İçindekiler

In [10]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Giriş

Büyük Resmi Görmek ve Veriyi Temsil Etmek

Keşifçi Veri Analizi:

- Betimsel istatistikler, veri görselleştirme teknikleri ve iş çıktısı hedefiyle veri üzerinde çalışmaktır.
- Veri bilimcinin özgürce çalışabildiği, söz konusu olmayan yeni bulgulara ulaşabileceği yeni iş fikirleri, yeni işe yarar sonuçlara ulaşabileceği, sorular sorarak, hipotezler kurarak ilerlediği süreçtir.

Python'da Veri Görselleştirme

Veri görselleştirme kütüphaneleri başlıca şunlardır:

Matplotlib:

- Python veri görselleştirme dünyasının büyük babasıdır.
- Matlab benzeri bir arayüzü vardır.
- Çok güçlüdür fakat biraz karmaşıktır.
- Low level(düşük seviye) erişim sağlar. Yani yapılmak istenen işlemi ifade etmek için daha fazla çaba harcanmalıdır.
- Diğer veri görselleştirme kütüphaneleri Matplotlib'in özelliklerini kullanır ve onun üzerine inşa edilmiştir.
- Bu diğer kütüphaneler, işleri kolaylaştırmak açısından komutlar daha basit hale indirgenmiştir.

Pandas:

- Veri analizi ve veri manipülasyon işlemlerinin dışında veri görselleştirme için de çok zengin bir kullanımı vardır.
- Matplotlib üzerine inşa edilmiştir.
- Kolay bir kullanıma sahiptir. Bu yüzden high level(yüksek seviyeli) bir yapıdır.
- Matplotlib'e göre daha az kod ile daha fazla iş yapılabilir.

Seaborn:

- Matplotlib üzerine inşa edilmiş bir veri görselleştirme kütüphanesidir.
- Pandas gibi high level bir kullanıma sahiptir.
- İstatistiksel ve bilgi taşıyan grafikleri kolay bir şekilde edinebilmemizi sağlar.

ggplot:

- Aslında R programlama dilinde yer alan bir kütüphanedir fakat Python'a uyarlanmış halidir.
- **Dünyanın en iyi veri görselleştirme kütüphanesi** olarak kabul görmektedir. Haliyle high level bir kullanım sunmaktadır.
- Kullanıcı başlangıç seviyesinde olsa bile onu ileri seviyeye taşıyacak bir yapıya sahiptir.
- R dünyasındaki başarısından dolayı Python dünyasında da kendisine yer verilmiştir.

Bokeh:

- Seaborn ve Pandas'tan farklı olarak Matplotlib üzerine inşa edilmiş bir kütüphane değildir.
- İnteraktif bir görselleştirme kütüphanesidir.
- Büyük verisetleri ve akan verisetlerini yüksek performanslı olarak interaktif şekilde sunabilmektedir. Bu interaktif grafikleri(dashboardları) ve veri odaklı uygulamaları hızlı ve kolay bir şekilde gerçekleştirebilmektedir.
- En önemli yanlarından birisi bu işlemleri modern web tarayıcılar aracılığıyla gerçekleştirmeye imkan sağlamasıdır.

Plot.ly:

- Veri görselleştirme alanına sonradan girip birçok programalama diliyle beraber çalışabilme imkanından dolayı çok güzel olanaklar sunan bir kütüphanedir.
- Yine interaktif veri görselleştirme imkanı sağlar.
- Diğer kütüphaneler arasında en profesyonel kütüphanelerden birisidir.
- Hem Python hem de R'da ve diğer bazı dillerde de kullanılabilmektedir.

Bu kursta daha çok Seaborn kütüphanesi kullanılacaktır. Bazen Pandas ve Matplotlib'ten de destek alarak görselleştirme işlemleri yapılacaktır.

Veri Seti & İlk Adımlar

Veriye İlk Bakış

Genelde veri ilk geldiği anda ne yapacağımız konusunda kararsız kalabiliriz. Burada veri geldiği anda yapacağımız ilk adımlar anlatılmaktadır.

1. Adım: Veri Setinin Hikayesi ve Yapısının İncelenmesi

In [2]:
```
planets = sns.load_dataset("planets")
planets.head()
```

Out[2]:

	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

1.1: Veri Setinin Hikayesi

Veri setimizi getirdik. Bir veri seti getirilirken ilk sorulacak soru şudur:
Veri setinin hikayesi nedir?

Veri setinin nasıl oluşturulduğu, isimlendirmelerin hangi mantıkla yapıldığı bilinmelidir. Çünkü bu sonuçlar üzerine yorumlar yapılacaktır. Yanlış yapılan herhangi bir yerde yorumlar da tutarsız olacaktır. Bu yüzden veri setinin hikayesi çok iyi bir şekilde bilinmelidir.
Yani veriye ilk bakış demek, veri setinin teorik olarak nasıl oluştuğunun sorgulanmasıdır.

Bu veri setine gelecek olursak, NASA'nın yayınladığı galaksi keşfi ile ilgili bir veri setidir.

Değişkenlere bakacak olursak:
method: Gezegenlerin, galaksilerin bulunma şeklini ifade etmektedir.
number: Bulunan sistemlerdeki gezegen sayısını ifade etmektedir.

orbital_period: Teknik bir ifadedir. Yörünge dönemini ifade eder.
mass: Teknik bir ifadedir. Formülü `m*sin` şeklindedir. Dolayısıyla kütleyi ifade eder diyebiliriz.
distance: Uzaklığı ifade eder.
year: Galaksinin bulunma yılını ifade eder.

1.2: Veri Setinin Kopyası

Veri setini anladıktan sonra ilk yapılması gereken işlem veri setinin bir kopyasını almaktır. Birçok işlem yapılacağı için orijinal veri seti korunmalıdır.

In [3]:
```
df = planets.copy()
```

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

Out[4]:

	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 [5]:
```
df.tail()
```

Out[5]:

	method	number	orbital_period	mass	distance	year
1030	Transit	1	3.941507	NaN	172.0	2006
1031	Transit	1	2.615864	NaN	148.0	2007
1032	Transit	1	3.191524	NaN	174.0	2007
1033	Transit	1	4.125083	NaN	293.0	2008
1034	Transit	1	4.187757	NaN	260.0	2008

1.3: Veri Setinin Yapısal Bilgileri

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1035 entries, 0 to 1034
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   method          1035 non-null   object 
 1   number          1035 non-null   int64  
 2   orbital_period  992 non-null    float64
 3   mass            513 non-null    float64
 4   distance        808 non-null    float64
 5   year            1035 non-null   int64  
dtypes: float64(3), int64(2), object(1)
memory usage: 48.6+ KB
```

In [7]:
```
df.dtypes
```

Out[7]:
```
method             object
number              int64
orbital_period    float64
mass              float64
distance          float64
year                int64
dtype: object
```

Buradaki **method** değişkenini ***object*** türünden ***category*** türüne dönüştürülmesi tavsiye edilir. Makine öğrenmesi aşamasında bunu yapmaya gerek yoktur fakat Vahit hocamızın tavsiyesidir. Bunun sebebi bazı gelişmiş fonksiyonlar bunu kategorik değişken olarak algılasa da bazı fonksiyonlar bunu sadece string olarak algılamaktadır.

Dönüştürme işlemini yapmak için:

In [8]:
```
df.method = pd.Categorical(df.method)
```

In [9]:
```
df.dtypes
```

Out[9]:
```
method            category
number               int64
orbital_period     float64
mass               float64
distance           float64
year                 int64
dtype: object
```

Görüldüğü üzere **method** değişkeni ***category*** türüne dönüştürülmüş oldu.

In [10]:
```
df.head()
```

Out[10]:

	method	number	orbital_period	mass	distance	year

	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

Veri Setinin Betimlenmesi

In [11]:
```
planets = sns.load_dataset("planets")
df = planets.copy()
```

In [12]:
```
df.head()
```

Out[12]:

	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 [13]:
```
df.shape
```

Out[13]:
```
(1035, 6)
```

In [14]:
```
df.columns
```

Out[14]:
```
Index(['method', 'number', 'orbital_period', 'mass', 'distance', 'year'], dtype='object')
```

In [15]:
```
df.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%	max
number	1035.0	1.785507	1.240976	1.000000	1.00000	1.0000	2.000	7.0
orbital_period	992.0	2002.917596	26014.728304	0.090706	5.44254	39.9795	526.005	730000.0
mass	513.0	2.638161	3.818617	0.003600	0.22900	1.2600	3.040	25.0
distance	808.0	264.069282	733.116493	1.350000	32.56000	55.2500	178.500	8500.0
year	1035.0	2009.070531	3.972567	1989.000000	2007.00000	2010.0000	2012.000	2014.0

Not: `describe()` fonksiyonu eksik gözlemleri göz ardı eder ve kategorik değişkenleri dışarıda bırakır.
Fakat değişken değerleri sayısal diye kategorik olmayacak anlamına gelmiyor. 0,1,2,... şeklinde de kategorilere ayrılmış olabilir. Bu yüzden veri seti hikayesi çok iyi anlaşılmalıdır.

Şimdi çıktıyı anlayacak olursak: **number:** Ortalaması ve standart sapması çok çok düşük. min ve max değerlerine baktığımız zaman bu değişkenin 1-7 arasında olduğunu anlıyoruz.
orbital_period: Ortalama, standart sapma, min, max değerleri çok uçuk seviyede. Bizim bildiğimiz şekliyle bir değişken olmadığını anlıyoruz.
mass: Ortalama ve standart sapma uygun gibi duruyor. Max değer 25 olması ***number*** değişkeni ile bir ilişkisi olduğunu anlıyoruz. Yani galaksi sayısı ne kadar artarsa kütle de o kadar artacağı için bir ilişki mümkündür.
distance: Standart sapmasının çok fazla olduğu gözlemleniyor. Min ve max değerlerine bakarak çok geniş bir dağılım olduğunu anlayabiliriz.
year: Yıl değişkeni olduğundan ort, std, min, max değerlerine bakmak hatalı olacaktır. Programa bunun bir yıl değişkeni olduğunu belirtmemiz gerekiyor. Fakat bu konuyu zaman serisi konusunda ele alacağız.

Not: Buradaki ort, std vs. değerlerini diğer değişkenler ile karşılaştırmak yanlıştır. Çünkü her değişkenin kendisine göre açıklaması olduğundan sadece o değişkene göre yorum yapmak gerekir. **Eğer diğer değişkenlerle bir ilişki varsa** o zaman diğer değişkenlere bakılabilir.

Eğer tüm değişkenleri görmek istersek:

In [16]:
```
df.describe(include = "all").T
```

Out[16]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
method	1035	10	Radial Velocity	553	NaN	NaN	NaN	NaN	NaN	NaN	NaN
number	1035	NaN	NaN	NaN	1.78551	1.24098	1	1	1	2	7
orbital_period	992	NaN	NaN	NaN	2002.92	26014.7	0.0907063	5.44254	39.9795	526.005	730000
mass	513	NaN	NaN	NaN	2.63816	3.81862	0.0036	0.229	1.26	3.04	25
distance	808	NaN	NaN	NaN	264.069	733.116	1.35	32.56	55.25	178.5	8500
year	1035	NaN	NaN	NaN	2009.07	3.97257	1989	2007	2010	2012	2014

şeklinde yazılmalıdır. Fakat bu mantıklı olmayacağından yazmak gereksizdir. Kategorik ve sürekli değişken için ayrı ayrı analizler ileride yapılacaktır.

Eksik Değerlerin İncelenmesi

Not: Bu bölüm daha geniş kapsamlı olarak ileride **Veri Ön İşleme** dersinde ele alınacaktır. Fakat bu bölümde eksik değerlere sadece veri seti özelinde bakılacak ve eksik değerleri gidermeye yönelik yöntemler önerilecektir.

In [17]:
```
planets = sns.load_dataset("planets")
df = planets.copy()
df.head()
```

Out[17]:

	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

Eksik Gözlem(Değer) Var Mı?

In [18]:
```
df.isnull().values.any()
```

Out[18]:
```
True
```

Hangi Değişkende Kaçar Tane Var?

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
method              0
number              0
orbital_period     43
mass              522
distance          227
year                0
dtype: int64
```

Görüldüğü üzere **number** değişkenini ele alacak olursak hiç eksik veri yok. Zaten olmaması da gerekir çünkü gezegen sayısı olmazsa galaksi de olamayacağından bu değişkende boş veri olmamalıdır. Eğer varsa NASA'nın veri ekibinde veya veri çekilirken ara katmanlarda bir problem olmuş olabilir.

Not: Bu eksikliklerin nasıl ortaya çıktığını **Veri Ön İşleme** dersinde ele alınacaktır. Şu an sadece hangi değişkenlerde kaç tane eksik veri olduğunu gözlemledik.

Eksik Değerleri Doldurmak İçin Yöntemler

1. Yöntem: Sıfır Atamak

In [20]:
```
df["orbital_period"].fillna(0, inplace = True)
```

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
method              0
number              0
orbital_period      0
mass              522
distance          227
year                0
dtype: int64
```

2. Yöntem: Ortalama Atamak

In [22]:
```
df["mass"].fillna(df.mass.mean(), inplace = True)
```

In [23]:
```
df.isnull().sum()
```

Out[23]:
```
method              0
number              0
orbital_period      0
mass                0
distance          227
year                0
dtype: int64
```

Eğer tüm eksik değerler ortalama atamak istersek:

In [24]:
```
df.fillna(df.mean(), inplace = True)
```

In [25]:
```
df.isnull().sum()
```

Out[25]:
```
method            0
number            0
orbital_period    0
mass              0
distance          0
year              0
dtype: int64
```

Görüldüğü üzere eksik veri kalmamış oldu.

Önemli Not: Eksik değerler konusu **çok hassas** bir konudur. Kafamıza göre bir yöntem kullanamayız. Bunların haricinde birçok yöntem vardır. İlerideki **Veri Ön İşleme** dersinde bu detaylı bir şekilde ele alınacaktır.

Yalnız bu doldurma işlemlerini örnek amaçlı yaptığımızdan veri setimizin yapısı bozulmuş oldu. `copy()` fonksiyonun faydası burada ortaya çıkıyor. Orijinal halini geri getirelim:

In [26]:
```
df = planets.copy()
df.head()
```

Out[26]:

	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 [27]:
```
df.isnull().sum()
```

Out[27]:
```
method              0
number              0
orbital_period     43
mass              522
distance          227
year                0
dtype: int64
```

Kategorik Değişken Özetleri

In [28]:
```
planets = sns.load_dataset("planets")
df = planets.copy()
df.head()
```

Out[28]:

	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

	method	number	orbital_period	mass	distance	year
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

Sadece Kategorik Değişkenler ve Özetleri

In [29]:
```
kat_df = df.select_dtypes(include = ["object"])
```

In [30]:
```
kat_df.head()
```

Out[30]:

	method
0	Radial Velocity
1	Radial Velocity
2	Radial Velocity
3	Radial Velocity
4	Radial Velocity

Bu fonksiyon ile istediğimiz türdeki değişkenleri seçebiliyoruz. Burada **object** türünde olan değişken sadece **method** değişkenidir.

Kategorik Değişkenin Sınıflarına ve Sınıf Sayısına Erişmek

In [31]:
```
kat_df.method.unique()
```

Out[31]:
```
array(['Radial Velocity', 'Imaging', 'Eclipse Timing Variations',
       'Transit', 'Astrometry', 'Transit Timing Variations',
       'Orbital Brightness Modulation', 'Microlensing', 'Pulsar Timing',
       'Pulsation Timing Variations'], dtype=object)
```

method kategorik değişkeninin sınıflarına eriştik fakat kaç tane olduğunu görmek istersek:

In [32]:
```
kat_df["method"].value_counts().count()

#ya da

kat_df["method"].nunique()
```

Out[32]:
```
10
```

şeklinde görebiliriz.

Kategorik Değişkenin Sınıflarının Frekanslarına Erişmek

In [33]:
```
kat_df["method"].value_counts()
```

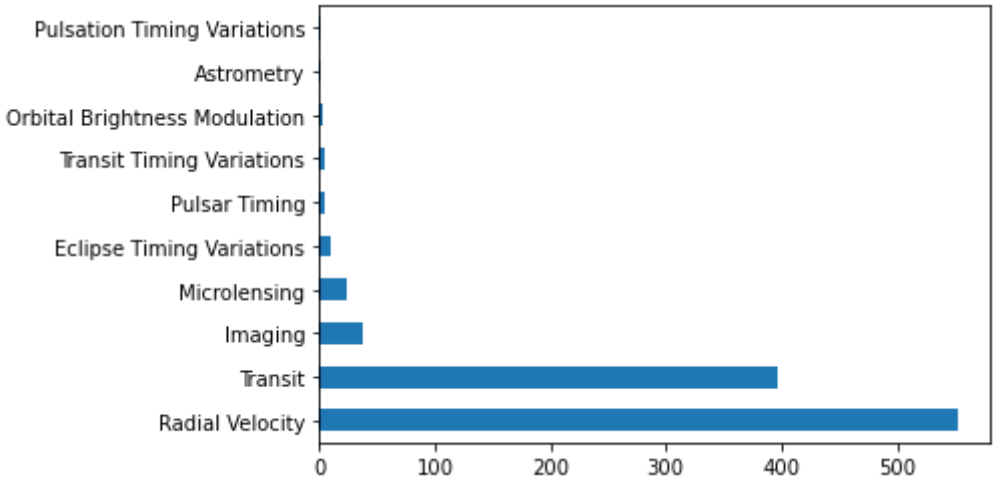
Out[33]:
```
Radial Velocity                553
Transit                        397
Imaging                         38
Microlensing                    23
Eclipse Timing Variations        9
Pulsar Timing                    5
Transit Timing Variations        4
Orbital Brightness Modulation    3
Astrometry                       2
Pulsation Timing Variations      1
Name: method, dtype: int64
```

Görüldüğü üzere her bir sınıfın frekanslarına da erişmiş olduk. Grafik ile görmek istersek:

In [34]:
```
df["method"].value_counts().plot.barh()

#df["method"].value_counts().plot.barh();
```

Out[34]:
```
<AxesSubplot:>
```



Görüldüğü üzere sınıfların frekanslarını Pandas'ın `plot.barh()` fonksiyonu yardımıyla görmüş olduk.

Dipnot: Çıktıdan önceki **<<AxesSubplot:>>** ifadesini ve genel olarak çıktıdan önce yazılan bellekte nesne oluşturma ile ilgili bilgileri kaldırmak için kodun sonuna

";" koymalıyız.

Sürekli Değişken Özetleri

In [35]:
```
planets = sns.load_dataset("planets")
df = planets.copy()
df.head()
```

Out[35]:

	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 [36]:
```
df_num = df.select_dtypes(include = ["float64","int64"])
```

In [37]:
```
df_num.head()
```

Out[37]:

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

In [38]:
```
df_num.describe().T
```

Out[38]:

	count	mean	std	min	25%	50%	75%	max
number	1035.0	1.785507	1.240976	1.000000	1.00000	1.0000	2.000	7.0
orbital_period	992.0	2002.917596	26014.728304	0.090706	5.44254	39.9795	526.005	730000.0

	count	mean	std	min	25%	50%	75%	max
mass	513.0	2.638161	3.818617	0.003600	0.22900	1.2600	3.040	25.0
distance	808.0	264.069282	733.116493	1.350000	32.56000	55.2500	178.500	8500.0
year	1035.0	2009.070531	3.972567	1989.000000	2007.00000	2010.0000	2012.000	2014.0

In [39]:
```
df_num["distance"].describe()
```

Out[39]:
```
count     808.000000
mean      264.069282
std       733.116493
min         1.350000
25%        32.560000
50%        55.250000
75%       178.500000
max      8500.000000
Name: distance, dtype: float64
```

Bu çıktıları Türkçe olarak görmek istersek:

In [40]:
```
"""
print("Ortalama: ", str(df_num["distance"].mean()))
print("Dolu Gözlem Sayısı: " + str(df_num["distance"].count()))
print("Maksimum Değer: " + str(df_num["distance"].max()))
print("Minimum Değer: " + str(df_num["distance"].min()))
print("Medyan: " + str(df_num["distance"].median()))
print("Standart Sapma: " + str(df_num["distance"].std()))
"""
#ya da

print("Ortalama:", df_num["distance"].mean())
print("Dolu Gözlem Sayısı:", df_num["distance"].count())
print("Maksimum Değer:", df_num["distance"].max())
print("Minimum Değer:", df_num["distance"].min())
print("Medyan:", df_num["distance"].median())
print("Standart Sapma:", df_num["distance"].std())
```

```
Ortalama: 264.06928217821786
Dolu Gözlem Sayısı: 808
Maksimum Değer: 8500.0
Minimum Değer: 1.35
Medyan: 55.25
Standart Sapma: 733.1164929404421
```

Sütun Grafik (Bar Plot)

Elimizdeki **kategorik değişkenleri görselleştirmek için** kullanılır.

