```
msno.matrix(df);
```



Bu grafikte hangi değerlerin dolu ve boş olduğu bilgisini bizlere veriyor. Sol taraf veri sayısını, sağ taraf ise her satırda kaç tane dolu değer var onu temsil ediyor (Aynı anda 0 ve 3 dolu değer olduğu için onları gösteriyor).

Örneğin;

1.satır için  $V_1$  ve  $V_2$ 'nin verisi dolu;  $V_3$ 'ün verisi boş,

6.satır tamamen boş şeklinde yorum yapabiliyoruz.

In [82]:

```
df
```

Out[82]:

	$V_1$	$V_2$	$V_3$
1	1	1	0
2	0	3	0
3	0	3	0
4	0	3	0
5	0	3	0
6	0	3	0
7	0	3	0
8	0	3	0
9	0	3	0

```
V1    V2    V3
0    1.0    7.0   NaN
1    3.0   NaN  12.0
2    6.0    5.0   5.0
3    NaN    8.0   6.0
4    7.0  12.0  14.0
5    1.0   NaN   7.0
6    NaN   NaN   NaN
7    9.0    2.0   2.0
8   15.0    3.0  31.0
```

df'ye bakarak bunun sağlamasını yapabiliriz.

Şimdi `seaborn` kütüphanesindeki başka bir veri seti için eksik değer analizi yapalım:

In [83]:

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

Out[83]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

In [84]:

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

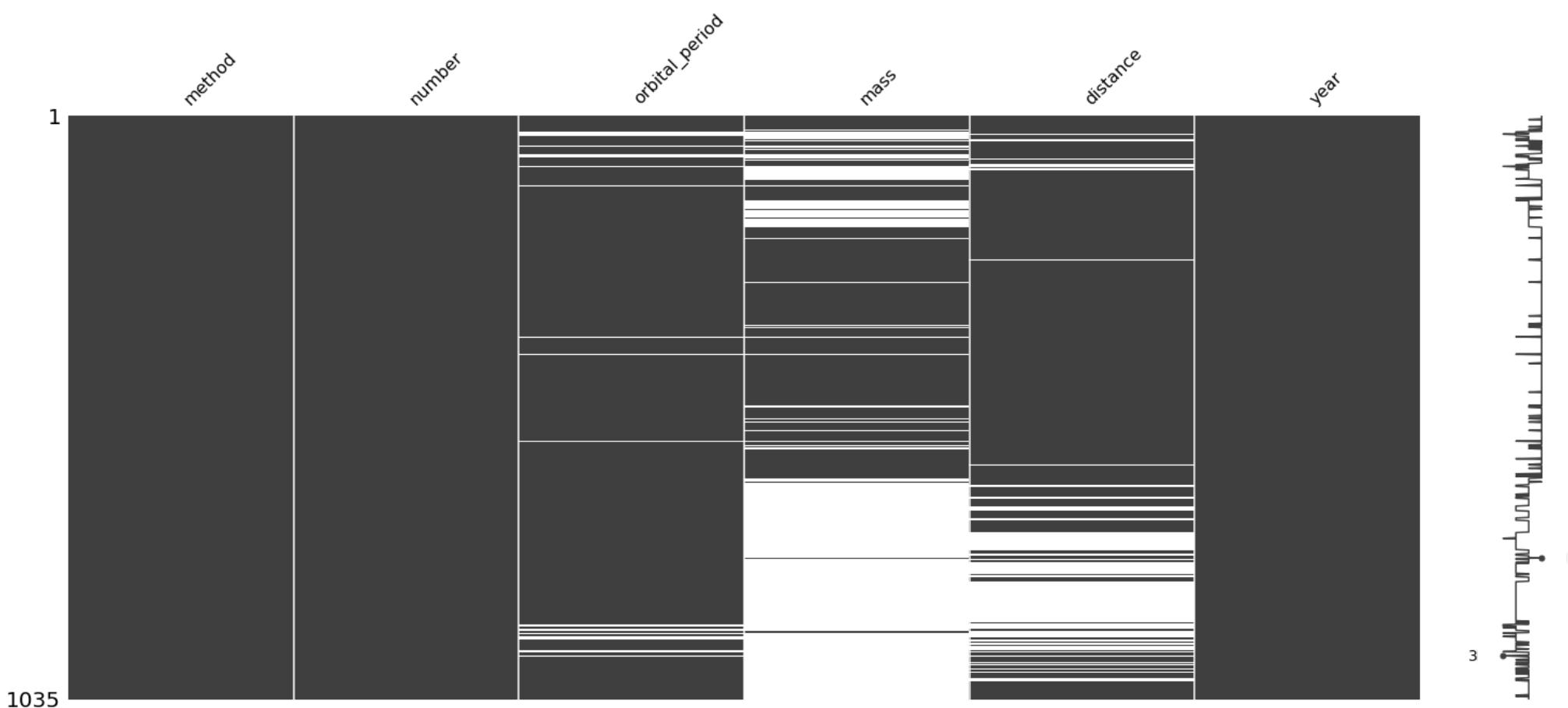
Out[84]:

```
method          0
number          0
orbital_period  43
mass           522
distance        227
year            0
dtype: int64
```

Bu eksik değerleri anlatıldığı üzere direkt silmek ve doldurmadan önce rassallığını incelemeliyiz:

In [85]:

```
msno.matrix(df);
```



Bu grafikte dikkat edilecek olursa **orbital\_period** değişkeninde ne zaman boş bir değer varsa genellikle **mass** değişkeninde de boş bir değer olduğu gözlemleniyor. Kredi kartı örneğini hatırlarsak kredi kartı olmayan kişilerin harcama yapması söz konusu değildir. Yani harcama değeri otomatik olarak boş olmalıdır. Dolayısıyla burada bu değişkenler arasında bir bağımlılık durumu söz konusudur. Aynı durum **mass** ve **distance** değişkenleri için de geçerlidir. Bu durumda **bu eksik değerler rastgele oluşmamıştır.** yorumu yapabiliriz.

