1. **Basit Rastgele Örnekleme (Simple Random Sampling):**

Bu yöntem, popülasyondan tamamen **rastgele** örnekler alır ve her birimin seçilme olasılığı eşittir.

Ne Zaman Kullanılır?

- Eğer popülasyonun her biriminden eşit bir şekilde temsil almak istiyorsanız, bu yöntemi kullanabilirsiniz.
- Örneğin, bir okuldaki tüm öğrencilerden rastgele bir grup seçmek isterseniz, her öğrenciye eşit bir seçilme şansı verirseniz, bu basit rastgele örneklemedir.

Avantajları:

- Kolay uygulanabilir.
- Bütün popülasyonun eşit şekilde temsil edilmesini sağlar.

Dezavantajları:

- Büyük bir popülasyona sahipseniz, örneklem almak zorlaşabilir.
- Popülasyona tüm veriye ulaşmanın mümkün olmadığı durumlarda, bazı grupların temsili düşük olabilir.

Python Örneği:

```
import pandas as pd

# %10 oranında rastgele örnekleme
simple_random_sample = df_coffee.sample(frac=0.1, random_state=2021)

# Sonuçları görüntüleme
print(simple_random_sample.head())
```

Bu örnek, **df_coffee** veri çerçevesinden **%10** oranında tamamen **rastgele** bir örnek alır.

---

2. **Tabakalı Örnekleme (Stratified Sampling):**

Bu yöntemde, popülasyon belirli gruplara veya **katmanlara (tabakalara)** ayrılır ve her gruptan belirli oranda örnek alınır. Her grup homojen olmalı ve belirli bir özelliğe göre gruplandırılmalıdır.

Ne Zaman Kullanılır?

- Eğer belirli bir grubun **eşit temsili** gerektiği durumlarda kullanılır. Örneğin, ülkeler arasındaki dengeli bir dağılım sağlamak için kullanılabilir.

Avantajları:

- Her grup homojen şekilde temsil edilir, böylece her grup hakkında daha doğru sonuçlar elde edilir.
- Düşük varyans sağlar ve daha güvenilir sonuçlar elde edilebilir.

Dezavantajları:

- Grupların nasıl tanımlanacağı ve her grup için ne kadar örnek alınacağı gibi kararlar karmaşık olabilir.

Python Örneği:

```
# Veri setini ülkelere göre tabakalara ayırıp, her ülkeden %10
oranında rastgele örnek alıyoruz
stratified_sample =
df_coffee.groupby('country_of_origin').sample(frac=0.1,
random_state=2021)

# Sonuçları görüntüleme
print(stratified_sample['country_of_origin'].value_counts())
```

Bu kod, **country_of_origin** sütununa göre **gruplama** yapar ve her gruptan **%10** oranında rastgele örnekler alır. Bu yöntem, her bir ülkenin eşit temsil edilmesini sağlar.

---

3. **Kümeleme Örneklemesi (Cluster Sampling)**:

Kümeleme örneklemesinde, popülasyon **küme** adı verilen küçük gruplara ayrılır ve bu kümelerden bazıları rastgele seçilir. Seçilen kümelerdeki **tüm veriler** analiz edilir. Yani, örnekleme bir kümeyle sınırlıdır.

Ne Zaman Kullanılır?

- Eğer tüm popülasyonu incelemek yerine, popülasyonu küçük kümelere ayırarak bazı kümeler üzerinden analiz yapmayı tercih ediyorsanız kullanılır.
- Büyük ve dağıtık popülasyonlarda maliyet ve zaman tasarrufu sağlamak için ideal bir yöntemdir.

Avantajları:

- Kümeleme örneklemesi, maliyet ve zaman tasarrufu sağlar.
- Özellikle büyük veri kümelerinde daha verimli olabilir.

Dezavantajları:

- Eğer kümeler **homojen değilse**, temsil gücü azalabilir. Yani, kümeler arasında büyük farklar varsa, bazı kümeler yeterince iyi temsil etmeyebilir.

Python Örneği:

```
import random

# Kahve çeşitleri arasından rastgele 3 çeşit seçiyoruz
varieties = df_coffee['variety'].unique()
selected_varieties = random.sample(list(varieties), k=3)

# Sadece seçilen kümelere ait verileri filtreliyoruz
cluster_sample =
```

```
df_coffee[df_coffee['variety'].isin(selected_varieties)]

# Sonuçları görüntüleme
print("Seçilen çeşitler:", selected_varieties)
print(cluster_sample.head())
```

Bu örnek, **variety** sütunundaki **benzersiz çeşitleri** rastgele seçer ve sadece seçilen kümelerdeki (çeşitlerdeki) tüm verileri alır.

Özet Karşılaştırma:

Örnekleme Türü	Ne Zaman Kullanılır?	Avantajları	Dezavantajları
Basit Rastgele Örnekleme	Popülasyonun rastgele seçildiği durumlar	Kolay, her birime eşit şans	Büyük popülasyonlar için zor olabilir, temsili düşük olabilir
Tabakalı Örnekleme	Grupların eşit temsil edilmesi gerektiğinde	Her gruptan orantılı örnek alınır	Grupları tanımlamak ve eşit dağılım zor olabilir
Kümeleme Örneklemesi	Tüm popülasyon yerine bazı gruplar üzerinden çalışmak istendiğinde	Maliyet ve zaman tasarrufu sağlar	Küme homojen değilse temsil gücü azalabilir

Özetle:

- **Basit Rastgele Örnekleme**, **tüm popülasyonu rastgele seçmek** için kullanılır ve en basit yöntemdir.
- **Tabakalı Örnekleme**, belirli **grupların eşit temsilini sağlamak** için kullanılır.
- **Kümeleme Örneklemesi**, **gruplar üzerinden çalışmak** isteyen durumlarda tercih edilir, genellikle **zaman ve maliyet tasarrufu** sağlar.

Her üç yöntem de farklı veri analizleri ve araştırmalar için avantajlı olabilir, hangi yöntemi kullanacağınız ise araştırma sorunuza ve veri kümenizin yapısına bağlıdır.

Standart Hata Nedir?

Standart hata (Standard Error, SE), bir örneklem ortalamasının popülasyon ortalamasına ne kadar yakın olduğunu gösteren bir ölçüdür. Başka bir deyişle, örneklemden elde edilen ortalamaların **popülasyon ortalamasına ne kadar yakın** olduğunu gösterir. Bu, örneklem verilerinin ne kadar tutarlı ve güvenilir olduğunu anlamamıza yardımcı olur.

Standart hata, **örneklem ortalamalarının standart sapması** olarak da düşünülebilir ve genellikle şu formülle hesaplanır:

$$SE = \frac{\sigma}{\sqrt{n}}$$

- **(\sigma)**: Popülasyonun standart sapması (veya örneklemdeki veriler için örneklem standart sapması).
- **(n)**: Örneklem büyüklüğü.

Standart hata, örneklem büyüklüğüyle ters orantılıdır. Yani, örneklem büyüklüğü arttıkça, standart hata azalır, çünkü daha büyük örneklemler daha güvenilir ortalamalar sağlar.

Özetle: Standart hata, örneklemden elde edilen ortalamaların ne kadar değişken olduğunu ve popülasyon ortalamasına ne kadar yakın olduğunu ölçer.

Göreli Hata Nedir?

Göreli hata (Relative Error), bir ölçümdeki hatanın, gerçek (doğru) değere oranıdır. Bu, hatanın büyüklüğünün doğru değere ne kadar yakın olduğunu gösteren bir oran veya yüzde değeridir. Göreli hata, genellikle **hatayı doğru değere böler** ve yüzde olarak ifade edilir.

Göreli hata şu şekilde hesaplanır:

$$\text{Göreli Hata} = \frac{|\text{Gerçek Değer} - \text{Ölçülen Değer}|}{|\text{Gerçek Değer}|}$$

Burada:

- **Gerçek Değer**: Doğru kabul edilen değer (örneğin, popülasyon ortalaması).
- **Ölçülen Değer**: Elde edilen örnekleme göre ölçülen değer (örneğin, örneklem ortalaması).

Özetle: Göreli hata, hatanın büyüklüğünü, gerçek değere göre ne kadar büyük olduğunu gösteren bir orandır. Bu oran, genellikle yüzde olarak ifade edilir.

Standart Hata ve Göreli Hata Arasındaki Farklar:

Özellik	Standart Hata (SE)	Göreli Hata
Tanım	Bir örneklem ortalamasının popülasyon ortalamasına ne kadar yakın olduğunu gösterir.	Ölçülen değerin, gerçek değere göre ne kadar uzak olduğunu gösterir.
Hesaplama	$\frac{\sigma}{\sqrt{n}}$ (popülasyon standart sapması bölü karekök örneklem büyüklüğü)	$\frac{\lvert\text{Gerçek Değer} - \text{Ölçülen Değer}\rvert}{\lvert\text{Gerçek Değer}\rvert}$
Kullanım Alanı	Örneklemden elde edilen sonuçların güvenilirliğini gösterir.	Hata oranını ve ölçüm doğruluğunu gösterir.
Birim	Aynı birime sahiptir (örneğin, kilogram, metre, vb.).	Genellikle bir oran ya da yüzde (%) olarak ifade edilir.
Bağlantılı	Örneklem büyüklüğü (büyüdükçe	Gerçek ve ölçülen değer arasındaki

Özellik	Standart Hata (SE)	Göreli Hata
Olduğu Faktörler	standart hata azalır).	fark.

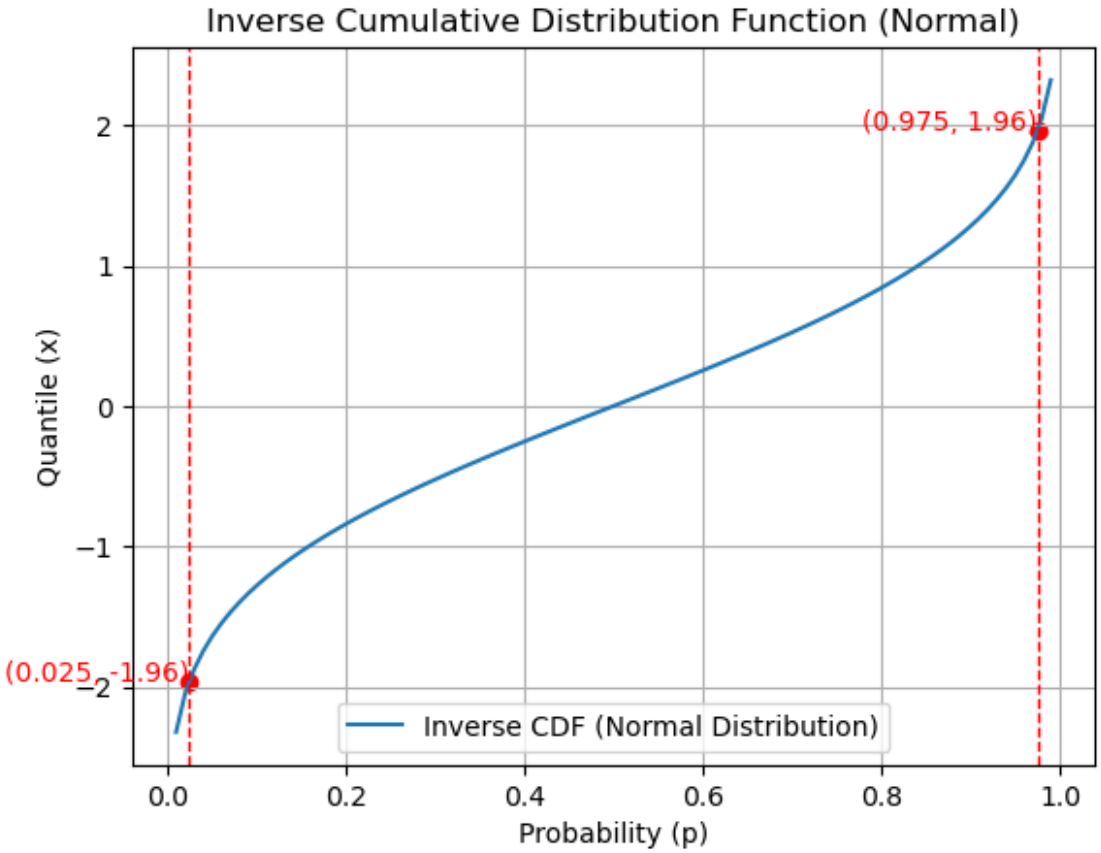
Özet:

- **Standart hata**, bir örneklem ortalamasının ne kadar değişken olduğunu ve popülasyon ortalamasına ne kadar yakın olduğunu gösterir.
- **Göreli hata**, ölçülen bir değerin, gerçek değere olan oranını ifade eder ve hatanın büyüklüğünü anlamamıza yardımcı olur.

Genel olarak, standart hata örneklem büyüklüğü ve verinin dağılımıyla ilgilidir, göreli hata ise ölçümün doğruluğu ve hatasının gerçek değere oranıyla ilgilidir.

```
from scipy.stats import norm

p_values = np.linspace(0, 1, 100)

quantiles = norm.ppf(p_values, loc=0, scale=1) #verilen olasılık
değerlerine karşılık gelen kuantil (quantile) değerlerini hesaplar.
Burada loc=0 ve scale=1 parametreleri, standart normal dağılım
(ortalama 0, standart sapma 1) kullanılarak hesaplama yapılmasını
sağlar.quantiles listesi, her bir olasılık değeri için karşılık gelen
normal dağılımdaki kuantil değerlerini tutar.

plt.plot(p_values, quantiles, label="Inverse CDF (Normal
Distribution)")

p1, p2 = 0.025, 0.975
quantile_025 = norm.ppf(p1)
quantile_975 = norm.ppf(p2)

plt.scatter([p1, p2], [quantile_025, quantile_975], color='red')  #
Noktaları çiz
plt.axvline(p1, color='red', linestyle='dashed', linewidth=1)  # p1
için dikey çizgi
plt.axvline(p2, color='red', linestyle='dashed', linewidth=1)  # p2
için dikey çizgi

# X ve Y değerlerini grafikte gösterelim
plt.text(p1, quantile_025, f'({p1:.3f}, {quantile_025:.2f})',
color='red', fontsize=10, ha='right')
plt.text(p2, quantile_975, f'({p2:.3f}, {quantile_975:.2f})',
color='red', fontsize=10, ha='right')



plt.xlabel("Probability (p)")
plt.ylabel("Quantile (x)")
```

```
plt.title("Inverse Cumulative Distribution Function (Normal)")
plt.grid(True)
plt.legend()
plt.show()
```



Bu tür hipotez testlerini seçerken hangi testin kullanılacağını anlamak için **bağımlı ve bağımsız değişkenlerin türünü** ve **veri tipini** dikkate almak önemlidir. İşte bu testleri nasıl seçeceğinize dair bazı kısa yollar:

1. T-Testi (T-Statistiği):

- **Kullanım Durumu**: İki bağımsız grup arasında **sürekli** bir değişkenin **ortalama farkı** olup olmadığını test etmek için kullanılır.
- **Durum**: Bağımlı değişken **sürekli**, bağımsız değişken **iki grup** (kategorik) olmalı.
- **Örnek**: Erkekler ve kadınlar arasındaki maaş farkı. ünkü burada aynı öğrencilerden alınan iki ölçüm (önceki sınav puanları ve final sınavı puanları) arasındaki farkı test edeceğiz.

  Alternatif: Eğer verileriniz normal dağılmıyorsa, **Mann-Whitney U testi** kullanılabilir.

2. ANOVA (Varyans Analizi):

- **Kullanım Durumu**: Üç veya daha fazla grubun ortalama farklarını test etmek için kullanılır.
- **Durum**: Bağımlı değişken **sürekli**, bağımsız değişken **iki veya daha fazla grup** (kategorik) olmalı.
- **Örnek**: Farklı eğitim yöntemlerinin öğrenci başarısı üzerindeki etkisi.

  Alternatif: Eğer verileriniz normal dağılmıyorsa, **Kruskal-Wallis testi** kullanılabilir.

3. Pairwise Test (Çiftli Karşılaştırma Testi):

- **Kullanım Durumu**: Gruplar arası **ikili karşılaştırmalar** yapmak için kullanılır.
- **Durum**: ANOVA testinden sonra, eğer ANOVA **farklılık gösterdiyse**, hangi grupların farklı olduğunu görmek için kullanılır.
- **Örnek**: Üç eğitim yöntemi arasında hangi ikisinin daha başarılı olduğunu test etmek.

4. Bonferroni Düzeltmesi (Bonferroni Correction):

- **Kullanım Durumu**: Birden fazla karşılaştırma yaparken **tip I hata** riskini azaltmak için kullanılır.
- **Durum**: Çiftli karşılaştırmalar yapıyorsanız ve p-değeri çok küçükse, **Bonferroni düzeltmesi** uygulanabilir.
- **Örnek**: Birden fazla grup arasındaki farkları test etmek (ANOVA sonrası).

5. One-Sample Proportion Test (Tek Örneklem Oranı Testi):

- **Kullanım Durumu**: Bir popülasyonun oranının belirli bir değere eşit olup olmadığını test eder.
- **Durum**: Bağımlı değişken **kategorik**, örneklem oranını test ediyoruz.
- **Örnek**: Bir anketin sonucunda, katılımcıların %70'inin belirli bir görüşü benimsediğini test etmek.

6. Two-Sample Proportion Test (İki Örneklem Oranı Testi):

- **Kullanım Durumu**: İki bağımsız grup arasındaki oran farklarını test eder.
- **Durum**: Bağımlı değişken **kategorik**, bağımsız değişken **iki grup** (örneğin erkek ve kadın).
- **Örnek**: Kadınlar ve erkekler arasındaki bir görüş oran farkını test etmek.

7. Ki-Kare Testi (Chi-Square Test):

- **Kullanım Durumu**: **Kategorik verilerde bağımsızlık** veya **beklenen dağılımla gözlemlenen dağılım arasındaki farkı** test etmek için kullanılır.
- **Durum**: Bağımlı ve bağımsız değişkenler **kategorik** olmalı.
- **Örnek**: Eğitim seviyesi ve iş tatmini arasındaki ilişkiyi test etmek.

8. Ki-Kare İyi Uyum Testi (Chi-Square Goodness of Fit Test):

- **Kullanım Durumu**: Gözlemlenen frekansların, beklenen frekanslarla uyumunu test eder.

- **Durum**: Tek bir kategorik değişkenin, beklenen dağılımla ne kadar uyumlu olduğunu test ederiz.
- **Örnek**: Eşit oranlarla dört farklı yanıtın dağılımı.

9. Wilcoxon Testi:

- **Kullanım Durumu**: İki bağımlı grup arasındaki farkı **non-parametrik** olarak test etmek için kullanılır. Bu test, **verilerin normal dağılmadığı durumlarda** kullanılır.
- **Durum**: Bağımlı değişken **sürekli** ve **bağımsız örneklem** ya da **bağımlı örneklem** olmalı.
- **Örnek**: Aynı grup için **önce** ve **sonra** ölçülen değerlerin karşılaştırılması.

10. Mann-Whitney U Testi:

- **Kullanım Durumu**: İki bağımsız grup arasındaki farkı **non-parametrik** olarak test eder.
- **Durum**: Bağımlı değişken **sürekli**, bağımsız değişken **iki grup** (kategorik).
- **Örnek**: Erkekler ve kadınlar arasındaki maaş farkı.

11. Kruskal-Wallis Testi:

- **Kullanım Durumu**: Üç veya daha fazla bağımsız grup arasındaki farkları **non-parametrik** olarak test eder.
- **Durum**: Bağımlı değişken **sürekli**, bağımsız değişken **kategorik** olmalı.
- **Örnek**: Üç farklı eğitim yönteminin başarıya etkisini test etmek.

Kısa Yol Seçimi:

- **Bir grup (sürekli değişken)**: **T-Testi** veya **One-Sample Proportion Testi**.
- **İki grup karşılaştırması (sürekli değişken)**: **T-Testi (bağımsız)** veya **Mann-Whitney U Testi**.
- **İki grup karşılaştırması (oran)**: **Two-Sample Proportion Testi**.
- **Birden fazla grup karşılaştırması (sürekli değişken)**: **ANOVA** veya **Kruskal-Wallis** (veriler normal dağılmıyorsa).
- **Birden fazla grup karşılaştırması (oran)**: **Ki-Kare Testi (bağımsızlık)**.
- **Veri normal dağılmıyorsa**: **Mann-Whitney U Testi**, **Wilcoxon Testi**, **Kruskal-Wallis Testi**.

Bu şekilde, test seçimi yaparken hangi testin uygulanacağını belirlemek çok daha kolay olur.

Test Seçim Rehberi:

- **İki grup karşılaştırması (sürekli değişken)**:
  - **T-Testi**: Veriler normal dağılım gösteriyorsa.
  - **Mann-Whitney U Testi**: Veriler normal dağılmıyorsa veya ordinal ölçek kullanılıyorsa.
- **Bir grup karşılaştırması (oran)**:
  - **One-Sample Proportion Testi**: Bir grubun belirli bir orandan farklı olup olmadığını test etmek için.
- **İki grup karşılaştırması (oran)**:

  - **Two-Sample Proportion Testi**: İki farklı grubun oranlarını karşılaştırmak için.
- **Birden fazla grup karşılaştırması (sürekli değişken)**:
  - **ANOVA**: Veriler normal dağılım gösteriyorsa.
  - **Kruskal-Wallis Testi**: Veriler normal dağılmıyorsa veya ordinal ölçek kullanılıyorsa.
- **Ki-Kare Testi (Kategorik Veriler)**:
  - **Bağımsızlık Testi**: İki kategorik değişken arasındaki ilişkinin bağımsız olup olmadığını test etmek için.
  - **Uyum Testi**: Gözlemlenen frekansların beklenen frekanslarla uyumunu test etmek için.
- **Bir grup karşılaştırması (sürekli değişken)**:
  - **One-Sample T-Test**: Bir grubun ortalamasının bilinen bir değerden farklı olup olmadığını test etmek için.
- **İki grup karşılaştırması (aynı grubun farklı zamanlardaki ölçümleri)**:
  - **Paired T-Test**: Veriler normal dağılım gösteriyorsa.
  - **Wilcoxon İşaretli Sıra Testi**: Veriler normal dağılmıyorsa.