Veri Seti Hikayesi

Burada başka bir veri seti kullanılacaktır. Bu veri setinin ismi "diamonds"tır. Bu veri setinin genel amacı pırlanlatar, mücevherler ile ilgili bilgiler tutmasıdır. Değişkenleri ve anlamları ise şu şekildedir:

price: dolar cinsinden fiyat (326 - 18,823)
carat: ağırlık (0,2 - 5,01)
cut: kalite (Fair, Good, Very Good, Premium, Ideal)
color: renk (From J (worst) to D (best))
clarity: temizliği, berraklığı (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
x: length in mm (0 - 10,74)
y: width in mm (0 - 58,9)
z: depth in mm (0 - 31,8)
depth: toplam derinlik yüzdesi = z / mean(x,y) = 2 *z / (x* y) (43 - 79)
table: elmasın en geniş noktasına göre genişliği (43 - 95)

In [41]:
```
diamonds = sns.load_dataset("diamonds")
df = diamonds.copy()
df.head()
```

Out[41]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Veri Setine Hızlı Bakış

In [42]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
```

```
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  category
 2   color    53940 non-null  category
 3   clarity  53940 non-null  category
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: category(3), float64(6), int64(1)
memory usage: 3.0 MB
```

In [43]:
```
df.describe().T
```

Out[43]:

	count	mean	std	min	25%	50%	75%	max
carat	53940.0	0.797940	0.474011	0.2	0.40	0.70	1.04	5.01
depth	53940.0	61.749405	1.432621	43.0	61.00	61.80	62.50	79.00
table	53940.0	57.457184	2.234491	43.0	56.00	57.00	59.00	95.00
price	53940.0	3932.799722	3989.439738	326.0	950.00	2401.00	5324.25	18823.00
x	53940.0	5.731157	1.121761	0.0	4.71	5.70	6.54	10.74
y	53940.0	5.734526	1.142135	0.0	4.72	5.71	6.54	58.90
z	53940.0	3.538734	0.705699	0.0	2.91	3.53	4.04	31.80

In [44]:
```
df["cut"].value_counts()
```

Out[44]:
```
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
```

In [45]:
```
df["color"].value_counts()
```

Out[45]:
```
G    11292
E     9797
F     9542
H     8304
D     6775
I     5422
```

```
J     2808
Name: color, dtype: int64
```

Buradaki kategorik değişkenlerin **nominal** değil, **ordinal** olduklarını görüyoruz. Bunu Python'a bildirmemiz lazım:

Ordinal Tanımlama

In [46]:
```
from pandas.api.types import CategoricalDtype
```

In [47]:
```
df.cut.head()
```

Out[47]:
```
0      Ideal
1    Premium
2       Good
3    Premium
4       Good
Name: cut, dtype: category
Categories (5, object): ['Ideal', 'Premium', 'Very Good', 'Good', 'Fair']
```

In [48]:
```
df.cut = df.cut.astype(CategoricalDtype(ordered = True))
```

In [49]:
```
df.dtypes
```

Out[49]:
```
carat       float64
cut        category
color      category
clarity    category
depth       float64
table       float64
price         int64
x           float64
y           float64
z           float64
dtype: object
```

In [50]:
```
df.cut.head(1)
```

Out[50]:
```
0    Ideal
Name: cut, dtype: category
Categories (5, object): ['Ideal' < 'Premium' < 'Very Good' < 'Good' < 'Fair']
```

Görüldüğü üzere değişkeni **ordinal** veri türüne çevirdik ve küçükten büyüğe doğru sıraladı. Fakat veri setine bir işlem yapmadan `df.cut.head()` dediğimizde çıktıda kategoriler **Ideal** ile başladığı için ve **Ideal** bu veri setine göre en büyük kategori olduğu için yanlış şekilde sıraladı. Bunu düzeltmek için fonksiyonun içerisine kategorileri **küçükten büyüğe** doğru yazmamız gerekiyor.

In [51]:
```
df.cut = df.cut.astype(CategoricalDtype(
    categories = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal'],
    ordered = True))
```

In [52]:
```
df.cut.head(1)
```

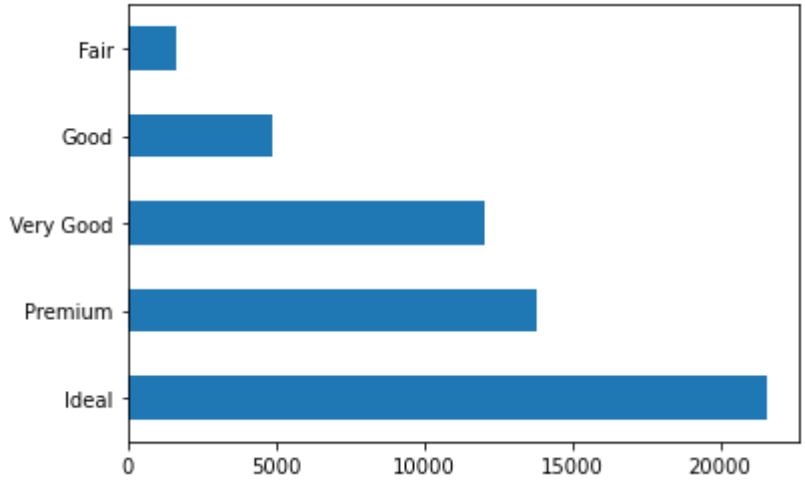
Out[52]:
```
0    Ideal
Name: cut, dtype: category
Categories (5, object): ['Fair' < 'Good' < 'Very Good' < 'Premium' < 'Ideal']
```

Görüldüğü üzere doğru şekilde sıralamış oldu.

Sütun Grafiğinin Oluşturulması

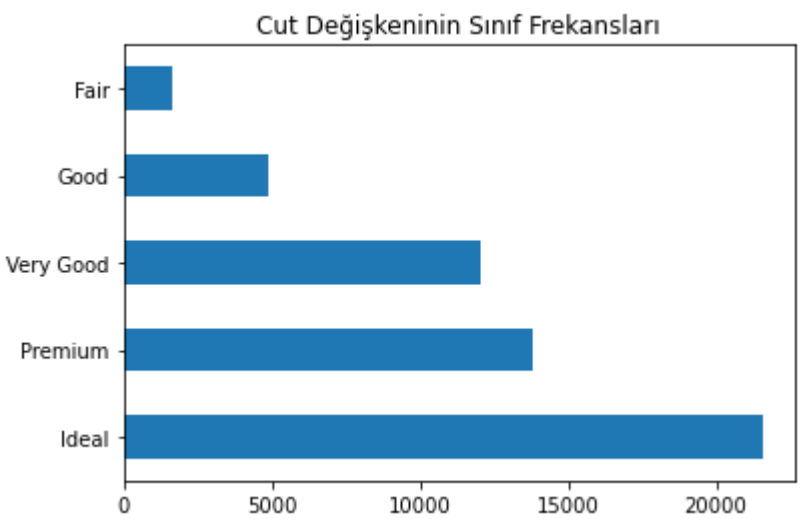
Pandas ile

In [53]:
```
df["cut"].value_counts().plot.barh();
```



Başlık eklemek için `set_title()` fonksiyonu kullanılır.

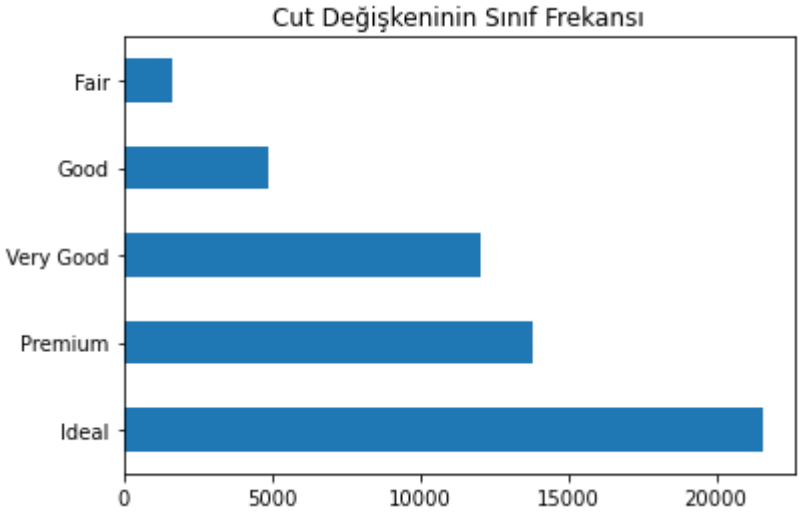
In [54]:
```
df["cut"].value_counts().plot.barh().set_title("Cut Değişkeninin Sınıf Frekansları");
```



Görüldüğü üzere başlığı eklemiş oldu.

Dipnot: Kodu yazdıkça kodlar sağa doğru kaymaktadır ve bu da kötü bir görüntüye sebep olmaktadır. Bunu engellemek için kodun başına ve sonuna parantez koyup noktalardan önce enter'a basılmalıdır.

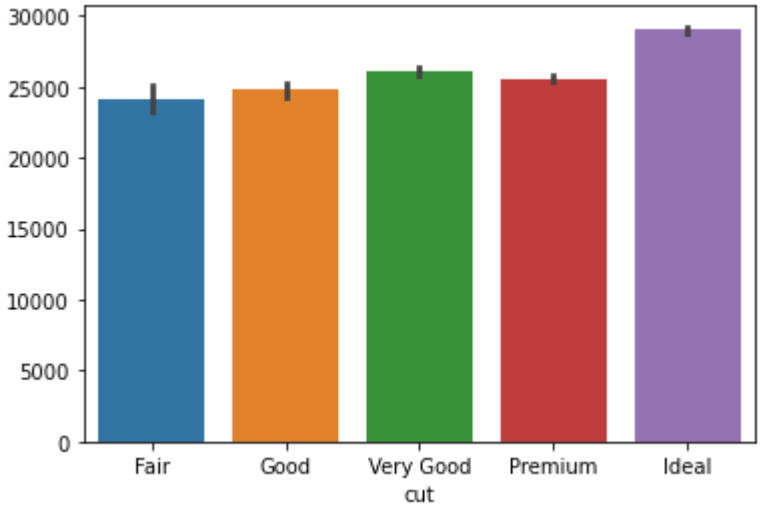
In [55]:
```
(df["cut"]
 .value_counts()
 .plot.barh()
 .set_title("Cut Değişkeninin Sınıf Frekansı"));
```



Görüldüğü üzere daha az yer kaplayarak kodlar okunur hale geldi ve hata vermeden çıktıyı vermiş oldu.

Seaborn ile

In [56]:
```
sns.barplot(x = "cut", y = df.cut.index, data=df);
```

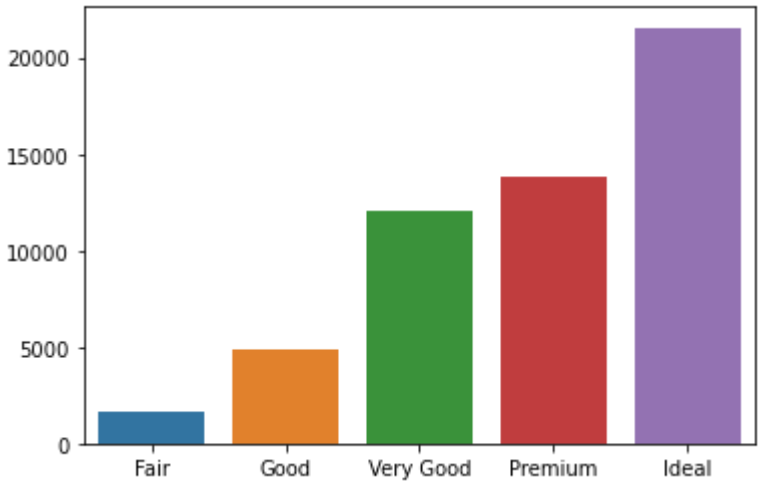


Görüldüğü üzere daha kolay bir yazım ile görselleştirmiş oldu.

Uyarı: Hocamız derste kodu bu şekilde yazdı fakat dikkat edilirse üstteki grafikle aynı değil, yani y eksenindeki değerleri farklı şekilde çizdirdi. Aynı grafik olması için argümanlar şu şekildedir:

In [57]:
```
cut_kategoriler = df["cut"].value_counts().index
cut_frekanslar = df["cut"].value_counts().values

sns.barplot(x = cut_kategoriler, y = cut_frekanslar);
```



Görüldüğü üzere Pandas'taki ile aynı grafik elde edilmiş oldu.

Sütun Grafik Çaprazlamalar

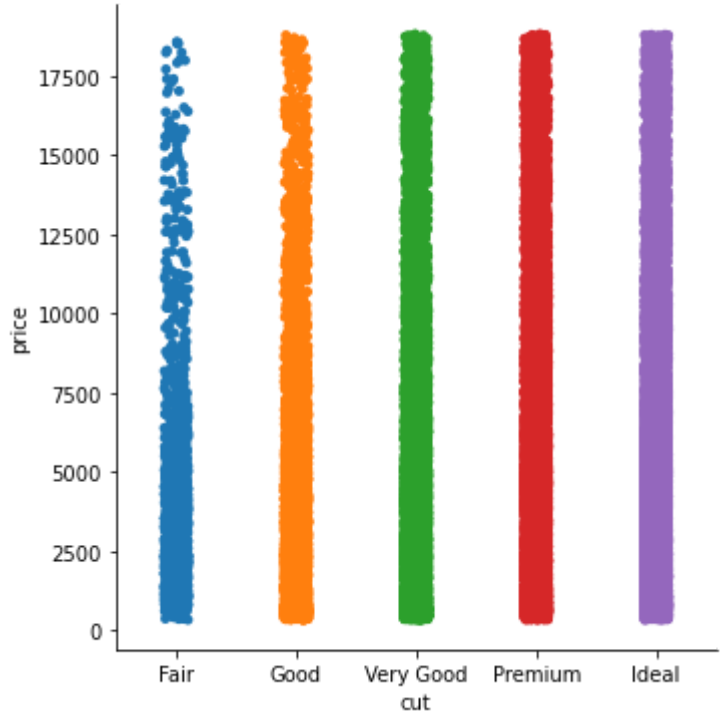
Çaprazlama: Veri seti içerisinde yer alan değişkenlerin birlikte değerlendirilmesi demektir. Diğer ifadelerle kırılımları göz önünde bulundurmak, değişkenlerin etkilerinin birlikte değerlendirilmesi denebilir. Kırılım, boyut ile aynı anlamdadır.

Bu bölüm ele alacak olduğumuz veri görselleştirme tekniklerinin her birisi için gerçekleştirilip veri seti üzerinde bazı veri görselleştirme tekniklerini kullanmaktan çok daha öte bu görsel tekniklerin üzerinde daha analitik anlamda somut yorumlar yapabilmek adına kullanacağımız teknikler olacaktır.

İlgili kütüphanelerin tutorial'larında bulunamayacak türden değerlendirmeler ve yorumlamalar yapılacaktır. Dolayısıyla bu bölümlerde ele alacağımız yorumlamalar grafiklerinin teknik özelliklerinin yanında bize daha detaylı, veriye değil de bilgiye erişmek için kullanacak olduğumuz yaklaşımlardır.

Şimdi **cut** kategorik değişkeni ile **price** sayısal değişkenini çaprazlayarak, bir arada değerlendirerek görselleştirme işlemini yapalım:

In [58]:
```
sns.catplot(x = "cut", y = "price", data=df);
```



Grafiği yorumlayacak olursak, öncelikle **price(fiyat)** değişkeni içerisinde bilgi taşıyan önemli bir değişkendir. Bu değişkendeki bilgiyi, diğer değişkenlerin oluşturma/belirleme ihtimali yüksektir. Bu bağlamda grafik incelendiğinde örneğin **Fair** sınıfında yaklaşık 8000 doların altı fazla yoğun, üstü ise az yoğun olarak gözlemlenmektedir ve **Ideal** sınıfına doğru arttıkça yani kaliteler arttıkça yoğunluğun üst taraflarda daha fazla biriktiği görülmektedir.

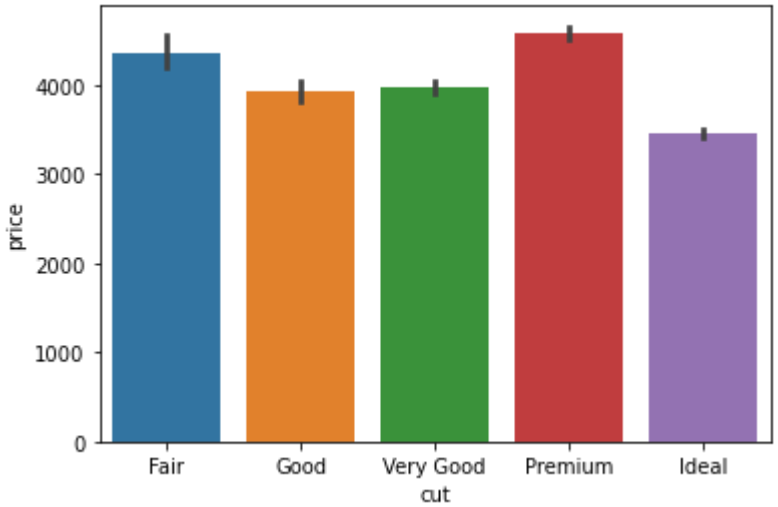
Buraya kadar olan kısım bilginin görünün kısmı idi. Bir de burada görülmeyen ve keşfedilmesi gereken daha fazla bilgi bulunmaktadır. Çaprazlama işlemlerine devam ederek bu bilgilere erişmeye çalışalım:

Bir tane daha boyut ekleyerek inceleyelim. Burada **color** değişkenini ekleyeceğiz.

"Bana Ait"

Öncelikle bir önceki grafiğin bar grafik halini çizdirelim:

In [59]:
```
sns.barplot(x = "cut", y = "price", data=df);
```



Burada **cut** değişkeninin her bir sınıfına göre ortalama değerlerini aldı. Kontrol için:

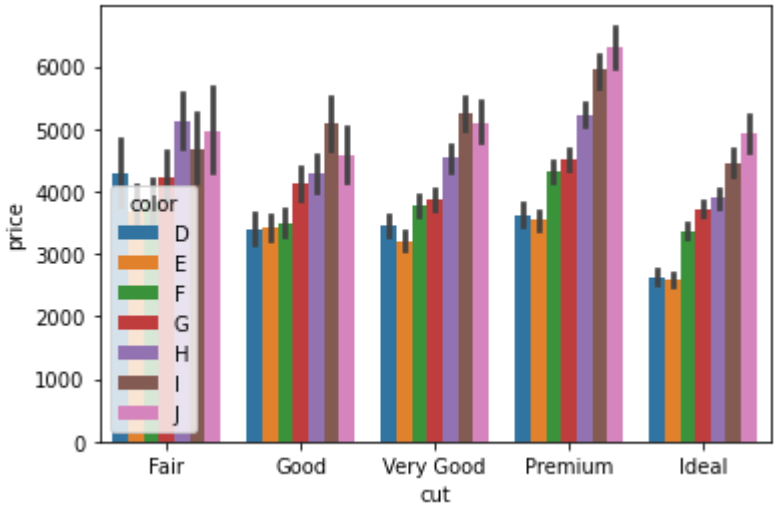
In [60]:
```
cut_price = df[["cut","price"]]
cut_price[cut_price.cut == "Fair"].mean()
```

Out[60]:
```
price    4358.757764
dtype: float64
```

Görüldüğü üzere **Fair** sınıfı grafikte de 4000 değerinin üzerinde olarak görünüyor.

Şimdi **color** değişkenini ekleyelim. Bunun için **hue** argümanı kullanılır.

In [61]:
```
sns.barplot(x = "cut", y = "price", hue = "color", data = df);
```



Görüldüğü üzere 3. bir değişken eklendiğinde bir önceki grafikten farklı olarak aslında **color** değişkeni bazında yoğunluğun nerelerde yer aldığını gördük. Kalite yükseldikçe **color** değişkenindeki **I ve J** sınıflarının fazla olduğunu görüyoruz. **Ideal** sınıfında farklı bir durum olarak **D ve E** sınıflarının yoğunluğu aşağıya çektiğini görüyoruz. Başka bir farklı durum ise **Premium** sınıfının **J** değeri, **Ideal** sınıfının **J** değerinden daha büyük. Bu, grafiğin bizden sakladığı bilgilerden bir tanesidir.

Bu grafikten çıkaracağımız yorum, elimizdeki değişkenleri görselleştirdiğimizde bu görsellerin üzerine yeni boyutlar, yeni kırılımlar, yeni değişkenler ekledikçe oluşturmuş olduğumuz grafiklerin bize sunmuş olduğu ilk bilginin nasıl ortaya çıktığı, nasıl oluştuğuna yönelik bazı ek bilgilere de erişebiliyoruz.

Bu grafikte fark edilen bir diğer husus **price** değişkeninin 6000'e kadar olmasıdır. Bunun sebebi, biz 3. boyutu eklediğimizde bunu teknik olarak bir arada göstermek sütun grafiği anlamında çok mümkün olmayacağından dolayı arka tarafta aslında kendisi bir veriyi temsil etme değeri oluşturup merkezi eğilimleri temsil edecek bir kategorik değişkenler kırılımı oluşturup buna göre bir yansıtma yaptı. Yani burada **price** ekseninde görmüş olduğumuz değerler iki kategorik değişken bir arada bulunduğunda iki `groupby` işlemi sonrasında **price** değişkeninin ortalamasıdır. Üzerine konulan çubuklar ise standart sapmalarıdır.

Bunu doğrulamak mümkün. Şimdi bunu yapalım:

In [62]:
```
df.groupby(["cut","color"])["price"].mean()
```

Out[62]:
```
cut        color
Fair       D        4291.061350
           E        3682.312500
           F        3827.003205
           G        4239.254777
           H        5135.683168
           I        4685.445714
           J        4975.655462
Good       D        3405.382175
           E        3423.644159
           F        3495.750275
           G        4123.482204
           H        4276.254986
```