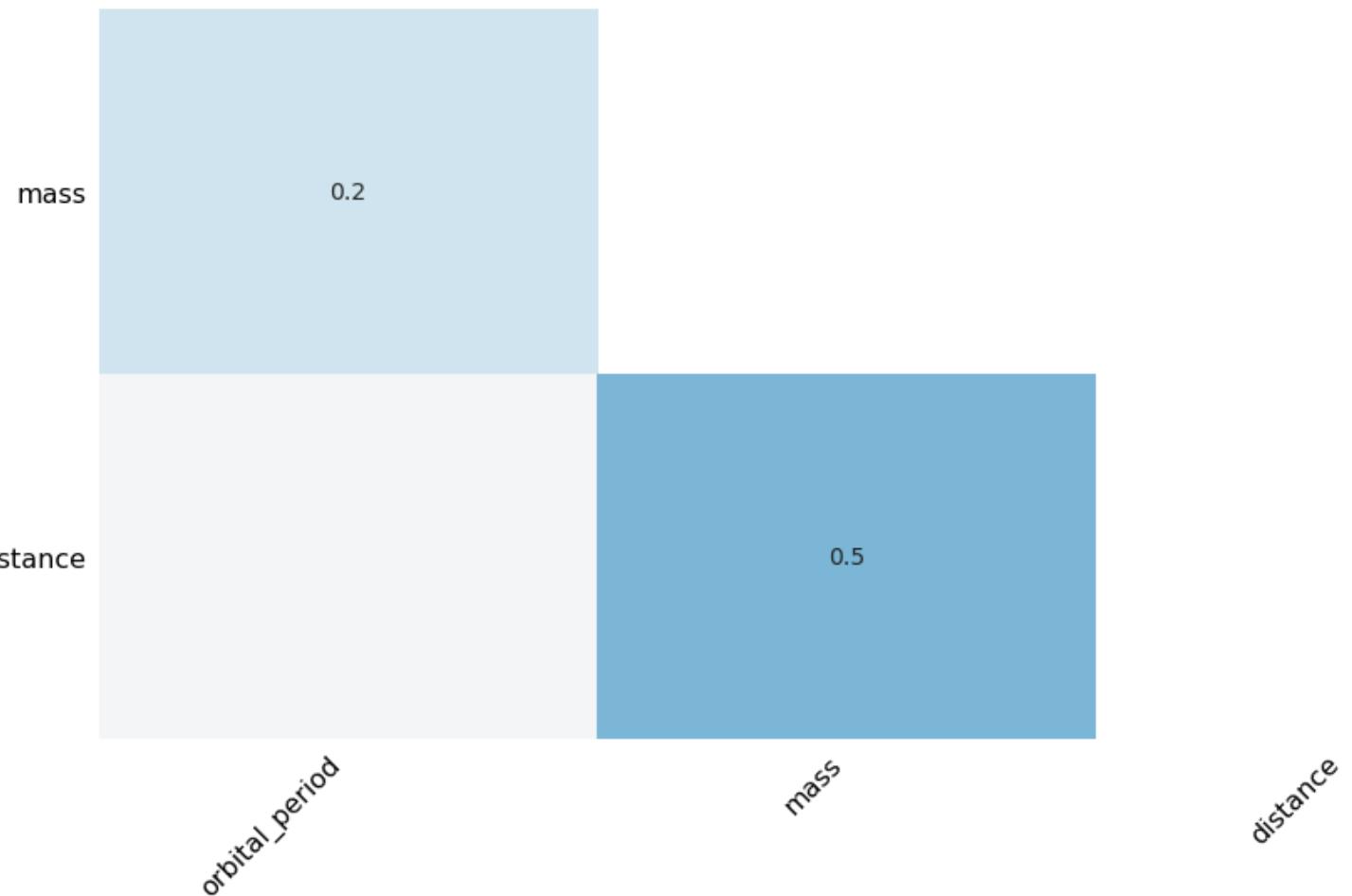
Şimdi bu değişkenler için ısı haritasını (heatmap) çizdirelim. Bu ısı haritası değişkenler arasındaki eksik değerlerin oluşma durumu için bir korelasyon grafiği (nullity correlation: boşluk korelasyonu) çizdiriyor. Örneğin bir değişkende eksik değerler varsa diğer değişkenlerde oluşup oluşmadığının korelasyonunu bizlere veriyor. Yani aslında eksik değerlerin rassallığı ile ilgili bilgi de bu grafik sayesinde anlayabiliyoruz.