Bu rehber, hangi testi kullanmanız gerektiğine hızlıca karar vermenizi sağlar! 😊

Bu testlerin nasıl seçileceğine dair bir rehber sundum. Bu sayede sınavda hangi testi kullanmanız gerektiğini rahatça belirleyebilirsiniz!

Bağımsız t-testi (Independent t-test): İki bağımsız grubun ortalamalarını karşılaştırmak için kullanılır. Örneğin, erkek ve kız öğrencilerin sınav notlarını karşılaştırmak.

Eşleştirilmiş t-testi (Paired t-test): Aynı grubun farklı zamanlardaki ölçümlerini karşılaştırmak için kullanılır. Örneğin, tedavi öncesi ve sonrası kan basıncı ölçümleri.

Tek örneklem t-testi (One-sample t-test): Bir grubun ortalamasının bilinen bir değerden farklı olup olmadığını test etmek için kullanılır.

ANOVA (Analysis of Variance): Üç veya daha fazla grubun ortalamalarını karşılaştırmak için kullanılır. Örneğin, üç farklı öğretim yönteminin etkilerini karşılaştırmak. Tek örneklem oran testi (One-sample proportion test): Bir grubun belirli bir oranın beklendiğinden farklı olup olmadığını test etmek için kullanılır. Örneğin, bir okulda öğrencilerin yüzde 70'inin sınavı geçmesini beklerken, gerçek geçme oranının bu değerden farklı olup olmadığını test etmek.

İki örneklem oran testi (Two-sample proportion test): İki farklı grubun oranlarını karşılaştırmak için kullanılır. Örneğin, erkek ve kız öğrenciler arasında sınav geçme oranlarını karşılaştırmak.

Ki-kare testi (Chi-square test): Kategorik veriler arasındaki ilişkiyi veya frekansların beklenen değerlerle uyumunu test etmek için kullanılır.

Ki-kare bağımsızlık testi (Chi-square test for independence): İki kategorik değişken arasındaki ilişkinin bağımsız olup olmadığını test etmek için kullanılır. Örneğin, sigara içme alışkanlığı ile yaş grupları arasındaki ilişkiyi test etmek. Ki-kare iyilik uyum testi (Chi-square goodness-of-fit test): Gözlemlenen frekansların beklenen frekanslara uyumunu test eder. Örneğin, bir zarın adil

olup olmadığını test etmek. Mann-Whitney U testi: İki bağımsız grubun medyanlarını karşılaştırmak için kullanılır. Veriler normal dağılmıyorsa ya da ölçek ordinal ise tercih edilir. Örneğin, tedavi ve kontrol gruplarının iyileşme sürelerini karşılaştırmak.

Wilcoxon işaretli sıra testi (Wilcoxon signed-rank test): Aynı grubun farklı zamanlardaki ölçümlerini karşılaştırmak için kullanılır (paired t-test'in parametrik olmayan eşdeğeri). Örneğin, tedavi öncesi ve sonrası ağrı skorlarını karşılaştırmak.

Kruskal-Wallis testi: Üç veya daha fazla bağımsız grubun medyanlarını karşılaştırmak için kullanılır. ANOVA'nın parametrik olmayan eşdeğeridir. Örneğin, üç farklı tedavi yönteminin iyileşme süresine etkisini test etmek.

İstatistiksel Test Seçim Rehberi

Bu rehber, hangi testi kullanmanız gerektiğini hızlıca anlamanıza yardımcı olur. Verilerinizin **grup sayısı**, **değişken tipi** ve **bağımlılık durumu** gibi özelliklerine göre seçim yapabilirsiniz.

---

1. Tek Grup Karşılaştırması

- **Sürekli değişken (ortalama) bilinen bir değerden farklı mı?**
  - **One-Sample T-Test**: Bir grubun ortalamasını sabit bir değere karşılaştırmak için.
    - Örnek: "Bir sınıfın sınav ortalaması 70 mi?"
- **Kategorik değişken (oran) bilinen bir orandan farklı mı?**
  - **One-Sample Proportion Test**: Bir grubun belirli bir orandan farklı olup olmadığını test etmek için.
    - Örnek: "Bir okulda sınavı geçenlerin oranı %80 mi?"

---

2. İki Grup Karşılaştırması

- **İki bağımsız grubun sürekli değişken ortalamalarını karşılaştırmak:**
  - **Two-Sample T-Test (Independent T-Test)**: Gruplar bağımsız ve veri sürekli ise.
    - Örnek: "Erkek ve kız öğrencilerin matematik sınavı ortalamaları farklı mı?"
- **İki bağımsız grubun medyanlarını karşılaştırmak:**
  - **Mann-Whitney U Test (Wilcoxon Rank-Sum Test)**: Veriler normal dağılmıyorsa veya ordinal ise.
    - Örnek: "İki farklı mağazanın müşteri memnuniyeti farklı mı?"
- **İki bağımlı (eşleştirilmiş) ölçümün sürekli değişken ortalamalarını karşılaştırmak:**
  - **Paired T-Test**: Aynı gruptan iki farklı durumda ölçülen veriler.
    - Örnek: "Öğrencilerin ders öncesi ve sonrası stres seviyeleri farklı mı?"
- **İki bağımlı ölçümün medyanlarını karşılaştırmak:**
  - **Wilcoxon Signed-Rank Test**: Veriler normal dağılmıyorsa.
    - Örnek: "Tedavi öncesi ve sonrası ağrı skorları farklı mı?"
- **İki bağımsız grubun kategorik değişken oranlarını karşılaştırmak:**
  - **Two-Sample Proportion Test**: İki grubun oranları karşılaştırılır.
    - Örnek: "Erkek ve kız öğrencilerin sınavı geçme oranları farklı mı?"

---

3. Üç veya Daha Fazla Grup Karşılaştırması

- **Üç veya daha fazla bağımsız grubun sürekli değişken ortalamalarını karşılaştırmak:**
  - **ANOVA (Analysis of Variance)**: Veriler normal dağılım gösteriyorsa.
    - Örnek: "Üç farklı öğretim yönteminin başarı üzerindeki etkisi farklı mı?"
- **Üç veya daha fazla bağımsız grubun medyanlarını karşılaştırmak:**
  - **Kruskal-Wallis Test**: Veriler normal dağılmıyorsa veya ordinal ise.
    - Örnek: "Üç farklı maaş grubunun iş memnuniyeti farklı mı?"

---

4. Kategorik Veri Karşılaştırmaları

- **Kategorik veriler arasındaki ilişkiyi test etmek:**
  - **Chi-Square Test for Independence (Ki-Kare Bağımsızlık Testi)**: İki kategorik değişken bağımsız mı?
    - Örnek: "Sigara içme durumu ile yaş grubu arasında ilişki var mı?"
- **Kategorik verilerin belirli bir dağılıma uyup uymadığını test etmek:**
  - **Chi-Square Goodness-of-Fit Test (Ki-Kare İyilik Uyum Testi)**: Gözlemlenen ve beklenen frekanslar arasında fark var mı?
    - Örnek: "Bir zar adil mi?"

Özet Tablo

Durum	Test
Tek grup - Ortalama (sürekli)	One T-Test
Tek grup - Oran (kategorik)	One-Sample Proportion Test
İki bağımsız grup - Ortalama (sürekli, oran)	Two-Sample T-Test
İki bağımsız grup - Medyan (ordinal)	Mann-Whitney U Test
İki bağımlı grup - Ortalama (sürekli)	ikili T-Test
İki bağımlı grup - Medyan (ordinal)	Wilcoxon Signed-Rank Test
İki bağımsız grup - Oran (kategorik)	Two-Sample Proportion Test
Üç veya daha fazla grup - Ortalama	ANOVA
Üç veya daha fazla grup - Medyan(ordinal)	Kruskal-Wallis Test
İki kategorik değişken ilişkisi	Chi-Square Independence Test
Kategorik veri uyumu	Chi-Square Goodness-of-Fit Test

Bu rehber ile doğru testi seçmek artık çok daha kolay! 😊

```
-> çıktı verileri sayısal ise regresyon kullanılır! doğru çizebilmeye
çalışıyorsun örneklere en yakın doğruyu çizmek
-> çıktıda kategorik değerler varsa sınıflandırma kullanılır.
kategorik değerleri en iyi şekilde birbirinden ayırabilen modeli
çizmeyi çalışır
```

Z SKORU:

Z-skoru, verinin ortalama değerden ne kadar uzak olduğunu gösterir. #z ortalaması 0 sapması 1 olan bir dağılımdır

$$ \[z = \frac{(\mu - X)}{\sigma} \] \(X\) : Gözlemlenen değer \(\mu\) : Ortalama \(\sigma\) : Standart sapma $$

Z-Skoru Sonuçlarının Yorumlanması:

- Z-skoru, verinin ortalama değerden ne kadar uzak olduğunu gösterir.
- Z-skoru **0** ise, veri ortalama ile tam olarak örtüşüyor demektir.
- Z-skoru **pozitif** ise, veri ortalamadan **büyük** demektir.
- Z-skoru **negatif** ise, veri ortalamadan **küçük** demektir.

```
import numpy as np
scores = [70, 80, 85, 90, 95, 100, 105, 110, 115, 120]
beklenen_deger= 95
mean_scores = np.mean(scores) #EĞER LİSTE ŞEKLİNDE VERİLİRSE NUMPY İLE MEAN HESAPLAMAK ZORUNDASIN!
mean_scores
#mean_scores2 = scores.mean()
#mean_scores2
```

```
97.0
```

```
ss = np.std(scores)
ss
```

```
15.198684153570664
```

```
z_score = (beklenen_deger - mean_scores) / ss
z_score
```

```
-0.1315903389919538
```

```
import pandas as pd
import numpy as np
data = {'Scores': [70, 80, 85, 90, 95, 100, 105, 110, 115, 120]}
df = pd.DataFrame(data)
beklenen_deger= 95
meanss = df['Scores'].mean()
std_dev = df['Scores'].std()
z_score2 = (beklenen_deger - meanss) / std_dev
z_score2
```

```
-0.12483755678647183
```

```
import pandas as pd
import numpy as np
data = {'Scores': [70, 80, 85, 90, 95, 100, 105, 110, 115, 120]}
df = pd.DataFrame(data)
```

```
scores2 = df['Scores']
beklenen_deger= 95
meanss = df['Scores'].mean()
std_dev = np.std(scores2, ddof=1) #burası önemli!! hepsinin alnacağını
belirtmek için numpy ile olduğunda ddof = 1 olmalı
z_score2 = (meanss - beklenen_deger) / std_dev
z_score2
```

```
0.12483755678647183
```

```
df_stck = pd.read_feather("data/stack_overflow.feather")
df_stck
```

```
      respondent                                         main_branch
hobbyist  \
0           36.0  I am not primarily a developer, but I write co...
Yes
1           47.0                    I am a developer by profession
Yes
2           69.0                    I am a developer by profession
Yes
3          125.0  I am not primarily a developer, but I write co...
Yes
4          147.0  I am not primarily a developer, but I write co...
No
...          ...                                                ...
...
2256     62812.0                    I am a developer by profession
Yes
2257     62835.0                    I am a developer by profession
Yes
2258     62837.0                    I am a developer by profession
Yes
2259     62867.0  I am not primarily a developer, but I write co...
Yes
2260     62882.0                    I am a developer by profession
Yes

       age  age_1st_code age_first_code_cut comp_freq  comp_total  \
0     34.0          30.0              adult    Yearly     60000.0
1     53.0          10.0              child    Yearly     58000.0
2     25.0          12.0              child    Yearly    550000.0
3     41.0          30.0              adult   Monthly    200000.0
4     28.0          15.0              adult    Yearly     50000.0
...    ...           ...                ...       ...         ...
2256  40.0          10.0              child    Yearly    145000.0
2257  23.0           9.0              child   Monthly    180000.0
2258  27.0           8.0              child   Monthly      7500.0
2259  33.0          13.0              child   Monthly      6000.0
2260  28.0          13.0              child    Yearly    180000.0
```

```
      converted_comp             country  ...          survey_length
trans  \
0            77556.0      United Kingdom  ...  Appropriate in length
No
1            74970.0      United Kingdom  ...  Appropriate in length
No
2           594539.0              France  ...              Too short
No
3          2000000.0       United States  ...  Appropriate in length
No
4            37816.0              Canada  ...  Appropriate in length
No
...              ...                 ...  ...                   ...
...
2256        145000.0       United States  ...               Too long
No
2257         33972.0  Russian Federation  ...              Too short
No
2258         97284.0             Germany  ...  Appropriate in length
No
2259         72000.0              Panama  ...               Too long
No
2260        180000.0       United States  ...  Appropriate in length
No

                                          undergrad_major  \
0     Computer science, computer engineering, or sof...
1     A natural science (such as biology, chemistry,...
2     Computer science, computer engineering, or sof...
3                                                  None
4     Another engineering discipline (such as civil,...
...                                                  ...
2256  Computer science, computer engineering, or sof...
2257  Computer science, computer engineering, or sof...
2258                       Mathematics or statistics
2259  Another engineering discipline (such as civil,...
2260  Computer science, computer engineering, or sof...

           webframe_desire_next_year  \
0                   Express;React.js
1                       Flask;Spring
2                       Django;Flask
3                               None
4                               None
...                              ...
2256                    Flask;jQuery
2257                    ASP.NET Core
2258                            None
2259                            None
```

```
2260  Angular;Express;Flask;React.js

                                webframe_worked_with  \
0                                   Express;React.js
1                                       Flask;Spring
2                                       Django;Flask
3                                               None
4                                     Express;Flask
...                                             ...
2256       Angular;Angular.js;Flask;jQuery;React.js
2257                              ASP.NET Core;Flask
2258                                            None
2259                                 Django;React.js
2260  Angular;Angular.js;Django;Drupal;Express;Flask

                                welcome_change work_week_hrs
years_code  \
0      Just as welcome now as I felt last year          40.0
4.0
1      Just as welcome now as I felt last year          40.0
43.0
2      Just as welcome now as I felt last year          40.0
13.0
3      Just as welcome now as I felt last year          40.0
11.0
4      Just as welcome now as I felt last year          40.0
5.0
...                                          ...           ...         ..
.
2256  Somewhat less welcome now than last year          50.0
30.0
2257   Just as welcome now as I felt last year          60.0
8.0
2258   Just as welcome now as I felt last year          42.0
12.0
2259     A lot less welcome now than last year          45.0
15.0
2260   Just as welcome now as I felt last year          40.0
11.0

     years_code_pro      age_cat
0               3.0  At least 30
1              28.0  At least 30
2               3.0     Under 30
3              11.0  At least 30
4               3.0     Under 30
...             ...          ...
2256           20.0  At least 30
2257            3.0     Under 30
2258            2.0     Under 30
```

```
2259            2.0  At least 30
2260            5.0     Under 30

[2261 rows x 63 columns]
```

Stack Overflow kullanıcıları arasında 'child' kategorisinde olanların oranı gerçekten %35'twn büyük mü? Null Hipotezi (H0): 'child' kategorisinde olan kullanıcıların oranı %35'ten büyüktür

```
first_code_boot_distn = []
for i in range(5000):
    first_code_boot_distn.append(
        np.mean(
            (df_stck.sample(frac=1, replace=True)
             ['age_first_code_cut'] == 'child').mean()
        )
    )

sample_means = (df_stck['age_first_code_cut'] == 'child').mean()
hyp = 0.35
ss = np.std(first_code_boot_distn, ddof=1)
z_score3 = (sample_means - hyp) / ss
z_score3
```

```
4.059655896408994
```

```
#büyüktür dediği için sağ kuyruk testi yapılır
from scipy.stats import norm

pvalue= 1- norm.cdf(z_score3)
pvalue #h0 reddedilir
```

```
2.457254345189508e-05
```

T İSTATİSTİĞİ:

iki örneklem arasındaki ortalama farkının anlamlı olup olmadığını test etmek için kullanılan bir istatistiksel testtir. Bu test, iki grup arasındaki farkın örneklem varyansı ile karşılaştırılmasını sağlar. Özellikle **iki bağımsız grup** için, örneklem büyüklükleri ve standart sapmaları göz önünde bulundurularak hesaplanır.

Formül:

İki bağımsız örneklem için t-istatistiği şu formülle hesaplanır:

$$t = \frac{\dot{x}_1 - \dot{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

- $\acute{x}_1$ ve $\acute{x}_2$: İlk ve ikinci örneklemin ortalamaları
- s_1 ve s_2: İlk ve ikinci örneklemin standart sapmaları
- n_1 ve n_2: İlk ve ikinci örneklemin büyüklükleri İki bağımsız örneklem için kullanılan, klasik t-testindeki **serbestlik derecesi** (degrees of freedom) yaklaşık olarak şu formülle hesaplanır:
  Serbestlik derecesi düşük olduğunda (df=1, 2), t dağılımı daha geniş kuyruklara sahip olur. Yani, daha fazla ağırlık uç değerlerde bulunur. Serbestlik derecesi arttıkça (df=4, 8), t dağılımı normal dağılıma daha çok benzer hale gelir. Normal dağılım, serbestlik derecesi yüksek olan t dağılımına yaklaşır.

$$df = n_1 + n_2 - 2$$

'adult' ve 'child' grupları arasındaki 'converted_comp' adlı değişkenin ortalamaları arasındaki farkın istatistiksel olarak anlamlı olup olmadığını test et. h0 = adult grubu ortalamasının child grubu ortalamasından büyüktür.

```
ort = df_stck.groupby('age_first_code_cut')['converted_comp'].mean()
ort
```

```
age_first_code_cut
adult    111313.311047
child    132419.570621
Name: converted_comp, dtype: float64
```

```
st = df_stck.groupby('age_first_code_cut')['converted_comp'].std()
st
```

```
age_first_code_cut
adult    271546.521729
child    255585.240115
Name: converted_comp, dtype: float64
```

```
n = df_stck.groupby('age_first_code_cut')['converted_comp'].count()
n
```

```
age_first_code_cut
adult    1376
child     885
Name: converted_comp, dtype: int64
```

```
import numpy as np
pay = ort.iloc[1] - ort.iloc[0]
payda = np.sqrt(st.iloc[1]**2 / n.iloc[1] + st.iloc[0]**2 / n.iloc[0])
t_stat = pay /payda
t_stat
```

```
1.8699313316221844
```

```
degrees_of_freedom = n.iloc[1] + n.iloc[0] - 2
degrees_of_freedom
```

```
2259
```

```
from scipy.stats import t
p_deger = 1- t.cdf(t_stat , df=degrees_of_freedom)
p_deger #reddedilir h0i çünkü 0.05 yani alpha değerinden küçüktür.
```

```
0.030811302165157595
```

Tek Örneklem T-Testi

- **Tanım**: Tek örneklem t-testi, bir örneklemin ortalamasının belirli bir teorik veya beklenen değerden (genellikle 0, 100 vb.) anlamlı şekilde farklı olup olmadığını test etmek için kullanılır.

- **Formül** (basit hali):

$$t = \frac{\acute{x} - \mu_0}{\sqrt{\frac{s^2}{n}}}$$

  - (\bar{x}): Örneklem ortalaması
  - (\mu_0): Beklenen/popülasyon ortalaması (hipotezdeki ortalama)
  - (s): Örneklemin standart sapması
  - (n): Örneklem büyüklüğü

Bunda degrees_of_freedom n-1'dir!!! sonrasında sağ ya da sol kuyruk testi yapılır!!

örneğin 2008 ve 2012 verilerinden yaratılan **`diff`** sütunu) tek bir örneklem verisi (farklar) üzerinden, "bu ortalama fark sıfır mı?" hipotezi test ediliyor. h0 = 2008 değerinin ort 2012 değerinin ort'ından eşit ya da küçüktür.

```
import pandas as pd

df_election = pd.read_feather('data/repub_votes_potus_08_12.feather')
sample_data = df_election.copy()
sample_data['diff'] = sample_data['repub_percent_08'] -
sample_data['repub_percent_12']
sample_data
```

```
         state       county  repub_percent_08  repub_percent_12
diff
0      Alabama         Hale         38.957877         37.139882
1.817995
```

```
1     Arkansas       Nevada         56.726272         58.983452  -
2.257179
2   California         Lake         38.896719         39.331367  -
0.434648
3   California      Ventura         42.923190         45.250693  -
2.327503
4     Colorado      Lincoln         74.522569         73.764757
0.757812
..         ...          ...               ...               ...
...
95  Wisconsin      Burnett         48.342541         52.437478  -
4.094937
96  Wisconsin    La Crosse         37.490904         40.577038  -
3.086134
97  Wisconsin    Lafayette         38.104967         41.675050  -
3.570083
98    Wyoming       Weston         76.684241         83.983328  -
7.299087
99     Alaska  District 34         77.063259         40.789626
36.273633

[100 rows x 5 columns]
```

```
xbar = sample_data['diff'].mean()
mu = 0
s2 = sample_data['diff'].std()
n_diff = len(sample_data) #count değil len kullanılmalı!!!!
n_diff
```

```
100
```

```
pay = xbar - mu
payda = np.sqrt(s2**2 / n_diff)
cevap = pay / payda
cevap
```

```
-5.601043121928489
```

```
from scipy.stats import t
degrees_of_fre = n_diff - 1
t.cdf(cevap, df= degrees_of_fre) #alpha değerinden küçük olduğu için
h0 reddedilir!
```

```
9.572537285272413e-08
```

```
# KISA YOL :
import pingouin

pingouin.ttest(sample_data['diff'], y=0, alternative='less')
```

```
               T  dof alternative         p-val          CI95%
cohen-d  \
T-test -5.601043   99        less  9.572537e-08  [-inf, -2.02]
0.560104

             BF10     power
T-test  1.323e+05  0.999955
```

ANOVA :

ANOVA (Analysis of Variance), bir veya daha fazla bağımsız değişkenin, bir bağımlı değişken üzerindeki etkilerini test etmek için kullanılan istatistiksel bir yöntemdir.ANOVA'nın amacı, gruplar arasındaki ortalamaların birbirinden anlamlı bir şekilde farklı olup olmadığını belirlemektir. Bu test, özellikle çoklu gruplar arasında karşılaştırmalar yapmak için kullanılır.