```
            I        5078.532567
            J        4574.172638
Very Good   D        3470.467284
            E        3214.652083
            F        3778.820240
            G        3872.753806
            H        4535.390351
            I        5255.879568
            J        5103.513274
Premium     D        3631.292576
            E        3538.914420
            F        4324.890176
            G        4500.742134
            H        5216.706780
            I        5946.180672
            J        6294.591584
Ideal       D        2629.094566
            E        2597.550090
            F        3374.939362
            G        3720.706388
            H        3889.334831
            I        4451.970377
            J        4918.186384
Name: price, dtype: float64
```

Görüldüğü üzere grafikteki değerler ile aynı olduğu gözlemledi.

Yani özetle bu grafik, önce **cut** değişkenine göre grupladı, sonra **color** değişkenine göre gruplayıp ortalama değerleri gösterdi.

Histogram ve Yoğunluk Grafikleri

Bu grafikler sayısal değişkenler için kullanılan, **sayısal değişkenlerin dağılımını** ifade etmek için kullanılan grafiklerdir.

Histogram ve Yoğunluk Grafiğinin Oluşturulması

In [2]:
```
diamonds = sns.load_dataset("diamonds")
df = diamonds.copy()
df.head()
```

Out[2]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63

	carat	cut	color	clarity	depth	table	price	x	y	z
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

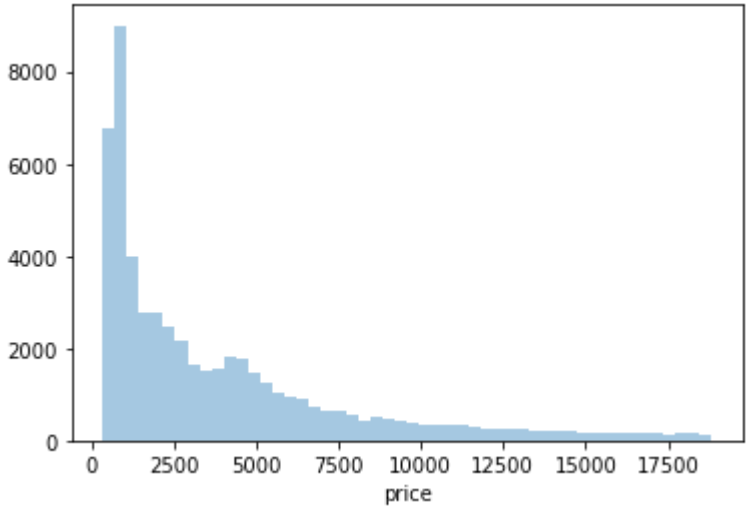
price değişkeninin histogramını bir inceleyelim:

In [78]:
```
sns.distplot(df.price, kde = False);
#ya da

#sns.histplot(df.price, kde = False);

# kde: Dağılımın yoğunluk eğrisinin(çizgisinin) olup olmama durumu.
```

```
C:\Users\ertug\anaconda3\envs\tf\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

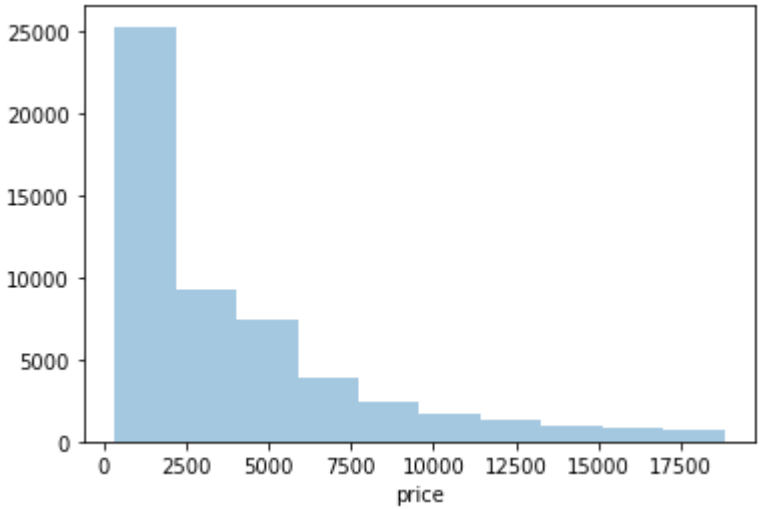


Histogram grafiği bize sayısal değişkenlerin dağılımını göstermesi adına çok önemlidir ve dağılım kavramı genellikle sayısal değişkenler için kullanılır. İstatistikte dağılım kavramı oldukça önem arz etmektedir. **Veri Bilimi için İstatistik** dersinde bunlar çok detaylı ele alınacaktır.

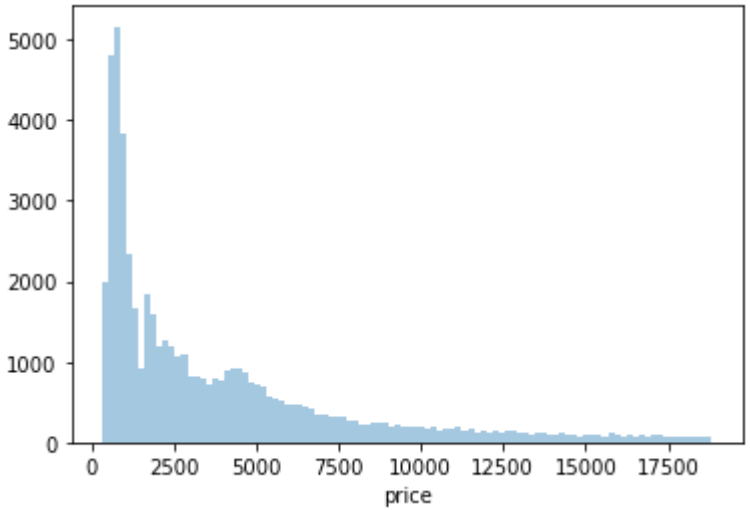
Histogramın mantığı ise elimizdeki sayısal değişkenin değerlerini belirli aralıklara böler ve belirli aralıklardaki ilgili değerlerin gözlem ve frekanslarını yansıtır. Örneğin der ki: **0-100 arasında bu kadar, 100-200 arasında bu kadar değer gözlenmiş vs.** şeklinde bize değişkenimizin belirli değerlerdeki dağılımıyla ilgili bilgi vermiş olur.

Şimdi bu fonksiyonun **bins** argümanını kullanalım. Bu argüman, en önemli argümanlardan bir tanesidir ve histogramdaki kutuların sayısını belirtmek için kullanılır.

In [67]:
```
sns.distplot(df.price, bins = 10, kde = False);
```



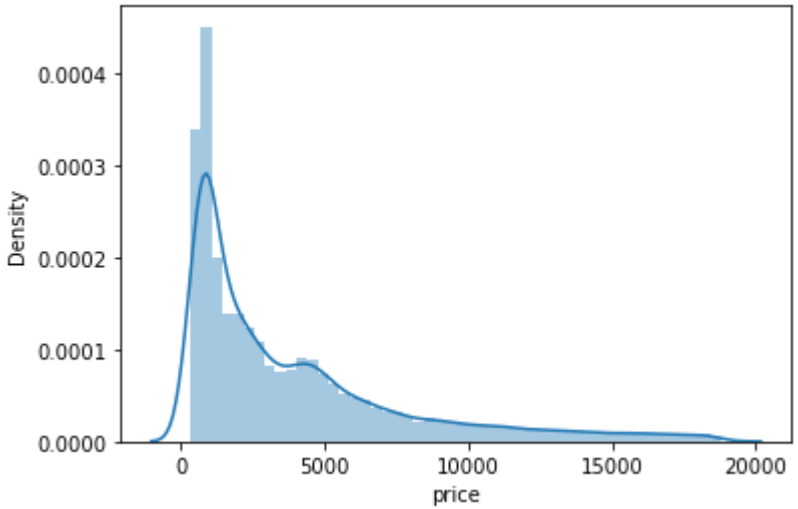
In [68]:
```
sns.distplot(df.price, bins = 100, kde = False);
```



Görüldüğü üzere kutuların sayısı arttı ve temsil gücü daha fazla olmuş oldu.

Şimdi bu grafiğin yoğunluk eğrisini çizdirelim. Bunun için **kde** argümanını silmemiz veya **True** yazmamız yeterlidir. Çünkü ön tanımlı değeri **True** dur.

In [69]:
```
sns.distplot(df.price);
```



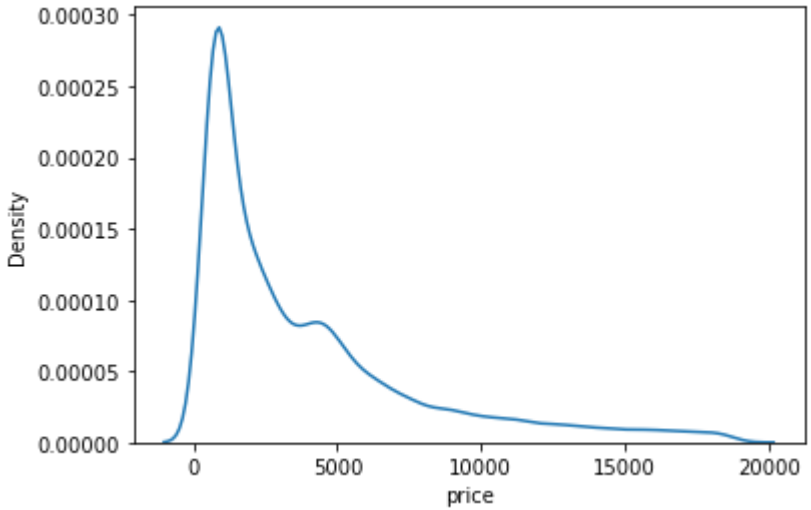
Görüldüğü üzere çizdirmiş oldu. Aslında bu çizgi(eğri) olasılık yoğunluk fonksiyonunun grafiğidir. Dikkat edilirse y ekseninin değerleri değişti. Yani histogram ile yoğunluk grafiğinin birlikte gösterilmesi istendiğinde y ekseninde ölçeklerin birbirinden farklı olacağından dolayı, görselleştirme imkanı olmayacağından ya da çok mantıksız olacağından dolayı arka planda seaborn otomatik olarak ölçeği **0-1** arasına, daha doğrusu olasılık yoğunluk fonksiyonunun ölçeğine göre gerçekleştirip(bu ölçeğe indirgeyip) histogramla yoğunluk grafiğini birlikte sunma imkanı veriyor.

Burada y ekseni, frekansların standartlaştırılmış halidir.

Eğer sadece yoğunluk grafiğini görmek istersek argüman olarak `hist = False` yazmamız yeterlidir.

In [70]:
```
sns.distplot(df.price, hist = False);
```

```
C:\Users\ertug\anaconda3\envs\tf\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```



İki grafik türü de bir sayısal değişkenin konumlanmasını/dağılmasını ifade etmek için kullanılır.

Grafiği yorumlayacak olursak **price** değişkeninin standart sapmasının çok olduğunu, çarpık olduğunu, medyan ile ortalaması arasındaki farkın olduğunu söyleyebiliriz.

price ın betimsel istatistiklerine bir bakalım:

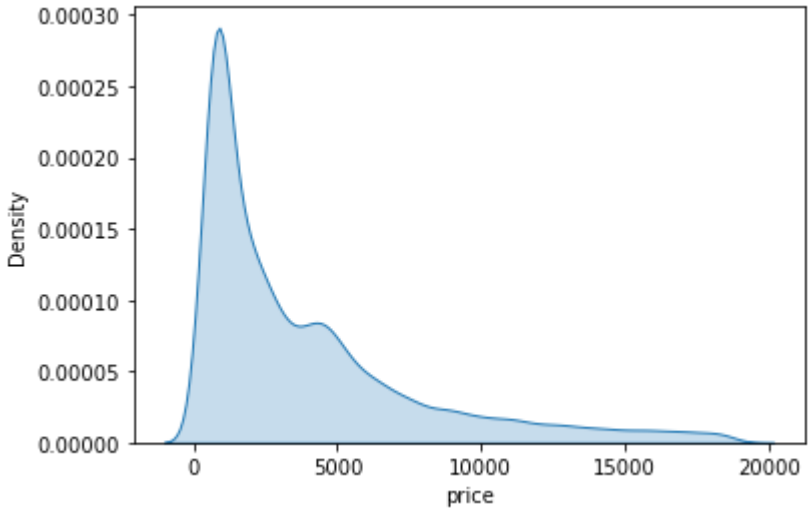
In [71]:
```
df["price"].describe().T
```

Out[71]:
```
count    53940.000000
mean      3932.799722
std       3989.439738
min        326.000000
25%        950.000000
50%       2401.000000
75%       5324.250000
max      18823.000000
Name: price, dtype: float64
```

Çaprazlamalar bölümünde(bir sonraki bölümde) bu grafiğin nasıl oluştuğu, neyden kaynaklı olduğu ele alınacaktır.

Bu yoğunluk grafiğinin altını doldurmak için başka bir fonksiyon kullanmalıyız.

In [72]:
```
sns.kdeplot(df.price, shade = True);
```



Görüldüğü üzere altını doldurmuş oldu.

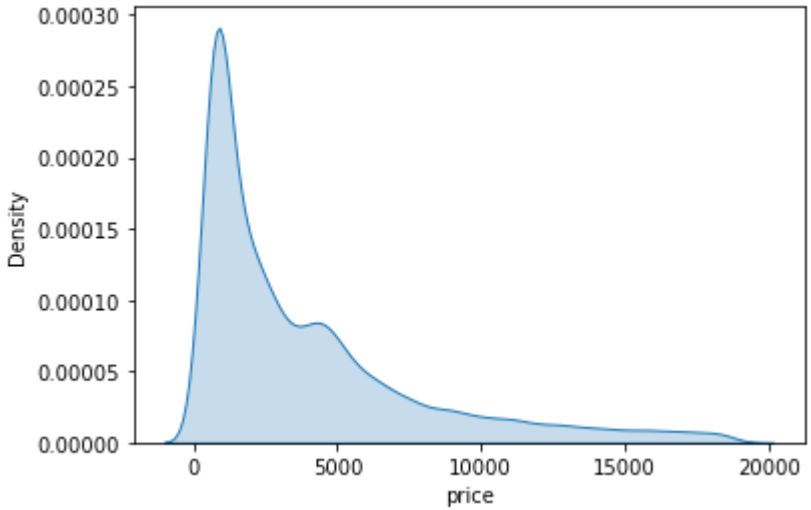
Histogram ve Yoğunluk Çaprazlamalar

In [79]:
```
df.head()
```

Out[79]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [80]:
```
sns.kdeplot(df.price, shade = True);
```

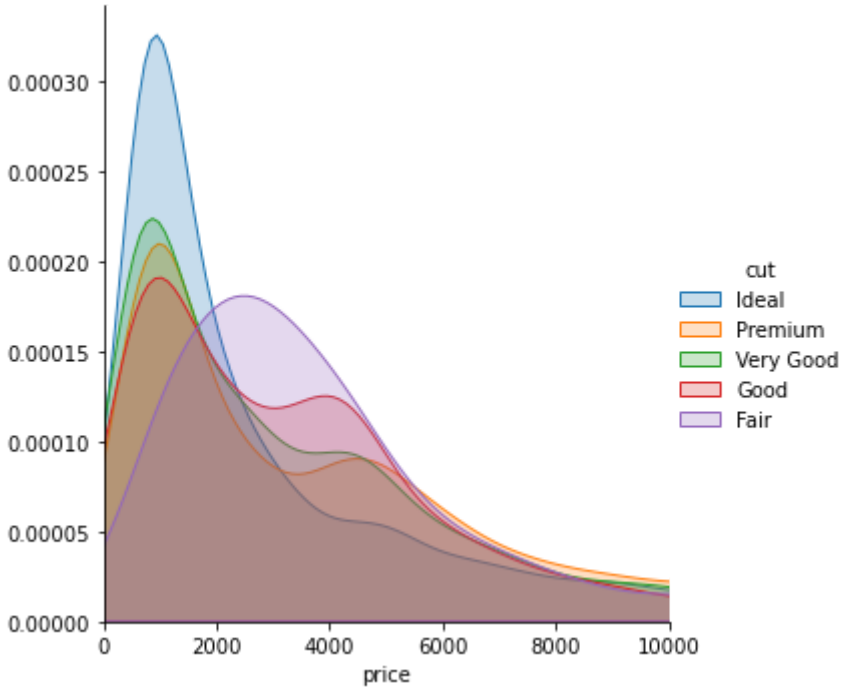


Şimdi burada **price** *bağımlı(Y)* değişkenin grafiğinin bu şekilde olmasının sebebini yani bizden saklanan bilgiyi araştıracağız. Çünkü sadece bu grafikten yorum yapmak, iş kararları almak, özellikle değişken bağımlı bir değişkense yanıltıcı olacaktır. Bu yüzden titizlikte bu süreç ilerletilmelidir.

Yani çaprazlamaların asıl amacı, bizden saklanan bilgiyi başka değişkenlerce tespit etmektir.

In [90]:

```
(sns
 .FacetGrid(df,
            hue = "cut",
            height = 5,
            xlim = (0, 10000))
 .map(sns.kdeplot, "price", shade = True)
 .add_legend()
);
```



`FacetGrid()` fonksiyonu grafiğe eklenen boyutları bölerek göstermek için kullandık. **xlim** argümanı ise x ekseninin nereye kadar gideceğini belirttik. 10000'e kadar yazmamızın sebebi ise o sayıdan sonra pek bir değişim olmadığından ve biz tepelerle ilgilendiğimizden dolayı neden o şekilde(çarpık) olduklarını anlamak adına ve daha yakından görmek adına x eksenini 10000'e kadar ayarladık.

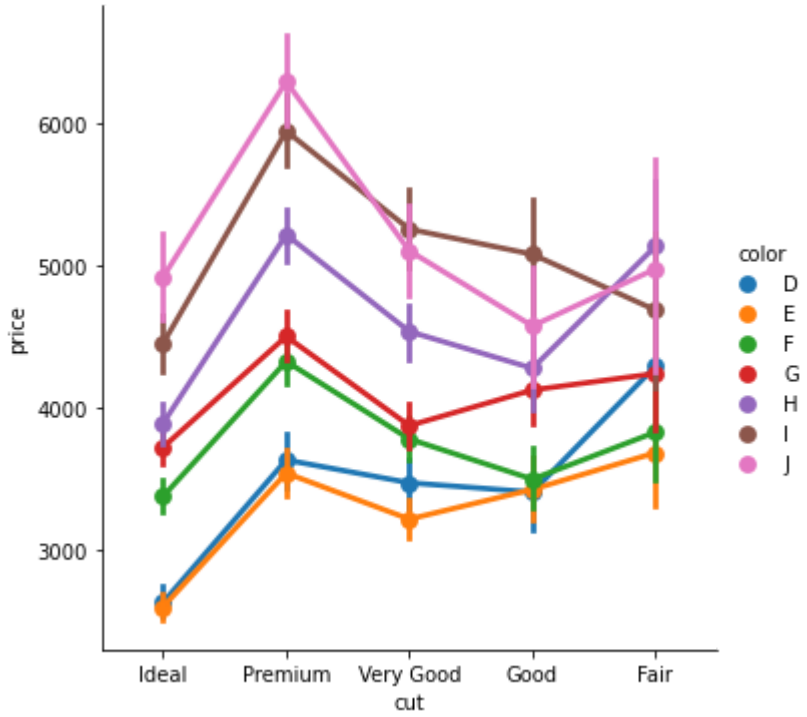
Yazılan kodun anlamı ise, **price**'a göre bir `kdeplot()` grafiği oluştur fakat bunu `FacetGrid()` fonksiyonunu kullanarak tüm **cut** değişkeninin sınıflarıyla maple(eşle).
Yani **price**'a göre bir yoğunluk grafiği oluşturacağız fakat **cut** değişkenin sınıflarını bir boyut olarak eklemek istiyoruz. Bunun için de **cut** ile `kdeplot()` 'ı maplememiz(eşlememiz) gerekiyor.
`add.legend()` fonksiyonunu ise grafiğin üzerine eklemiş olduğumuz **cut** kategorik değişkeninin sınıf bilgilerini eklemek için kullandık.

Önemli Bir Bilgi: Normal hayatta sayısal değişkenin normal dağılmasını bekleriz. Fakat burada iki tepeli bir yapı yani çarpık bir yapı var. Bunun genelde anlamı şudur: ***Odaklanılan sayısal değişkenin içerisindeki bilgiyi, yapıyı oluşturan birden fazla faktör var demektir. Yani bu grafiğin normal olmamasının sebebi başka değişken(ler)inin (genelde kategorik değişkenler) bu ilgilendiğimiz değişkene(price) bir etkisi vardır.***

Grafiği yorumlarsak: Normalde kalite arttıkça(**cut**) fiyatın da artar şeklinde bir yorum yapmıştık. Fakat ilginç bir şekilde **Ideal** sınıfı **0-2000** arasında diğerlerine göre daha fazla yoğunlaşmıştır. Hakeza diğerleri de neredeyse **Ideal** sınıfına yakın bir grafik göstermektedir. Ortak olan durum ise yaklaşık **6000** değerinden sonra bütün sınıflar ortak yoğunluktadır. Dolayısıyla bu grafik, bir önceki **price** grafiğinin neden çarpık olduğunu açıklamaktadır.

Bir örnek daha yapalım:

In [84]:
```
sns.catplot(x = "cut", y = "price", hue = "color", kind = "point", data = df);
```



Bu grafik, **cut** kategorik değişkeninin sınıflarının içerisinde yer alan birden fazla renk olması ve bunların bazılarının birbirlerinden uzak olması, **color** aracılığıyla **cut** değişkeni sınıfları kesişiminde birbirinden farklı bir bilgi taşındığı anlamına geliyor. Birbirine yakın olan **color** sınıfları da aynı bilgileri taşıyor anlamına gelir.

Önemli Bir Bilgi: Makine öğrenmesinde amacımız, hedeflediğimiz bağımlı değişkende var olan bilgiyi/değişimi, farklı değişkenlerce açıklamaya çalışmaktır. Bu değişimi farklı değişkenlerce açıklamaya çalışmak amacının en büyük hedefi, veri setinin içerisindeki hedef değişken hedefiyle değişkenliklerin bulunmaya çalışılmasıdır. Veri setini makine öğrenmesiyle modellemeye başladığımızda arka taraftaki algoritmaların en büyük amacı şudur: Bağımlı değişkenin(ilgili değişken, Y) içerisindeki bilgiyi/değişimi diğer değişkenlerce açıklamaya çalışmak oluyor. Diğer değişkenlerce bağımlı değişkenin açıklanma çabasının en önemli noktası ise bağımlı değişkenin içerisinde bu değişkenlerin ayırt edici bilgi taşıması oluyor.

Zaten grafikten görüldüğü üzere **Ideal** sınıfının altında **color** kategorik değişkeninin farklı sınıflarının farklı değerler oluşturmuş olması, burada **color**'ın **cut** ile kesiştirildiğinde **Ideal** sınıfı altında **price** değişkenine göre ayırt edici bir bilgi taşıdığı anlamına geliyor. Yine dikkat edilirse **price** değişkeni **6000**'e kadar gitmiş. Bu da sınıfların ortalama değerleri belirttiği anlamına geliyor.

Kutu Grafik (Box Plot)

Bu grafik türü, histogram gibi elimizdeki sayısal(sürekli) değişkenleri görselleştirmek için çok sık kullanılan histogram ile beraber eşit ağırlığa sahip bir grafik türüdür. Örneğin bir sayısal değişkenin dağılımı incelenmek isteniyorsa **mutlaka ve mutlaka** histogram ile beraber bir kutu grafiğinin de anlaşılması gerekmektedir.

Hatırlatma: Bizim burada amacımız veri setini tanımaktır ve **en en önemli** kısımdır. Veri biliminin asıl amacı veriden bir değer üretmektir. Tanınmayan bir veriden değer üretmek mümkün değildir. İnsanlardan, kurumlardan veya sistemlerden elde edilen veriden optimize işlemleri yapmak, geliştirmeye muhtaç kısımları görmek

ve bir gelir artışı elde etmek istiyorsak veriyi yani aslında değişkenleri çok iyi anlamamız gerekiyor.

Veri Seti Hikayesi

total_bill: yemeğin toplam fiyatı (bahşiş ve vergi dahil)
tip: bahşiş
sex: ücreti ödeyen kişinin cinsiyeti (0=male, 1=female)
smoker: grupta sigara içen var mı? (0=No, 1=Yes)
day: gün (3=Thur, 4=Fri, 5=Sat, 6=Sun)
time: ne zaman? (0=Lunch, 1=Dinner)
size: grupta kaç kişi var?

In [2]:
```
tips = sns.load_dataset("tips")
df = tips.copy()
df.head()
```

Out[2]:

	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

In [15]:
```
df.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%	max
total_bill	244.0	19.785943	8.902412	3.07	13.3475	17.795	24.1275	50.81
tip	244.0	2.998279	1.383638	1.00	2.0000	2.900	3.5625	10.00
size	244.0	2.569672	0.951100	1.00	2.0000	2.000	3.0000	6.00

Genel olarak betimsel istatistiklere bakıldığında restoranda yemek yenilen ve bunun fiyatı ile ilgili olduğundan insan aklına daha yatkın bir veri setidir. Bu veri seti ile ilgili şu sorular sorulabilir:

- **Yemek yiyen kişi sayısının fazla olması bahşiş fiyatını arttırıyor mu?**

- **Sigara içme durumu yemek fiyatını etkiliyor mu?**
- **Hesabı ödeyen kişinin cinsiyetine göre bahşiş fiyatı nasıl etkileniyor?**

gibi birçok soru sorulabilir. Aslında bu soruların cevabı genel olarak istatistik ile cevaplanabilir fakat şu an görselleştirme yardımı ile soruların cevaplarını bulmaya çalışacağız.

In [16]:
```
df["sex"].value_counts()
```

Out[16]:
```
Male      157
Female     87
Name: sex, dtype: int64
```

In [17]:
```
df["smoker"].value_counts()
```

Out[17]:
```
No     151
Yes     93
Name: smoker, dtype: int64
```

In [18]:
```
df["day"].value_counts()
```

Out[18]:
```
Sat     87
Sun     76
Thur    62
Fri     19
Name: day, dtype: int64
```

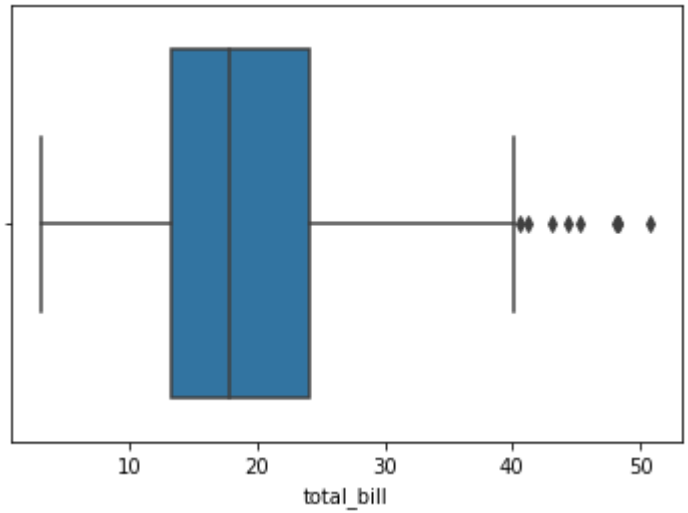
In [19]:
```
df["time"].value_counts()
```

Out[19]:
```
Dinner    176
Lunch      68
Name: time, dtype: int64
```

Kutu Grafiğinin Oluşturulması

Kutu grafiği bize kartiller(çeyrekler) aracılığıyla çok değerli bilgi sunan bir görselleştirme tekniğidir. Yapılacak olan çaprazlamalar ile de bize çok değerli bilgiler sunar.

In [20]:
```
sns.boxplot(x = df["total_bill"]);
```



Bu kutu grafiğinde solda ve sağda bulunan uzun çubuklar minimum ve maksimum değerlerini, ortadaki mavi kutu verilerin yoğunlaştığı bölgeyi, mavi alanın ortasındaki çizgi medyanı, dikdörtgeni oluşturan sağ ve soldaki dik çizgiler verinin %25 ve %75'lik değerleri, en sağdaki noktalar ise aykırı(artık) değerleri ifade ediyor. Yani bu grafikte maksimum değerden sonra bazı aykırı değerler var. Bu aykırı değerler minimum değerinden önce de olabilir.

Dikkat edilirse bu grafiğe bir eğri çizdirildiğinde yoğunluk grafiğine benzeyecektir. Histogram grafiğine benzemesinin ve beraber değerlendirilmesinin sebebi budur.

Normalde hiçbir işlem yapılmadan veriler incelendiğinde **total_bill** değişkeni için **en sağdaki nokta maksimum değerdir**. Bunu `describe()` fonksiyonu ile de görmüştük. Fakat bu grafik onu ve diğer noktaları aykırı değer olarak algıladı ve noktalar halinde gösterdi. Bu yüzden noktalardaki bir önce dik çizgi **maksimum değerdir**. Bu grafik türü bize aykırı değerleri gösterme konusunda çok faydalıdır.

Aykırı değerler konusu ve nasıl hesaplandığı **Veri Ön İşleme - Aykırı Gözlem Analizi** bölümünde gösterilecektir.

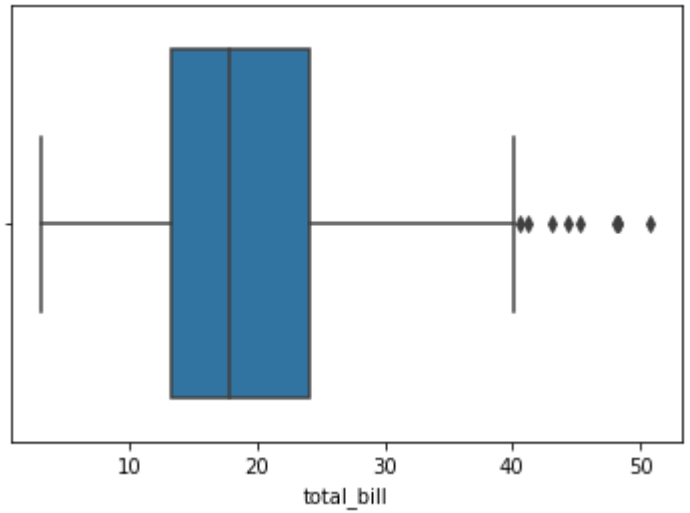
Bu grafiği dik şekilde de çizdirebiliriz.

In [ ]:

In [ ]:

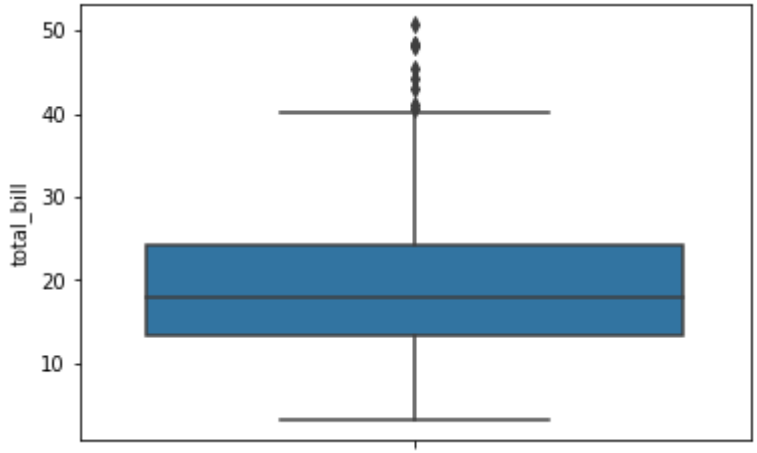
In [39]:
```
sns.boxplot(x = df["total_bill"], orient="v");
```

```
C:\Users\ertug\anaconda3\envs\tf\lib\site-packages\seaborn\_core.py:1303: UserWarning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```



Uyarı: Hocamız kodu derste bu şekilde yazdı fakat bir uyarı ile karşılaştık ve dik olarak çizdirmedi. Bunun için değişkenimizi x'e değil, y'ye eşitlememiz gerekiyor.

In [36]:
```
sns.boxplot(y = df["total_bill"]);
```



Görüldüğü üzere dik hale getirmiş oldu.

Kutu Grafik Çaprazlamalar

Soru: Hangi günlerde daha fazla kazanılıyor?

In [4]:
```
df.describe().T
```

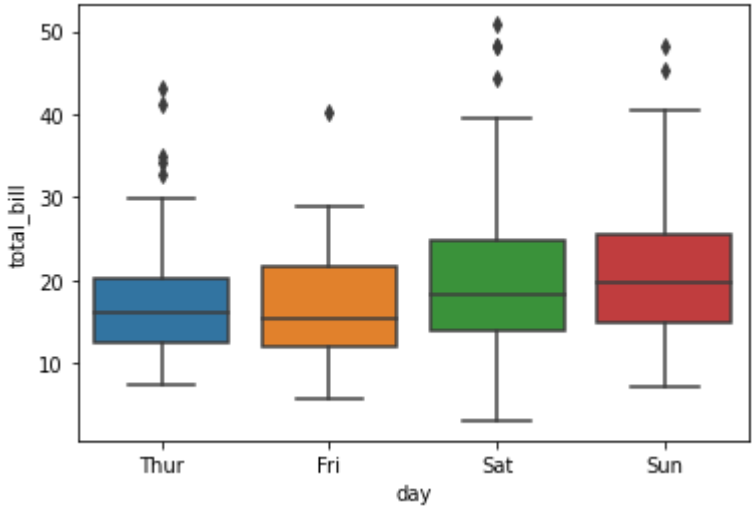
Out[4]:

	count	mean	std	min	25%	50%	75%	max

	count	mean	std	min	25%	50%	75%	max
total_bill	244.0	19.785943	8.902412	3.07	13.3475	17.795	24.1275	50.81
tip	244.0	2.998279	1.383638	1.00	2.0000	2.900	3.5625	10.00
size	244.0	2.569672	0.951100	1.00	2.0000	2.000	3.0000	6.00

Not: İnsan aklına yakın veri setlerinde işe başlanmadan önce bazı sorular hazırlanmalı ve ilerleyen süreçlerde onların cevapları araştırılmalıdır.

In [5]:
```
sns.boxplot(x = "day", y = "total_bill", data = df);
```

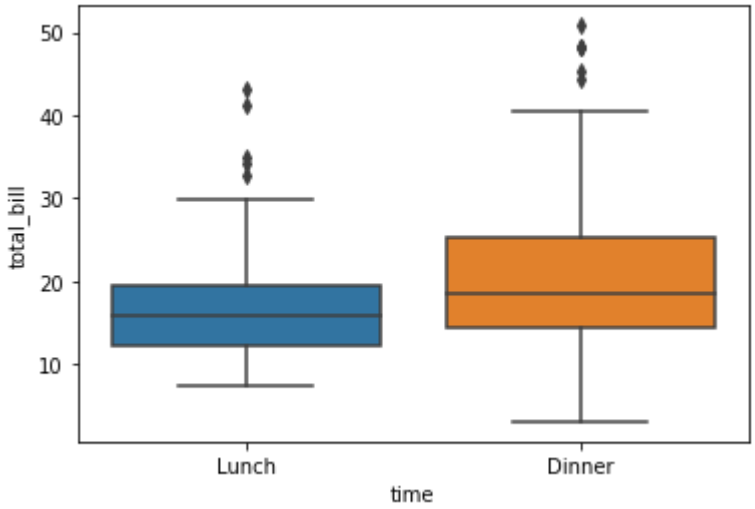


Grafikten görüldüğü üzere:

- Günlerin frekanslarını hatırlarsak cumartesi günü daha fazla işlem yapılmasına rağmen pazar günü daha fazla kâr elde edilmiş.

Soru: Öğle yemeğinde mi yoksa akşam yemeğinde mi daha çok kazanılıyor?

In [6]:
```
sns.boxplot(x = "time", y = "total_bill", data = df);
```



Grafikten görüldüğü üzere:

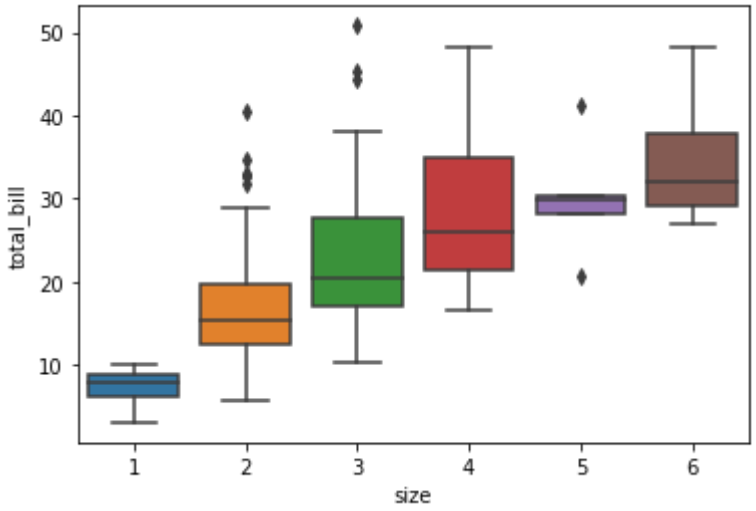
- Akşam yemeklerinden daha fazla para kazanılıyor.

Bu grafiklerden yola çıkarak şirket şu kararı verebilir:

- Garsonlar genelde hafta sonları ve akşamları çalıştırılmalı ki daha fazla kâr elde edilsin.

Soru: Yemeğe gelen grupların kişi sayısı ile kazanç doğru orantılı mıdır?

In [7]:
```
sns.boxplot(x = "size", y = "total_bill", data = df);
```

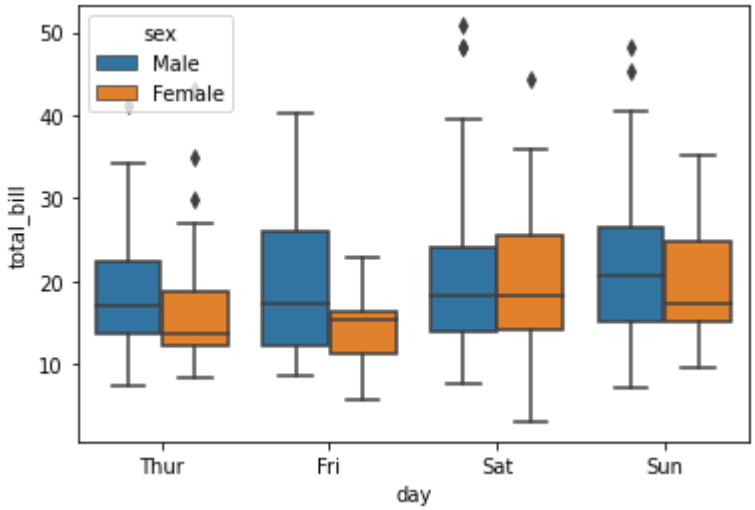


Grafikten görüldüğü üzere:

- Beklenildiği gibi kişi sayısı arttıkça kazanç da artmaktadır.

Soru: Cinsiyete göre gün bazında kazanç ne kadar değişiyor?

In [9]:
```
sns.boxplot(x = "day", y = "total_bill", hue = "sex", data = df);
```



Grafikten görüldüğü üzere:

- Cumartesi hariç genelde erkekler ödeme yapmışlardır.

Violin Grafik

- Kutu grafiğine benzerdir.
- Biraz daha dağılım anlamında bize bilgi sunar.
- Yoğunluk grafiği ile kutu grafiğinin kesişimi gibi düşünülebilir.

Violin Grafiğinin Oluşturulması

In [2]:
```
tips = sns.load_dataset("tips")
df = tips.copy()
df.head()
```

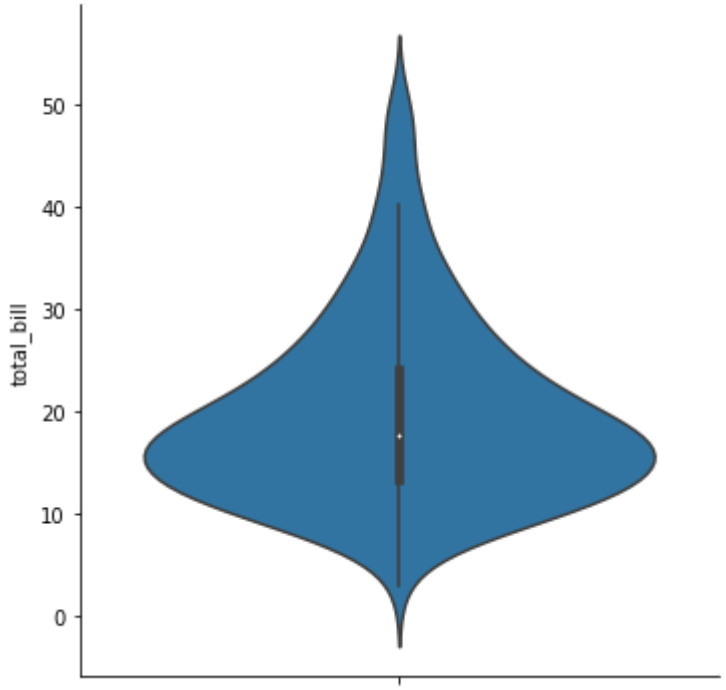
Out[2]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2

	total_bill	tip	sex	smoker	day	time	size
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

`catplot()` fonksiyonunu kullanarak violin grafiğini çizdirelim.
Bunun için **kind** argümanına **violin** yazmalıyız.

In [4]:
```
sns.catplot(y="total_bill", kind="violin", data=df);
```



Görüldüğü üzere bu grafik:

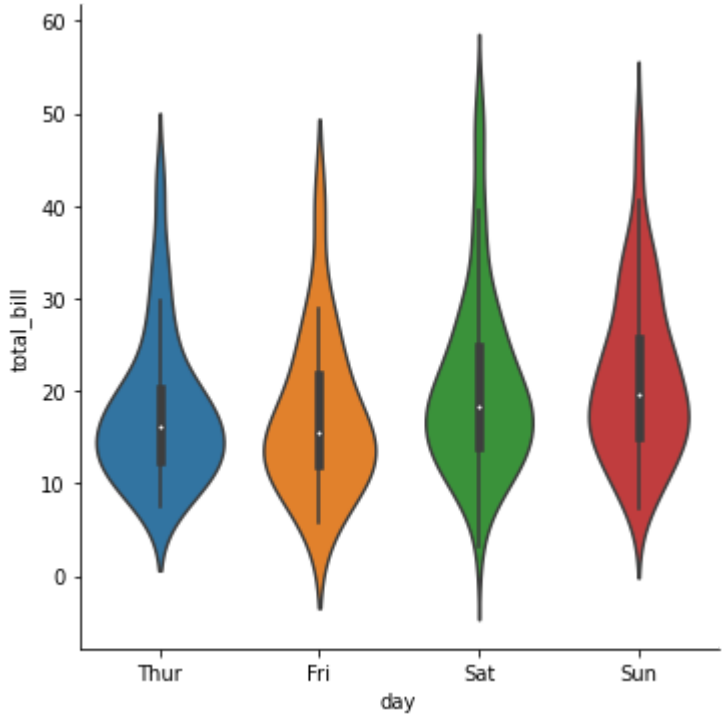
- Boxplot grafiğinin kutu olan kısmın kenarlarının yumuşatılmış şeklidir.
- Tercihen kullanılan bir grafik türüdür.
- Boxplot grafiği ile yorumlaması aynıdır.
- Ortadaki beyaz nokta medyanı, kalın kısımlarının uçları çeyreklikleri, çizginin biten yerleri de max ve min'leri ifade eder.

Violin Çaprazlamalar

Yeni bir boyut olarak **günleri** ekleyip inceleyelim:

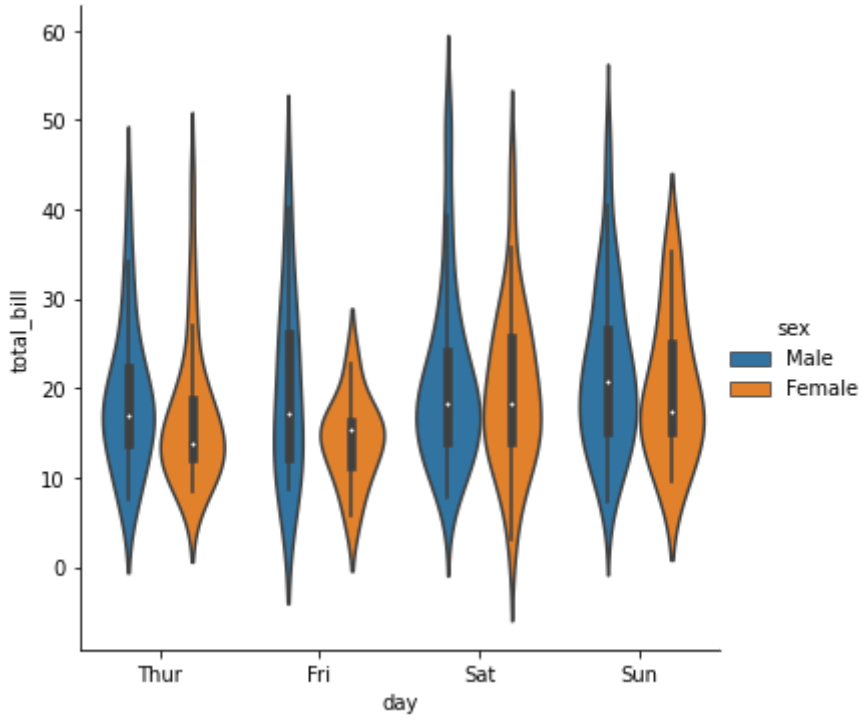
Bunun için argüman olarak `x="day"` şeklinde yazmalıyız.

In [5]:
```
sns.catplot(x="day", y="total_bill", kind="violin", data=df);
```



Yeni bir boyut olarak cinsiyeti de ekleyip gözlemleyelim:

In [6]:
```
sns.catplot(x="day", y="total_bill", hue="sex", kind="violin", data=df);
```



Korelasyon Grafikleri

Korelasyon:

- Değişkenler arasındaki ilişkiyi ifade eden istatistiksel bir terimdir.
- İki değişken arasındaki ilişkiyi ifade etmek için kullanılan ve en çok bilinen yaklaşım **scatter plot(saçılım grafiği)** yaklaşımıdır.

Önceki bölümlerde tek sayısal değişken için grafikler incelenirken bu bölümde iki sayısal değişkenin göstermiş olduğu yapıyı inceleyeceğiz.

Yine bu sefer iki sayısal değişken için çaprazlama anlamında da yeni değişkenler ekleyip inceleyeceğiz.

Korelasyon Grafiğinin Oluşturulması

Scatter Plot

Bu grafik türü sayısal değişkenler arasındaki ilişkiyi gösterir.

Yine **"tips"** veri seti kullanılacaktır. Değişkenleri hatırlayacak olursak:

total_bill: yemeğin toplam fiyatı (bahşiş ve vergi dahil)
tip: bahşiş
sex: ücreti ödeyen kişinin cinsiyeti (0=male, 1=female)

smoker: grupta sigara içen var mı? (0=No, 1=Yes)

day: gün (3=Thur, 4=Fri, 5=Sat, 6=Sun)

time: ne zaman? (0=Lunch, 1=Dinner)

size: grupta kaç kişi var?

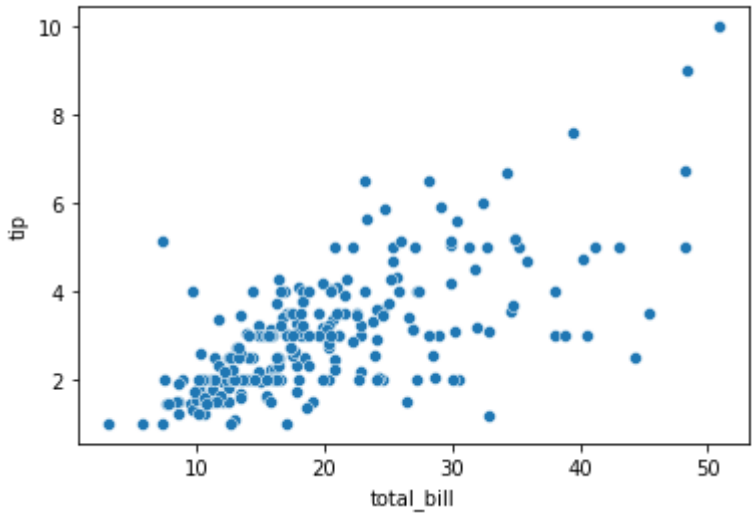
In [3]:
```
tips = sns.load_dataset("tips")
df = tips.copy()
df.head()
```

Out[3]:

	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

total_bill ile **tip** değişkenkeri arasındaki ilişkiyi gözlemleyelim:

In [8]:
```
sns.scatterplot(x="total_bill",y="tip",data=df);
```



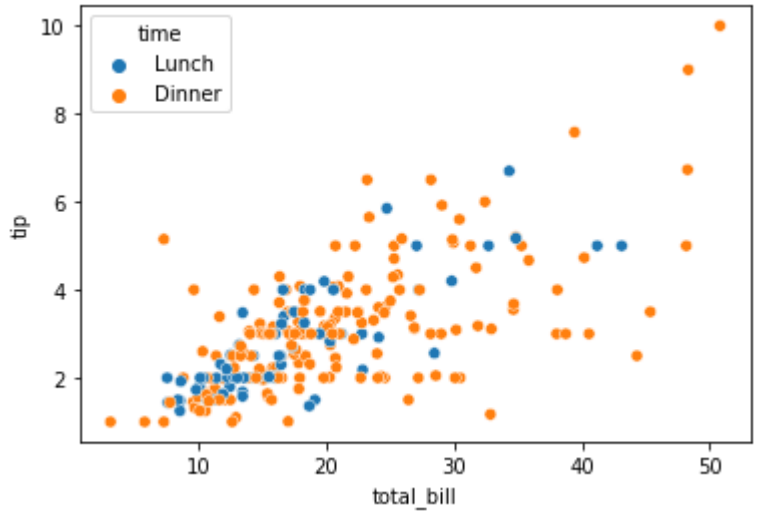
Grafikten görüldüğü üzere:

- Fiyat arttıkça verilen bahşiş de arttığı gözlemlenmektedir. Yani pozitif bir ilişki söz konusudur.

Korelasyon Çaprazlamalar

Boyut olarak **time** değişkenini ekleyelim:

In [3]:
```
sns.scatterplot(x="total_bill",y="tip",hue="time",data=df);
```

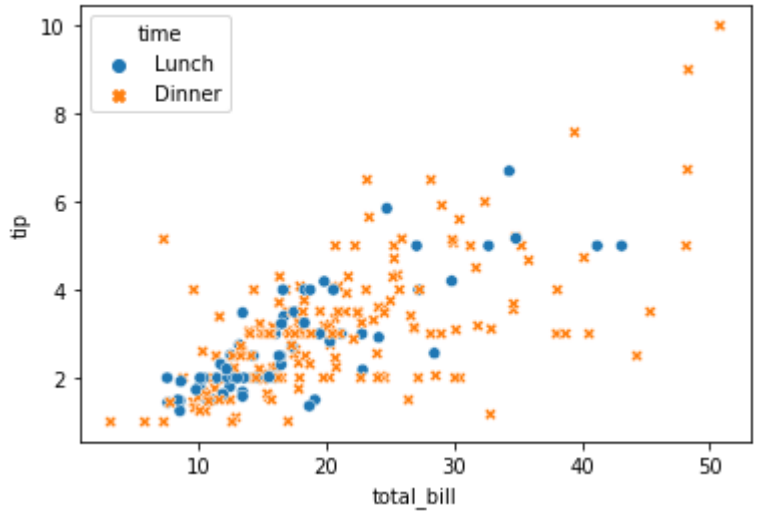


Grafikte görüldüğü üzere:

- Akşam yemeği için fiyat arttıkça öğle yemeğine göre bahşiş daha fazla artmaktadır.
- Bir önceki grafikte görülemeyen bilgi, bir boyut ekleme ile görülebildi.
- **İşte bu çaprazlamalar(boyut, kırılım) ile yeni iş karaları alınabilir.**

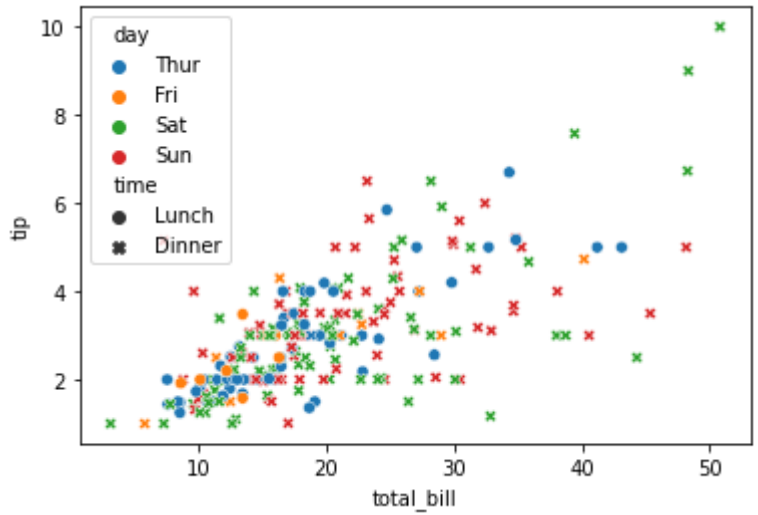
`scatterplot()` fonksiyonunun `style` argümanı ile boyuta göre farklı simgelendirme yapılabilir. O argümana da **time** değişkenini yazalım:

In [4]:
```
sns.scatterplot(x="total_bill",y="tip",hue="time",style="time",data=df);
```



Şimdi ise boyut olarak **day** değişkenini ekleyelim. Diğerleri aynı kalsın.

In [5]:
```
sns.scatterplot(x="total_bill",y="tip",hue="day",style="time",data=df);
```

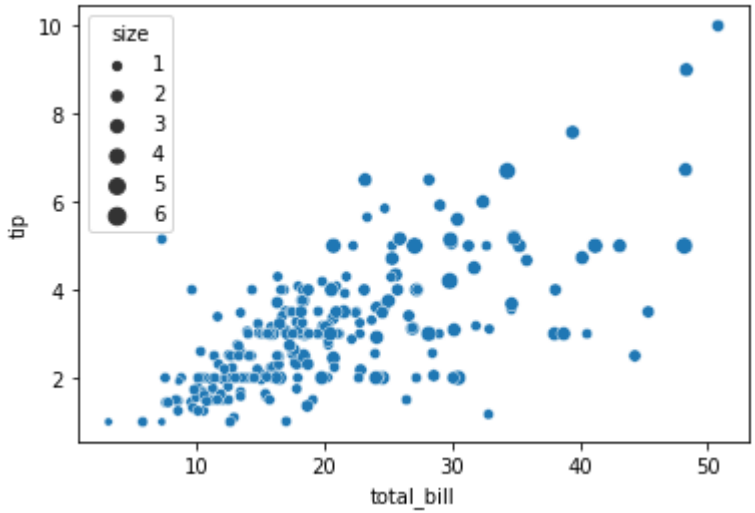


Grafikten görüldüğü üzere:

- **day** değişkenini ekleyerek daha fazla detay elde edebildik.

Şimdi ise bir **sürekli değişkeni** boyut olarak ekleyelim. Bunun için `size` argümanını kullanmalıyız. Değişken olarak da **size** değişkenini kullacağız.

In [6]:
```
sns.scatterplot(x="total_bill",y="tip",size="size",data=df);
```

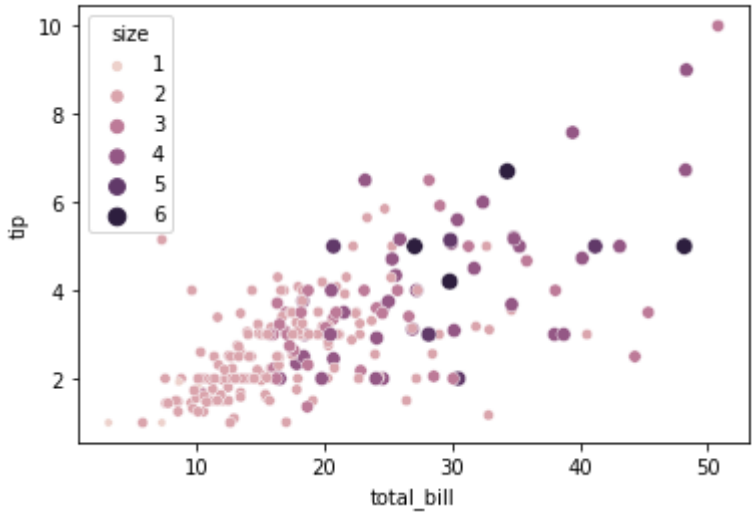


Grafikten görüldüğü üzere:

- **size** değişkeni sayısal olarak arttıkça yuvarlakların boyutu da artmaktadır. **Aslında bu yüzden `size` argümanını kullandık.**

Daah anlaşılır bir grafik için bu değişkeni `hue` argümanına yazalım. Diğerleri aynı kalsın.

In [11]:
```
sns.scatterplot(x="total_bill",y="tip",hue="size",size="size",data=df);
```



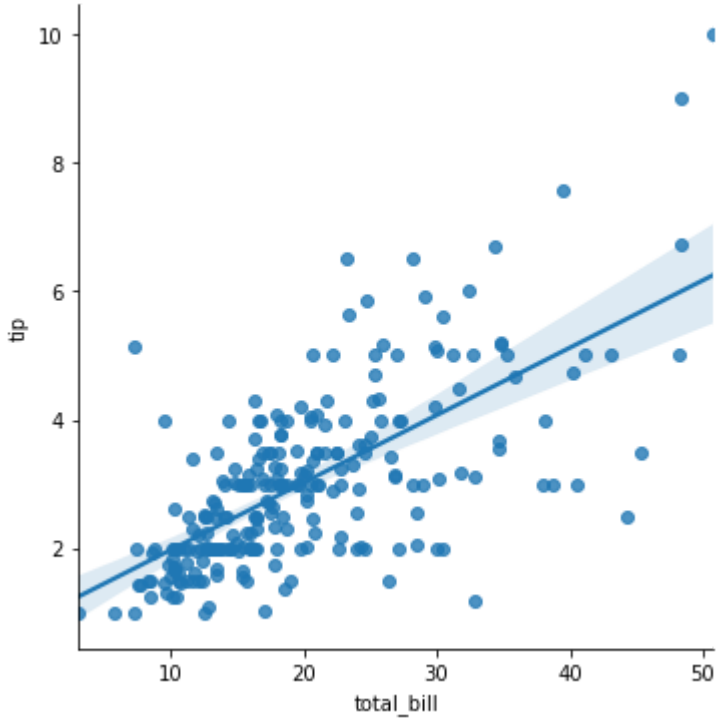
Görüldüğü üzere renklendirme yaparak daha anlaşılır bir grafik elde ettik.

Doğrusal İlişkinin Gösterilmesi

İki sürekli değişken arasındaki ilişkiyi bu sefer bir çizgi yardımıyla göstereceğiz. Bunun için **linear modeling** ifadesinin kısaltması olan `lmplot` fonksiyonunu

kullanacağız.

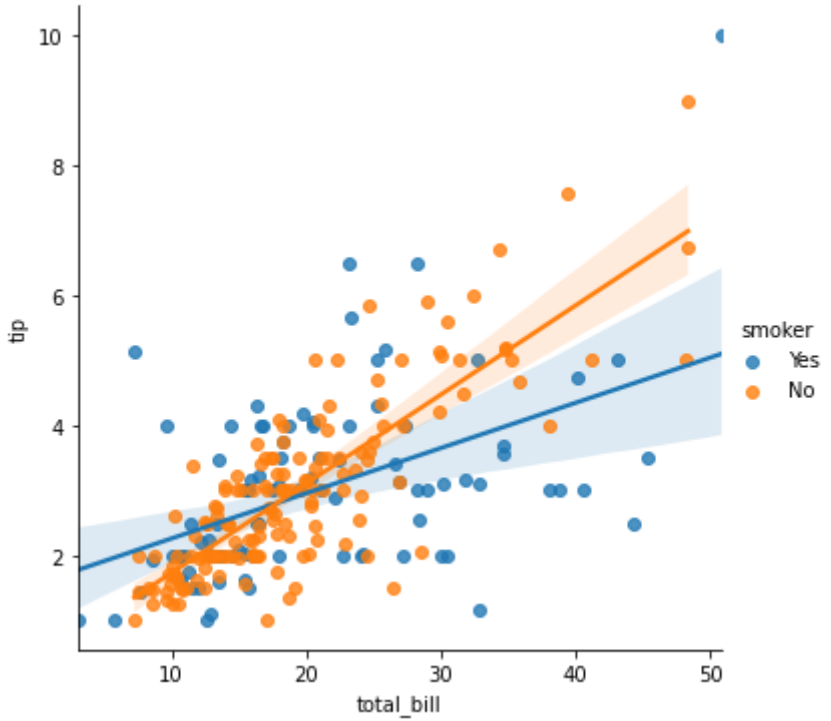
In [5]:
```
sns.lmplot(x="total_bill",y="tip",data=df);
```



Görüldüğü üzere ilişkiyi bir çizgi halinde gösterdi. Çizginin etrafındaki açık maviler ise çizgi etrafındaki ortalama sapmayı ifade eder.

Şimdi boyut olarak **smoker** değişkenini ekleyelim:

In [6]:
```
sns.lmplot(x="total_bill",y="tip",hue="smoker",data=df);
```

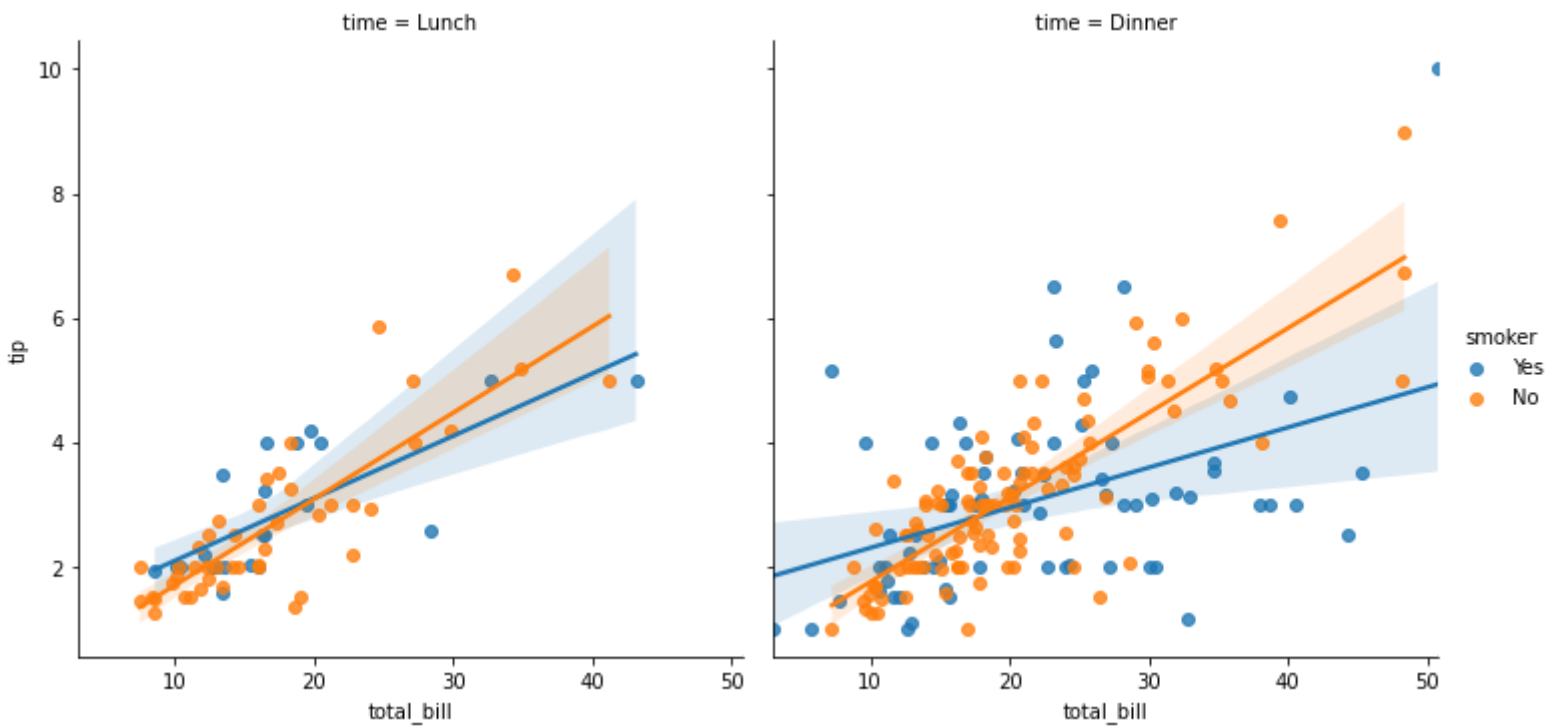


Görüldüğü üzere:

- **smoker** değişkeni eklendiğinde çizgilerin eğimi değişti.
- **Bizim için ve makine öğrenmesi için çok önemli grafiklerden bir tanesidir.**
- Yorum olarak ise sigara içmeyenlerin toplam fiyatı arttıkça bahşiş de arttığı gözlemlenmektedir.

Şimdi ise `col` argümanına **time** değişkenini ekleyelim:

In [7]:
```
sns.lmplot(x="total_bill",y="tip",hue="smoker",col="time",data=df);
```

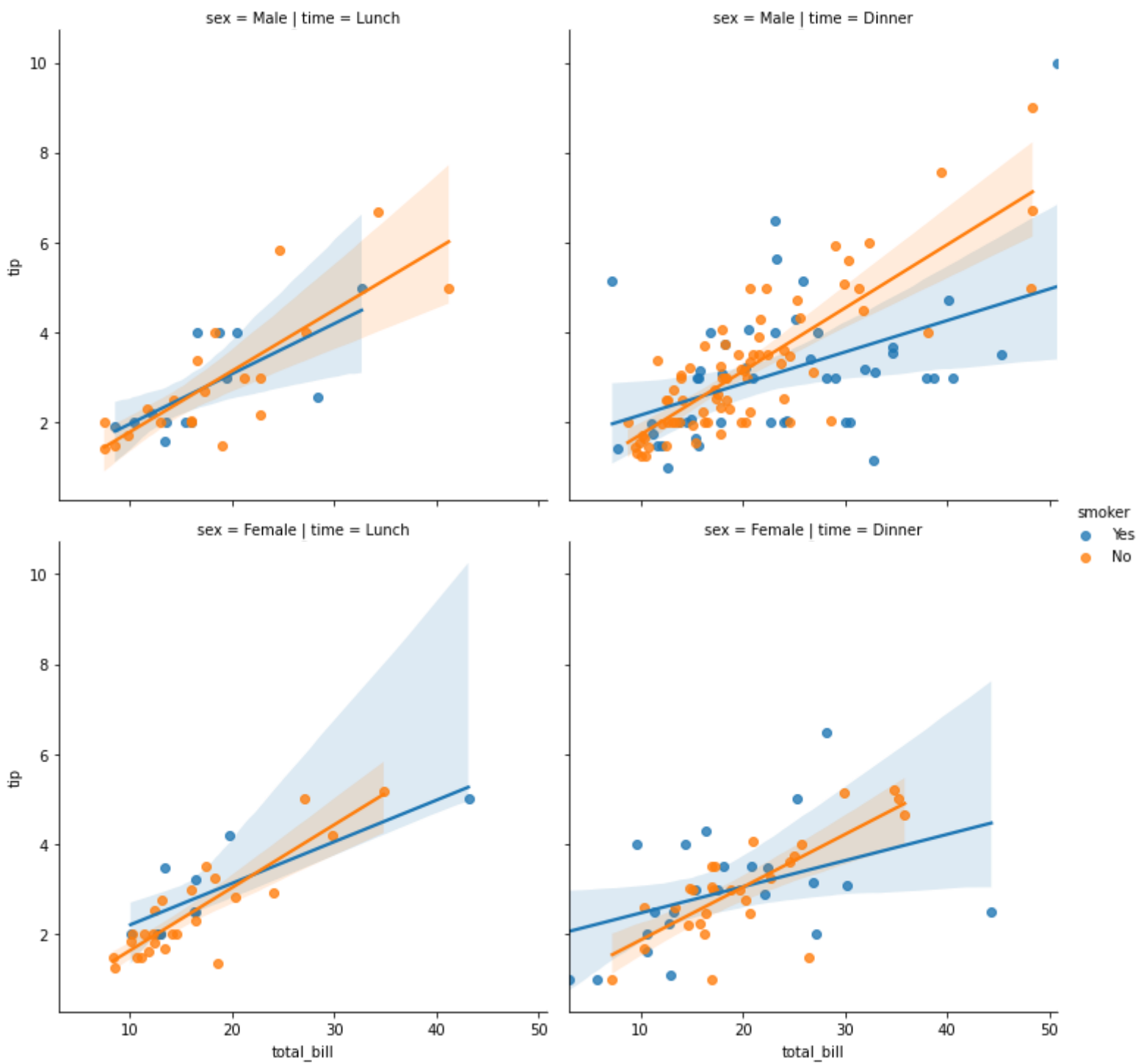


Görüldüğü üzere:

- Sütun olarak öğünlere göre de ayırmış olduk.
- Bu şekilde yapmamızın sebebi saklı bilgiyi öğrenip ona göre iş kararları almaktır.
- Makine öğrenmesinde bu çok çok önemlidir. Makine öğrenmeden önce bizim bazı bilgileri öğrenmemiz gerekiyor ki makine öğrenmesine hazırlık olsun.

Şimdi ise `row` değişkenine cinsiyeti ekleyelim:

In [8]:
```
sns.lmplot(x="total_bill",y="tip",hue="smoker",col="time",row="sex",data=df);
```



Görüldüğü üzere:

- Satırlara göre de cinsiyeti ayırmış olduk.
- Yorum olarak ise cinsiyet kadın olduğunda öğle yemeği ve akşam yemeği, cinsiyet erkek olduğunda öğle yemeği ve akşam yemeğinin toplam fiyat, bahşiş ve sigara içme durumuna göre ne yönde ilişkili olduğunu gösterir.
- Aslında makine öğrenmesi algoritmaları arka planda bu ilişkileri yapıyor. Biz de bu şekilde yaptıklarının bir kısmını görebiliyoruz.

Scatter Plot Matrisi

- **Scatter plot(saçılım grafiği)**, sayısal değişkenler arasındaki ilişkiyi ifade eden bir grafik idi.
- Burada veri seti içerisindeki tüm sayısal değişkenler için matris formunda bu ilişkiyi ifade ettiğimizde bir scatter plot matrisi oluşturmuş olacağız.

In [2]:
```
iris = sns.load_dataset("iris")
df = iris.copy()
df.head()
```

Out[2]:

	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

In [3]:
```
df.dtypes
```

Out[3]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

Not: species kategorik değişkende sınıfları arasında büyüklük-küçüklük ilişkisi olmadığından, yani nominal olduğundan dönüştürme işlemi yapmamıza gerek yok. **object** türünde kalabilir.

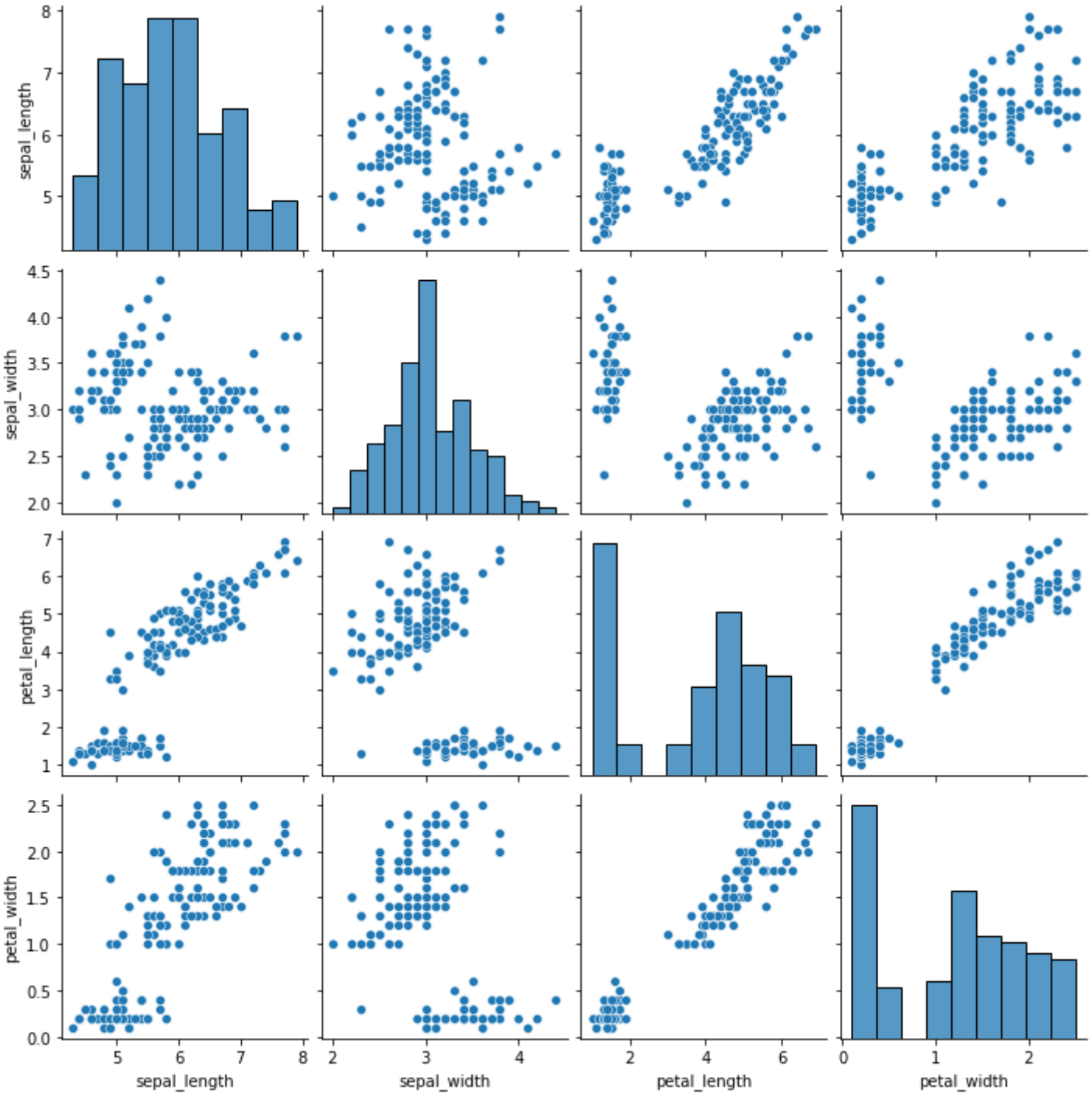
In [4]:
```
df.shape
```

Out[4]:
```
(150, 5)
```

Tüm sayısal değişkenler arasındaki ilişkiyi göstermek için `pairplot` fonksiyonunu kullanacağız.

In [5]:

```
sns.pairplot(df);
```



Bu grafikte bize iki bilgi veriliyor:

- Köşegende yer alan histogram grafikleri bize o değişkenin dağılımı gösteriyor.

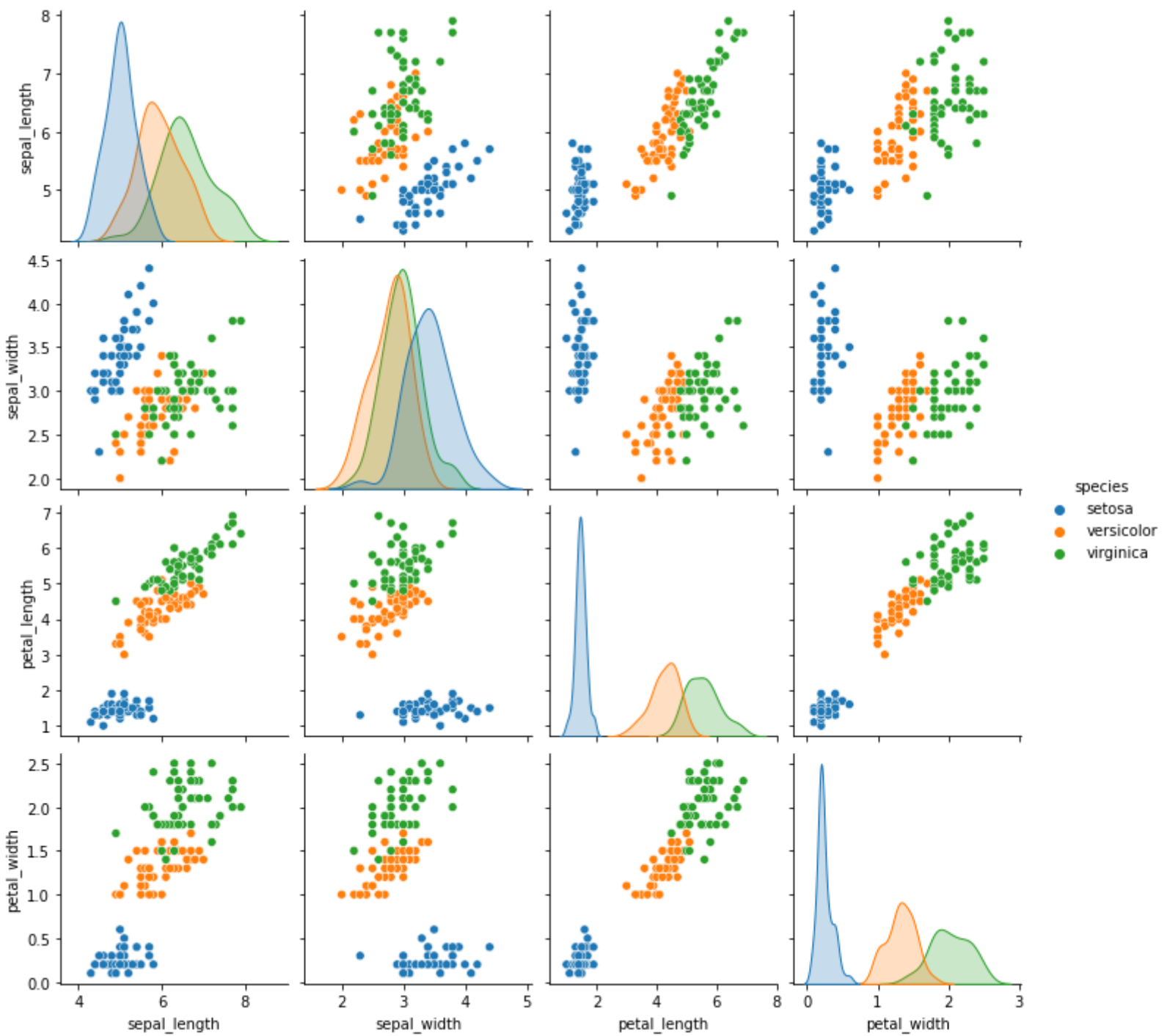
- Diğer saçılım grafikleri ise o değişkenin diğer değişkenler arasındaki ilişkiyi gösteriyor.

ÖNEMLİ:

- Scatter plot görsellerinde eğer oluşturulmuş olan grafik bir toz bulutu şeklindeyse veya yapısal bir formu yoksa o iki değişken arasında bir ilişkinin olmadığını gösterir.
- Eğer oluşturulan scatter plot içerisinde gözlem noktaları incelendiğinde farklı köşelerde kümeleniyorsa bu durumda çaprazlama ile yani başka bir boyut eklenerek incelenmelidir.

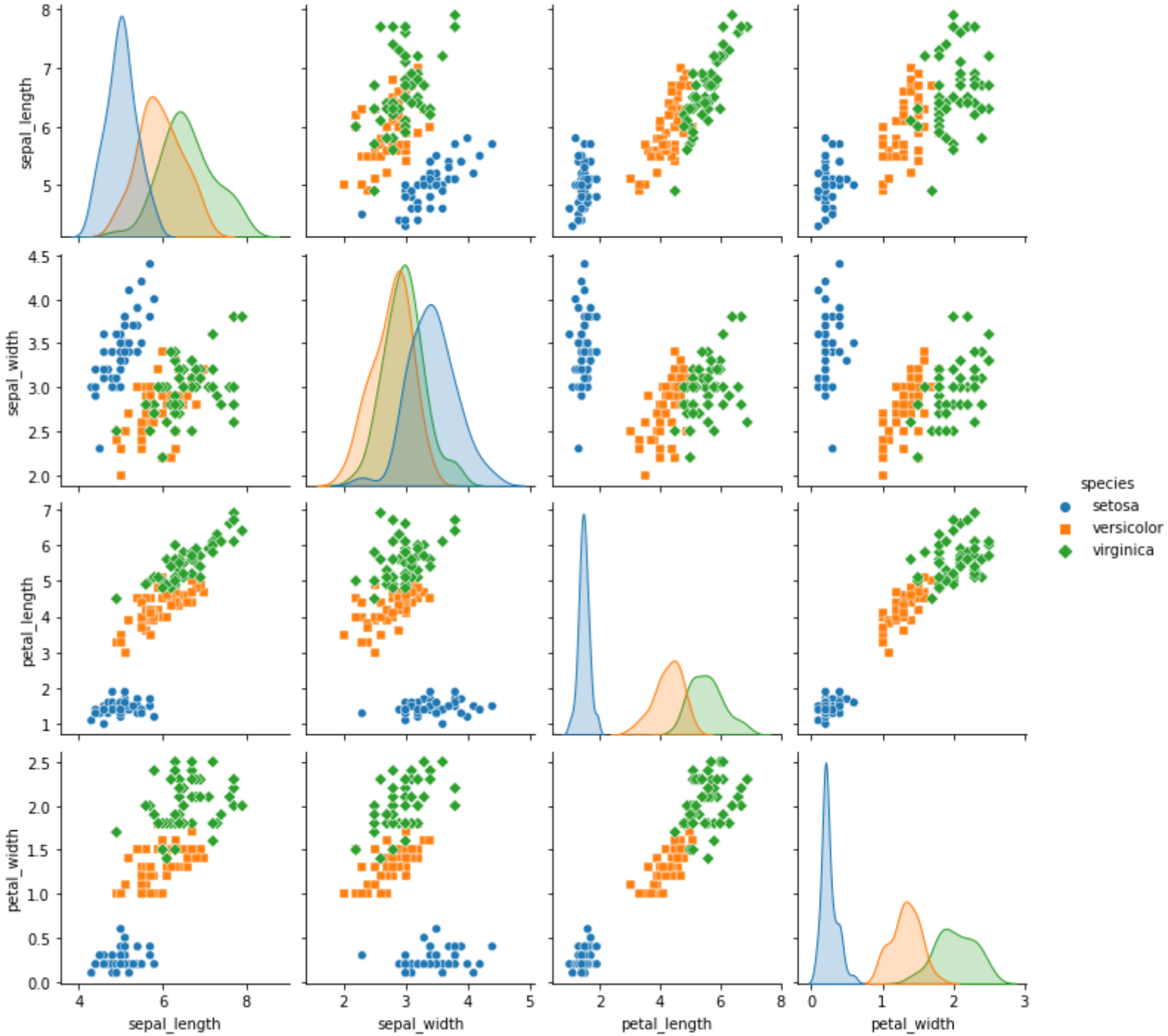
Şimdi bu grafiğe boyut olarak **species** değişkenini ekleyelim:

In [6]:
```
sns.pairplot(df, hue="species");
```



Türleri ayırt etmek için **markers** argümanını kullanalım:

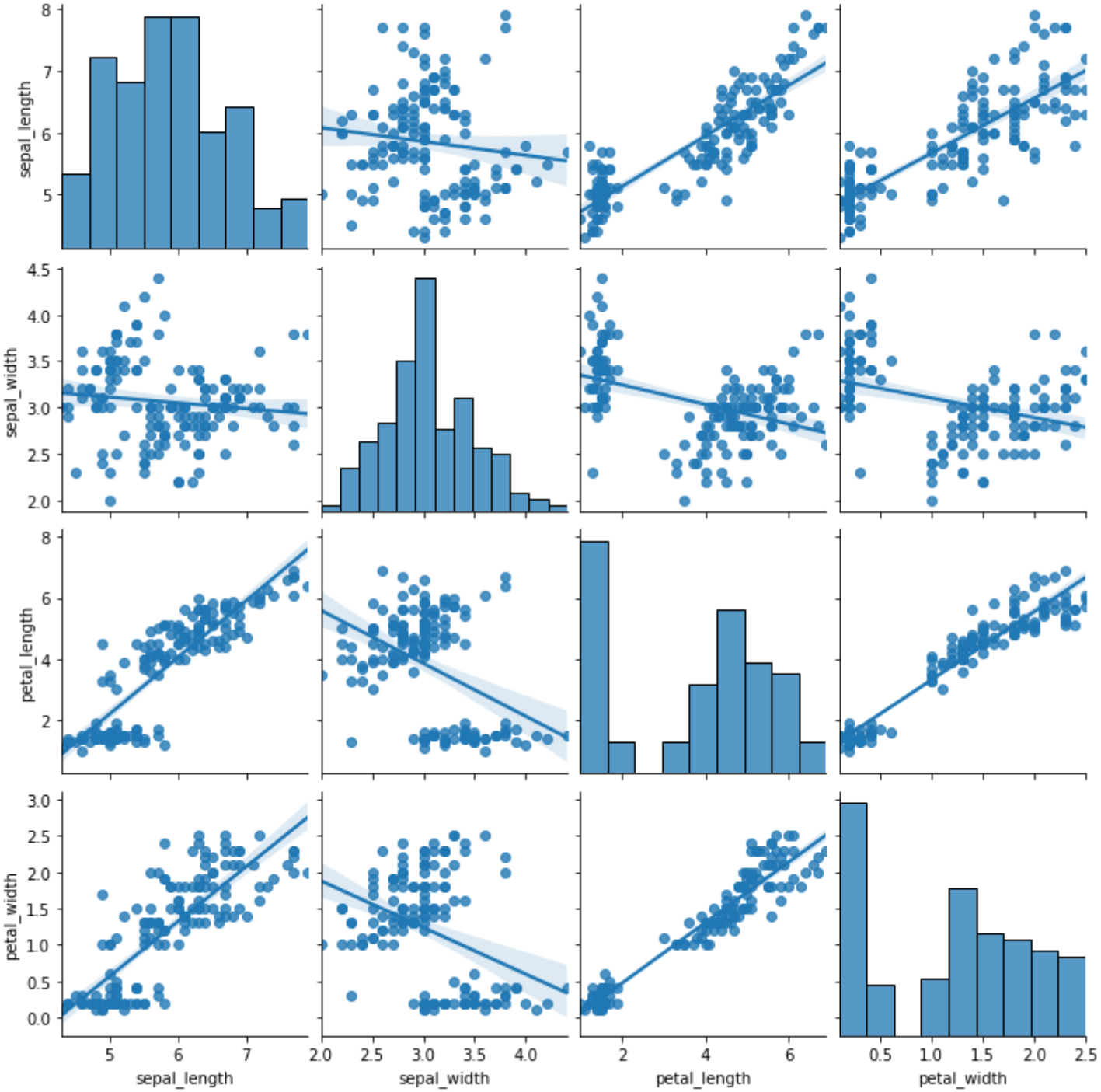
In [7]:
```
sns.pairplot(df, hue="species",markers=["o","s","D"]);
```



Scatter plot grafiklerine bir doğru ekleyelim. Bunun için **kind** argümanını kullanmalıyız.

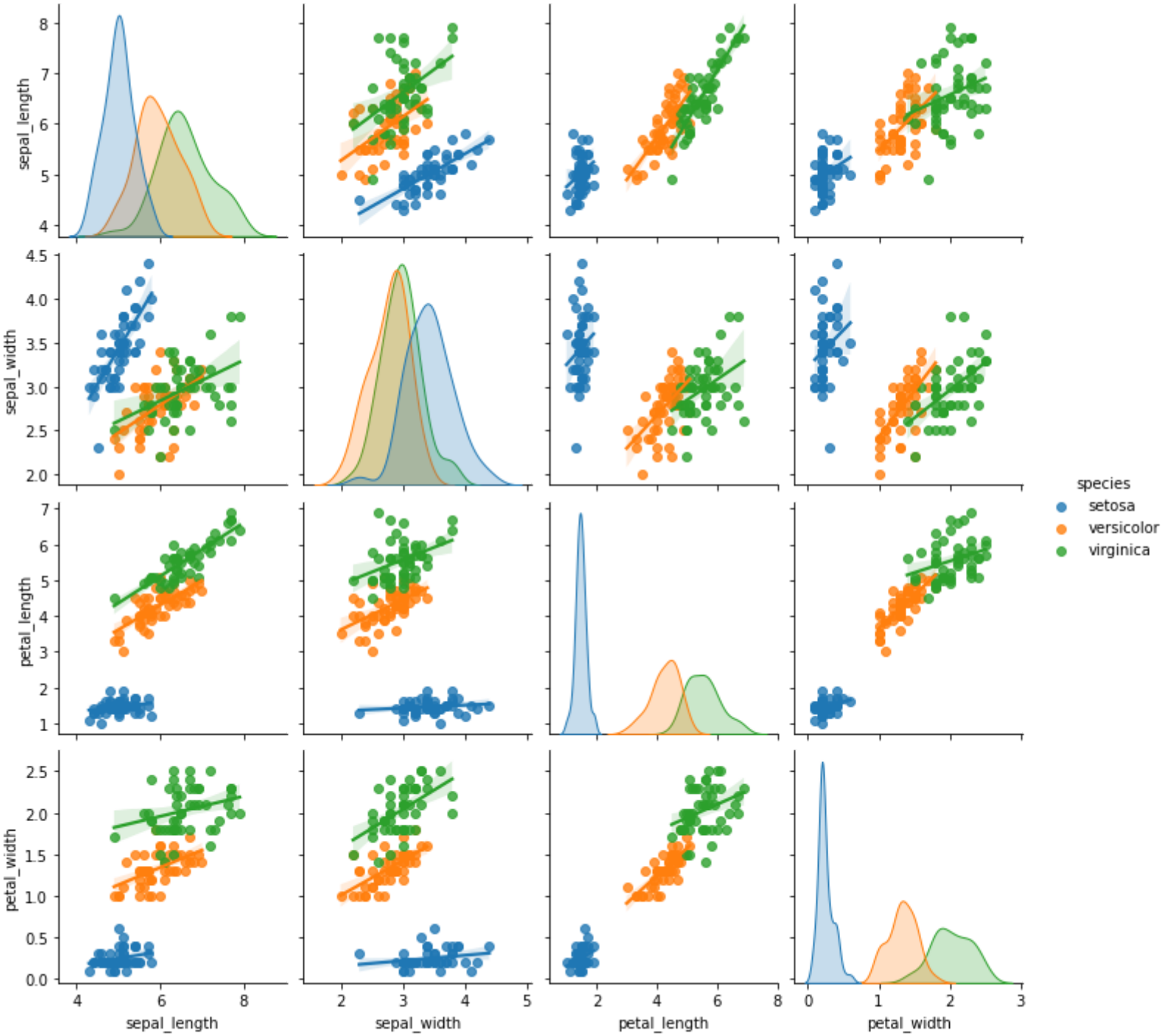
In [8]:

```
sns.pairplot(df, kind="reg");
```



species değişkenini de ekleyelim:

In [9]:
```
sns.pairplot(df, kind="reg",hue="species");
```



Görüldüğü üzere **species** değişkenindeki her bir sınıfa göre eğimi çizmiş oldu.

Örneğin **petal_length** ile **sepal_length** değişkenleri dikkat edilecek olursa **versicolor** ve **virginica** sınıfları için pozitif yönde kuvvetli bir ilişki söz konusudur. Fakat **setosa** sınıfı için bu durum söylenemez.

Isı Haritası (Heat Map)

- Isı haritası, elimizdeki değişkenleri biraz daha yapısal anlamda daha geniş perspektiften görmek istediğimizde kullanılabilecek olan grafik görselleştirme tekniklerinden birisidir.
- Uzun vadeli verilerde yani içerisinde zaman serisi verileri olduğunda(yıl, ay, vs.) bu dönemlere karşılık gözlemlenenbilecek olan bazı sayısal değişken değerleri olduğunda ya da bunun dışında daha büyük ölçekli belirli periyotlarla tekrar eden olayları görmek istediğimizde bize çok güzel bilgiler sunan bir grafiktir.
- Sadece zamansal bağlamda değil, eğer elimizde çok sınıflı kategorik değişken ve bunu belirli bir sayısal değişken açısından görselleştirme ihtiyacımız varsa bu durumda da çok işe yaramaktadır.

Veri Seti Hikayesi

Yılların aylara göre toplamda kaç yolcunun uçuş yaptığını gösteren bir veri setini kullanacağız.

In [3]:
```
flights = sns.load_dataset("flights")
df = flights.copy()
df.head()
```

Out[3]:

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

In [4]:
```
df.shape
```

Out[4]:
```
(144, 3)
```

In [5]:
```
df["passengers"].describe()
```

Out[5]:
```
count    144.000000
mean     280.298611
std      119.966317
```