Grafiği çizecek olursak:

In [86]:

```
msno.heatmap(df);
```

orbital\_period



Grafik incelendiğinde **mass** ile **distance** değişkenleri arasında 0.5 değerinde bir korelasyon olduğunu görüyoruz. Yani değişkenin birinde bir eksik değer varsa orta şiddette diğer değişkende de vardır diyebiliriz.

Burada da aynı şekilde -1 ve +1 arasında değerler alır. +1, birinde eksik değer varsa diğerinde de yüksek ihtimalle var demek iken; -1, birinde eksik değer varsa diğerinde yoktur demektir. 0 ise ilişki yoktur anlamına gelir.

Sonuç olarak **planets** veri seti için "eksik değerler rastgele oluşmamıştır, direkt silme veya doldurma işlemi yapılmamalıdır" yorumu yapılabilir.

## Silme Yöntemleri

In [87]:

```
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
df = pd.DataFrame({
    "V1" : V1,
    "V2" : V2,
    "V3" : V3
})
df
```

Out[87]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

### 1. En Az 1 Eksik Değeri Sahip Satırın Silinmesi

In [88]:

```
df.dropna()
```

Out[88]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

In [89]:

```
df
```

Out[89]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

## 2. Bütün Satırı NaN olan Satırın Silinmesi

In [90]:

```
#Bunun için how = "all" argümanı kullanılır.
```

```
df.dropna(how = "all")
```

Out[90]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Göründüğü üzere 6. satır silinmiş oldu. Bunu sütun için yapsaydık axis = 1 demeliydik.

### 3. En Az 1 Eksik Değere Sahip Sütunun Silinmesi

In [91]:

```
df.dropna(axis = 1)
```

Out[91]:

```
0  
1  
2  
3  
4  
5  
6  
7  
8
```

Göründüğü üzere tüm sütunlarda en az 1 tane `NaN` olduğu için tüm sütunları sildi.

### 4. Tüm Sütunu `NaN` olan Sütunun Silinmesi

In [92]:

```
df.dropna(axis = 1, how = "all")
```

Out[92]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Tüm değerleri `NaN` olan bir sütun olmadığından dolayı hiçbir değişkeni silmedi. Örnek olması açısından tüm değerleri `NaN` olan bir değişken ekleyelim:

In [93]: `df["sil_beni"] = np.nan`

In [94]: `df`

Out[94]:

	V1	V2	V3	sil_beni
0	1.0	7.0	NaN	NaN
1	3.0	NaN	12.0	NaN
2	6.0	5.0	5.0	NaN
3	NaN	8.0	6.0	NaN
4	7.0	12.0	14.0	NaN
5	1.0	NaN	7.0	NaN
6	NaN	NaN	NaN	NaN
7	9.0	2.0	2.0	NaN
8	15.0	3.0	31.0	NaN

In [95]: `df.dropna(axis = 1, how = "all", inplace = True)`

In [96]: `df`

Out[96]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN

	V1	V2	V3
7	9.0	2.0	2.0
8	15.0	3.0	31.0

## Basit Değer Atama Yöntemleri

In [97]:

```
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
df = pd.DataFrame({
    "V1" : V1,
    "V2" : V2,
    "V3" : V3
})
df
```

Out[97]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

## Sayısal Değişkenlerde Atama İşlemi

In [98]:

```
df["V1"].fillna(0)
```

Out[98]:

0	1.0
1	3.0
2	6.0
3	0.0

```
4    7.0
5    1.0
6    0.0
7    9.0
8   15.0
Name: V1, dtype: float64
```

In [99]:

```
df
```

Out[99]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

In [100...]

```
df["V1"].fillna(df["V1"].mean())
```

Out[100...]

```
0    1.0
1    3.0
2    6.0
3    6.0
4    7.0
5    1.0
6    6.0
7    9.0
8   15.0
Name: V1, dtype: float64
```

## Tüm Değişkenler için 1. yol

In [101...]

```
df.apply(lambda x: x.fillna(x.mean()), axis = 0)
```

Out[101...]

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

## Tüm Değişkenler için 2. Yol

In [102...]

```
df.fillna(df.mean()[::])  
#df.fillna(df.mean())
```

Out[102...]

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

Tüm değişkenler normal dağılığında eksik değerleri ortalamaya atamak mantıklıdır. Fakat bazı değişkenler normal dağılmıyorsa (sağa çarpık veya sola çarpık) ortalamaya atanmamalıdır.

Bu yüzden bazı değişkenlerin eksik değerlerini farklı değerlerle atamak istediğimizi düşünelim.

```
In [103...]: df.fillna(df.mean()["V1":"V2"])
```

```
Out[103...]:
```

	V1	V2	V3
0	1.0	7.000000	NaN
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	NaN
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

V3 değişkeninin eksik değerlerini V3'ün medyan değerini atayalım:

```
In [104...]: df["V3"].fillna(df["V3"].median())
```

```
Out[104...]:
```

0	7.0
1	12.0
2	5.0
3	6.0
4	14.0
5	7.0
6	7.0
7	2.0
8	31.0

Name: V3, dtype: float64

### Tüm Değişkenler için 3. Yol

```
In [105...]: df.where(pd.notna(df), df.mean(), axis = "columns")
```

```
Out[105...]:
```

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0

	V1	V2	V3
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

## Kategorik Değişken Kırılımında Değer Atama

Örneğin bir şirketin maaşları ile ilgili bir veri setinde maaş değişkeninde bazı değerler boş olmuş olsun. Bunu direkt ortalama ile atamak bazı durumlarda mantıklı olmayacağıdır. Çünkü örneğin maaşı boş olan kişi alt skalada veya üst skalada bir departmanda ise genel kitleye göre ortalama atamak doğru değildir, **çalıştığı departmanın maaş ortalamasına göre boş değer doldurulmalıdır**. Bu yüzden böyle durumlarda diğer kategorik değişkenleri inceleyip ona göre atama işlemi yapmak daha mantıklıdır.