Anlamlılık düzeyi: 0.2

```
#H₀: 'job_sat' kategorisindeki grupların 'converted_comp' ortalamaları
eşittir.
#Bu test, "job_sat" (iş tatmini) gibi bağımsız bir değişkenin,
"converted_comp" (ücret düzeyi) gibi bağımlı değişken üzerindeki
etkilerini incelemeye yarar
import pingouin
pingouin.anova(data=df_stck , dv = "converted_comp" , between =
"job_sat") # p değeri anlamlılık düzeyinden küçük. h0 reddedilir!
```

```
    Source  ddof1  ddof2         F     p-unc       np2
0  job_sat      4   2256  4.480485  0.001315  0.007882
```

Pairwise tests

```
import pingouin
pingouin.pairwise_tests(data=df_stck , dv="converted_comp" , between =
"job_sat" , padjust="none")
```

```
  Contrast                     A                     B  Paired
Parametric  \
0  job_sat      Very dissatisfied  Slightly dissatisfied   False
True
1  job_sat      Very dissatisfied               Neither   False
True
2  job_sat      Very dissatisfied     Slightly satisfied   False
True
3  job_sat      Very dissatisfied         Very satisfied   False
True
4  job_sat  Slightly dissatisfied               Neither   False
True
5  job_sat  Slightly dissatisfied     Slightly satisfied   False
True
6  job_sat  Slightly dissatisfied         Very satisfied   False
```

```
True
7  job_sat                Neither     Slightly satisfied   False
True
8  job_sat                Neither         Very satisfied   False
True
9  job_sat     Slightly satisfied         Very satisfied   False
True

          T          dof alternative     p-unc     BF10    hedges
0  1.129951   247.570187   two-sided  0.259590    0.197  0.119131
1  0.545948   321.165726   two-sided  0.585481    0.135  0.058537
2  1.243665   187.153329   two-sided  0.215179    0.208  0.145624
3 -0.747379   221.666205   two-sided  0.455627    0.126 -0.063479
4 -0.602209   367.730081   two-sided  0.547406    0.118 -0.055707
5  0.038264   569.926329   two-sided  0.969491    0.074  0.002719
6 -3.076222   821.303063   two-sided  0.002166     7.43 -0.173247
7  0.700752   258.204546   two-sided  0.484088    0.114  0.068513
8 -1.662901   328.326639   two-sided  0.097286    0.337 -0.120115
9 -4.009935  1478.622799   two-sided  0.000064  158.564 -0.192931
```

Bonferroni correction

Bonferroni düzeltmesi (Bonferroni correction), çoklu karşılaştırmalar (multiple comparisons) yapıldığında, tip I hata oranını (false positive risk) kontrol altına almak için kullanılan bir yöntemdir. Çoklu karşılaştırma, aynı veri seti üzerinde birden fazla istatistiksel test yapmayı ifade eder ve bu da tip I hata oranını arttırabilir.

Diyelim ki, 5 bağımsız test yapıyorsunuz ve her bir testin p-değeri 0.05. Bonferroni düzeltmesi ile bu testlerin her birinin p-değerini şu şekilde düzeltebilirsiniz:

Düzeltmiş p-değeri = 0.05 × 5 = 0.25 Bu durumda, 0.25'ten küçük p-değerleri anlamlı kabul edilir. Yani, 5 testten her birinin p-değeri 0.25'ten küçükse, farklar anlamlıdır.

```
import pingouin
pingouin.pairwise_tests(data=df_stck , dv="converted_comp" , between =
"job_sat" , padjust="bonf")

  Contrast                     A                     B  Paired
Parametric  \
0  job_sat      Very dissatisfied  Slightly dissatisfied   False
True
1  job_sat      Very dissatisfied               Neither   False
True
2  job_sat      Very dissatisfied     Slightly satisfied   False
True
3  job_sat      Very dissatisfied         Very satisfied   False
True
4  job_sat  Slightly dissatisfied               Neither   False
True
5  job_sat  Slightly dissatisfied     Slightly satisfied   False
```

```
True
6  job_sat  Slightly dissatisfied         Very satisfied   False
True
7  job_sat                Neither     Slightly satisfied   False
True
8  job_sat                Neither         Very satisfied   False
True
9  job_sat     Slightly satisfied         Very satisfied   False
True

          T          dof alternative     p-unc    p-corr p-adjust
BF10  \
0  1.129951   247.570187   two-sided  0.259590  1.000000     bonf
0.197
1  0.545948   321.165726   two-sided  0.585481  1.000000     bonf
0.135
2  1.243665   187.153329   two-sided  0.215179  1.000000     bonf
0.208
3 -0.747379   221.666205   two-sided  0.455627  1.000000     bonf
0.126
4 -0.602209   367.730081   two-sided  0.547406  1.000000     bonf
0.118
5  0.038264   569.926329   two-sided  0.969491  1.000000     bonf
0.074
6 -3.076222   821.303063   two-sided  0.002166  0.021659     bonf
7.43
7  0.700752   258.204546   two-sided  0.484088  1.000000     bonf
0.114
8 -1.662901   328.326639   two-sided  0.097286  0.972864     bonf
0.337
9 -4.009935  1478.622799   two-sided  0.000064  0.000638     bonf
158.564

     hedges
0  0.119131
1  0.058537
2  0.145624
3 -0.063479
4 -0.055707
5  0.002719
6 -0.173247
7  0.068513
8 -0.120115
9 -0.192931
```

One-Sample Proportion Testi

belirli bir popülasyon oranının **beklenen bir değere** eşit olup olmadığını test etmek için kullanılan istatistiksel bir yöntemdir. Bu test, örneklemdeki oranı kullanarak, popülasyon oranının belirli bir değere sahip olup olmadığını anlamaya çalışır. Z-skoru şu şekilde hesaplanır:

$$z = \frac{\hat{p} - p_0}{\text{SE}}$$

- (\hat{p}): Örneklemdeki istenen şeyin ort
- (p_0): Popülasyondaki beklenen oran (null hipotezdeki oran)
- **SE** (Standart Hata): Örneklem oranının standart hatası

Standart hata (SE), örneklem büyüklüğüne ve popülasyon oranına bağlı olarak hesaplanır:

$$SE = \sqrt{\frac{p_0(1 - p_0)}{n}}$$

- (p_0): Popülasyon oranı (null hipotezdeki oran)
- (n): büyüklük

#Stackoverflow veri setindeki popülasyondaki kullanıcıların yarısının otuz yaşın altında olduğunu varsayalım ve bir fark olup olmadığını kontrol edelim. Anlamlılık düzeyini 0.01 olarak belirleyelim. Örneklemde , kullanıcıların yarısından biraz fazlası otuz yaşın altındadır.
H0 = 0.5
Ha != 0.5

```
alpha = 0.01
df_stck["age_cat"].value_counts()
```

```
age_cat
Under 30       1211
At least 30    1050
Name: count, dtype: int64
```

```
p_hat = (df_stck["age_cat"] == "Under 30").mean()
p_hat
```

```
0.5356037151702786
```

```
p0 = 0.5
n=len(df_stck)
se = np.sqrt( p0 * (1-p0) / n)
pay = p_hat - p0
sonuc = pay / se
sonuc
```

```
3.385911440783663
```

Calculating the p-value

Sol kuyruklu alternatif hipotezler için norm.cdf kullanarak z-skoru p-değerine dönüştürülür. Sağ kuyruklu alternatif hipotezler için norm.cdf sonucu birden çıkarılır. İki kuyruklu alternatif hipotezler için, test istatistiğinin her iki kuyrukta da yer alıp almadığı kontrol edilir, dolayısıyla p-değeri bu iki değerin toplamıdır: biri z-skoruna, diğeri de dağılımın diğer tarafındaki negatifine karşılık gelir. Normal dağılım PDF'si simetrik olduğundan, z-skoru pozitif olduğu için bu , sağ

kuyruklu p-değerinin iki katı olarak basitleştirilir. Burada , p-değeri sıfır-bir noktasının anlamlılık düzeyinden küçüktür, bu nedenle sıfır hipotezini reddererek otuz yaşın altındaki kullanıcıların oranının beş noktasına eşit olmadığı sonucuna varılır.

```
from scipy.stats import norm
#sol kuyruk
p_value = norm.cdf(sonuc)
p_value
```

```
0.999645288631595
```

```
#sağ kuyruk
p_value = 1- norm.cdf(sonuc)
p_value
```

```
0.00035471136840503625
```

```
#iki kuyruk
p_value= norm.cdf(-sonuc) + 1-norm.cdf(sonuc)
p_value
```

```
0.0007094227368100725
```

```
p_value = 2* (1-norm.cdf(sonuc))
p_value
```

```
0.0007094227368100725
```

```
p_value<alpha
```

```
True
```

Two-sample proportion test

İki örneklem oranı testi, iki farklı popülasyondan alınan örneklemlerdeki oranların birbirinden anlamlı derecede farklı olup olmadığını test etmek için kullanılır.

Stack Overflow anketi bir hobbyist değişkeni içermektedir. "Evet" değeri kullanıcının kendisini hobici olarak tanımladığı, "Hayır" değeri ise kendisini profesyonel olarak tanımladığı anlamına gelmektedir. Hobi sahibi kullanıcıların oranının otuz yaş altı kategorisi ile otuz yaş ve üstü kategorisi için aynı olduğu varsayılabilir, bu da iki kuyruklu bir testtir. Daha açık bir ifadeyle , boş hipotez her bir grup için popülasyon parametreleri arasındaki farkın sıfır olduğudur. Anlamlılık düzeyini 0.05 olarak belirleyelim.
H0 : p>=30 - p<30 =0
Ha : ? p>=30- p<30=! 0

Z-skoru şu şekilde hesaplanır:

$$z = \frac{p_{\geq 30} - p_{\text{altı } 30}}{\text{SE}}$$

- (p_{\geq 30}): Otuz yaş ve üstü kullanıcıların hobi sahibi olma oranı.

- (p_{\text{altı 30}}): Otuz yaş altı kullanıcıların hobi sahibi olma oranı.
- **SE** (Standart Hata): İki oran arasındaki farkın standart hatasıdır.

Standart hata (SE) şu şekilde hesaplanır:

$$SE = \sqrt{\frac{p_{\geq 30}\left(1 - p_{\geq 30}\right)}{n_{\geq 30}} + \frac{p_{\text{altı 30}}\left(1 - p_{\text{altı 30}}\right)}{n_{\text{altı 30}}}}$$

- (p_{\geq 30}): Otuz yaş ve üstü oranı.
- (p_{\text{altı 30}}): Otuz yaş altı oranı.
- (n_{\geq 30}): Otuz yaş ve üstü kullanıcı sayısı.
- (n_{\text{altı 30}}): Otuz yaş altı kullanıcı sayısı.

```
alpha= 0.05
p_hat = df_stck.groupby('age_cat')
['hobbyist'].value_counts(normalize=True)
p_hat
```

```
age_cat      hobbyist
At least 30  Yes         0.773333
             No          0.226667
Under 30     Yes         0.843105
             No          0.156895
Name: proportion, dtype: float64
```

```
n = df_stck.groupby('age_cat')['hobbyist'].count()
n
```

```
age_cat
At least 30    1050
Under 30       1211
Name: hobbyist, dtype: int64
```

```
otuz_buyuk = p_hat[('At least 30', 'Yes')]
otuz_kucuk = p_hat[('Under 30', 'Yes')]
otuz_kucuk
```

```
0.8431048720066061
```

```
n_at_least_30 = n['At least 30']
n_at_under_30 = n['Under 30']
```

```
pay = (otuz_buyuk - otuz_kucuk)
payda = np.sqrt((otuz_buyuk * (1 - otuz_buyuk)) / n_at_least_30 +
(otuz_kucuk * (1 - otuz_kucuk) / n_at_under_30))
z_score = pay / payda
z_score #bu neden yanlış anlamadım!!
```

```
-4.1984371024335285
```

```
import numpy as np
p_hat = df_stck.groupby("age_cat")
["hobbyist"].value_counts(normalize=True)
n = df_stck.groupby("age_cat")["hobbyist"].count()
p_hat_at_least_30 = p_hat[('At least 30', 'Yes')]
p_hat_at_under_30 = p_hat[('Under 30', 'Yes')]
n_at_least_30 = n['At least 30']
n_at_under_30 = n['Under 30']

p_hat = (n_at_least_30 * p_hat_at_least_30 + n_at_under_30 *
p_hat_at_under_30) / (n_at_least_30 + n_at_under_30)

std_error = np.sqrt(p_hat * (1 - p_hat) / n_at_least_30 +
                    p_hat * (1 - p_hat) / n_at_under_30)

z_score = (p_hat_at_least_30 - p_hat_at_under_30) / std_error

z_score
```

```
-4.223691463320559
```

```
# n_hobbyist = np.array([812, 1021])
# Hobi sahibi kullanıcıları saymak (her bir grupta 'Yes' yanıtı
verenler)
n_hobbyist = df_stck[df_stck['hobbyist'] ==
'Yes'].groupby('age_cat').size().values
# Her bir grup için toplam kullanıcı sayısını hesaplamak
n_rows = df_stck.groupby('age_cat').size().values
#n_rows = np.array([812 + 238, 1021 + 190])

from statsmodels.stats.proportion import proportions_ztest

z_score, p_value = proportions_ztest(count=n_hobbyist, nobs=n_rows,
                                     alternative='two-sided')

(z_score, p_value)

# h0 red
```

```
(-4.223691463320559, 2.4033301426850675e-05)
```

Bağımsızlık Testi: Ki-Kare Testi (χ^2 Testi)

Ki-kare testi, iki kategorik değişkenin bağımsız olup olmadığını test etmek için kullanılır. Bu testin amacı, **gözlemlenen frekanslarla** beklenen frekanslar arasındaki farkı ölçmektir. Eğer gözlemlenen ve beklenen frekanslar arasında **büyük bir fark varsa**, bağımsızlık hipotezini reddederiz. Ki-kare testini değişkenler yer değiştirmiş olarak çalıştırırsak , sonuçlar aynı olacaktır. Bu nedenle , sorularımızı "X değişkeni Y değişkeninden bağımsız mı?" yerine " X ve Y değişkenleri bağımsız mı?" şeklinde ifade ediyoruz, çünkü sıralama önemli değil.

Bu değişkenlerin bağımsızlığını test etmek için hipotezler beyan edebiliriz. Burada yaş kategorisi yanıt(bağımlı) değişkeni, iş tatmini ise açıklayıcı(bağımsız) değişkendir: Boş hipotez, bağımsızlığın gerçekleştiğidir. 0.1'lik anlamlılık düzeyi kullanalım.Test istatistiği ki-kare(χ^2) olarak gösterilir. Bağımsızlığın doğru olması durumunda gözlenen sonuçların beklenen değerlerden ne kadar uzak olduğunu ölçer.
H0 : Yaş kategorileri iş tatmini düzeylerinden bağımsızdır.
Ha : Yaş kategorileri iş tatmini düzeylerinden bağımsız değildir.

```
props = df_stck.groupby('job_sat')
['age_cat'].value_counts(normalize=True)
props
```

```
/var/folders/zf/gnk755n14rj48wl97n0lv_0w0000gn/T/
ipykernel_1908/384739958.py:1: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
  props = df_stck.groupby('job_sat')
['age_cat'].value_counts(normalize=True)
```

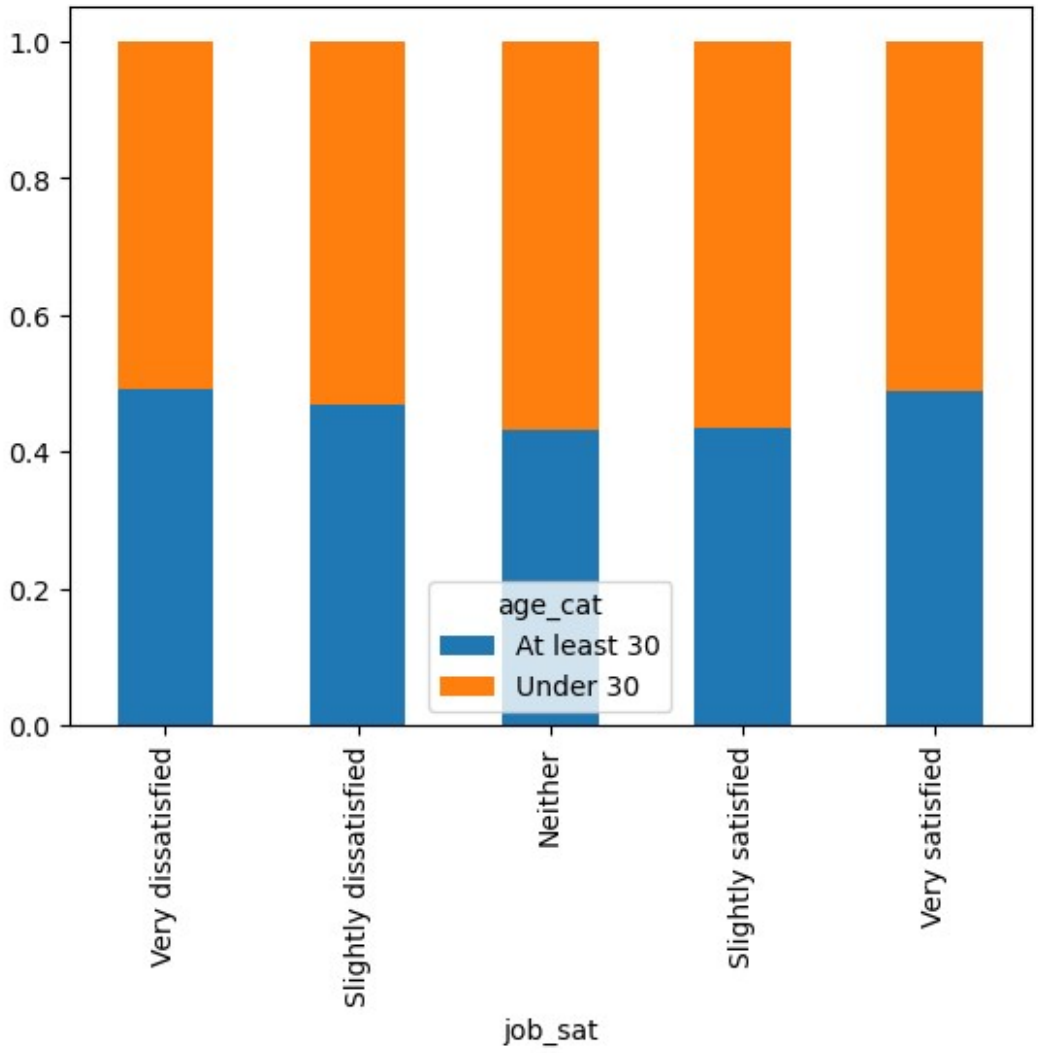
```
job_sat                 age_cat
Very dissatisfied       Under 30       0.509434
                        At least 30    0.490566
Slightly dissatisfied   Under 30       0.532164
                        At least 30    0.467836
Neither                 Under 30       0.567164
                        At least 30    0.432836
Slightly satisfied      Under 30       0.564706
                        At least 30    0.435294
Very satisfied          Under 30       0.511945
                        At least 30    0.488055
Name: proportion, dtype: float64
```

```
wide_props = props.unstack() #age_cat değerlerinin iş tatmini
kategorilerine göre sütunlara dönüştürülmesini sağlar. Bu işlem,
Seri'yi DataFrame'e dönüştürür.
wide_props
```

```
age_cat                 At least 30  Under 30
job_sat
Very dissatisfied          0.490566  0.509434
Slightly dissatisfied      0.467836  0.532164
Neither                    0.432836  0.567164
Slightly satisfied         0.435294  0.564706
Very satisfied             0.488055  0.511945
```

```
wide_props.plot(kind='bar', stacked=True)
```

```
<Axes: xlabel='job_sat'>
```




```
expected, observed, stats = pingouin.chi2_independence(data=df_stck,
x='job_sat',
                                                       y='age_cat')
stats
```

	test	lambda	chi2	dof	pval	cramer	power
0	pearson	1.000000	5.552373	4.0	0.235164	0.049555	0.437417
1	cressie-read	0.666667	5.554106	4.0	0.235014	0.049563	0.437545
2	log-likelihood	0.000000	5.558529	4.0	0.234632	0.049583	0.437871
3	freeman-tukey	-0.500000	5.562688	4.0	0.234274	0.049601	0.438178
4	mod-log-likelihood	-1.000000	5.567570	4.0	0.233854	0.049623	0.438538

```
5              neyman -2.000000  5.579519  4.0  0.232828  0.049676
0.439419
```

Ki-Kare Uyum İyiliği Testleri

Önceki örnekte, **iki kategorik değişkenin oranlarını** karşılaştırmak için **ki-kare testi** kullanılmıştı. Bu sefer ise, **tek bir kategorik değişkeni**, **varsayılan bir dağılımla** karşılaştırmak için ki-kare testinin başka bir çeşidi kullanılacaktır.

Ki-Kare Uyum İyiliği Testi (Goodness-of-Fit Test)

Ki-kare uyum iyiliği testi, tek bir kategorik değişkenin, belirli bir dağılım ile ne kadar iyi uyum sağladığını test etmek için kullanılır. Bu testin amacı, örneklemdeki verilerin, beklenen (varsayılan) dağılımla ne kadar uyuştuğunu belirlemektir.