```
min      104.000000
25%      180.000000
50%      265.500000
75%      360.500000
max      622.000000
Name: passengers, dtype: float64
```

Heat map fonksiyonunu kullanabilmemiz için öncelikle bu veri setini **pivot table** şekline dönüştürmemiz gerekiyor:

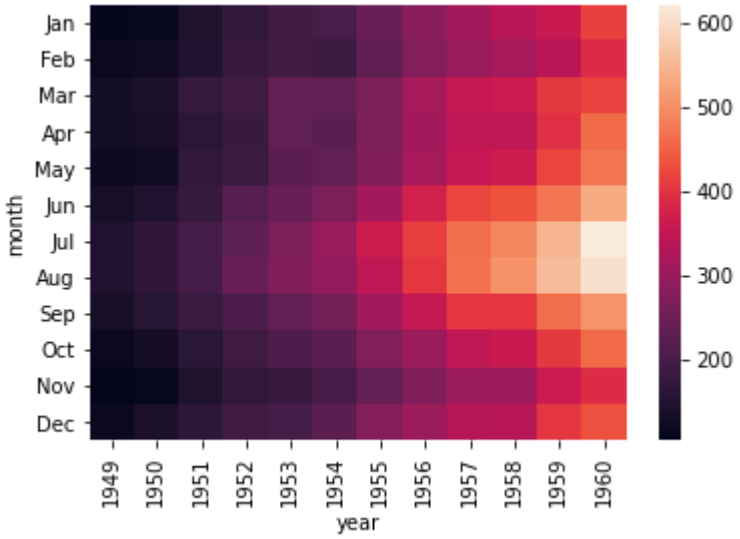
In [6]:
```
df = df.pivot("month", "year", "passengers");
df
```

Out[6]:

month	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
Jan	112	115	145	171	196	204	242	284	315	340	360	417
Feb	118	126	150	180	196	188	233	277	301	318	342	391
Mar	132	141	178	193	236	235	267	317	356	362	406	419
Apr	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
Jun	135	149	178	218	243	264	315	374	422	435	472	535
Jul	148	170	199	230	264	302	364	413	465	491	548	622
Aug	148	170	199	242	272	293	347	405	467	505	559	606
Sep	136	158	184	209	237	259	312	355	404	404	463	508
Oct	119	133	162	191	211	229	274	306	347	359	407	461
Nov	104	114	146	172	180	203	237	271	305	310	362	390
Dec	118	140	166	194	201	229	278	306	336	337	405	432

Veri setimizi pivot table'a dönüştürmüş olduk. Şimdi fonksiyonumuzu kullanabiliriz.

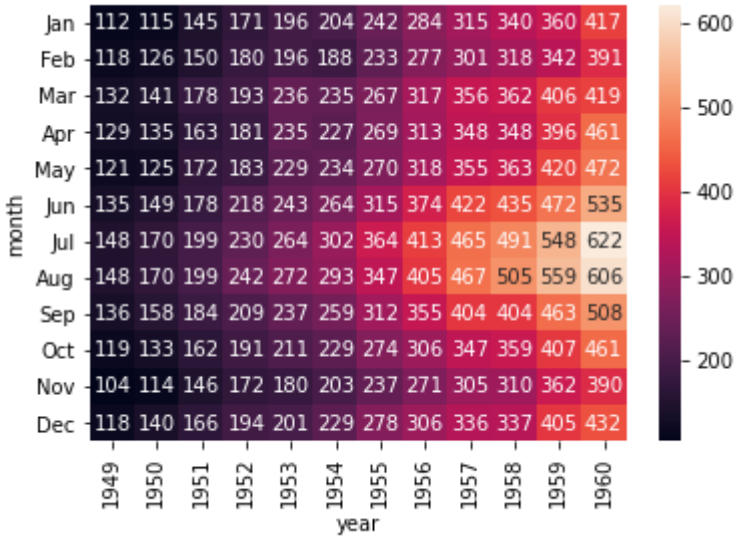
In [8]:
```
sns.heatmap(df);
```



Görüldüğü üzere yıllara göre yolcu sayılarında bir artış gözleniyor. Bunu sağ taraftaki renk ölçeğinden anlayabiliyoruz.
Yorum olarak ise yolcular yaz aylarında daha çok uçuş yapmışlar.

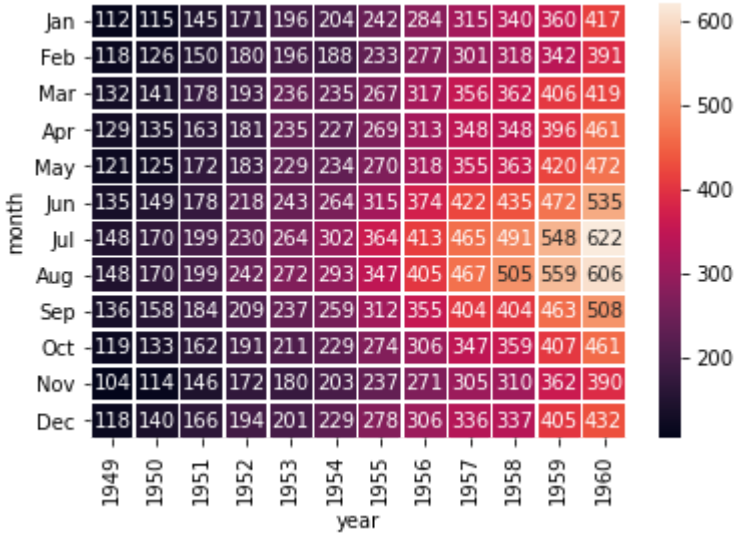
Renklerin içerisine değerleri de yazdırabiliriz. Bunun için **annot** argümanı ve değerlerinin formatını ayarlamak için **fmt** argümanı kullanılır.

In [11]:
```
sns.heatmap(df, annot=True, fmt="d");
```



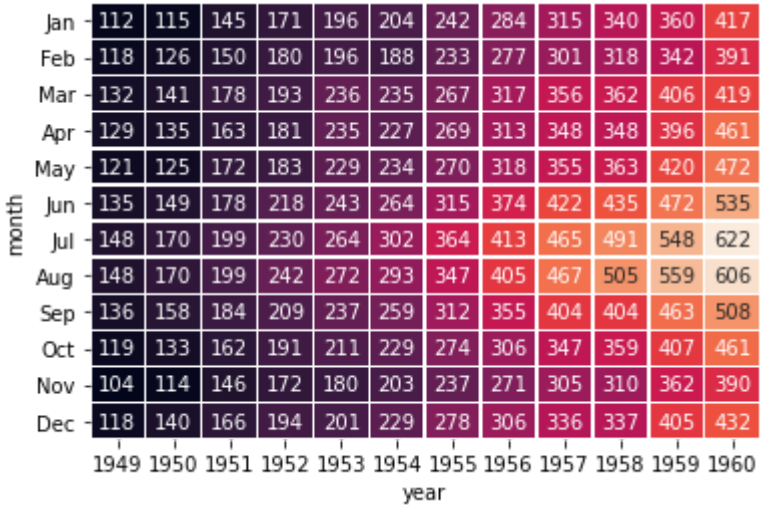
Okunmasını kolaylaştırmak için hücre aralarına boşluk ekleyebiliriz. Bunun için **linewidths** argümanı kullanılır.

In [16]:
```
sns.heatmap(df, annot=True, fmt="d", linewidths=.5);
```



Sağ taraftaki ölçeği de kaldırabiliriz. Bunun için **cbar** argümanı kullanılır.

In [17]:
```
sns.heatmap(df, annot=True, fmt="d", linewidths=.5, cbar=False);
```



Çizgi Grafik (Line Plot)

- Bilinmesi gereken grafik türünden birisidir.
- Diğer grafik türlerine göre daha zor problemlerde kullanılır. Zaman serileri, nesnelerin interneti gibi senaryolarda makinelerin ürettiği verileri görselleştirmek için kullanılır.

Veri Seti Hikayesi

Beyne bağlanan bir cihaz aracılığıyla toplanan sinyalleri ifade eden bir veri seti ile çalışacağız. IoT'de çok sık karşılaşılan bir veri seti türüdür.

subject: verilerin toplandığı kişiler
timepoint: zaman noktaları
event: oluşan olaylar
region: sinyalin toplandığı beyin bölgesi
signal: sinyal şiddeti

In [2]:
```
fmri = sns.load_dataset("fmri")
df = fmri.copy()
df.head()
```

Out[2]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

In [3]:
```
df.shape
```

Out[3]:
```
(1064, 5)
```

Sayısal değişkenleri gözlemleyelim:

In [4]:
```
df["timepoint"].describe()
```

Out[4]:
```
count    1064.000000
mean        9.000000
std         5.479801
min         0.000000
25%         4.000000
50%         9.000000
75%        14.000000
max        18.000000
Name: timepoint, dtype: float64
```

In [5]:
```
df["signal"].describe()
```

Out[5]:
```
count    1064.000000
mean        0.003540
std         0.093930
min        -0.255486
25%        -0.046070
50%        -0.013653
75%         0.024293
max         0.564985
Name: signal, dtype: float64
```

In [6]:
```
df.groupby("timepoint")["signal"].count()
```

Out[6]:
```
timepoint
0     56
1     56
2     56
3     56
4     56
5     56
6     56
7     56
8     56
9     56
10    56
11    56
12    56
13    56
14    56
15    56
16    56
17    56
18    56
Name: signal, dtype: int64
```

Her bir zaman noktasından eşit sayıda sinyal alınmış.

In [7]:
```
df.groupby("signal").count()
```

Out[7]:

signal	subject	timepoint	event	region
-0.255486	1	1	1	1
-0.238474	1	1	1	1
-0.224351	1	1	1	1

signal	subject	timepoint	event	region
-0.181241	1	1	1	1
-0.178510	1	1	1	1
...	...	...	...	...
0.455575	1	1	1	1
0.460896	1	1	1	1
0.476055	1	1	1	1
0.494787	1	1	1	1
0.564985	1	1	1	1

1064 rows × 4 columns

Tam tersi düşünülürse her bir sinyalin değeri beklenildiği üzere eşsiz olarak gözlemleniyor.

In [8]:
```
df.groupby("timepoint")["signal"].describe()
```

Out[8]:

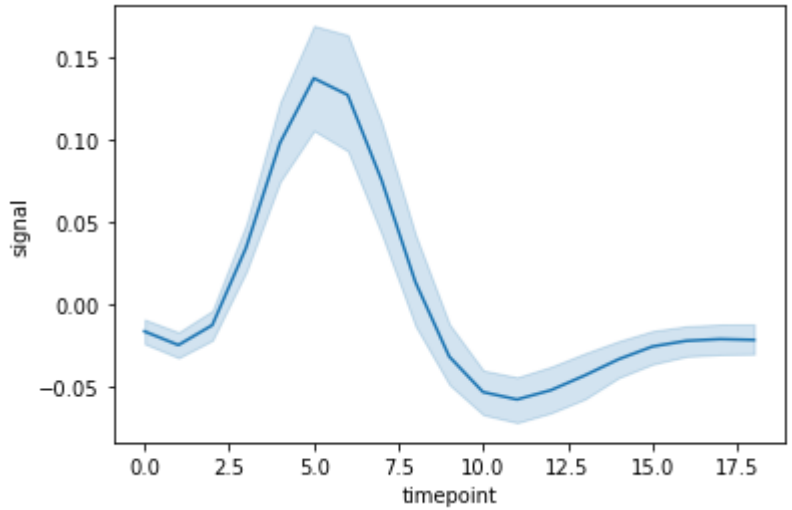
timepoint	count	mean	std	min	25%	50%	75%	max
0	56.0	-0.016662	0.028326	-0.064454	-0.039169	-0.018382	0.003539	0.074399
1	56.0	-0.025002	0.030641	-0.082174	-0.046299	-0.024533	-0.005388	0.063558
2	56.0	-0.012873	0.035440	-0.110565	-0.034944	-0.013183	0.009318	0.077277
3	56.0	0.034446	0.058260	-0.089708	-0.001157	0.028430	0.061840	0.185581
4	56.0	0.098194	0.092838	-0.046347	0.030912	0.070166	0.144911	0.346775
5	56.0	0.137725	0.123353	-0.017946	0.042762	0.096535	0.211638	0.476055
6	56.0	0.127515	0.137332	-0.054405	0.022409	0.068850	0.218919	0.564985
7	56.0	0.075660	0.129704	-0.108222	-0.016252	0.032486	0.144781	0.494787
8	56.0	0.013420	0.104216	-0.181241	-0.049453	-0.012834	0.030396	0.337143
9	56.0	-0.032041	0.072728	-0.152929	-0.075693	-0.038496	0.008717	0.221716
10	56.0	-0.053685	0.053148	-0.176453	-0.078893	-0.052906	-0.015302	0.089231

	count	mean	std	min	25%	50%	75%	max
timepoint								
11	56.0	-0.058194	0.053828	-0.238474	-0.093127	-0.045699	-0.022522	0.030528
12	56.0	-0.052526	0.056991	-0.255486	-0.090391	-0.042294	-0.016239	0.055766
13	56.0	-0.043532	0.053598	-0.224351	-0.069285	-0.031612	-0.012958	0.059510
14	56.0	-0.033660	0.045983	-0.169312	-0.055110	-0.022165	-0.006797	0.050133
15	56.0	-0.025880	0.039092	-0.134828	-0.050536	-0.018207	0.000486	0.047102
16	56.0	-0.022414	0.035035	-0.131641	-0.041122	-0.020777	-0.001380	0.057105
17	56.0	-0.021368	0.034797	-0.121574	-0.042946	-0.017070	-0.000026	0.073757
18	56.0	-0.021867	0.036322	-0.103513	-0.046781	-0.020225	-0.002821	0.090520

Görüldüğü üzere her bir zaman noktasına göre sinyallerin ortalama ve standart sapmaları değişiyor. Dolayısıyla bazı değişkenlerin bunu etkilediğini fark ediyoruz. Çaprazlamalar ile bunu inceleyelim:

Çizgi Grafiğinin Oluşturulması ve Çaprazlamalar

In [10]:
```
sns.lineplot(x="timepoint", y="signal",data=df);
```



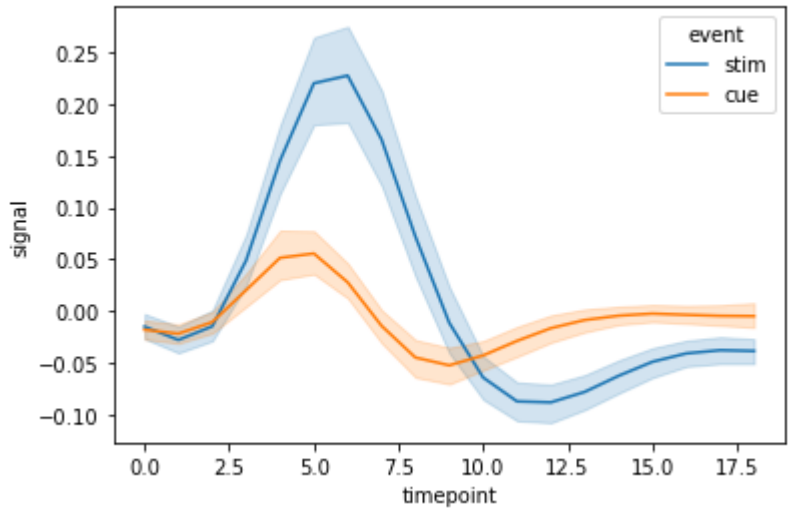
Grafiği yorumlayarak olursak:

- Artış sonra azalış şeklinde devam ediyor.

- Kalın çizgiler timepoint'e göre ortalama sinyaller, etrafındaki açık maviler ise standart sapmayı temsil ediyor. Yani her bir kişi için 0-18 arasındaki zaman noktasından 56 tane sinyal alınmış, her bir zaman noktası için ise ortalama ve standart sapma çizdirilmiştir.
- Seaborn kütüphanesi genel olarak bu şekilde grafikler çizdiriyor ve karmaşık işlemler yapmadan bize yardımcı oluyor.

Boyut eklemeden ilk haliyle grafiği çizdirmek, yorumlama açısından yanıltıcı olacağını söylemiştik. Bu yüzden boyut olarak **event** değişkenini bir ekleyelim:

In [11]:
```
sns.lineplot(x="timepoint", y="signal", hue="event",data=df);
```

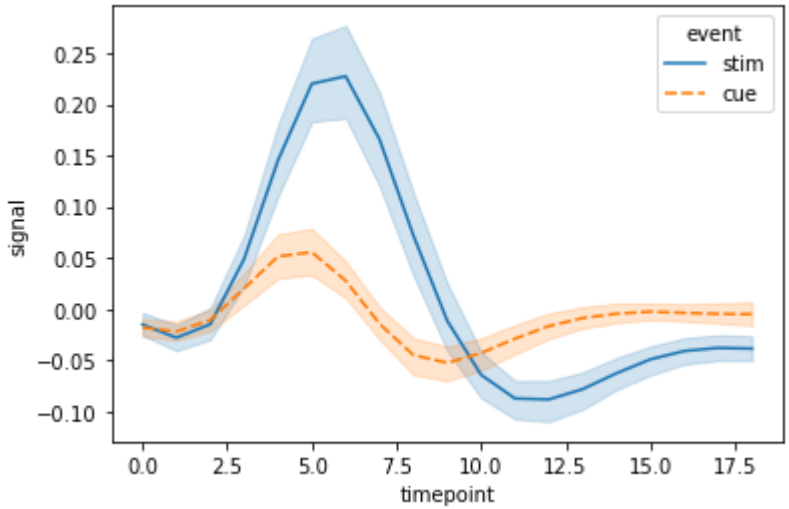


Görüldüğü üzere çok güzel bir bilgi sunuldu. Yüksek değerleri aslında **stim** sınıfı oluşturuyormuş.

Uyarı: Unutulmamalıdır ki bu ***yapısal*** bir veri setidir. Yani insan ilişkileri veya kurumsal bazı ilişkilerde neden-sonuç çıkarımı çok da yapılamayan fakat mekanik, teknik olarak yorum yapılabilen bir veri setidir.

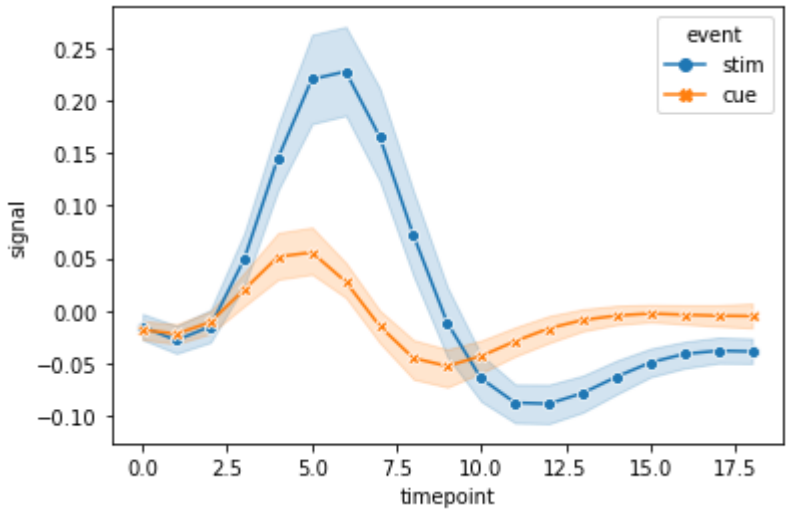
Çizgileri ayırt etmek için `style` argümanına **event** değişkenini ekleyelim.

In [12]:
```
sns.lineplot(x="timepoint", y="signal", hue="event",style="event",data=df);
```



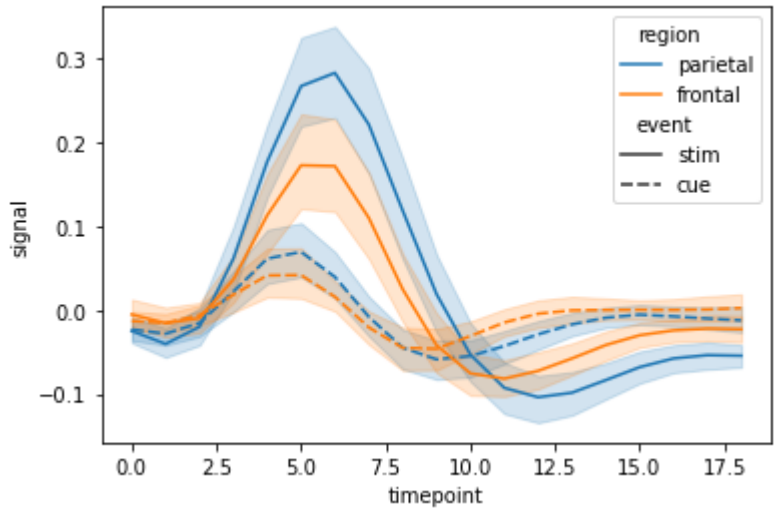
Şimdi ise ortalamaları ifade eden noktaları belirginleştirmek isteyelim. Bunun için `markers` argümanını **True** yapmalıyız. Diğer sınıflardaki kesik çizgileri kaldırmak için ise `dashes` argümanını **False** yapmalıyız.

In [21]:
```
sns.lineplot(x="timepoint",
             y="signal",
             hue="event",
             style="event",
             markers=True,
             dashes=False,
             data=df);
```



Şimdi ise **region** değişkenini yeni bir boyut olarak ekleyelim:

In [23]:
```
sns.lineplot(x="timepoint",
             y="signal",
             hue="region",
             style="event",
             data=df);
```



Görüldüğü üzere **event** kategorileri arasında çok fazla boşluk olarak gözlemlerken **region** değişkenini eklediğimizde aralarında çok boşluk olmadığını görüyoruz.

Basit Zaman Serisi Grafiği

Basit Zaman Serisi Grafiğinin Oluşturulması

Bu derste kullanacağımız veri setine erişmek için bir kütüphaneye ihtiyacımız var. Öncelikle onu indirelim:

In [1]:
```
!pip install pandas_datareader
import pandas_datareader as pr
```

```
Collecting pandas_datareader
  Downloading pandas_datareader-0.10.0-py3-none-any.whl (109 kB)
Requirement already satisfied: pandas>=0.23 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from pandas_datareader) (1.1.5)
Requirement already satisfied: requests>=2.19.0 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from pandas_datareader) (2.25.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from pandas>=0.23->pandas_datareader) (202
0.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from pandas>=0.23->pandas_datare
ader) (2.8.1)
Requirement already satisfied: numpy>=1.15.4 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from pandas>=0.23->pandas_datareader) (1.
19.2)
Requirement already satisfied: six>=1.5 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from python-dateutil>=2.7.3->pandas>=0.23->pan
das_datareader) (1.15.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from requests>=2.19.0->pandas_datare
```

```
ader) (2021.5.30)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from requests>=2.19.0->pandas_datarea
der) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from requests>=2.19.0->pandas_datareader)
(2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\ertug\anaconda3\envs\tf\lib\site-packages (from requests>=2.19.0->pandas_dat
areader) (1.26.2)
Collecting lxml
  Downloading lxml-4.7.1-cp38-cp38-win_amd64.whl (3.7 MB)
Installing collected packages: lxml, pandas-datareader
Successfully installed lxml-4.7.1 pandas-datareader-0.10.0
```

Bu veri seti Apple'ın borsadaki zamana bağlı hisse senedi değerleri olacak.

In [2]:
```
df = pr.get_data_yahoo("AAPL", start="2016-01-01", end="2019-08-25")
```

Zamana bağlı bir veri seti olduğu için başlangıç tarihini ve bitiş tarihini girdik.

In [3]:
```
df.head()
```

Out[3]:

Date	High	Low	Open	Close	Volume	Adj Close
2016-01-04	26.342501	25.500000	25.652500	26.337500	270597600.0	24.251436
2016-01-05	26.462500	25.602501	26.437500	25.677500	223164000.0	23.643711
2016-01-06	25.592501	24.967501	25.139999	25.174999	273829600.0	23.181017
2016-01-07	25.032499	24.107500	24.670000	24.112499	324377600.0	22.202667
2016-01-08	24.777500	24.190001	24.637501	24.240000	283192000.0	22.320068

In [4]:
```
df.shape
```

Out[4]:
```
(917, 6)
```

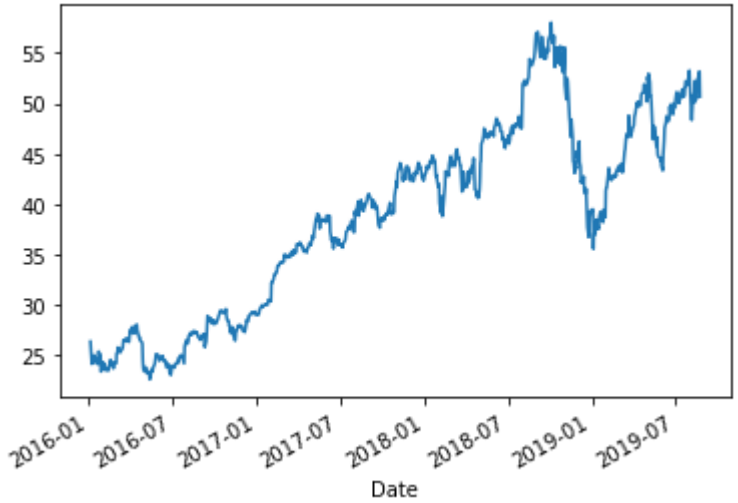
In [5]:
```
kapanis = df["Close"]
```

In [6]:
```
kapanis.head()
```

Out[6]:
```
Date
2016-01-04    26.337500
```

```
2016-01-05    25.677500
2016-01-06    25.174999
2016-01-07    24.112499
2016-01-08    24.240000
Name: Close, dtype: float64
```

In [8]:
```
kapanis.plot();
```



Not: Normalde bizim açımızdan ilgili değişken zaman değişkeni olarak gözükürken bilgisayar açısından bu kategorik değişken veya normal bir index olarak gözükebilir. Bu yüzden kodsal olarak zaman değişkeni olduğunu bilgisayara söylememiz gerekebilir.

In [9]:
```
kapanis.index
```

Out[9]:
```
DatetimeIndex(['2016-01-04', '2016-01-05', '2016-01-06', '2016-01-07',
               '2016-01-08', '2016-01-11', '2016-01-12', '2016-01-13',
               '2016-01-14', '2016-01-15',
               ...
               '2019-08-12', '2019-08-13', '2019-08-14', '2019-08-15',
               '2019-08-16', '2019-08-19', '2019-08-20', '2019-08-21',
               '2019-08-22', '2019-08-23'],
              dtype='datetime64[ns]', name='Date', length=917, freq=None)
```

Zaten zaman değişkeni olarak tanımlanmış. Eğer **tanımlanmasaydı** nasıl yapıldığına bir bakalım:

In [11]:
```
kapanis.index = pd.DatetimeIndex(kapanis.index)
```

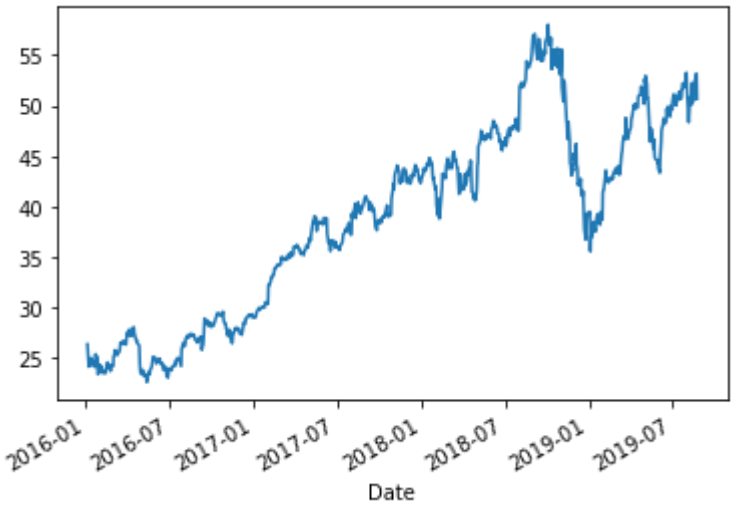
In [12]:
```
kapanis.index
```

Out[12]:
```
DatetimeIndex(['2016-01-04', '2016-01-05', '2016-01-06', '2016-01-07',
               '2016-01-08', '2016-01-11', '2016-01-12', '2016-01-13',
               '2016-01-14', '2016-01-15',
               ...
               '2019-08-12', '2019-08-13', '2019-08-14', '2019-08-15',
               '2019-08-16', '2019-08-19', '2019-08-20', '2019-08-21',
               '2019-08-22', '2019-08-23'],
              dtype='datetime64[ns]', name='Date', length=917, freq=None)
```

In [13]:
```
kapanis.head()
```

Out[13]:
```
Date
2016-01-04    26.337500
2016-01-05    25.677500
2016-01-06    25.174999
2016-01-07    24.112499
2016-01-08    24.240000
Name: Close, dtype: float64
```

In [14]:
```
kapanis.plot();
```



Görüldüğü üzere bir problem yok.

Özet

- Dağılım Grafikleri
- Korelasyon Grafikleri
- Çizgi Grafik

- Basit Zaman Serisi

Dağılım Grafikleri

Değişkenleri tek başına incelemek için kullanılır.

- Kategorik Değişkenler için: **Sütun Grafiği (Bar Plot)**
- Sayısal Değişkenler için: **Histogram, Kutu Grafik (Box Plot) ve Violin Grafiği**

Korelasyon Grafikleri

İki **sayısal değişkeni** bir arada incelemek için kullanılır.

Çizgi Grafik

Nesnelerin interneti (IoT) çalışmalarında veya makinelerin ürettiği biraz daha yapısal, insan davranışları veya kurum davranışlarından ziyade biraz daha mekanik veri setleri olduğunda bunlari görselleştirmek için kullanılır.

Basit Zaman Serisi

Zamana bağlı grafikleri (saat, tarih vs.) görselleştirmek için kullanılır.

Vakit ayırdığınız için teşekkür ederim. ☺