Örnek üzerinden bunu inceleyelim:

```
In [106...]: V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
V4 = np.array(["IT","IT","IK","IK","IK","IK","IT","IT"])
df = pd.DataFrame({
    "maas" : V1,
    "V2" : V2,
    "V3" : V3,
    "departman" : V4
})
df
```

Out[106...]:

	maas	V2	V3	departman
0	1.0	7.0	NaN	IT
1	3.0	NaN	12.0	IT
2	6.0	5.0	5.0	IK
3	NaN	8.0	6.0	IK

	maas	V2	V3	departman
4	7.0	12.0	14.0	IK
5	1.0	NaN	7.0	IK
6	NaN	NaN	NaN	IK
7	9.0	2.0	2.0	IT
8	15.0	3.0	31.0	IT

```
In [107...]: df.groupby("departman")["maas"].mean()
```

```
Out[107...]: departman
IK      4.666667
IT      7.000000
Name: maas, dtype: float64
```

Şimdi maaş değişkenindeki boş değerleri departman bazında ortalamalarına göre dolduralım:

```
In [108...]: df["maas"].fillna(df.groupby("departman")["maas"].transform("mean"))
```

```
Out[108...]: 0      1.000000
1      3.000000
2      6.000000
3      4.666667
4      7.000000
5      1.000000
6      4.666667
7      9.000000
8     15.000000
Name: maas, dtype: float64
```

Göründüğü üzere departman bazında eksik değerler dolmuş oldu. `transform()` fonksiyonunu yazmamızın sebebi eksik değerleri ortalama ile "dönüşürme" yapmamız gerektiği içindir. Zaten `fillna()` içerisinde `df.groupby("departman")["maas"].mean()` şeklinde yazınca bir değişiklik olmayacağından emin olursak:

```
In [109...]: df["maas"].fillna(df.groupby("departman")["maas"].mean())
```

```
Out[109...]: 0      1.0
1      3.0
2      6.0
3      NaN
4      7.0
5      1.0
6      NaN
```

```
7    9.0
8   15.0
Name: maas, dtype: float64
```

Görüldüğü üzere bir değişiklik olmadı. Çünkü `df.groupby("departman")["maas"].mean()` ifadesi bir dönüştürme işlemi değil, sadece gösterme kodudur. Bu yüzden `transform()` fonksiyonu kullanılmalıdır.

## Kategorik Değişkenlerde Değer Atama

Kategorik değişkenler için değer atamanın en mantıklı ve verimli yolu mod işlemidir. Yani değişkende en sık tekrar eden kategori ne ise o kategori ile doldurulur.

In [110...]

```
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V4 = np.array(["IT",np.nan,"IK","IK","IK","IK","IK","IT","IT"], dtype=object)
df = pd.DataFrame({
    "maas" : V1,
    "departman" : V4
})
df
```

Out[110...]

	maas	departman
0	1.0	IT
1	3.0	NaN
2	6.0	IK
3	NaN	IK
4	7.0	IK
5	1.0	IK
6	NaN	IK
7	9.0	IT
8	15.0	IT

Departman değişkeninin modunu alalım:

In [111...]

```
df["departman"].mode()
```

Out[111...]

```
0    IK
dtype: object
```

0 indexini al dersek:

```
In [112... df["departman"].mode()[0]
```

```
Out[112... 'IK'
```

Bizlere "IK" sonucunu döndürür. Şimdi bu kodumuzu `fillna()` fonksiyonuna yazalım:

```
In [113... df["departman"].fillna(df["departman"].mode()[0])
```

```
Out[113... 0    IT  
1    IK  
2    IK  
3    IK  
4    IK  
5    IK  
6    IK  
7    IT  
8    IT  
Name: departman, dtype: object
```

Görüldüğü üzere eksik değer değişkenin modu olan "IK" ile dolmuş oldu. Kalıcı yapmak için ise `inplace` argümanı kullanılabilir.

Modunu almak yerine bir önceki veya bir sonraki kategori ile doldurmak isteyebiliriz. Şimdi bunu yapalım:

Sonraki kategoriyle doldurmak için:

```
In [114... df["departman"].fillna(method = "bfill")
```

```
Out[114... 0    IT  
1    IK  
2    IK  
3    IK  
4    IK  
5    IK  
6    IK  
7    IT  
8    IT  
Name: departman, dtype: object
```

Önceki değerle doldurmak için:

```
In [115... df["departman"].fillna(method = "ffill")
```

```
Out[115... 0    IT  
1    IT  
2    IK  
3    IK  
4    IK
```

```
5    IK  
6    IK  
7    IT  
8    IT  
Name: departman, dtype: object
```

Göründüğü üzere `method` argümanını kullanarak önceki ve sonraki kategoriyle doldurmuş olduk.

## Tahmine Dayalı Değer Atama Yöntemleri

İleri bir teknik olan bu yöntem, makine öğrenmesi algoritmalarını kullanarak eksik değerler doldurulabilmektedir.