Mor Bağlantılar(Purple links)

Stack Overflow anketi ,kullanıcıların bir kodlama problemini çözmeye çalışırken mor bağlantı olarak da adlandırılan en iyi kaynağı zaten ziyaret ettiklerini keşfettiklerinde nasıl hissettiklerine dair eğlenceli bir soru içeriyor. purple_link sütunundaki her bir grubun sayılarını almak için value_counts fonksiyonu kullanılabilir. Ayrıca , daha sonra üzerinde çalışabilecek güzel yapılandırılmış bir DataFrame elde etmek için burada biraz manipülasyon yapılmaktadır. İlk olarak en soldaki sütunu purple_link olarak yeniden adlandırılır, sayılar n'ye atılır ve son olarak purple_link'e göre sıranır, böylece yanıtlar alfabetik sırada olur, purple_link sütununda saklanan dört olası yanıt vardır.

Popülasyondaki kullanıcıların yarısının "Merhaba, eski dostum" yanıtını vereceğini ve diğer üç yanıtın her birinin altıda bir alacağını varsayalım. Her yanıt için anahtar-değer anahtar-değer çiftlerinden oluşan bir sözlükten bu varsayılan sonuçlar için bir DataFrame oluşturabiliriz. Hipotezleri , örneğin bu varsayılan dağılıma uyup uymadığı şeklinde belirleriz. Ki-kare test istatistiği, çözümlenen örneklem oran dağılımının hipotezlenen dağılımdan ne kadar uzak olduğunu ölçer. Anlamlılık düzeyini 0.01 olarak belirleyelim.

purple_link değişkeni üzerinden, verilen varsayılan (hipotez edilen) oranların, gözlemlenen dağılımla ne kadar uyumlu olduğunu görmek istiyoruz.

H0 : Örneklem varsayılan dağılımla eşleşmektedir.
Ha : Örneklem varsayılan dağılımla eşleşmemektedir.
χ^2: Her grupta gözlemlenen sonuçların beklentilerden ne kadar uzak olduğunu ölçer.

```
purple_link_counts = df_stck['purple_link'].value_counts()
purple_link_counts
```

```
purple_link
Hello, old friend    1225
Indifferent           405
Amused                368
Annoyed               263
Name: count, dtype: int64
```

```
purple_link_counts =
purple_link_counts.rename_axis('purple_link').reset_index(name='n').so
rt_values('purple_link')
purple_link_counts
```

```
         purple_link     n
2             Amused   368
3            Annoyed   263
0  Hello, old friend  1225
1        Indifferent   405
```

```
hypothesized = pd.DataFrame({
    'purple_link' : ['Amused', 'Annoyed', 'Hello, old friend',
'Indifferent'],
    'prop' : [1/6, 1/6, 1/2, 1/6]
}
)
hypothesized
```

```
         purple_link      prop
0             Amused  0.166667
1            Annoyed  0.166667
2  Hello, old friend  0.500000
3        Indifferent  0.166667
```

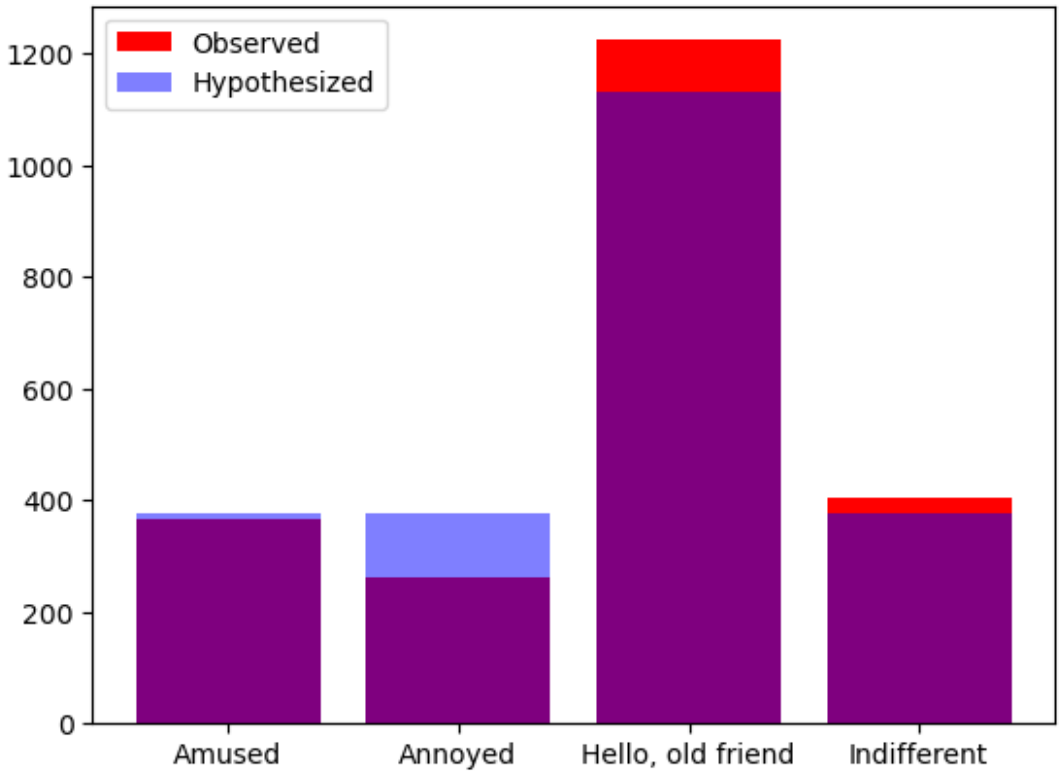
```
alpha = 0.01
#  Kategoriye göre varsayılan sayılar
# Purple_link dağılımını görselleştirmek için,
# her bir yanıt için varsayılan sayıların bulunması yardımcı
olacaktır.
# Bu sayılar, varsayılan oranların örneklemdeki
# toplam gözlem sayısıyla çarpılmasıyla hesaplanır.
n_total = len(df_stck)
hypothesized['n'] = hypothesized['prop'] * n_total
hypothesized
```

```
         purple_link      prop            n
0             Amused  0.166667   376.833333
1            Annoyed  0.166667   376.833333
2  Hello, old friend  0.500000  1130.500000
3        Indifferent  0.166667   376.833333
```

```
import matplotlib.pyplot as plt
plt.bar(purple_link_counts['purple_link'], purple_link_counts['n'],
color='red', label='Observed')
plt.bar(hypothesized['purple_link'], hypothesized['n'], alpha=0.5,
color='blue', label='Hypothesized')
plt.legend()

#Yanıtlardan ikisinin varsayılan dağılım tarafından makul ölçüde iyi
modellendiğini ve diğer ikisinin oldukça farklı göründüğünü
```

```
görebiliriz, ancak farkın istatiksel olarak anlamlı olup olmadığını
görmek için bir hipotez testi yapmamız gerekecek.
```

```
<matplotlib.legend.Legend at 0x1545ec3b0>
```




```
from scipy.stats import chisquare
chisquare(f_obs=purple_link_counts["n"] , f_exp=hypothesized["n"])
```

```
Power_divergenceResult(statistic=44.59840778416629,
pvalue=1.1261810719413717e-09)
```

Bu kod, **ki-kare iyelik testi (Chi-squared goodness-of-fit test)** uygulayarak, **gözlemlenen dağılım** ile **varsayılan dağılım** arasındaki farkın **istatistiksel olarak anlamlı olup olmadığını** test eder.

İyilik Testi (Goodness-of-Fit Test) Nedir?

Ki-kare iyilik testi (Chi-squared goodness-of-fit test), bir gözlem grubunun **beklenen bir dağılıma** (örneğin, teorik bir dağılım) uygun olup olmadığını test etmek için kullanılır. Bu test, genellikle **kategorik veriler** için uygulanır.

Örneğin, **purple_link** değişkeni üzerinden, verilen **varsayılan (hipotez edilen)** oranların, **gözlemlenen** dağılımla ne kadar uyumlu olduğunu görmek istiyoruz.

Kodu Adım Adım Açıklayalım:

1. **Gözlemlenen Dağılım (`purple_link_counts`)**:

```
purple_link_counts =
df_stck['purple_link'].value_counts(normalize=True)
```

Bu satırda, **purple_link** sütunundaki her kategorinin **gözlemlenen frekanslarını** (oranlarını) hesaplıyoruz.

- `value_counts(normalize=True)` komutu, **her bir kategorinin** toplam gözlem sayısına oranını verir. Bu, gözlemlenen **oranları** verir.

2. **Varsayılan Dağılım (`hypothesized`)**:

```
hypothesized = pd.DataFrame({
    'purple_link' : ['Amused', 'Annoyed', 'Hello, old friend',
'Indifferent'],
    'prop' : [1/6, 1/6, 1/2, 1/6]
})
```

Burada, **hipotez edilen** (varsayılan) dağılımı tanımlıyoruz. Bu dağılımda, her kategorinin **beklenen oranları** verilmiştir:

- **'Amused'**, **'Annoyed'**, **'Hello, old friend'**, ve **'Indifferent'** kategorilerinin beklenen oranları sırasıyla **1/6**, **1/6**, **1/2**, ve **1/6** olarak belirlenmiştir.

3. **Varsayılan Dağılım İçin Gözlemler (`n`)**:

```
n_total = len(df_stck)
hypothesized['n'] = hypothesized['prop'] * n_total
```

Bu satırda, **varsayılan dağılımdaki oranları** kullanarak, her kategorinin **beklenen gözlem sayısını** hesaplıyoruz. Beklenen gözlem sayısı, **her kategorinin oranı** ile **toplam gözlem sayısı** (`n_total`) çarpılarak bulunur.

4. **Dağılımı Görselleştirme**:

```
plt.bar(purple_link_counts['purple_link'],
purple_link_counts['n'], color='red', label='Observed')
plt.bar(hypothesized['purple_link'], hypothesized['n'],
alpha=0.5, color='blue', label='Hypothesized')
plt.legend()
```

Burada, **gözlemlenen** ve **varsayılan** dağılımlarını **bar chart (çubuk grafiği)** şeklinde görselleştiriyoruz. **Kırmızı çubuklar**, **gözlemlenen dağılımı**, **mavi çubuklar** ise **varsayılan dağılımı** temsil ediyor.

5. **Ki-Kare İyilik Testi (Chi-Squared Goodness of Fit Test)**:

```
from scipy.stats import chisquare
chisquare(f_obs=purple_link_counts["n"], f_exp=hypothesized["n"])
```

Bu satırda, `chisquare()` fonksiyonu kullanılarak **ki-kare iyilik testi** yapılır. Burada:

- `f_obs`: **Gözlemlenen değerler** (yani, her kategorideki gözlemler).
- `f_exp`: **Beklenen değerler** (yani, her kategorideki varsayılan sayılar).

Bu test, **gözlemlenen değerlerin** (gerçek verinin) **beklenen değerlerle (varsayılan dağılım)** ne kadar uyumlu olduğunu test eder.

Ki-Kare İyilik Testi (Chi-Squared Goodness of Fit Test):

Ki-kare testi şu şekilde çalışır:

1. **H0 (Boş Hipotez)**: Gözlemlenen dağılım, beklenen (varsayılan) dağılımla uyumludur. Yani, **veri beklenen dağılıma uyar**.
2. **Ha (Alternatif Hipotez)**: Gözlemlenen dağılım, beklenen dağılımla uyumsuzdur. Yani, **veri beklenen dağılıma uymaz**.

Testin Sonuçları:

- **p-değeri**: Eğer **p-değeri küçükse** (genellikle 0.05'ten küçükse), **null hipotezi reddedilir** ve gözlemlenen dağılımın beklenen dağılımdan anlamlı derecede farklı olduğu kabul edilir.
- **Ki-kare değeri** (χ^2): Bu, gözlemlenen ve beklenen dağılımlar arasındaki farkın büyüklüğünü ölçen bir test istatistiğidir.

Özet:

Bu kodda **ki-kare iyilik testi** yapılarak, **gözlemlenen** ve **beklenen** dağılımlar arasındaki farkın **istatistiksel olarak anlamlı olup olmadığı** test ediliyor. Eğer **p-değeri** küçükse, **gözlemlenen dağılımın beklenen dağılımdan farklı olduğu** sonucuna varılacaktır.

Hipotez Testinde Varsayımlar

Şimdiye kadar görülen her hipotez testi, veriler hakkında belirli varsayımlarda bulunur. Sadece bu varsayımlar karşılandığında, o hipotez testini kullanmak uygun olur.

Rastgelelik (Randomness)

İster bir örnek kullanılsın, ister birden fazla örnek kullanılsın, her hipotez testi **bir örneğin popülasyondan rastgele seçildiğini** varsayar. Eğer **rastgele bir örneklem** yoksa, o zaman popülasyon **temsil edilmeyecektir**.

Bu varsayımı kontrol etmek için verilerin **nereden geldiğinin bilinmesi gerekir**. Bunun için yapılabilecek herhangi bir istatistiksel veya kodlama testi yoktur. Eğer şüphe duyuluyorsa, **veri**

toplamaya dahil olan kişilere veya **örneklenen popülasyonu anlayan bir alan uzmanına** sorulması gerekir.

Gözlemlerin Bağımsızlığı (Independence of Observations)

Hipotez testleri ayrıca her bir gözlemin **bağımsız olduğunu** varsayar. Ancak, bazı özel durumlarda, örneğin **eşleştirilmiş t-testleri** gibi, iki örnek arasındaki bağımlılıklara izin verilebilir. Bu tür testler, hesaplamaları değiştirir, bu nedenle bu tür bağımlılıkların **nerede meydana geldiğini** anlamak önemlidir.

Eşleştirilmiş t-testleri gibi durumlarda, **bağımlılıkların hesaba katılmaması**, **yanlış negatif** ve **yanlış pozitif hata olasılığının artmasına** neden olabilir. Bağımlılıkların hesaba katılmaması, analiz sırasında **teşhis edilmesi zor** bir sorundur. Bu nedenle, ideal olarak veriler analiz edilmeden önce tartışılmalıdır.

Büyük Örneklem Büyüklüğü (Large Sample Size)

Hipotez testleri ayrıca örneklemin **Merkez Limit Teoremi'nin (CLT)** geçerli olacağı kadar büyük olduğunu ve **örneklem dağılımının normal dağıldığını** varsayar.

- **Küçük örneklemler**, **daha büyük belirsizliğe** neden olabilir. Bu da, **Merkez Limit Teoremi'nin geçerli olmadığı** ve örneklem dağılımının **normal dağılmayabileceği** anlamına gelir.
- **Küçük örneklemler**, daha geniş **güven aralıkları** elde edilmesine yol açabilir.

Eğer **Merkez Limit Teoremi geçerli değilse**, örneklem üzerinde yapılan hesaplamalar ve bunlardan çıkarılan sonuçlar **saçma olabilir**. Bu da, **yanlış negatif** ve **yanlış pozitif hata olasılığını artırır**.

Örneklemin "yeterince büyük" olması için ne kadar büyük olması gerektiği, **teste bağlıdır**.

Özetle:

1. **Rastgelelik (Randomness)**: Her hipotez testi, örneklemin rastgele seçildiğini varsayar.
2. **Gözlemlerin Bağımsızlığı**: Testlerin doğru sonuçlar verebilmesi için her gözlem bağımsız olmalıdır.
3. **Büyük Örneklem Büyüklüğü**: Merkez Limit Teoremi'nin geçerli olabilmesi için örneklem yeterince büyük olmalıdır.

Wilcoxon Testi Nedir?

Wilcoxon işaretli sıralar testi (Wilcoxon Signed-Rank Test), iki **eşleştirilmiş** örneği karşılaştırmak için kullanılan parametrik olmayan bir testtir. Bu test, özellikle örneklemin **normal dağılıma** uymadığı durumlarda, **t-testinin** yerine kullanılabilir. Genellikle **sıralama** ve **işaretli farklar** kullanılarak test yapılır.

```
import pandas as pd
from scipy.stats import rankdata
df_small =pd.read_feather("data/repub_votes_potus_08_12.feather")
df_small['diff'] = df_small['repub_percent_08'] -
df_small['repub_percent_12']
df_small['abs_diff'] = df_small['diff'].abs()
df_small['rank_abs_diff'] = rankdata(df_small['abs_diff'])
df_small
```

```
         state       county  repub_percent_08  repub_percent_12
diff  \
0      Alabama         Hale         38.957877         37.139882
1.817995
1     Arkansas       Nevada         56.726272         58.983452  -
2.257179
2   California         Lake         38.896719         39.331367  -
0.434648
3   California      Ventura         42.923190         45.250693  -
2.327503
4     Colorado      Lincoln         74.522569         73.764757
0.757812
..         ...          ...               ...               ...
...
95   Wisconsin      Burnett         48.342541         52.437478  -
4.094937
96   Wisconsin    La Crosse         37.490904         40.577038  -
3.086134
97   Wisconsin    Lafayette         38.104967         41.675050  -
3.570083
98     Wyoming       Weston         76.684241         83.983328  -
7.299087
99      Alaska  District 34         77.063259         40.789626
36.273633

     abs_diff  rank_abs_diff
0    1.817995           33.0
1    2.257179           42.0
2    0.434648           10.0
3    2.327503           44.0
4    0.757812           16.0
..        ...            ...
95   4.094937           63.0
96   3.086134           50.0
97   3.570083           54.0
98   7.299087           87.0
99  36.273633          100.0

[100 rows x 7 columns]
```

```
alpha = 0.01
pingouin.wilcoxon(x=df_small['repub_percent_08'],
                  y=df_small['repub_percent_12'],
                  alternative='less') #Yani, 2012 oranının 2008
oranından küçük olduğunu test ediyoruz.
#Wilcoxon testi, 2008 ve 2012 arasındaki farkların anlamlı olup
olmadığını test ederken, sıralamaları ve farkları kullanır.
```

	W-val	alternative	p-val	RBC	CLES
Wilcoxon	386.0	less	9.700674e-14	-0.847129	0.5729

Wilcoxon-Mann-Whitney Testi Nedir?

Wilcoxon-Mann-Whitney testi, **non-parametrik bir testtir** ve genellikle **iki bağımsız grup** arasındaki farkları test etmek için kullanılır. Bu testin temel amacı, iki grubun sıralamaları arasında anlamlı bir fark olup olmadığını belirlemektir.

- **Parametrik olmayan bir alternatif** olarak **t-testine** benzer bir işlev görür.
- **Eşleştirilmiş verilerle çalışmaz**, yani test, **bağımsız** (unpaired) iki grubun sıralamalarını karşılaştırır.
- **Wilcoxon testi** gibi **sıralı** veriler üzerinde çalışır, ancak burada **eşleştirilmiş veriler** yerine **bağımsız (unpaired)** veriler kullanılır.

çocukların gelirinin yetişkinlerin gelirinden daha yüksek olup olmadığını test ediyoruz:

```
import pandas as pd
df_stck= pd.read_feather("data/stack_overflow.feather")
age_vs_comp = df_stck[['converted_comp', 'age_first_code_cut']]
age_vs_comp_wide = age_vs_comp.pivot(columns='age_first_code_cut',
                                     values='converted_comp')
#pivot() fonksiyonu, pandas kütüphanesinde kullanılan bir fonksiyondur
ve verilerin geniş formata dönüştürülmesini sağlar. Bu fonksiyon, bir
uzun formatta olan veri çerçevesini, daha uygun analizler yapabilmek
için geniş formata çevirir.
import pingouin

alpha = 0.01

pingouin.mwu(x=age_vs_comp_wide['child'],
             y=age_vs_comp_wide['adult'],
             alternative='greater')

#p-val = 1.902723e-19: p-değeri, testin sonucudur ve yaklaşık olarak
1.9 x 10⁻¹⁹ (çok küçük bir değere sahiptir). Bu, null hipotezini
reddetmek için güçlü bir kanıt sağlar. Yani, çocukların gelirinin
yetişkinlerin gelirinden anlamlı şekilde daha yüksek olduğunu
gösterir. Bu p-değeri, anlamlılık düzeyinden (α = 0.01) çok daha
küçüktür, bu yüzden null hipotezini reddederiz
```

```
        U-val alternative         p-val       RBC      CLES
MWU  744365.5     greater  1.902723e-19  0.222516  0.611258
```

Kruskal-Wallis Testi Nedir?

Kruskal-Wallis testi, **non-parametrik bir testtir** ve **ANOVA**'nın parametrik olmayan versiyonudur. Bu test, üç veya daha fazla **bağımsız grubun medyanları arasında fark olup olmadığını** test eder. Kruskal-Wallis, verilerin sıralamalı olduğu varsayımıyla çalışır ve verilerin normal dağılmadığı durumlarda, parametrik ANOVA testlerinin yerine kullanılabilir.

H_0: 'job_sat' kategorisindeki tüm grupların 'converted_comp' medyanları birbirine eşittir.

```
alpha = 0.01

pingouin.kruskal(data=df_stck, dv='converted_comp', between='job_sat')

#p-unc = 5.772915e-15: Bu, p-değeridir. p-değeri çok küçüktür
(yaklaşık 5.77 x 10⁻¹⁵), bu da null hipotezini reddetmek için güçlü
bir kanıt sağlar. Yani, iş tatmini grupları arasında anlamlı bir fark
vardır. Burada p-değeri, anlamlılık düzeyinden (α = 0.01) çok daha
küçüktür. Bu durumda null hipotezini reddederiz ve iş tatmini
gruplarının ortalama gelirlerinin birbirinden farklı olduğunu kabul
ederiz.
```

```
          Source  ddof1          H         p-unc
Kruskal  job_sat      4  72.814939  5.772915e-15
```

What is Machine Learning?

KNN SEZGİSİ:

KNN (K-Nearest Neighbors), gözetimsiz bir öğrenme algoritmasıdır. Bu algoritma, bir örneğin sınıfını belirlemek için en yakın komşularının sınıflarını gözlemler ve çoğunluğa göre tahmin yapar.