In [116...]

```
import missingno as msno  
df = sns.load_dataset("titanic")  
df = df.select_dtypes(include = ["float64", "int64"])  
print(df.head())  
df.isnull().sum()
```

	survived	pclass	age	sibsp	parch	fare
0	0	3	22.0	1	0	7.2500
1	1	1	38.0	1	0	71.2833
2	1	3	26.0	0	0	7.9250
3	1	1	35.0	1	0	53.1000
4	0	3	35.0	0	0	8.0500

Out[116...]

```
survived      0  
pclass        0  
age         177  
sibsp        0  
parch        0  
fare         0  
dtype: int64
```

`age` değişkeninde 177 adet eksik değerlerin olduğunu görmekteyiz.

Bu doldurma işlemini yapmak için `ycimpute` isminde bir kütüphane indirmeliyiz.

In [117...]

```
!pip install ycimpute
```

```
Requirement already satisfied: ycimpute in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (0.1.1)  
Requirement already satisfied: six in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from ycimpute) (1.15.0)  
Requirement already satisfied: scipy in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from ycimpute) (1.7.1)  
Requirement already satisfied: numpy>=1.10 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from ycimpute) (1.19.2)  
Requirement already satisfied: scikit-learn>=0.17.1 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from ycimpute) (0.23.2)  
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from scikit-learn>=0.17.1->ycimpute) (2.1.0)  
Requirement already satisfied: joblib>=0.11 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from scikit-learn>=0.17.1->ycimpute) (1.0.0)
```

3 makine öğrenmesi algoritmasını kullanarak doldurma işlemini ele alacağız.

1. KNN (k en yakın komşu algoritması)
2. Random Forests
3. EM (Expectation-Maximization)

## 1. KNN

```
In [118...]: from ycimpute.imputer import knnimput
```

`knnimput` modülü bizden bir numpy array'i beklemektedir. Bu yüzden DataFrame'mimizi numpy'a çevirmeliyiz.

Öncelikle df'mizin değişken isimlerini bir saklayalım:

```
In [119...]: var_names = list(df)
var_names
```

```
Out[119...]: ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
```

Sonra df'mizi numpy array'ine çevirelim:

```
In [120...]: n_df = np.array(df)
```

```
In [121...]: n_df[0:10]
```

```
Out[121...]: array([[ 0.    ,  3.    , 22.    ,  1.    ,  0.    ,  7.25  ],
       [ 1.    ,  1.    , 38.    ,  1.    ,  0.    , 71.2833],
       [ 1.    ,  3.    , 26.    ,  0.    ,  0.    ,  7.925 ],
       [ 1.    ,  1.    , 35.    ,  1.    ,  0.    ,  53.1   ],
       [ 0.    ,  3.    , 35.    ,  0.    ,  0.    ,  8.05   ],
       [ 0.    ,  3.    ,  nan   ,  0.    ,  0.    ,  8.4583],
       [ 0.    ,  1.    , 54.    ,  0.    ,  0.    , 51.8625],
       [ 0.    ,  3.    ,  2.    ,  3.    ,  1.    , 21.075  ],
       [ 1.    ,  3.    , 27.    ,  0.    ,  2.    , 11.1333],
       [ 1.    ,  2.    , 14.    ,  1.    ,  0.    , 30.0708]])
```

```
In [122...]: n_df.shape
```

```
Out[122...]: (891, 6)
```

Şimdi algoritmayı kullanarak doldurma işlemini yapalım:

```
In [123...]: dff = knnimput.KNN(k = 4).complete(n_df)
```

```
#k: Komşu sayısı
```

```
Imputing row 1/891 with 0 missing, elapsed time: 0.297
Imputing row 101/891 with 0 missing, elapsed time: 0.300
Imputing row 201/891 with 0 missing, elapsed time: 0.302
Imputing row 301/891 with 1 missing, elapsed time: 0.303
Imputing row 401/891 with 0 missing, elapsed time: 0.305
Imputing row 501/891 with 0 missing, elapsed time: 0.307
Imputing row 601/891 with 0 missing, elapsed time: 0.309
Imputing row 701/891 with 0 missing, elapsed time: 0.311
Imputing row 801/891 with 0 missing, elapsed time: 0.312
```

Doldurma işlemini gerçekleştirdik. `dff` değişkenimiz şu an numpy arrayi olduğundan onu dataframe'e çevirelim:

```
In [124...]
```

```
type(dff)
```

```
Out[124...]: numpy.ndarray
```

```
In [125...]
```

```
dff = pd.DataFrame(dff, columns = var_names)
```

```
In [126...]
```

```
type(dff)
```

```
Out[126...]: pandas.core.frame.DataFrame
```

```
In [127...]
```

```
dff.isnull().sum()
```

```
Out[127...]: survived      0
pclass          0
age            0
sibsp          0
parch          0
fare           0
dtype: int64
```

Göründüğü üzere boş değer kalmadı. Herhangi bir görselleştirme işlemi yapmadan doldurma işlemini gerçekleştirdik.