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

df_churn = pd.read_csv("data/telecom_churn_clean.csv")
X = df_churn[['total_day_charge', 'total_eve_charge']].values
#bağımsız değişken
y= df_churn['churn'].values #bağımlı değişken
model_knn = KNeighborsClassifier(n_neighbors=15)
model_knn.fit(X,y) # şu anda veri tahmini yapmaya hazır geldi, model
eğitildi!
import numpy as np
X_new = np.array([
    [56.8, 17.5],
    [24.4, 24.1],
```

```
    [50.1, 10.9]
])
#Yeni veriler için tahmin yapmak amacıyla, her bir yeni müşteri
verisini eğitim verisindeki en yakın 15 komşusuyla karşılaştırır.
#Bu komşu sınıflarına bakarak, her bir yeni müşteri için "churn"
(hizmet iptali) olup olmadığını tahmin eder.
predictions = model_knn.predict(X_new) #tahmin ediyor burada
predictions
```

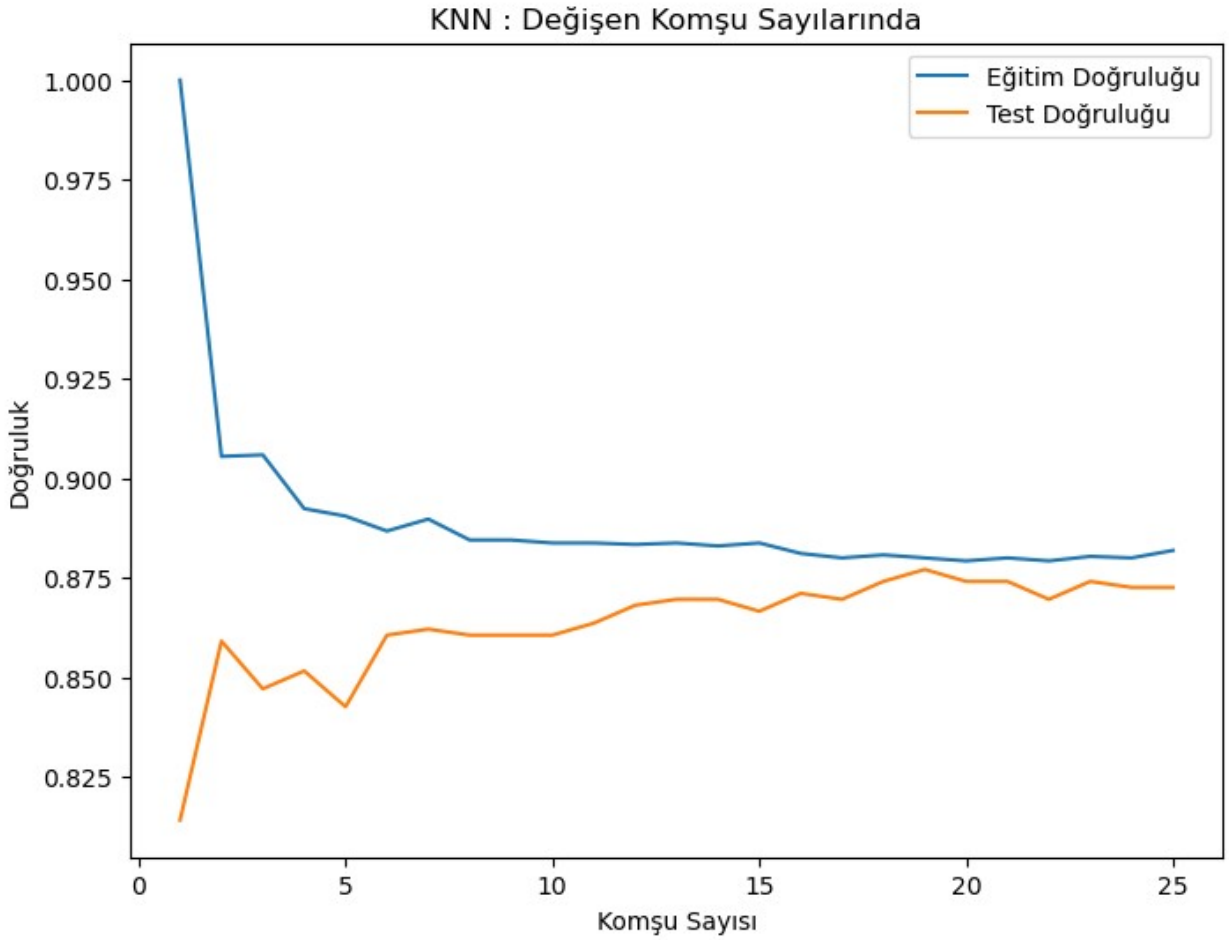
```
array([1, 0, 0])
```

```
# Train/test split
# Verilerimizi nasil ikiye boluyoruz

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =train_test_split(X, y,
                                                   test_size=0.2,
                                                   random_state=42,
                                                   stratify=y)

knn = KNeighborsClassifier(n_neighbors=6) #model
knn.fit(X_train, y_train) #model eğitildi
knn.score(X_test, y_test) #ne kadar doğru veriler

#sonucu 0.8605697151424287 olarak verilmiş, bu da modelin test seti
üzerindeki doğruluğunun yaklaşık %86 olduğunu gösterir.
```

```
0.8605697151424287
```

```
import matplotlib.pyplot as plt

train_accuracies = {}
test_accuracies = {}

neighbours = np.arange(1, 26)

for neighbour in neighbours:
    knn = KNeighborsClassifier(n_neighbors=neighbour)
    knn.fit(X_train, y_train)
    train_accuracies[neighbour] = knn.score(X_train, y_train)
    test_accuracies[neighbour] = knn.score(X_test, y_test)

plt.figure(figsize=(8, 6))
plt.title('KNN : Değişen Komşu Sayılarında')
plt.plot(neighbours, train_accuracies.values(), label='Eğitim
Doğruluğu')
plt.plot(neighbours, test_accuracies.values(), label='Test Doğruluğu')
plt.legend()
plt.xlabel('Komşu Sayısı')
plt.ylabel('Doğruluk')
```

```
Text(0, 0.5, 'Doğruluk')
```



komşu sayısının (k) modelin doğruluğu üzerindeki etkisini görselleştiren bir grafik elde edersiniz. Eğitim doğruluğu genellikle komşu sayısı arttıkça artar çünkü model daha fazla komşuyu dikkate alır ve daha kesin sonuçlar elde etmeye çalışır. Test doğruluğu ise başlangıçta artabilir, ancak çok yüksek komşu sayıları ile modelin overfitting (aşırı uyum) yapmaya başlaması olasılığı vardır. Yani, model eğitim verisine aşırı uyum sağlayarak test verisi üzerinde başarısını kaybedebilir. Özet: Komşu sayısı (k), KNN modelinin başarısı için önemli bir parametredir. Bu kod, farklı k değerleri ile modelin eğitim ve test doğruluklarını inceleyerek, doğru k değerinin bulunmasına yardımcı olur.

Az komşu sayısı (1-3) genellikle eğitim verilerine çok fazla uyum sağlar ve aşırı öğrenme (overfitting) sorunu yaratır. Bu durumda eğitim doğruluğu çok yüksek olsa da test doğruluğu düşük olabilir. Yüksek komşu sayıları (yaklaşık 15 ve sonrası), modelin genelleme yeteneğini artırır, ancak eğitim doğruluğu daha düşük olabilir. Bu durum underfitting riskini azaltır. Sonuç olarak, test doğruluğunun arttığı ve eğitim doğruluğunun stabilize olduğu komşu sayısı, model için en iyi genelleme performansını sağlar.

Regresyon:

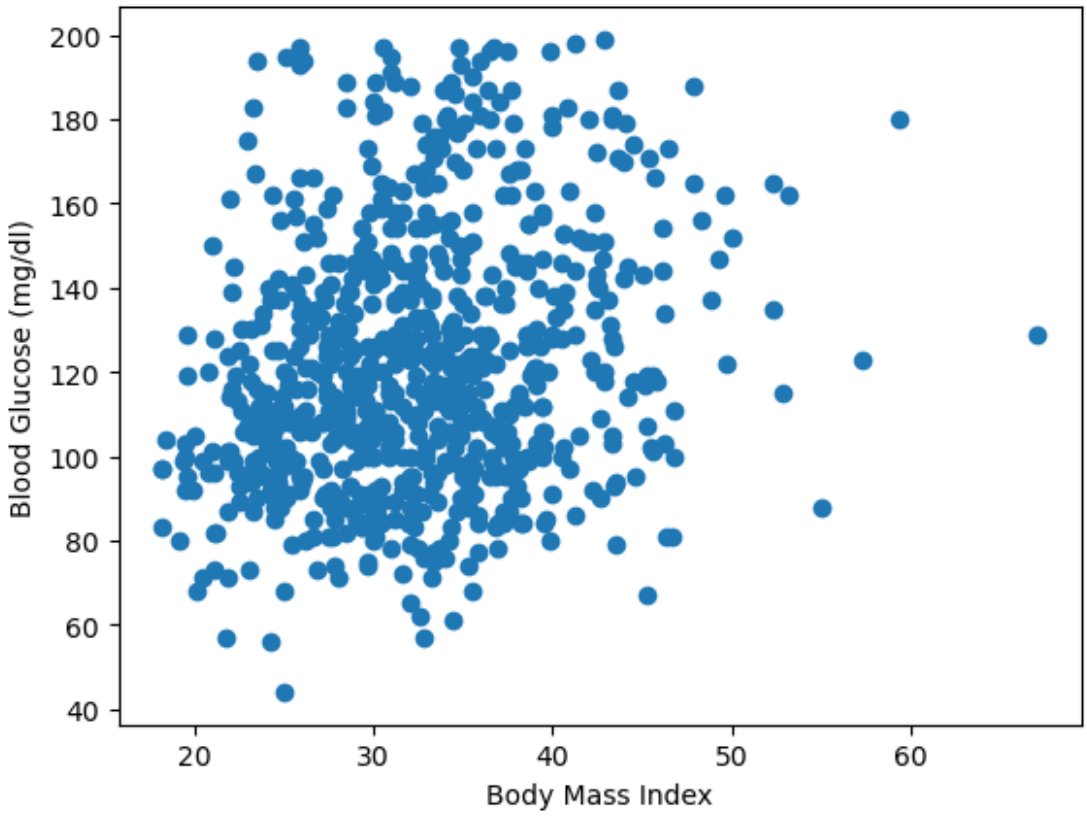
bir veri seti içindeki değişkenler arasındaki ilişkileri modelleme ve tahmin yapma amacıyla kullanılan bir istatistiksel tekniktir. Özellikle, bir bağımlı değişkenin (yani hedef değişken) bir veya daha fazla bağımsız değişkenin (yani özellikler veya girdiler) etkisiyle nasıl değiştiğini anlamak için kullanılır.

Regresyonun Temel Amacı: Bağımlı değişkeni tahmin etmek için bağımsız değişkenleri kullanmaktır. Örneğin, bir şirketin satış rakamlarını tahmin etmek için, reklam harcamaları, fiyatlar, promosyonlar gibi bağımsız değişkenler kullanılabilir.

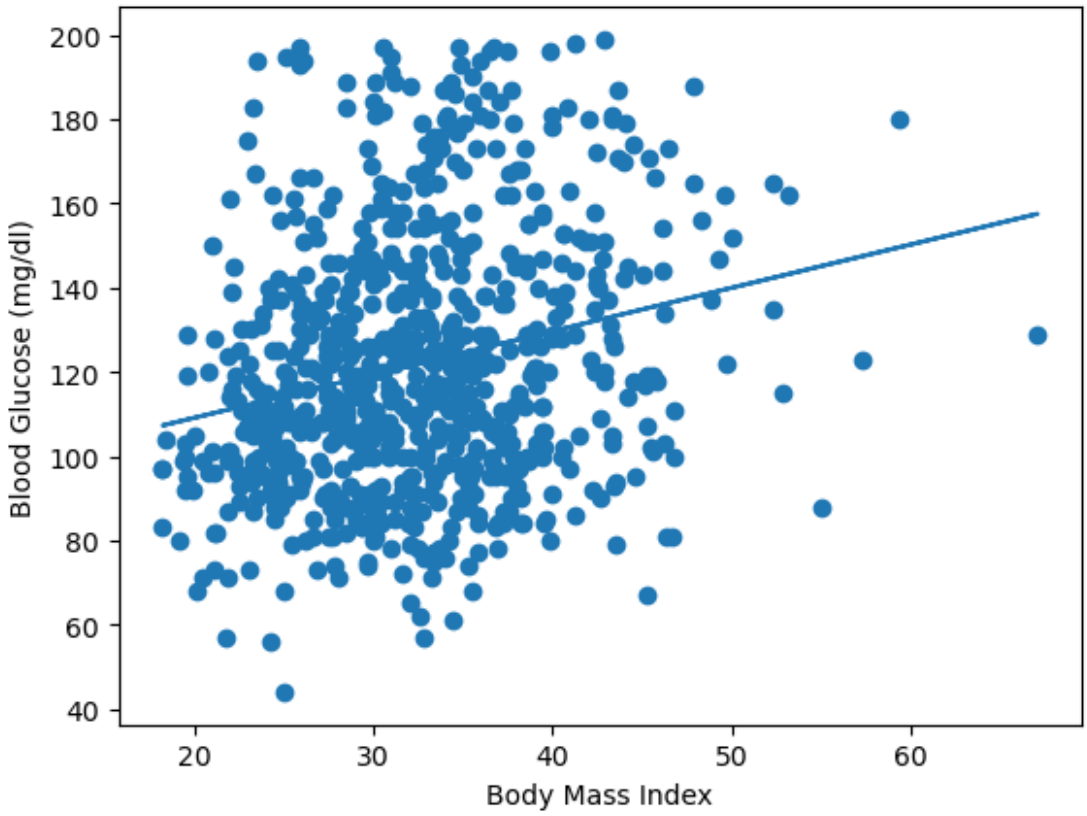
```
import pandas as pd

df_diabets = pd.read_csv('data/diabetes_clean.csv')
df_filteredx = df_diabets[(df_diabets['bmi'] == 0.0) |
(df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)
X = df_diabets.drop('glucose', axis=1).values #: Bu satırda,
df_diabets DataFrame'inden glucose sütunu hariç diğer tüm sütunlar
seçilir. drop('glucose', axis=1) ifadesi, glucose sütununu kaldırır.
axis=1 parametresi, sütunları ifade eder (satırları ifade etmek için
axis=0 kullanılır). Sonuçta X'te, glucose haricindeki bağımsız
değişkenlerin verileri saklanır.
y = df_diabets['glucose'].values #Bu satırda ise glucose sütunu
bağımlı değişken olarak seçilir. Bu, modelin tahmin etmeye çalışacağı
hedef değişkendir.
```

Lineer Regresyon, istatistiksel bir modelleme tekniğidir ve bağımlı bir değişkenin (y) bağımsız değişkenler (x1, x2, ..., xn) ile olan doğrusal ilişkisini modellemeye çalışır. Genellikle, bir hedef değişkenin (y) tahmin edilmesinde kullanılır ve bu hedef değişkenin diğer değişkenlerle (özellikler) ilişkisini anlamak için çok yaygın olarak kullanılır.

```
X_bmi = X[:, 4] #X[:, 4]: Bu komut, X veri setinin 4. sütununu seçer.
0 tabanlı indekslemeye dikkat edilmelidir. Bu satırda, BMI sütunu X
veri setinin 4. sütunu olarak seçilmiştir.
X_bmi = X_bmi.reshape(-1, 1) #1d veriyi 2d hale getirmek için BAĞIMSIZ
DEĞİŞKEN 2D OLMALI!!!!!!!!
import matplotlib.pyplot as plt

plt.scatter(X_bmi,y)
plt.ylabel('Blood Glucose (mg/dl)')
plt.xlabel('Body Mass Index')

Text(0.5, 0, 'Body Mass Index')
```



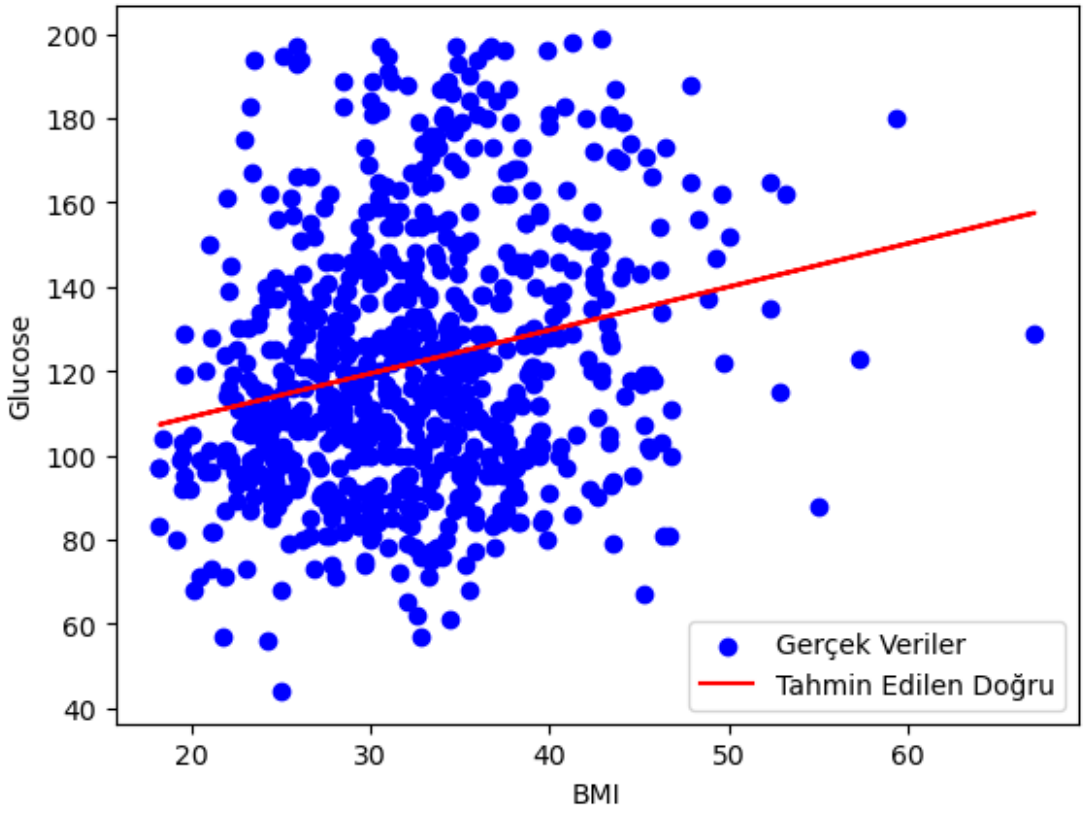

```
from sklearn.linear_model import LinearRegression

model_reg = LinearRegression()
model_reg.fit(X_bmi, y) #model BMI ile kan şekeri arasındaki doğrusal
ilişkiyi öğrenir

predictions = model_reg.predict(X_bmi) #tahmin yürütür

plt.scatter(X_bmi, y)
plt.plot(X_bmi, predictions) #tahmin, oğrusal regresyon modelinin
tahmin ettiği ilişkiyi gösteren çizgidir. Model bu çizgiyi, verilen
BMI değerlerine karşılık gelen kan şekeri tahminlerini yaparak
çiziyor.
plt.ylabel('Blood Glucose (mg/dl)')
plt.xlabel('Body Mass Index')

Text(0.5, 0, 'Body Mass Index')
```




```
from sklearn.linear_model import LinearRegression
import pandas as pd

# Veri setini yükle
df_diabetes = pd.read_csv('data/diabetes_clean.csv')

# 'bmi' ve 'glucose' sütununda 0 değeri olan satırları filtrele ve
kaldır
df_filtered = df_diabetes[(df_diabetes['bmi'] == 0.0) |
(df_diabetes['glucose'] == 0)]
df_diabetes.drop(df_filtered.index, inplace=True)

# Bağımsız değişken (X) ve bağımlı değişken (y) olarak veriyi ayır
X = df_diabetes['bmi'].values.reshape(-1, 1)  # X'yi 2D formata
dönüştür. eğer tek veri ise
y = df_diabetes['glucose'].values

# Linear Regression modelini oluştur ve eğit
model_reg = LinearRegression()
model_reg.fit(X, y)

# Model ile tahmin yap (eğitim verisiyle de test yapabilirsiniz)
predictions = model_reg.predict(X)

# Modelin katsayılarını ve kesişim noktasını yazdır
```

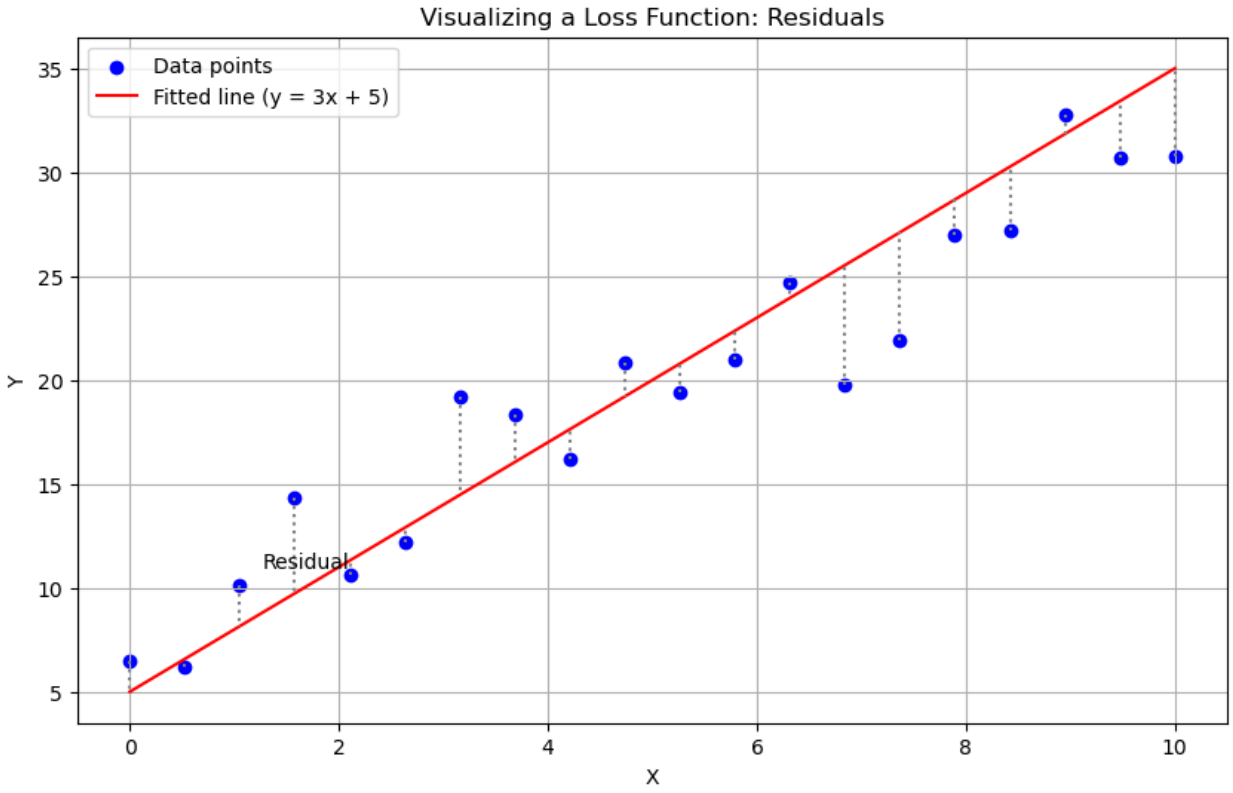
```
print("Katsayı (slope):", model_reg.coef_)
print("Kesim (intercept):", model_reg.intercept_)

# Tahmin sonuçlarını görselleştirelim
import matplotlib.pyplot as plt

plt.scatter(X, y, color='blue', label='Gerçek Veriler')
plt.plot(X, predictions, color='red', label='Tahmin Edilen Doğru')
plt.xlabel('BMI')
plt.ylabel('Glucose')
plt.legend()
plt.show()
```

```
Katsayı (slope): [1.02801737]
Kesim (intercept): 88.57754093395485
```




```
import numpy as np
import matplotlib.pyplot as plt

# Generate random data for the scatter plot
np.random.seed(42)
X = np.linspace(0, 10, 20) #bağımsiz değişken
Y = 3 * X + 5 + np.random.normal(0, 3, size=len(X)) #bağımlı değişken
```

```
# Fit a line (y = mx + c) manually
m = 3  # slope
c = 5  # intercept
Y_fit = m * X + c #Bu doğrunun tahmin edilen değerlerini hesaplar
(yani, bağımsız değişken X için Y'nin tahmin edilen değerleri).