Şimdi bunu **Random Forests** algoritması için yapalım:

## 2. Random Forests

```
In [128...]
```

```
import missingno as msno
df = sns.load_dataset("titanic")
df = df.select_dtypes(include = ["float64", "int64"])
```

```
In [129... df.isnull().sum()
```

```
Out[129... survived      0
         pclass       0
         age        177
         sibsp       0
         parch       0
         fare        0
         dtype: int64
```

```
In [130... var_names = list(df)
         var_names
```

```
Out[130... ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
```

```
In [131... n_df = np.array(df)
```

```
In [132... from ycimpute.imputer import IterForest
         dff = IterForest.IterImputer().complete(n_df)
```

```
In [133... dff = pd.DataFrame(dff, columns = var_names)
```

```
In [134... dff.isnull().sum()
```

```
Out[134... survived      0
         pclass       0
         age        177
         sibsp       0
         parch       0
         fare        0
         dtype: int64
```

Şimdi EM algoritması ile bunu yapalım:

### 3. EM (Expectation-Maximization)

```
In [135... import missingno as msno
         df = sns.load_dataset("titanic")
         df = df.select_dtypes(include = ["float64", "int64"])
```

```
In [136... from ycimpute.imputer import EM
```

```
In [137...]  
var_names = list(df)  
var_names
```

```
Out[137...]: ['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
```

```
In [138...]  
n_df = np.array(df)
```

```
In [139...]  
dff = EM().complete(n_df)
```

```
In [140...]  
dff = pd.DataFrame(dff, columns = var_names)
```

```
In [141...]  
dff.isnull().sum()
```

```
Out[141...]:  
survived    0  
pclass      0  
age         0  
sibsp       0  
parch       0  
fare        0  
dtype: int64
```

Göründüğü üzere çeşitli algoritmalar ile doldurma işlemini gerçekleştirdik. Öneri olarak bunları yapmadan önce eksik değerlerin rastgele mi olduğu yoksa başka değişkenlere bağlı olarak mı eksik olduğu mutlaka incelenmelidir.

## Değişken Dönüşümleri

### Değişken Standardizasyonu (Veri Standardizasyonu)

Değişken dönüşümünden farklı olarak bu yöntem, değişkenin kendi içinde varyans/bilgi yapısı bozulmaz. Fakat belirli bir standarda oturtulur.

Değişken dönüşümünü ise değişkenin taşıdığı bilgiyle kalamamasıdır. (örn: kadın-erkek yerine 0-1 yazmak.) Aslında özünde bilgi değişmez fakat yapısı değişir.

Bu dönüştürme işlemini yapmamızın sebebi veriyi göndereceğimiz fonksiyonların genelde bu şekilde istemesinden dolayıdır.

```
In [142...]  
V1 = np.array([1,3,6,5,7])  
V2 = np.array([7,7,5,8,12])  
V3 = np.array([6,12,5,6,14])  
df = pd.DataFrame({  
    "V1" : V1,
```

```
        "V2" : V2,  
        "V3" : V3,  
    })  
df = df.astype(float)  
df
```

```
Out[142...  V1   V2   V3  
0  1.0    7.0    6.0  
1  3.0    7.0   12.0  
2  6.0    5.0    5.0  
3  5.0    8.0    6.0  
4  7.0   12.0   14.0
```

## Standardizasyon

Verileri standart normal dağılıma dönüştürme işlemidir. (0 ortalamalı 1 varyanslı)

```
In [143... from sklearn import preprocessing
```

```
In [144... preprocessing.scale(df)
```

```
Out[144... array([[-1.57841037, -0.34554737, -0.70920814],  
                  [-0.64993368, -0.34554737,  0.92742603],  
                  [ 0.74278135, -1.2094158 , -0.98198051],  
                  [ 0.27854301,  0.08638684, -0.70920814],  
                  [ 1.2070197 ,  1.81412369,  1.47297076]])
```

**Not:** Makine öğrenmesi algoritmaları *genelde* farklı ölçek türlerdeki verileri çok sevmemektedir. Bu yüzden verileri standartlaştırmak gereklidir. Fakat bazı makine öğrenmesi algoritmaları için bu gerekmemektedir.

```
In [145... df
```

```
Out[145...  V1   V2   V3  
0  1.0    7.0    6.0  
1  3.0    7.0   12.0  
2  6.0    5.0    5.0  
3  5.0    8.0    6.0
```

V1	V2	V3
----	----	----

4	7.0	12.0	14.0
---	-----	------	------

Orjinal yapısı bozulmadı. Dönüşürme fonksiyonları genelde orijinal yapıyı bozmaz. ( `copy = True` veya `inplace = False` )

In [146...]

```
df.mean()
```

Out[146...]  
V1 4.4  
V2 7.8  
V3 8.6  
dtype: float64

## Normalizasyon

Değerleri 0 ile 1 arasına dönüştürür.

In [147...]

```
preprocessing.normalize(df)
```

Out[147...]  
array([[0.10783277, 0.75482941, 0.64699664],  
 [0.21107926, 0.49251828, 0.84431705],  
 [0.64699664, 0.53916387, 0.53916387],  
 [0.4472136 , 0.71554175, 0.53665631],  
 [0.35491409, 0.60842415, 0.70982818]])

## Max - Min Dönüşümü

Bizim istediğimiz 2 aralığa dönüştürmek için kullanılır.

In [148...]

```
scaler = preprocessing.MinMaxScaler(feature_range = (10,20))
```

#MinMaxScaler fonksiyonundan bir örnek oluşturduk. Bu nesne artık belirli özelliklerini olan bir dönüştürücü nesnesidir.  
#Şu anki özelliği, verileri 10-20 arasına çekmektir.

In [149...]

```
scaler.fit_transform(df)
```

#Bu nesneye beraber fit\_transform fonksiyonunu kullanarak dönüştürme işlemini gerçekleştirdik.  
#Yani belirli özellikte bir nesne oluşturduk ve bunu artık fit\_transform fonksiyonıyla df'e uygula dedik.

Out[149...]

[10.          , 12.85714286, 11.11111111],
[13.33333333, 12.85714286, 17.77777778],
[18.33333333, 10.          , 10.          ],
[16.66666667, 14.28571429, 11.11111111],
[20.          , 20.          , 20.          ]])

# Değişken Dönüşümleri

Değişken dönüşümleri genel bir başlıktır ve bütün alt başlıklar bu başlığa girmektedir. (örn: deg. dönüşümü > standardizasyon)

Burada önemli olan değişkendeki mevcut bilginin yapısının bozulup bozulmamasıdır. Standartlaştırmada aynı değerleri farklı aralıklarda gösterildiği için taşıdığı bilgi aynıdır. Fakat dönüştürmede örneğin kadın-erkek yerine 0-1 yazıldığından yapısı değişmektedir.

In [150...]

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

Out[150...]

	total_bill	tip	sex	smoker	day	time	size
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
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

## 0-1 Dönüşümü

Cinsiyet değişkenini 0-1 şeklinde dönüştürelim. Bunun için `LabelEncoder` fonksiyonunu kullanmalıyız.

In [151...]

```
from sklearn.preprocessing import LabelEncoder

lbe = LabelEncoder()
```

In [152...]

```
lbe.fit_transform(df["sex"])
```

Out[152...]

```
array([0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0])
```

Gördüğü üzere dönüştürme işlemini gerçekleştirdi. Fakat orijinal df'de bir değişiklik olmayacağındır. Çünkü bu gibi fonksiyonlar genellikle verilerin kopyasını alarak

ya da `inplace` argümanı `False` şeklinde çalışırlar.

In [153...]

df

Out[153...]

	total_bill	tip	sex	smoker	day	time	size
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
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

Bu oluşturduğumuz dönüştürmeyi yeni bir sütun olarak ekleyelim. Çünkü orijinal değişken sonradan işimize yarayabilir.

In [154...]

```
df["yeni_sex"] = lbe.fit_transform(df["sex"])
```

In [155...]

df

Out[155...]

	total_bill	tip	sex	smoker	day	time	size	yeni_sex
0	16.99	1.01	Female	No	Sun	Dinner	2	0
1	10.34	1.66	Male	No	Sun	Dinner	3	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0

	total_bill	tip	sex	smoker	day	time	size	yeni_sex
...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	1
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0
241	22.67	2.00	Male	Yes	Sat	Dinner	2	1
242	17.82	1.75	Male	No	Sat	Dinner	2	1
243	18.78	3.00	Female	No	Thur	Dinner	2	0

244 rows × 8 columns

**Not:** Dönüşürme yaptıktan sonra hangi kategorik ifadenin neye karşılık geldiği bilmemiz gerekiyor. Burada "**Female = 0, Male = 1**" şeklinde dönüşüm gerçekleştirildi.

## 1 ve Diğerleri (0) Dönüşümü

Örneğin birden fazla kategori varsa ilgilenilen değişkene 1, diğerlerine 0 vermek gibi durumlarda kullanılır.

In [156...]

```
df.head()
```

Out[156...]

	total_bill	tip	sex	smoker	day	time	size	yeni_sex
0	16.99	1.01	Female	No	Sun	Dinner	2	0
1	10.34	1.66	Male	No	Sun	Dinner	3	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0

In [157...]

```
df["yeni_day"] = np.where(df["day"].str.contains("Sun"), 1, 0)
```

#Burada "where" fonksiyonunu kullanarak bir koşul belirttik ve bu koşula göre dönüşümü gerçekleştirdi.  
#df["day"] değişkenini str olarak değerlendir ve içinde "Sun" olanları 1, diğerleri 0 ata demektir.

In [158...]

```
df
```

Out[158...]

	total_bill	tip	sex	smoker	day	time	size	yeni_sex	yeni_day
...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	1	0
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0	1
241	22.67	2.00	Male	Yes	Sat	Dinner	2	1	0
242	17.82	1.75	Male	No	Sat	Dinner	2	1	0
243	18.78	3.00	Female	No	Thur	Dinner	2	0	0

	total_bill	tip	sex	smoker	day	time	size	yeni_sex	yeni_day
0	16.99	1.01	Female	No	Sun	Dinner	2	0	1
1	10.34	1.66	Male	No	Sun	Dinner	3	1	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0	1
...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	1	0
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0	0
241	22.67	2.00	Male	Yes	Sat	Dinner	2	1	0
242	17.82	1.75	Male	No	Sat	Dinner	2	1	0
243	18.78	3.00	Female	No	Thur	Dinner	2	0	0

244 rows × 9 columns

## Çok Sınıflı Dönüşüm

Birden fazla kategori olan değişkeni kategori sayısı kadar değerle dönüştürmek için kullanılır. (örn: 4 kategori için 0,1,2,3 şeklinde dönüstürülür)