# Calculate residuals (vertical distances between points and the line)
residuals = Y - Y_fit #her bir veri noktası ile modelin tahmin ettiği
değerler arasındaki farktır.

# Plot the scatter plot and the fitted line
plt.figure(figsize=(10, 6))
plt.scatter(X, Y, color='blue', label='Data points')
plt.plot(X, Y_fit, color='red', label='Fitted line (y = 3x + 5)')

# Add vertical lines to represent residuals
for i in range(len(X)):
    plt.plot([X[i], X[i]], [Y[i], Y_fit[i]], color='gray',
linestyle='dotted')
    if i == 4:  # Add a label to one residual line
        plt.text(X[i], (Y[i] + Y_fit[i]) / 2, 'Residual',
color='black', fontsize=10, ha='right')

# Customize the plot
plt.title('Visualizing a Loss Function: Residuals')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



Bu grafik, regresyon modelinin doğruluğunu ve kayıp fonksiyonunun (loss function) nasıl çalıştığını görselleştirir:

Kayıp fonksiyonu, modelin tahminlerinin ne kadar hatalı olduğunu ölçen bir ölçüttür. Genellikle residuals'ın karelerinin toplamı kayıp fonksiyonu olarak kullanılır (bu "mean squared error" (MSE) olarak bilinir). Residuals her bir veri noktasının modelin tahmininden ne kadar uzak olduğunu gösterir ve bu uzaklıkları minimize etmeye çalışarak modelin doğruluğunu artırmaya çalışırız.

```
#bu kod, **doğrusal regresyon (Linear Regression)** kullanarak
**diabetes veri kümesi** üzerinde bir model oluşturuyor ve bu modelin
**kan şekeri seviyelerini** tahmin etmesini sağlıyor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import pandas as pd

df_diabets = pd.read_csv('data/diabetes_clean.csv')

df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) |
(df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)


X = df_diabets.drop('glucose', axis=1).values #bağımsız değişkenler
y = df_diabets['glucose'].values #bağımlı değişken
```

```
X_train, X_test, y_train, y_test =train_test_split(X, y,
test_size=0.2,
                                                   random_state=42)


model_reg = LinearRegression()
model_reg.fit(X_train, y_train) #model eğitildi

y_pred = model_reg.predict(X_test) #tahminler yapıldı
model_reg.score(X_test, y_test) #model ne kadar başarılı onu ölçer.
R^2 skoru kullanarak yapar


#Diyabet veri kümesindeki tüm özellikleri kullanarak kan şekeri
seviyelerini tahmin etmek için doğrusal regresyon gerçekleştirelim.
#LinearRegression'ı sklearn-dot-linear_model'den içe aktarıyoruz.
Ardından verileri eğitim ve test kümelerine ayırıyoruz, modeli
#örneklendiriyoruz, eğitim kümesine yerleştiriyoruz ve test kümesinde
tahmin ediyoruz.
#Scikit-learn'deki doğrusal regresyonun kaputun altında OLS
gerçekleştirdiğini unutmayın.

#Doğrusal regresyon, en küçük kareler yöntemi (OLS, Ordinary Least
Squares) kullanarak, veri noktalarına en uygun doğrusal çizgiyi çizer.
Yani, model y = mx + c formülüne dayalı olarak, m (eğim) ve c (kesim)
parametrelerini öğrenir.
#Eğim (m), bağımsız değişkenin (X) her bir birim değiştiğinde bağımlı
değişkenin (y) ne kadar değişeceğini gösterir.
#Kesim (c), X = 0 olduğunda y değerinin ne olduğunu belirtir.
```

```
0.3282802627263198
```

```
model_reg.coef_
# Bu, eğitilen doğrusal regresyon modelinin katsayılarını
(coefficients) döndüren bir özelliktir.
#Katsayılar, her bağımsız değişkenin (özelliklerin) bağımlı değişken
üzerindeki etkisini gösterir. Yani, modelin öğrenmiş olduğu eğim (m)
değerleridir. Örneğin, BMI gibi bir bağımsız değişkenin katsayısı,
BMI'deki bir birimlik değişikliğin kan şekeri üzerindeki etkisini
gösterir.
#Eğer model birden fazla bağımsız değişken içeriyorsa, her bağımsız
değişken için ayrı bir katsayı değeri olacaktır.
```

```
array([-0.32654116,  0.14686555, -0.27590315,  0.08606826,
0.36160446,
        1.8382773 ,  0.42185562, 25.08247323])
```

```
from sklearn.metrics import root_mean_squared_error

root_mean_squared_error(y_test, y_pred) #modelin tahmin ettiği
```

```
değerlerle gerçek değerler arasındaki farkların karelerinin
ortalamasıdır. Daha düşük MSE, modelin daha iyi olduğunu gösterir.
```

```
25.695203763480208
```

Cross-validation

Cross-validation, modelin genelleme yeteneğini değerlendirmek amacıyla kullanılan bir tekniktir. Modelin eğitim verisi üzerinde ne kadar başarılı olduğunu değerlendirmek yerine, modelin yeni, görülmemiş verilere karşı nasıl performans gösterdiğini test eder.

Cross-validation, bir modelin genelleme gücünü değerlendirmek için kullanılan bir tekniktir. Bir modelin doğruluğunu değerlendirirken, modelin eğitim verileri üzerinde aşırı öğrenme (overfitting) yapıp yapmadığını kontrol etmek önemlidir. Cross-validation, bu aşırı öğrenme sorununu ortadan kaldırarak daha sağlam ve güvenilir bir model değerlendirmesi yapmamıza yardımcı olur.

Çapraz doğrulama, genellikle veriyi daha küçük parçalara bölme (folds) ve her parça üzerinde modelin doğruluğunu test etme işlemini içerir. Çapraz doğrulama sayesinde, verilerin her kısmı hem eğitim hem de test için kullanılarak modelin performansı daha güvenilir bir şekilde ölçülür.

```
#Bu kod, doğrusal regresyon modelinin cross-validation (çapraz
doğrulama) ile değerlendirilmesini sağlar. Yani, modelin genel
performansını daha güvenilir bir şekilde ölçmek için veriyi birkaç
farklı alt küme (fold) üzerinde test eder.

import pandas as pd
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
df_diabets = pd.read_csv('data/diabetes_clean.csv')

df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) |
(df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)

X = df_diabets.drop('glucose', axis=1).kfvalues
y = df_diabets['glucose'].values

kf = KFold(n_splits=6, shuffle=True, random_state=42)
model_reg = LinearRegression()

cv_results = cross_val_score(model_reg, X, y, cv=kf) #modelin
performansını çapraz doğrulama ile değerlendirir.
#model_reg: Doğrusal regresyon modelini belirtir.
#X ve y: Bağımsız ve bağımlı değişkenleri belirtir.
#v=kf: KFold çapraz doğrulama nesnesi burada kullanılır. Bu, veri
setinin 6 alt kümeye (fold) bölünmesini sağlar. Her seferinde 5 katman
eğitim için, 1 katman test için kullanılır.
#Sonuç: cross_val_score fonksiyonu, her bir fold'da elde edilen model
doğruluğunu döndüren bir dizi (array) oluşturur. Bu doğruluklar
```

```
cv_results değişkeninde saklanır.

cv_results

array([0.31239631, 0.39992274, 0.38698031, 0.19731639, 0.32317527,
       0.3320924 ])

import numpy as np
(np.mean(cv_results), np.std(cv_results))

(0.32531390402925003, 0.06579003824551613)

np.quantile(cv_results, [0.025, 0.975])

array([0.21170138, 0.39830494])
```

Ortalama doğruluk 0.325 olarak hesaplanmış. Bu, modelin genel olarak yüzde 32.5 doğrulukla performans gösterdiğini belirtiyor. Bu değer, modelin ortalama başarısını gösterir. Standart Sapma (Standard Deviation):

0.0658'lik bir standart sapma değeri, fold'lar arasındaki doğruluk farklılıklarının ne kadar yaygın olduğunu gösterir. Düşük standart sapma (0.0658), modelin göreceli olarak tutarlı olduğunu ve farklı fold'larda büyük değişiklikler yaşamadığını gösterir. **0.025 Persentili**:

- **0.21170138**: Bu, **cross-validation sonuçlarının en düşük %2.5'lik kısmındaki doğruluk** değerini gösterir. Modelin **en kötü performansı** bu değere yakın bir yer alır. Yani, modelin doğruluğu, en kötü durumda yaklaşık **21.17%**'dir.
- **0.975 Persentili**:
  - **0.39830494**: Bu, **cross-validation sonuçlarının en yüksek %2.5'lik kısmındaki doğruluk** değerini gösterir. Modelin **en iyi performansı** bu değere yakın bir yer alır. Yani, modelin doğruluğu, en iyi durumda yaklaşık **39.83%**'tir.
- Bu **persentil aralığı**, modelin **%95 güven aralığında** ne kadar değişkenlik gösterdiğini anlamamıza yardımcı olur. Verilen aralık, modelin doğruluğunun genellikle **0.2117 ile 0.3983** arasında değiştiğini gösteriyor.

Regularized Reggession

Regularization (Düzenleme), makine öğrenmesinde aşırı öğrenme (overfitting) ve aşırı uyum (overfitting) gibi problemleri önlemek için kullanılan bir tekniktir. Aşırı öğrenme, modelin eğitim verisine çok fazla uyum sağlaması sonucu, yeni ve görülmemiş verilere karşı kötü performans göstermesi durumudur. Regularization bu durumu engellemeye yardımcı olur.

RİDGE REGRESYONU:

Ridge regresyonu kullanarak **düzenli regresyon (regularized regression)** performansını değerlendirir. **Ridge regresyonu**, **L2 düzenlemesi** (regularization) kullanarak modelin aşırı uyum (overfitting) yapmasını engellemeye çalışır. Kodda farklı **alpha** (düzenleme parametresi) değerleri ile modelin performansı değerlendirilmekte ve sonuçlar karşılaştırılmaktadır.

```
#  Alfa arttıkça performansın kötüleştiğini görüyoruz.

from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split

df_diabets = pd.read_csv('data/diabetes_clean.csv')

df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) |
(df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)

X = df_diabets.drop('glucose', axis=1).values
y = df_diabets['glucose'].values

X_train, X_test, y_train, y_test =train_test_split(X, y,
test_size=0.2,
                                                   random_state=42)

scores = []
for alpha in [0.1, 1.0, 10.0, 100.0, 1000.0]:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
    scores.append(ridge.score(X_test, y_test))

scores
#alpha ne kadar büyükse, ceza daha fazla olur ve modelin karmaşıklığı
daha fazla sınırlanır.
#alpha değeri küçükse, ceza daha azdır ve model daha serbest çalışır,
bu da aşırı öğrenmeye yol açabilir.
```

```
[0.32825526615552425,
 0.3280240795994781,
 0.3252049078298792,
 0.28836032637802334,
 0.20299309688977707]
```

LASSO REGRESYONU:

Lasso regresyonu kullanarak **farklı alpha** değerleri ile modelin performansını değerlendirir. **Lasso regresyonu**, **L1 düzenlemesi** (regularization) kullanarak, modelin **gereksiz özellikleri** (parametreleri) **sıfırlamasını** sağlar ve böylece daha **basit ve genelleştirilebilir bir model** oluşturur. Lasso, özellikle çok sayıda özellik olduğunda **önemli parametrelerin seçilmesine** yardımcı olur.

```
# Lasso Regression: Etkin parametre secilirken kullanilir #  Alfa
20'nin üzerine çıktıkça performans önemli ölçüde düşer!

from sklearn.linear_model import Lasso
```

```
scores = []
for alpha in [0.1, 1.0, 10.0, 20.0, 50.0]:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    y_pred = lasso.predict(X_test)
    scores.append(lasso.score(X_test, y_test))

scores
```

```
[0.3284857694292622,
 0.3166121180165745,
 0.17121386697851626,
 0.156847521532139,
 0.11477890284329806]
```

```
#Bu kod, Lasso regresyonu kullanarak, diabetes veri kümesindeki her bir özelliğin kan şekeri seviyesi üzerindeki etkisini görselleştirir. Katsayıları çubuk grafikle göstererek, hangi özelliklerin modelin tahminlerine daha fazla katkı sağladığını anlayabilirsiniz.
import matplotlib.pyplot as plt

df_diabets = pd.read_csv('data/diabetes_clean.csv')

df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) |
(df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)

X = df_diabets.drop('glucose', axis=1).values
y = df_diabets['glucose'].values

names = df_diabets.drop('glucose', axis=1).columns

lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_

plt.bar(names, lasso_coef)
plt.xticks(rotation=45)
```

Classifications Metrics

accuracy kullanilir

Classification Metrics (Sınıflandırma Metriği), sınıflandırma problemlerinde bir modelin performansını değerlendirmek için kullanılan ölçütlerdir:

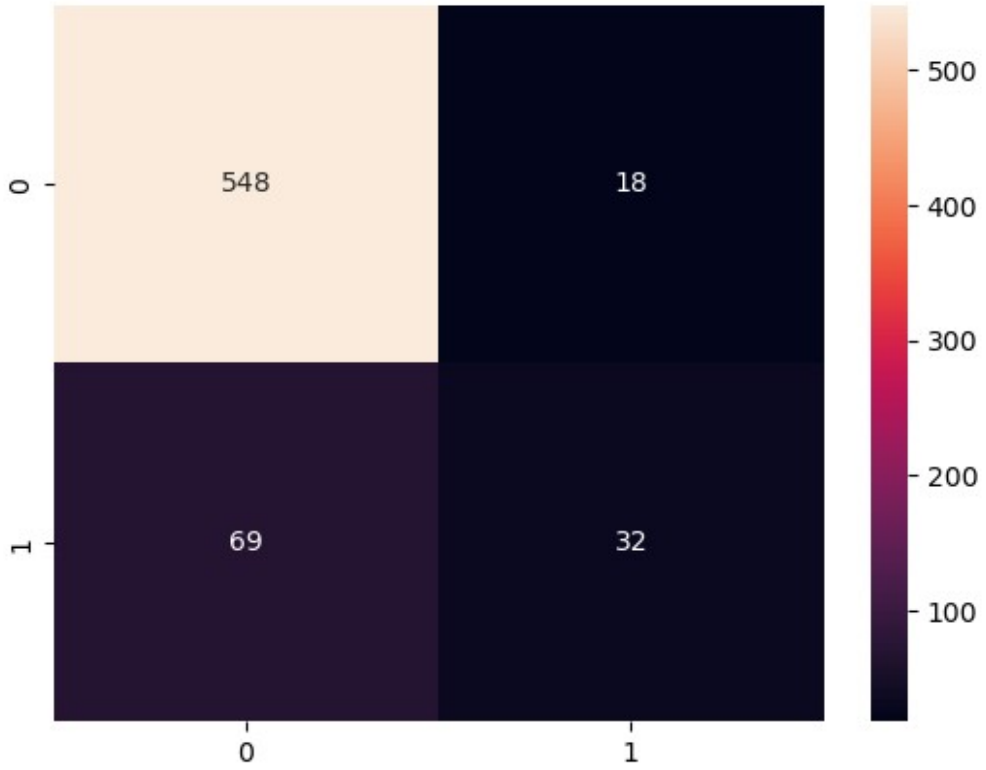
```
# 1. Confusion matrix in scikit-learn

import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.model_selection import train_test_split

df_churn = pd.read_csv("data/telecom_churn_clean.csv")
X = df_churn[['total_day_charge', 'total_eve_charge']].values
y= df_churn['churn'].values

X_train, X_test, y_train, y_test =train_test_split(X, y,
test_size=0.2, random_state=42)


knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train) #model eğitildi
y_pred = knn.predict(X_test) #tahmin yapılıyor

confusion_matrix(y_test, y_pred)
```

```
array([[548,  18],
       [ 69,  32]])
```

```
import seaborn as sns

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='g')
```

```
<Axes: >
```



```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.97      0.93       566
           1       0.64      0.32      0.42       101

    accuracy                           0.87       667
   macro avg       0.76      0.64      0.68       667
weighted avg       0.85      0.87      0.85       667
```

`classification_report(y_test, y_pred)`: Bu fonksiyon, **precision**, **recall**, **f1-score** ve **accuracy** gibi performans metriklerini hesaplar ve **raporlar**.

Ölçütler:

- a. **Precision**: **Doğruluk**; pozitif tahminlerin ne kadar doğru olduğunu gösterir. Formül:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- a. **Recall**: **Duyarlılık**; gerçek pozitiflerin ne kadarını doğru tahmin ettiğini gösterir. Formül:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- a. **F1-score**: Precision ve recall arasında **dengeleme** yapar. Formül:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- a. **Accuracy**: Modelin genel doğruluğu; doğru tahminlerin oranını gösterir:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Logistic Regression:

sinfilandirmak icin kullanilan model.

bağımlı değişkenin (hedef değişkenin) kategorik olduğu durumlarda kullanılır. Yani, sınıflandırma problemlerinde 0 ya da 1 gibi ikili sınıflar veya daha fazla sınıf tahmin etmek için kullanılır.

```
# Confusion matrix in scikit-learn

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.model_selection import train_test_split

df_churn = pd.read_csv("data/telecom_churn_clean.csv")
X = df_churn[['total_day_charge', 'total_eve_charge']].values
#bağımsız değişkenler
y= df_churn['churn'].values #bağımlı değişken

X_train, X_test, y_train, y_test =train_test_split(X, y,
test_size=0.2, random_state=42)

model_log = LogisticRegression()
model_log.fit(X_train, y_train) #model eğitildi

y_predict = model_log.predict(X_test) #doğrudan tahmin. 0 ya da 1
olur.(iptal etti - etmedi)
y_predict
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0])

y_pred_probs = model_log.predict_proba(X_test) #modelin her örnek için
her sınıfa ait olasılıkları döndürür. ilk sütun 0 olma olasılığı
ikinci satır 1 olma olasılığı
y_pred_probs
y_pred_probs = y_pred_probs[:,1] # 1 olma olasılığı seçiliyor
```

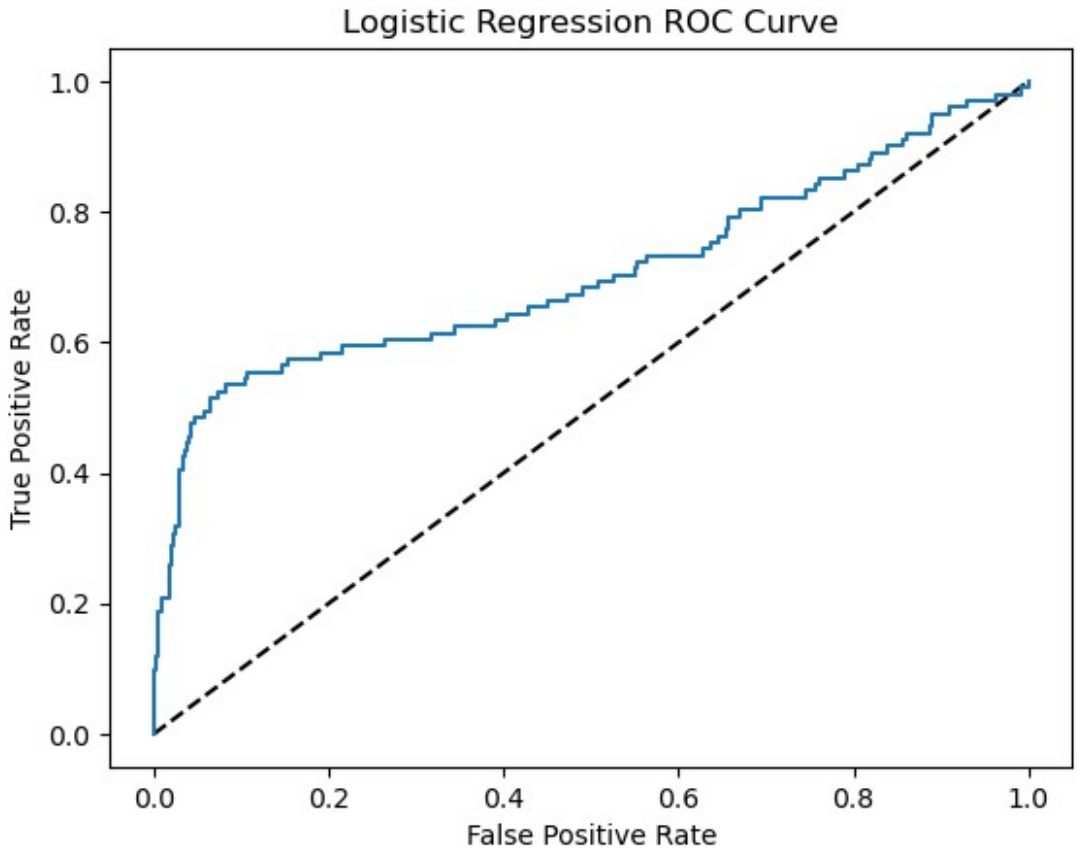
ROC CURVE:

sınıflandırma modelinin performansını değerlendirmek için kullanılan önemli bir görselleştirme aracıdır. ROC eğrisi, modelin gerçek pozitif (True Positive) ve yanlış pozitif (False Positive) oranlarını göstererek modelin sınıflandırma başarısını analiz eder.

```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Logistic Regression ROC Curve')

# Bu grafiğe dayanarak modelin performansı nasıl ölçülebilir?
```

```
Text(0.5, 1.0, 'Logistic Regression ROC Curve')
```




```
#**AUC (Area Under the Curve)**, **ROC eğrisinin altındaki alanı** ölçer ve modelin **genel performansını** bir sayı ile ifade eder. AUC şu şekilde yorumlanır:
#  - **AUC = 1**: Model mükemmel, tüm pozitifleri doğru sınıflandırmış ve yanlış pozitifleri hiç yapmamış.
#  - **AUC = 0.5**: Model, **rastgele tahmin** yapıyor.
#  - **AUC < 0.5**: Model, **tersine çalışıyor**, yani her tahminin zıttını yapıyor.

from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_pred_probs)
```

```
0.7049469964664311
```

Regression ve Classification için Performans Metrikleri Cheat Sheet

Aşağıda, **regresyon** ve **sınıflandırma** modelleri için kullanabileceğiniz uygun performans metriklerini ayırarak özetledim:

---

1. Regresyon Modelleri

Regresyon modelleri, sürekli bir bağımlı değişkeni tahmin eder.

Metot	Açıklama
R^2 (R-Squared)	Modelin bağımlı değişken varyansını ne kadar açıkladığını gösterir.
MSE (Mean Squared Error)	Hataların karelerinin ortalamasını hesaplar. Daha küçük değer daha iyidir.
RMSE (Root Mean Squared Error)	Hataların karekökünü alarak hatayı bağımlı değişken birimine indirger.
MAE (Mean Absolute Error)	Hataların mutlak değerlerinin ortalamasını hesaplar. Daha küçük değer daha iyidir.

Regresyon Modelleri:

- **Linear Regression**
- **Ridge Regression**
- **Lasso Regression**
- **Polynomial Regression**

---

2. Sınıflandırma Modelleri

Sınıflandırma modelleri, kategorik bir bağımlı değişkeni tahmin eder.

Metot	Açıklama
Accuracy	Doğru tahminlerin oranını verir.
Confusion Matrix	Doğru ve yanlış sınıflandırmaları gösterir.
Precision	Pozitif sınıf için doğru tahmin oranını gösterir (TP / (TP + FP)).
Recall (Sensitivity)	Pozitif sınıf için hatırlama oranını gösterir (TP / (TP + FN)).
F1-Score	Precision ve Recall'ün harmonik ortalamasıdır.
ROC-AUC	Sınıflandırıcı performansını ölçmek için Receiver Operating Characteristic eğrisi.

Sınıflandırma Modelleri:

- **Logistic Regression**
- **KNN (K-Nearest Neighbors)**
- **Decision Trees**
- **Random Forest**

- **SVM (Support Vector Machines)**

Hangi Metrik Nerede Kullanılmalı?

1. **Regresyon Modelleri**:
   - **R^2, MSE, RMSE, MAE** kullanılır.
   - **Confusion Matrix veya Accuracy gibi sınıflandırma metrikleri kullanılmaz.**
2. **Sınıflandırma Modelleri**:
   - **Confusion Matrix, Precision, Recall, F1-Score, Accuracy** gibi metrikler kullanılır.
   - **R^2, MSE, RMSE gibi regresyon metrikleri kullanılmaz.**

Örnek Kullanım Alanları

Regresyon:

- Bir çalışanın maaşını tahmin etmek.
- Ev fiyatlarını belirlemek.

Sınıflandırma:

- Bir müşterinin ürünü satın alıp almayacağını tahmin etmek.
- Bir hastanın hastalığa sahip olup olmadığını sınıflandırmak.

Grid Search Cross Validation

Grid Search Cross Validation (Grid Search Çapraz Doğrulama), makine öğrenmesi modelinin hiperparametrelerini optimize etmek için kullanılan bir yöntemdir. Bu yöntem, belirli bir modelin performansını değerlendirirken farklı hiperparametre kombinasyonlarını dener ve en iyi sonucu veren parametreleri seçer. Grid Search, "grid" adı verilen bir hiperparametre ızgarası üzerinde sistematik olarak arama yaparak en iyi model parametrelerini bulur.

Grid Search'in Çalışma Prensibi

Grid Search, parametre arama alanını tanımladıktan sonra, her bir parametre kombinasyonunu belirli bir çapraz doğrulama stratejisi (örneğin,

k-fold çapraz doğrulama) ile değerlendirir. Bu süreç, aşağıdaki adımları takip eder:

Modeli parametrelerin her kombinasyonu için eğit.

Çapraz doğrulama kullanarak her parametre kombinasyonu için doğrulama performansını değerlendir.

En yüksek doğrulama performansına sahip parametre kombinasyonunu seç.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split, GridSearchCV,
KFold
import numpy as np
```

```
# Veri setini okuma
df_diabets = pd.read_csv('data/diabetes_clean.csv')

# Boş verileri (bmi=0 veya glucose=0) filtreleme
df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) |
(df_diabets['glucose'] == 0)]
df_diabets.drop(df_filtered.index, inplace=True)

# Bağımsız değişkenler (X) ve bağımlı değişken (y) oluşturma
X = df_diabets.drop('glucose', axis=1).values
y = df_diabets['glucose'].values

# Eğitim ve test verilerine ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# K-Fold çapraz doğrulama için KFold sınıfını oluşturuyoruz
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Hiperparametreler için arama alanını (grid) oluşturuyoruz
param_grid = {
    'alpha': np.arange(0.0001, 1, 0.1),  # alpha parametresinin farklı
değerlerini deniyoruz
    'solver': ['sag', 'lsqr']  # Ridge regresyonu için farklı çözüm
yöntemleri
}
#- **GridSearchCV**, verilen **param_grid** üzerinde
hiperparametreleri optimize etmek için kullanılır. Burada, **Ridge**
modelinin **α (alpha)** parametresi üzerinde en uygun değeri bulmak
için **5 katlı çapraz doğrulama** yapılır.

# Ridge regresyon modelini oluşturuyoruz
ridge = Ridge()

# GridSearchCV ile Grid Search başlatıyoruz
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf)

# Modeli eğitim verileriyle eğitiyoruz
ridge_cv.fit(X_train, y_train)

# En iyi parametreleri ve en iyi skoru yazdırıyoruz
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/
_sag.py:349: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
```

```
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
```

```
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
```

```
coef_ did not converge
  warnings.warn(
```

```
{'alpha': 0.0001, 'solver': 'lsqr'} 0.3404176885506861
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/
_sag.py:349: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
```

RandomizedSearchCV

RandomizedSearchCV: Grid search'in daha verimli bir alternatifidir. Grid search'te tüm kombinasyonları denemek yerine, belirli bir sayıda rastgele kombinasyon seçer ve optimize eder. Bu, daha az hesaplama maliyetiyle hızlı sonuçlar elde etmenizi sağlar.

```python
from sklearn.model_selection import RandomizedSearchCV

# İsteğe bağlı olarak, test edilen hiperparametre değerlerinin
sayısını belirleyen
# n_iter bağımsız değişkenini ayarlanabilir.
# Böylece n_iter iki olarak ayarlandığında
# beş katlı çapraz doğrulama 10 fit() gerçekleştirir.

#RandomizedSearchCV:
#Grid Search'ten farklı olarak RandomizedSearchCV, belirli bir
parametre kombinasyonu yerine rastgele bir seçki ile
hiperparametreleri dener. Bu yöntem, büyük veri setlerinde ve
parametrelerin çok olduğu durumlarda daha hızlı sonuçlar verebilir.
from sklearn.model_selection import RandomizedSearchCV

# RandomizedSearchCV kullanımı
ridge_cv = RandomizedSearchCV(ridge, param_grid, cv=kf, n_iter=2)
ridge_cv.fit(X_train, y_train)

# Sonuçları görüntüle
print(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
{'solver': 'sag', 'alpha': 0.10010000000000001} 0.24297917653517223
```

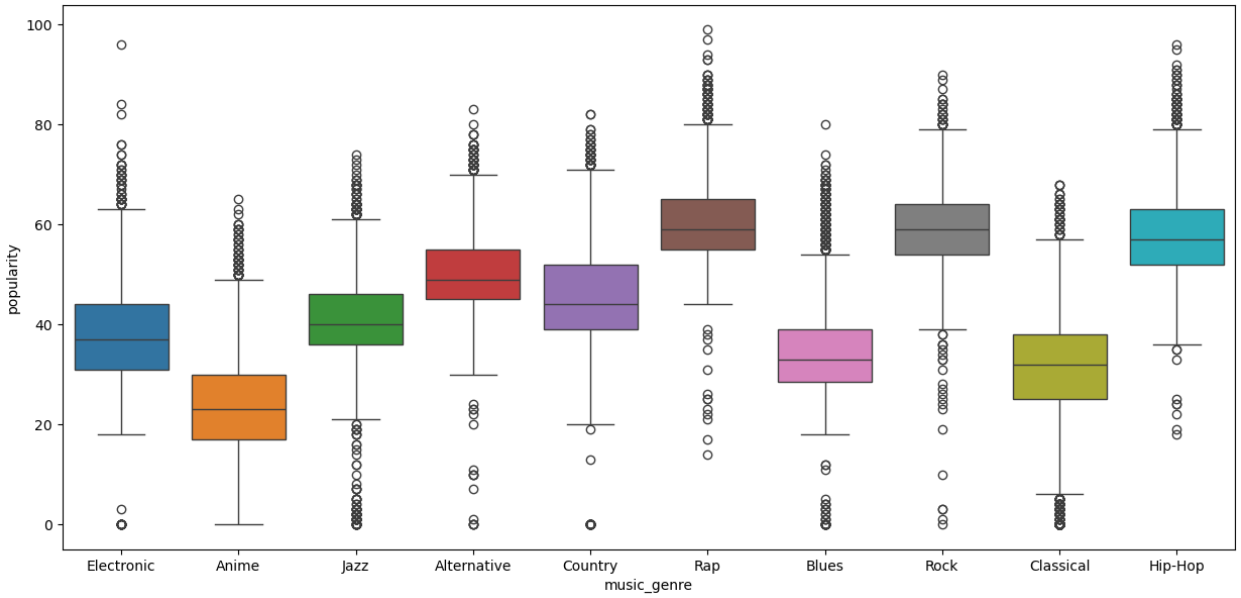
```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/
_sag.py:349: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/linear_model/_sag.
py:349: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  warnings.warn(
```

Preprocessing Data

```python
import pandas as pd

df_music = pd.read_csv('data/music_genre.csv')
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 7))
sns.boxplot(data=df_music, x='music_genre', y='popularity',
hue='music_genre')
```

```
<Axes: xlabel='music_genre', ylabel='popularity'>
```




```
music_dummies = pd.get_dummies(df_music['music_genre'],
drop_first=True, dtype='int')
music_dummies.head()
```

	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock
0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0	0
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	0	0	0

Dummy değişkenleri, **kategorik verileri** sayısal verilere dönüştürmek için kullanılan bir tekniktir. Bu, makine öğrenmesi ve istatistiksel analizlerde yaygın olarak kullanılır çünkü çoğu algoritma **sayılarla** çalışır ve **kategorik** (metin, etiket gibi) verilerle doğrudan işlem yapamaz.

Dummy Değişkenleri Nedir?

Bir **kategorik değişkeni** sayısal verilere dönüştürmenin en yaygın yolu **dummy değişkenleri** kullanmaktır. Örneğin, **müzik türü (music_genre)** bir kategorik değişken olabilir. Bu tür bir veriyi sayısal hale getirmek için **her bir kategori için bir sütun** (veya dummy değişkeni) oluşturulur.

Örneğin, aşağıdaki **music_genre** kategorik verisini düşünelim:

music_genre
Pop

music_genre
Jazz
Rock
Pop
Jazz

Bu veriyi **dummy değişkenlerine** dönüştürmek için şu adımlar izlenir:

Dummy Değişkenleri Oluşturma

1. **Her Bir Kategoriyi Yeni Bir Sütunla Temsil Etme:**

Her müzik türü için bir sütun oluşturulacak ve her satırda, o müzik türüne ait olma durumu **1 (vardır)** veya **0 (yoktur)** ile gösterilecektir.

music_genre	Pop	Jazz	Rock
Pop	1	0	0
Jazz	0	1	0
Rock	0	0	1
Pop	1	0	0
Jazz	0	1	0

Bu işlem, **kategorik değişkenleri** (örneğin **music_genre**) **sayısal verilere** dönüştürmüş olur. Bu sayede makine öğrenmesi algoritmaları, bu veriler üzerinde işlem yapabilir.

2. `drop_first=True` Parametresi:

- Bazı durumlarda, **ilk kategori** (veya "ilk sütun") çıkarılabilir. Bu, **multicollinearity** sorununu engeller, çünkü **ilk sütun** diğerlerinin **kombinasyonu** olarak zaten temsil edilebilir.
- Bu teknik, regresyon modellerinde genellikle kullanılır çünkü **aşırı parametre sayısı** modelin performansını düşürebilir.

Örneğin, yukarıdaki tablodan **Pop** sütununu çıkaralım (ilk sütun):

music_genre	Jazz	Rock
Pop	0	0
Jazz	1	0
Rock	0	1
Pop	0	0
Jazz	1	0

Burada, **Pop** müzik türü **referans kategori** olarak kabul edilir, ve **Jazz** ve **Rock** sütunları **bu kategoriye göre** ölçülür.

Kod Açıklaması:

```
music_dummies = pd.get_dummies(df_music['music_genre'],
drop_first=True, dtype='int')
```

- **`pd.get_dummies(df_music['music_genre'])`**: Bu, **music_genre** sütunundaki her kategori için bir **dummy değişkeni** oluşturur. Kategorik veriler sayısal verilere dönüştürülür.
- **`drop_first=True`**: İlk sütunu (ilk kategori, örneğin **Pop**) çıkarır. Bu, **çoklu doğrusal ilişkiyi** (multicollinearity) engeller. Çünkü ilk kategori zaten diğerlerinin **kombinasyonu** olarak temsil edilebilir.
- **`dtype='int'`**: Dummy değişkenleri **tam sayı (integer)** türünde olacak şekilde ayarlanır.

Dummy Değişkenlerini Kullanmanın Avantajları:

- **Makine öğrenmesi modelleri** kategorik verileri doğrudan işleyemez. Dummy değişkenleri, algoritmaların bu tür verilerle **verimli bir şekilde çalışabilmesini** sağlar.
- **Modelin daha doğru öğrenmesi** için gereklidir, çünkü kategorik veriler modelin öğrenmesi için gereklidir.

Özet:

Dummy değişkenleri, kategorik değişkenleri **sayısal verilere dönüştürmek** için kullanılır ve bu sayede makine öğrenmesi algoritmaları bu verilerle işlem yapabilir. **`drop_first=True`** parametresi, modelin daha verimli çalışması için genellikle kullanılır, çünkü bu, multicollinearity problemini önler.

```
music_dummies = pd.concat([df_music, music_dummies], axis=1)
#oluşturulan dummy değişkenleri işe veri seti birleştirilir
music_dummies = music_dummies.drop(['music_genre', 'instance_id'],
axis=1) #dummy değişkeniyle değiştirildiği için music_genre veri
setinden çıkarılır.
music_dummies
```

```
       popularity  acousticness  danceability  duration_ms  energy  \
0              27       0.00468         0.652           -1   0.941
1              31       0.01270         0.622       218293   0.890
2              28       0.00306         0.620       215613   0.755
3              34       0.02540         0.774       166875   0.700
4              32       0.00465         0.638       222369   0.587
...           ...           ...           ...          ...     ...
44991          59       0.03340         0.913           -1   0.574
44992          72       0.15700         0.709       251860   0.362
44993          51       0.00597         0.693       189483   0.763
44994          65       0.08310         0.782       262773   0.472
```

```
44995          67       0.10200         0.862       267267   0.642

       instrumentalness  liveness  loudness  speechiness    tempo  ...
\
0               0.79200     0.115    -5.201       0.0748  100.889  ...

1               0.95000     0.124    -7.043       0.0300  115.002  ...

2               0.01180     0.534    -4.617       0.0345  127.994  ...

3               0.00253     0.157    -4.498       0.2390  128.014  ...

4               0.90900     0.157    -6.266       0.0413  145.036  ...

...                 ...       ...       ...          ...      ...  ...

44991           0.00000     0.119    -7.022       0.2980   98.028  ...

44992           0.00000     0.109    -9.814       0.0550  122.043  ...

44993           0.00000     0.143    -5.443       0.1460  131.079  ...

44994           0.00000     0.106    -5.016       0.0441   75.886  ...

44995           0.00000     0.272   -13.652       0.1010   99.201  ...


       valence  Anime  Blues  Classical  Country  Electronic  Hip-Hop
Jazz  \
0        0.759      0      0          0        0           1        0
0
1        0.531      0      0          0        0           1        0
0
2        0.333      0      0          0        0           1        0
0
3        0.270      0      0          0        0           1        0
0
4        0.323      0      0          0        0           1        0
0
...        ...    ...    ...        ...      ...         ...      ...
...
44991    0.330      0      0          0        0           0        1
0
44992    0.113      0      0          0        0           0        1
0
44993    0.395      0      0          0        0           0        1
0
44994    0.354      0      0          0        0           0        1
0
44995    0.765      0      0          0        0           0        1
```

```
0
```

```
       Rap  Rock
0        0     0
1        0     0
2        0     0
3        0     0
4        0     0
...    ...   ...
44991    0     0
44992    0     0
44993    0     0
44994    0     0
44995    0     0

[44996 rows x 31 columns]
```

```
# Linear regression with dummy variables

from sklearn.model_selection import cross_val_score, KFold,
train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np

X = music_dummies.drop('popularity', axis=1).values
y = music_dummies['popularity'].values

X_train, X_test, y_train, y_test =train_test_split(X, y,
test_size=0.2, random_state=42)

kf = KFold(n_splits=5, shuffle=True, random_state=42) #modelin
genelleme yeteneği ölçülmek için veri seti 5 parçaya bölünür

model_reg = LinearRegression() #lineer regreyon modeli oluşturuldu
model_reg_cv = cross_val_score(model_reg, X_train, y_train, cv=kf,
                               scoring='neg_mean_squared_error')
#kfold ile değerlendirilmesi sağlanır. parametresi, modelin
doğruluğunu ölçerken negatif ortalama kare hata (MSE) metriğini
kullanır. MSE, modelin tahminleri ile gerçek değerler arasındaki
farkların karesinin ortalamasıdır. Negatif işareti kullanmak,
sklearn'ın hata metriğini minimize etmek için gereklidir, çünkü
sklearn genellikle daha büyük değerleri iyi olarak değerlendirir.

np.sqrt(-model_reg_cv) #MSE'nin karekökü alınarak RMSE (Root Mean
Squared Error) hesaplanır. RMSE, modelin doğruluğunu ölçmek için
yaygın olarak kullanılan bir metriktir ve hata oranının daha anlaşılır
bir şekilde sunulmasını sağlar.

array([9.55195316, 9.3511157 , 9.426638  , 9.61542024, 9.60686241])
```

SIMPLE IMPUTER:

eksik değerleri bir stratejiye göre doldurmak için kullanılır.

```
import pandas as pd  # pandas kütüphanesini veri işleme ve analiz için içeri aktarır
df_house = pd.read_csv('data/AmesHousing.csv')  # 'AmesHousing.csv' dosyasını df_house veri çerçevesine yükler

# Veri çerçevesindeki her sütundaki eksik (NaN) değerlerin sayısını hesaplar ve en fazla eksik veriye sahip sütundan başlayarak sıralar
df_house.isna().sum().sort_values(ascending=False)

na_series = df_house.isna().sum()  # Her sütundaki eksik veri sayısını hesaplar ve na_series'e atar
data_len = len(df_house) * 0.05  # Toplam satır sayısının %5'ini hesaplar, eksik verilerin bu orandan az olup olmadığını görmek için
na_series[(na_series < data_len) & (na_series != 0)]  # Eksik veri sayısı toplam satır sayısının %5'inden az olan ve 0'dan farklı olan sütunları filtreler

# Eksik veri sayısı %5'ten az olan sütunların isimlerini alır
col_names = list(na_series[(na_series <= data_len) & (na_series != 0)].keys())

# Belirtilen sütunlardaki eksik verilere sahip satırları çıkarır.
'dropna' fonksiyonu, sadece 'col_names' listesinde yer alan sütunlardaki eksik verilere sahip satırları siler.
df_house = df_house.dropna(subset=col_names)

# df_house veri çerçevesindeki kategorik sütunların isimlerini alır
# select_dtypes(include='object') komutu ile kategorik veri türündeki tüm sütunları seçeriz.
object_cols = list(df_house.select_dtypes(include='object').columns)
X_cat = df_house[object_cols]  # Kategorik verileri X_cat değişkenine atar.

# Kategorik sütunları çıkararak sayısal sütunlardan oluşan yeni bir veri çerçevesi oluşturur.
# Bu, modelin eğitimi için sayısal verileri yalnızca kullanmak amacıyla yapılır.
X_nums = df_house.drop(object_cols, axis=1)

# Bağımlı değişkeni (SalePrice) bağımsız değişkenlerden ayırır ve bağımlı değişkeni y olarak atar.
# 'SalePrice' sütunu hedef değişken olup, modelin tahmin etmeye çalıştığı değeri temsil eder.
y = X_nums['SalePrice'].values.reshape(-1, 1)  # 'SalePrice' sütunu bağımlı değişken olarak seçilir ve 2D array'e dönüştürülür.
```

```
# 'SalePrice' sütununu X_nums veri çerçevesinden çıkarır, böylece
X_nums sadece bağımsız değişkenlerden oluşur.
# Burada, modelin eğitiminde kullanılacak olan bağımsız değişkenlerin
bulunduğu veri çerçevesi oluşturulmuş olur.
X_nums.drop('SalePrice', inplace=True, axis=1)  # 'SalePrice' bağımsız
değişkenler arasından çıkarılır ve X_nums sadece bağımsız değişkenleri
içerir.

from sklearn.model_selection import train_test_split

# X_cat ve X_nums veri setlerini, bağımlı değişken y ile birlikte %80
eğitim ve %20 test olarak ayırır.
X_train_cat, X_test_cat, y_train_cat, y_test_cat =
train_test_split(X_cat, y, test_size=0.2, random_state=42)
X_train_nums, X_test_nums, y_train_nums, y_test_nums =
train_test_split(X_nums, y, test_size=0.2, random_state=42)

from sklearn.impute import SimpleImputer

# Kategorik verilerdeki eksik değerleri doldurmak için SimpleImputer
kullanılır.
# Bu durumda, eksik değerler en sık görülen (most frequent) değere
göre doldurulacaktır.
imp_cat = SimpleImputer(strategy="most_frequent")
X_train_cat = imp_cat.fit_transform(X_train_cat)  # Eğitim setindeki
eksik değerler, en sık görülen değerle doldurulur
X_test_cat = imp_cat.fit_transform(X_test_cat)    # Test setindeki
eksik değerler de en sık görülen değerle doldurulur

# Sayısal verilerdeki eksik değerleri doldurmak için SimpleImputer
kullanılır.
# Burada eksik değerler, sayısal sütunların ortalamasıyla
doldurulacaktır.
imp_num = SimpleImputer()  # Varsayılan olarak strategy='mean' yani
ortalama ile doldurma yapılır.
X_train_nums = imp_num.fit_transform(X_train_nums)  # Eğitim setindeki
eksik sayısal değerler ortalama ile doldurulur
X_test_nums = imp_num.fit_transform(X_test_nums)    # Test setindeki
eksik sayısal değerler de ortalama ile doldurulur
X_train = np.append(X_train_nums, X_train_cat, axis=1)  # Sayısal
veriler (X_train_nums) ve kategorik veriler (X_train_cat)
birleştirilir. axis=1 parametresi, sütunlar boyunca birleştirme yapar.
X_test = np.append(X_test_nums, X_test_cat, axis=1)    # Test
setindeki sayısal veriler (X_test_nums) ve kategorik veriler
(X_test_cat) birleştirilir.
```

PIPELINE:

Pipeline, makine öğrenimi ve veri işleme işlemlerini adım adım düzenli bir şekilde birbirine bağlayan bir işlem hattı (workflow) oluşturur. Bu, özellikle veri ön işleme ve modelleme işlemlerini tek bir nesneyle yönetmek için oldukça faydalıdır.

`Pipeline` kullanarak verileri işlemeden başlayıp bir modelle tahmin yapmaya kadar bir dizi işlem gerçekleştiren bir makine öğrenimi modelini oluşturur. **Pipeline** kullanımı, işlem adımlarının düzenli ve tekrarlanabilir bir şekilde uygulanmasını sağlar

```
from sklearn.pipeline import Pipeline  # Pipeline sınıfını içeri
aktarır, ardışık işlemleri düzenlemek için kullanılır
import pandas as pd  # pandas kütüphanesini veri işleme ve analiz için
içeri aktarır

df_music = pd.read_csv('data/music_clean.csv')  # 'music_clean.csv'
dosyasını df_music veri çerçevesine yükler
df_music = df_music.dropna(subset=['genre', 'popularity', 'loudness',
'liveness', 'tempo'])  # Belirtilen sütunlarda eksik veri bulunan
satırlar çıkarılır

X = df_music.drop('genre', axis=1).values  # 'genre' dışında kalan tüm
sütunlar bağımsız değişkenler olarak X'e atanır
y = df_music['genre'].values  # 'genre' sütunu bağımlı değişken olarak
y'ye atanır

# Bir ardışık düzende, son adım hariç her adımın bir dönüştürücü
olması gerektiği unutulmamalıdır
# Bu açıklama, pipeline içerisinde kullanılan her bir adımın bir veri
dönüştürücü (transformer) olması gerektiğini belirtir.

from sklearn.linear_model import LogisticRegression  #
LogisticRegression modelini içeri aktarır

# Pipeline içerisinde yapılacak işlemler sırasıyla tanımlanır:
# 1. Eksik verilerin doldurulması (SimpleImputer ile)
# 2. Lojistik regresyon modelinin uygulanması
steps = [
    ('imputation', SimpleImputer()),  # Eksik veriler SimpleImputer
ile doldurulacak
    ('logistic_regression', LogisticRegression())  # Lojistik
regresyon modeli uygulanacak
]

pipeline = Pipeline(steps=steps)  # Tanımlanan adımları içeren
pipeline oluşturulur

# Veriyi eğitim ve test setlerine ayırır
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)  # %80 eğitim, %20 test seti olarak
```

```
ayırır

pipeline.fit(X_train, y_train)  # Pipeline içerisindeki tüm adımlar sırasıyla eğitim seti üzerinde çalıştırılır ve model eğitilir
pipeline.score(X_test, y_test)  # Test seti üzerinde modelin doğruluğu hesaplanır
```

STANDART SCALER:

StandardScaler (Standartlaştırıcı), veri ön işleme (preprocessing) adımlarında yaygın olarak kullanılan bir tekniktir. Verilerin özelliklerini (sütunlarını) ortalama 0 ve standart sapma 1 olacak şekilde ölçeklendiren bir araçtır. Veri setindeki her bir özelliğin (özellikle sayısal sütunların) dağılımını standartlaştırarak modelleme sürecine uygun hale getirir.

```
import pandas as pd  # pandas kütüphanesini veri işleme ve analiz için içeri aktarır

df_music = pd.read_csv('data/music_genre.csv')  # 'music_genre.csv' dosyasını df_music veri çerçevesine yükler

df_music.describe().T  # df_music veri çerçevesinin sayısal sütunlarıyla ilgili temel istatistiksel bilgileri gösterir. .T ile veriler transpose (satır-sütun değiştirerek) yapılır.

from sklearn.preprocessing import StandardScaler  # Verileri standartlaştırmak için StandardScaler sınıfını içeri aktarır
from sklearn.model_selection import train_test_split  # Eğitim ve test verisine ayırmak için train_test_split fonksiyonunu içeri aktarır
import numpy as np  # Sayısal işlemler için numpy kütüphanesini içeri aktarır

# 'music_genre' sütunu hariç tüm sütunları bağımsız değişkenler olarak seçer
X = df_music.drop('music_genre', axis=1).values  # Bağımsız değişkenler X olarak atanır
y = df_music['music_genre'].values  # Bağımlı değişken (hedef) olan 'music_genre' y olarak atanır

# Veriyi eğitim (%80) ve test (%20) setlerine ayırır. random_state=42, işlemin tekrar edilebilirliğini sağlar.
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

scaler = StandardScaler()  # Verileri standartlaştırmak için StandardScaler nesnesi oluşturulur
X_train_scaled = scaler.fit_transform(X_train)  # Eğitim verileri üzerinde standardizasyon işlemi yapılır
X_test_scaled = scaler.fit_transform(X_test)    # Test verileri üzerinde standardizasyon işlemi yapılır
```

```python
# Orijinal verilerin ortalaması ve standart sapması hesaplanır
print(np.mean(X), np.std(X))  # Orijinal X verilerinin (standartlaştırılmamış) ortalama ve standart sapması yazdırılır

# Standardize edilmiş eğitim verilerinin ortalaması ve standart sapması hesaplanır
print(np.mean(X_train_scaled), np.std(X_train_scaled))  # Standardize edilmiş X_train verilerinin ortalama ve standart sapması yazdırılır
```

```
23100.432767763512 72094.37978367604
-2.132128274398108e-15 1.000000000000005
```

```python
from sklearn.pipeline import Pipeline  # Pipeline sınıfını içeri aktarır, ardışık işlemleri düzenlemek için kullanılır
from sklearn.neighbors import KNeighborsClassifier  # KNN sınıflandırıcı sınıfını içeri aktarır

# Pipeline içerisindeki adımlar belirlenir:
# 1. 'scaler' adımında veriler StandardScaler ile standartlaştırılacak.
# 2. 'knn' adımında K-en yakın komşu (KNN) sınıflandırıcı kullanılacak.
steps = [
    ('scaler', StandardScaler()),  # İlk adım: Verileri standartlaştırmak için StandardScaler kullanılır.
    ('knn', KNeighborsClassifier(n_neighbors=6))  # İkinci adım: K-en yakın komşu sınıflandırıcı (KNN) kullanılır, n_neighbors=6 ile 6 komşu dikkate alınır.
]

# Pipeline içerisinde belirlenen adımlar birleştirilir ve pipeline nesnesi oluşturulur.
pipeline = Pipeline(steps)

# Eğitim verisini kullanarak pipeline'daki işlemler sırasıyla gerçekleştirilir:
# 1. Eğitim verileri standartlaştırılır (scaler).
# 2. KNN sınıflandırıcısı eğitim verileri üzerinde eğitilir.
knn_scaled = pipeline.fit(X_train, y_train)

# Test verisi üzerinde tahmin yapılır: Test verisi, ilk adımda standartlaştırılır ve ardından KNN sınıflandırıcısı ile tahmin yapılır.
y_pred = knn_scaled.predict(X_test)

# Modelin doğruluğu hesaplanır: Test verisi ile yapılan tahminlerin doğruluğu ölçülür.
# Sonuç, test verisindeki doğru sınıflandırmaların oranıdır.
```

```
knn_scaled.score(X_test, y_test)  # Test verisinde doğruluk oranı hesaplanır ve döndürülür
```

```
0.48088888888888887
```

```
# Comparing performance using unscaled data

# KNeighborsClassifier sınıfını kullanarak bir model oluşturuluyor.
# Bu model, **veri standartlaştırılmadan** (ölçeklenmeden) eğitim verileri üzerinde eğitilir.
knn_unscaled = KNeighborsClassifier(n_neighbors=6).fit(X_train, y_train)  # Eğitim verisi ile model eğitilir

# Modelin test verisi üzerindeki doğruluğu hesaplanır.
# Bu işlem, test setindeki doğru sınıflandırmaların oranını döndürür.
knn_unscaled.score(X_test, y_test)  # Test verisi ile doğruluk oranı hesaplanır
```

```
0.1272222222222222
```

```
from sklearn.model_selection import GridSearchCV  # GridSearchCV sınıfını içeri aktarır, hiperparametre optimizasyonu için kullanılır
import numpy as np  # Sayısal işlemler için numpy kütüphanesini içeri aktarır

# Pipeline içerisinde yapılacak işlemler sırasıyla tanımlanır:
# 1. 'scaler' adımında veriler StandardScaler ile standartlaştırılacak.
# 2. 'knn' adımında K-en yakın komşu (KNN) sınıflandırıcı kullanılacak.
steps = [
    ('scaler', StandardScaler()),  # İlk adım: Veriler standartlaştırılacak
    ('knn', KNeighborsClassifier())  # İkinci adım: K-en yakın komşu sınıflandırıcısı (KNN) uygulanacak
]

# Pipeline içerisinde belirlenen adımlar birleştirilir ve pipeline nesnesi oluşturulur.
pipeline = Pipeline(steps)

# Hiperparametre grid'ini tanımlar: Burada 'knn' modelinin 'n_neighbors' parametresi için 1'den 50'ye kadar olan değerler denenir.
parameters = {"knn__n_neighbors" : np.arange(1, 50)}  # 'knn__n_neighbors' parametresi 1 ile 50 arasında ayarlanacak

# GridSearchCV, verilen parametrelerin her kombinasyonunu deneyerek en iyi parametreyi bulur
cv = GridSearchCV(pipeline, param_grid=parameters)  # Pipeline içerisinde hiperparametre grid'ini kullanarak GridSearchCV nesnesi
```

```
oluşturulur

# Eğitim verisi üzerinde GridSearchCV uygulanır
cv.fit(X_train, y_train)  # GridSearchCV ile belirtilen parametrelerle model eğitilir

# En iyi performans ve parametreler yazdırılır
(cv.best_score_, cv.best_params_)  # GridSearchCV'nin bulduğu en iyi doğruluk (score) ve en iyi parametreler döndürülür
```

```
(0.5193075581485083, {'knn__n_neighbors': 38})
```

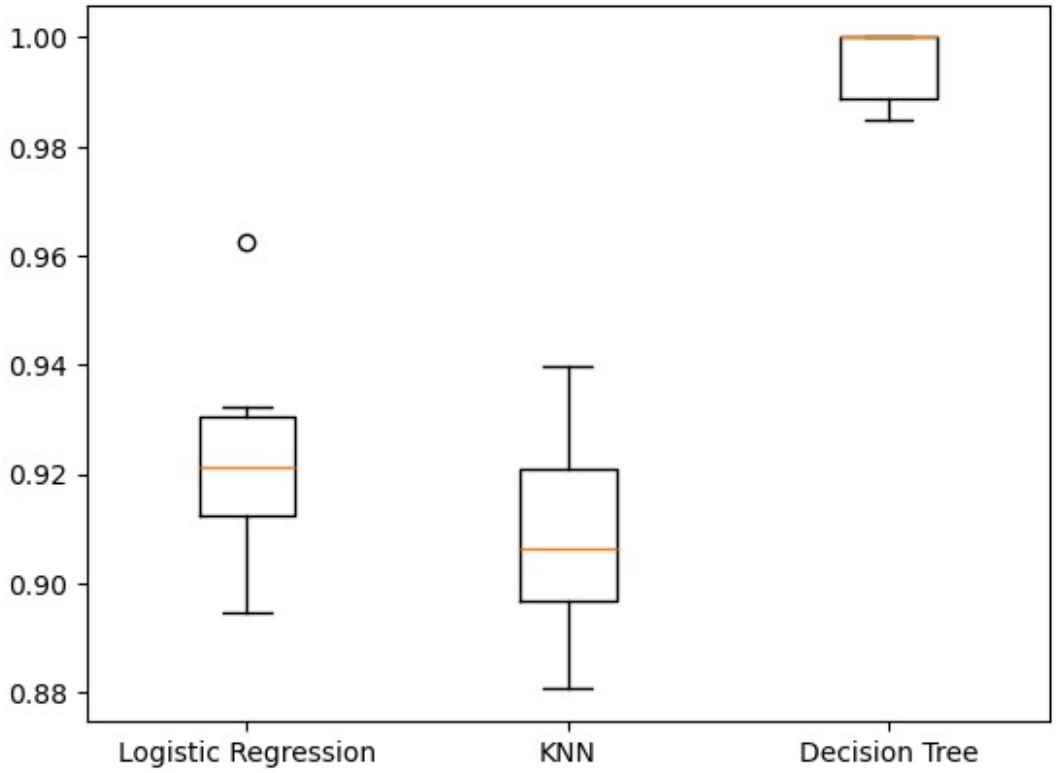
```
import matplotlib.pyplot as plt  # Görselleştirme için matplotlib kütüphanesini içeri aktarır

from sklearn.preprocessing import StandardScaler  # Veriyi standartlaştırmak için StandardScaler sınıfını içeri aktarır
from sklearn.model_selection import cross_val_score, KFold, train_test_split  # Veri bölme ve çapraz doğrulama için gerekli sınıflar içeri aktarılır
from sklearn.neighbors import KNeighborsClassifier  # KNN sınıflandırıcı sınıfını içeri aktarır
from sklearn.linear_model import LogisticRegression  # Lojistik regresyon sınıfını içeri aktarır
from sklearn.tree import DecisionTreeClassifier  # Karar ağacı sınıflandırıcı sınıfını içeri aktarır

df_music = pd.read_csv("data/music_clean.csv")  # 'music_clean.csv' dosyasını df_music veri çerçevesine yükler

X = df_music.drop('genre', axis=1).values  # 'genre' dışındaki tüm sütunlar bağımsız değişkenler (X) olarak seçilir
y = df_music['genre'].values  # 'genre' sütunu bağımlı değişken (y) olarak seçilir

# Eğitim ve test verisi olarak %80 eğitim, %20 test ayrılır.
# random_state=42, işlemi tekrarladığınızda aynı bölmeyi sağlar.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Veriyi standartlaştırmak için StandardScaler sınıfı kullanılır.
# Eğitim ve test verilerine ayrı ayrı uygulanır.
scaler = StandardScaler()  # StandardScaler nesnesi oluşturulur
X_train_scaled = scaler.fit_transform(X_train)  # Eğitim verileri üzerinde fit işlemi yapılır ve veriler dönüştürülür
X_test_scaled = scaler.fit_transform(X_test)    # Test verileri üzerinde fit işlemi yapılır ve veriler dönüştürülür

# Kullanılacak modellerin tanımlandığı bir sözlük oluşturulur.
models = {
```

```
    "Logistic Regression" : LogisticRegression(),  # Lojistik
regresyon modeli
    "KNN": KNeighborsClassifier(),  # KNN sınıflandırıcısı
    "Decision Tree": DecisionTreeClassifier()  # Karar ağacı
sınıflandırıcısı
}

results = []  # Çapraz doğrulama sonuçlarını saklayacak boş bir liste
oluşturulur

# Modellerin her birini sırasıyla test eder ve sonuçları 'results'
listesine ekler
for model in models.values():
    kf = KFold(n_splits=6, random_state=True, shuffle=True)  # KFold
ile 6 katmanlı çapraz doğrulama yapılır
    cv_results = cross_val_score(model, X_train_scaled, y_train,
cv=kf)  # Modelin doğruluğu çapraz doğrulama ile hesaplanır
    results.append(cv_results)  # Hesaplanan doğruluk sonuçları
'results' listesine eklenir

# Modellerin doğruluk sonuçlarını görselleştirmek için boxplot çizilir
plt.boxplot(results, labels=models.keys())  # Boxplot, her modelin
doğruluk dağılımını gösterir
```

```
/var/folders/zf/gnk755n14rj48wl97n0lv_0w0000gn/T/
ipykernel_1908/2990527257.py:38: MatplotlibDeprecationWarning: The
'labels' parameter of boxplot() has been renamed 'tick_labels' since
Matplotlib 3.9; support for the old name will be dropped in 3.11.
  plt.boxplot(results, labels=models.keys())  # Boxplot, her modelin
doğruluk dağılımını gösterir
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1636734d0>,
  <matplotlib.lines.Line2D at 0x163673c50>,
  <matplotlib.lines.Line2D at 0x1636714c0>,
  <matplotlib.lines.Line2D at 0x163671040>,
  <matplotlib.lines.Line2D at 0x161d502c0>,
  <matplotlib.lines.Line2D at 0x161d50a10>],
 'caps': [<matplotlib.lines.Line2D at 0x1636728a0>,
  <matplotlib.lines.Line2D at 0x1636728d0>,
  <matplotlib.lines.Line2D at 0x163670c20>,
  <matplotlib.lines.Line2D at 0x163673a10>,
  <matplotlib.lines.Line2D at 0x161d50e30>,
  <matplotlib.lines.Line2D at 0x161d51040>],
 'boxes': [<matplotlib.lines.Line2D at 0x161c92600>,
  <matplotlib.lines.Line2D at 0x1636717f0>,
  <matplotlib.lines.Line2D at 0x161d50860>],
 'medians': [<matplotlib.lines.Line2D at 0x163672510>,
  <matplotlib.lines.Line2D at 0x163670170>,
  <matplotlib.lines.Line2D at 0x161d51310>],
```

```
'fliers': [<matplotlib.lines.Line2D at 0x163671d90>,
 <matplotlib.lines.Line2D at 0x161d50980>,
 <matplotlib.lines.Line2D at 0x161d51610>],
'means': []}
```



KNN modelinin doğruluk dağılımı en geniş olan modeldir ve uç değerler dikkat çekicidir. Bu, parametre ayarlamaları (örneğin, komşu sayısı) ile iyileştirilebilir. Karar Ağacı daha belirsiz sonuçlar verir ve doğruluk dağılımı oldukça yayılmıştır. Bu, modelin bazen çok iyi, bazen de çok kötü sonuçlar verdiğini gösterir. Lojistik Regresyon ise genellikle sabit ve yüksek doğruluk değerleri üretmektedir. Bu, modelin doğruluğunun daha istikrarlı olduğunu gösterir.