In 「159...

```
from sklearn.preprocessing import LabelEncoder  
  
lbe = LabelEncoder()
```

In [160...]

```
lbe.fit_transform(df["day"])
```

## ÇOK DİKKAT

Çok kategorili değişkenleri dönüştürdüğümüzde artık kategoriler, sayısal ifadeler olduğundan artık bu değişken normal sayısal değişken gibi 0-3 aralığında bir değişken olacak ve makine öğrenmesi algoritmaların kafası karışacaktır. Yani bu dönüştürme sonucunda kategoriler arasında sanki bir fark/bir mesafe/bir ölçüm varmış gibi davranışlıdır. Dolayısıyla 3 olan değer sanki en büyüğün gibi olur. Halbuki bizim değişkenimiz nominal değişken türünde olduğundan kategoriler arasında hiçbir fark yoktur. Böyle bir durumda **One Hot Encoder** yöntemi kullanılır.

## One Hot Dönüşümü Ve Dummy Değişken Tuzağı

Bu dönüşüm yapılırken bir dummy (kukla) değişken tuzağı karımıza çıkarıyor. Bu durumları bir ele alalım:

In [161...]

```
df.head()
```

Out[161...]

	total_bill	tip	sex	smoker	day	time	size	yeni_sex	yeni_day
0	16.99	1.01	Female	No	Sun	Dinner	2	0	1
1	10.34	1.66	Male	No	Sun	Dinner	3	1	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0	1

In [162...]

```
df_one_hot = pd.get_dummies(df, columns=["sex"], prefix=["sex"])

#prefix: öne isimlendirme
```

In [163...]

```
df_one_hot.head()
```

Out[163...]

	total_bill	tip	smoker	day	time	size	yeni_sex	yeni_day	sex_Male	sex_Female
0	16.99	1.01	No	Sun	Dinner	2	0	1	0	1
1	10.34	1.66	No	Sun	Dinner	3	1	1	1	0
2	21.01	3.50	No	Sun	Dinner	3	1	1	1	0
3	23.68	3.31	No	Sun	Dinner	2	1	1	1	0
4	24.59	3.61	No	Sun	Dinner	4	0	1	0	1

Veriler dikkatle incelendiğinde orijinal cinsiyet değişkenin gittiğini ve en sağda cinsiyet kategorisinin sayısı kadar değişken olduğunu görüyoruz. 1 olan değerler aslında kendisini ifade etmektedir (yani male=1 ise bu kişi erkektir, kadın 0'dır; female=1 ise bu kişi kadındır, erkek 0'dır). Fakat burada ikisi de aynı bilgiyi verdiği için birisi dummy değişkendir. Bu dönüşüm sonrasında oluşturulan yeni değişkenler birbiri üzerinden oluşturabiliyorsa, ya da bir değişkeni ifade eden başka bir değişken varsa buna **dummy değişken** denir. Bir değişkende kaç kategori varsa, 1 eksiği kadar dummy değişken var demektir.

Şimdi birden fazla kategorisi olan `day` değişkeni için bunu yapalım:

In [164...]

```
pd.get_dummies(df, columns=["day"], prefix=["day"])
```

Out[164...]

	total_bill	tip	sex	smoker	time	size	yeni_sex	yeni_day	day_Thur	day_Fri	day_Sat	day_Sun
0	16.99	1.01	Female	No	Dinner	2	0	1	0	0	0	1
1	10.34	1.66	Male	No	Dinner	3	1	1	0	0	0	1
2	21.01	3.50	Male	No	Dinner	3	1	1	0	0	0	1
3	23.68	3.31	Male	No	Dinner	2	1	1	0	0	0	1
4	24.59	3.61	Female	No	Dinner	4	0	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Dinner	3	1	0	0	0	1	0
240	27.18	2.00	Female	Yes	Dinner	2	0	0	0	0	1	0
241	22.67	2.00	Male	Yes	Dinner	2	1	0	0	0	1	0
242	17.82	1.75	Male	No	Dinner	2	1	0	0	0	1	0
243	18.78	3.00	Female	No	Dinner	2	0	0	1	0	0	0

244 rows × 12 columns

Bu veri incelendiğinde değişkenin kategori sayısı kadar yeni değişkenler oluşturdu.

One-Hot dönüşümünün bize faydası ise

1. Kategorilerin etkileri, makine öğrenmesi algoritmasına hissettirilebilmektedir. Yani örneğin bir değişkenin herhangi bir kategorisinin etkisi büyükse, one-hot dönüşümü ile o kategori 1 olup diğerleri 0 olacağından dolayı bu fark anlaşılacak ve makine öğrenmesinin performansı artacaktır.
2. Oluşan değişkenler birbiri üzerinden oluşamayacağından ve bir değişkeni ifade eden başka değişkenler olamayacağından dolayı çok kategorili değişkenlerde dummy değişken olmayacağındır. Yani örneğin yeni oluşan tek bir değişken incelendiğinde oluşan diğer değişkenlerle aynı bilgiyi vermemektedir, kendine özgü bilgi vermektedir. Fakat 2 kategorili kadın-erkek değişkeni için aynı bilgileri verdiklerinden dolayı birisi dummy değişkendir. Bu yüzden çok kategorili durumlarda da one-hot dönüşümü bizim için faydalıdır.

Bir dersin sonuna geldik. Teşekkür ederim. 😊

In [ ]: