Bir okulda öğrencilerin sınav puanlarının ortalamasının 80 olup olmadığını test edin. bir grup ve oran olduğu için -> tek örneklem testi

Çift yönlü test (two-tailed test), H_0 (null hipotezi) ve H_1 (alternatif hipotez)'in belirli bir değeri eşit ya da farklı olduğunu test etmek için kullanılır. Bu tür testlerde, hem daha büyük hem de daha küçük farkların anlamlı olup olmadığına bakılır.

Neden Çift Yönlü Test Yapıyoruz? Tek örneklem t-testinin (one-sample t-test) formülünü ele alalım:

H_0: Örneklem ortalaması, popülasyon ortalamasına eşittir. ($\mu = 80$) H_1: Örneklem ortalaması, popülasyon ortalamasından farklıdır. ($\mu \neq 80$) Bu durumda, H_1'deki "farklıdır" ifadesi, iki olasılığı içerir:

Örneklem ortalaması 80'den büyükse. Örneklem ortalaması 80'den küçükse. Bu yüzden, hem sağ hem de sol kuyrukları kontrol etmemiz gerekir. Yani çift yönlü test yapılır.

```
import numpy as np
from scipy import stats
alpha = 0.05
# Öğrencilerin sınav puanları (örnek veri)
exam_scores = [85, 90, 78, 88, 92, 77, 80, 84, 91, 89]
ort = np.mean(exam_scores)
n = len(exam_scores)
hyp = 80
s = np.std(exam_scores, ddof=1)
payda = np.sqrt( s**2 / n)
sonuc = (ort - hyp) / payda
sonuc
```

```
3.10392674421936
```

```
degrees_of_freedom = n - 1
#çift kuyruk testi
t = 2 * (1 - stats.t.cdf(sonuc, df=degrees_of_freedom))
t
if t < alpha:
    print("h0 red")
else:
    print("h0 kabul")
```

```
h0 red
```

Bir şehirdeki kadınlar ve erkeklerin maaşları arasında bir fark olup olmadığını test edin.

```
from scipy import stats

# Kadınların maaşları (örnek veri)
```

```
female_salaries = [5000, 5200, 5300, 5500, 4800, 5100, 5400, 5000,
5600, 5300]

# Erkeklerin maaşları (örnek veri)
male_salaries = [6000, 6300, 6200, 6100, 6400, 5900, 6500, 6600, 6300,
6200]

x1 = np.mean(female_salaries)
x2=np.mean(male_salaries)
s1 = np.std(female_salaries, ddof=1)
s2 = np.std(male_salaries, ddof=1)
n1 = len(female_salaries)
n2= len(male_salaries)
pay = (x1 - x2)
payda = np.sqrt((s1**2 / n1) + (s2**2 / n2))

t_stat= pay / payda
t_stat

dof = n1 + n2 - 2

#iki yönlü kuyruk

p = s * ( 1 - stats.t.cdf(np.abs(t_stat), df=dof))

if p < alpha:
    print("h0 red")
else:
    print("h0 kabul")


h0 red
```

Bir okulda çocukların %40'ının A+ notu aldığına dair iddiaları doğru mu? -> bunu python ile kodlayalım

Bu soruda tek örneklem oran testi (One-Sample Proportion Test) kullanacağız. Çünkü burada tek bir grup (çocuklar) içerisindeki A+ notu alma oranının belirli bir değere (bu durumda %40) karşı test edilmesi isteniyor.

```
import numpy as np
from scipy import stats

alpha = 0.05
# Örnekleme veri (örnek olarak 100 öğrenci verisi kullanalım)
# A+ alan öğrenci sayısı
num_A_plus = 45
# Toplam öğrenci sayısı
total_students = 100
```

```
# Örneklem oranı
p = num_A_plus / total_students

# Popülasyon oranı
p0 = 0.45

# Standart hata (Standard Error)
se = np.sqrt((p0 * (1 - p0)) / total_students)

# Z-skoru hesaplama
z = (p - p0) / se

# Çift kuyruk testi (p-değerini hesaplama)
sonuc = 2 * (1 - stats.norm.cdf(np.abs(z)))

# Sonucu kontrol etme
if sonuc < alpha:
    print("H0 reddedilir: Çocukların A+ notu alma oranı %40'tan
farklıdır.")
else:
    print("H0 kabul edilir: Çocukların A+ notu alma oranı %40'tır.")
```

```
H0 kabul edilir: Çocukların A+ notu alma oranı %40'tır.
```

Öğrencilerin final sınavı puanları ile önceki sınav puanları arasındaki farkın anlamlı olup olmadığını test edin. bağımlı iki örneklem t-testini (paired sample t-test) kullanmamız gerekiyor. Çünkü burada aynı öğrencilerden alınan iki ölçüm (önceki sınav puanları ve final sınavı puanları) arasındaki farkı test edeceğiz.

```
import numpy as np
from scipy import stats
alpha = 0.05
# Öğrencilerin önceki sınav ve final sınavı puanları (örnek veri)
previous_scores = np.array([75, 80, 85, 90, 88, 70, 76, 82, 91, 84])
final_scores = np.array([80, 85, 90, 95, 92, 74, 78, 86, 95, 89])
x1 = np.mean(previous_scores)
x2 = np.mean(final_scores)
s1 = np.std(previous_scores, ddof=1)
s2 = np.std(final_scores, ddof=1)
n1 = len(previous_scores)
n2 = len(final_scores)
payda = np.sqrt((s1**2/n) + (s2**2/n))
pay = x1 - x2
t = pay / payda
dof = n1 + n2 - 2
#çift kuyruk
p = 2 * ( 1 - stats.t.cdf(np.abs(t),df=dof))
if p < alpha:
    print("H0 reddedilir: Çocukların A+ notu alma oranı %40'tan
farklıdır.")
```

```
else:
    print("H0 kabul edilir: Çocukların A+ notu alma oranı %40'tır.")
```

```
H0 kabul edilir: Çocukların A+ notu alma oranı %40'tır.
```

Bir okulda erkek ve kız öğrencilerin matematik sınavı ortalamalarını karşılaştırmak istiyoruz. iki grup ve sürekli değişken olduğu için iki örneklem t testi kulanılır

```
import numpy as np
import pandas as pd
alpha = 0.05
erkek_puanlar = np.array([78, 85, 88, 92, 76, 81, 95, 89, 84, 91])
kiz_puanlar = np.array([82, 79, 85, 88, 90, 83, 87, 80, 86, 78])

x1 = np.mean(erkek_puanlar)
x2 = np.mean(kiz_puanlar)
s1 = np.std(erkek_puanlar, ddof=1)
s2= np.std(kiz_puanlar, ddof=1)
n1 = len(erkek_puanlar)
n2=len(kiz_puanlar)
pay = x1 - x2
payda = np.sqrt( (s1**2)/n1 + (s2**2)/n2)
t = pay/payda
dof = n1 + n2 - 2
#çift kuyruk testi
from scipy import stats
p = 2 * ( 1 - stats.t.cdf(np.abs(t), df=dof))

if p < alpha:
    print("Sonuç: Erkek ve kız öğrencilerin matematik sınavı
ortalamaları arasında istatistiksel olarak anlamlı bir fark var.")
else:
    print("Sonuç: Erkek ve kız öğrencilerin matematik sınavı
ortalamaları arasında istatistiksel olarak anlamlı bir fark yok.")

print(t)
print(p)
```

```
Sonuç: Erkek ve kız öğrencilerin matematik sınavı ortalamaları
arasında istatistiksel olarak anlamlı bir fark yok.
0.8940893525776691
0.3830734770927189
```

```
import numpy as np
from scipy.stats import ttest_ind

# Örnek veri: Erkek ve kız öğrencilerin matematik sınavı puanları
erkek_puanlar = np.array([78, 85, 88, 92, 76, 81, 95, 89, 84, 91])
kiz_puanlar = np.array([82, 79, 85, 88, 90, 83, 87, 80, 86, 78])
```

```
# Bağımsız t-testi
t_stat, p_value = ttest_ind(erkek_puanlar, kiz_puanlar)

# Sonuçları yazdır
print("t-Statistik Değeri:", t_stat)
print("p-Değeri:", p_value)

# Karar
alpha = 0.05  # İstatistiksel anlamlılık seviyesi
if p_value < alpha:
    print("Sonuç: Erkek ve kız öğrencilerin matematik sınavı
ortalamaları arasında istatistiksel olarak anlamlı bir fark var.")
else:
    print("Sonuç: Erkek ve kız öğrencilerin matematik sınavı
ortalamaları arasında istatistiksel olarak anlamlı bir fark yok.")
```

```
t-Statistik Değeri: 0.8940893525776693
p-Değeri: 0.38307347709271866
Sonuç: Erkek ve kız öğrencilerin matematik sınavı ortalamaları
arasında istatistiksel olarak anlamlı bir fark yok.
```

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

#0ne sample proportion

```
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]
```

```
hyp = 0.35
alpha = 0.05
pz = len(df_stck[df_stck['age_first_code_cut'] == 'child']) /
len(df_stck)
n = len(df_stck)
se = np.sqrt( hyp * ( 1 - hyp ) / n)
pay = pz - hyp
z = pay / se
#sağ kuyruk testi
from scipy import stats
sonuc = 1 - stats.norm.cdf(z)
if sonuc < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 reddedilir.
```

'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.

#iki örneklem t testi kullanmalıyız

```
alpha= 0.05
x = df_stck.groupby('age_first_code_cut')['converted_comp'].mean()
x1 = x.iloc[0]
x2 = x.iloc[1]
s = df_stck.groupby('age_first_code_cut')['converted_comp'].std()
```

```
s1 = s.iloc[0]
s2 = s.iloc[1]
n = df_stck.groupby('age_first_code_cut')['converted_comp'].count()
n1 = n.iloc[0]
n2=n.iloc[1]
pay = x1 - x2
payda = np.sqrt((s1**2/n1) + (s2**2/n2))
t_d= pay / payda
dof = n1 + n2 - 2
#sağ kuyruk testi
from scipy import stats
p = 1 - stats.t.cdf(np.abs(t_d), df=dof)
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
p
```

```
H0 reddedilir.
```

```
0.030811302165157595
```

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 küçüktür.

#tek örneklem t testi kullanılır çünkü tek veri -> diff sütunu)

```
import pandas as pd
import numpy as np
df_election = pd.read_feather('data/repub_votes_potus_08_12.feather')
df_y = df_election.copy()
df_y['diff'] = df_y['repub_percent_08'] - df_y['repub_percent_12']
x = df_y['diff'].mean()
m = 0
s = np.std(df_y['diff'], ddof=1)
n = len(df_y)
payda = np.sqrt(s**2/n)
pay = x - m
t_d = pay /payda
dof = n - 1
#sol kuyruk
from scipy import stats
p = stats.t.cdf(np.abs(t_d), df=dof)
p = 1 - stats.t.cdf(np.abs(t_d), df=dof)
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
p
```

```
H0 reddedilir.

9.572537285063021e-08

#H₀: 'job_sat' kategorisindeki grupların 'converted_comp' ortalamaları
eşittir.
#birden fazla grup olduğu için anova

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
```

#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

```
p_hat = (df_stck["age_cat"] == "Under 30").mean()
p_hat
p0 = 0.5
n=len(df_stck)
se = np.sqrt( p0 * (1-p0) / n)
pay = p_hat - p0
sonuc = pay / se
sonuc
#çift kuyruk. çünkü küçüktür ya da byüktür diye belirtilmemiş
from scipy import stats
p = 2 * ( 1 - stats.norm.cdf(np.abs(sonuc)))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
sonuc

H0 reddedilir.

3.385911440783663
```

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

```
p_hat = df_stck.groupby('age_cat')
['hobbyist'].value_counts(normalize=True)
otuz_buyuk = p_hat[('At least 30', 'Yes')]
otuz_kucuk = p_hat[('Under 30', 'Yes')]
n = df_stck.groupby('age_cat')['hobbyist'].count()
n_30buyuk = n['At least 30']
n_30kucuk = n['Under 30']
se = np.sqrt( (otuz_buyuk * ( 1 - otuz_buyuk) / n_30buyuk) +
(otuz_kucuk * ( 1 - otuz_kucuk) / n_30kucuk))
z = (otuz_buyuk - otuz_kucuk) / se
z
#çift kuyruk testi olmalı:
from scipy import stats
p = 2 * (1 - stats.norm.cdf(np.abs(z)))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
p
```

```
H0 reddedilir.
```

```
2.6876347940296696e-05
```

Bir telekomünikasyon şirketinin müşteri verilerine dayalı olarak, total_day_charge ve total_eve_charge değişkenlerini kullanarak, churn (hizmet iptali) durumunu tahmin etmek için bir K-Nearest Neighbors (KNN) modeli geliştirin. Modeli eğitin ve verilen yeni müşteri verileri (X_new) için hizmet iptali durumunu tahmin edin.

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
X_new = np.array([
    [56.8, 17.5],
    [24.4, 24.1],
    [50.1, 10.9]
])
df_churn = pd.read_csv("data/telecom_churn_clean.csv")
X = df_churn[['total_day_charge', 'total_eve_charge']] #bağımsız
değişken
y = df_churn['churn'] #bağımlı değişken
model_knn = KNeighborsClassifier(n_neighbors=15)
model_knn.fit(X,y) #model eğitimi
predictions = model_knn.predict(X_new)
predictions
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:493:
UserWarning: X does not have valid feature names, but
KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

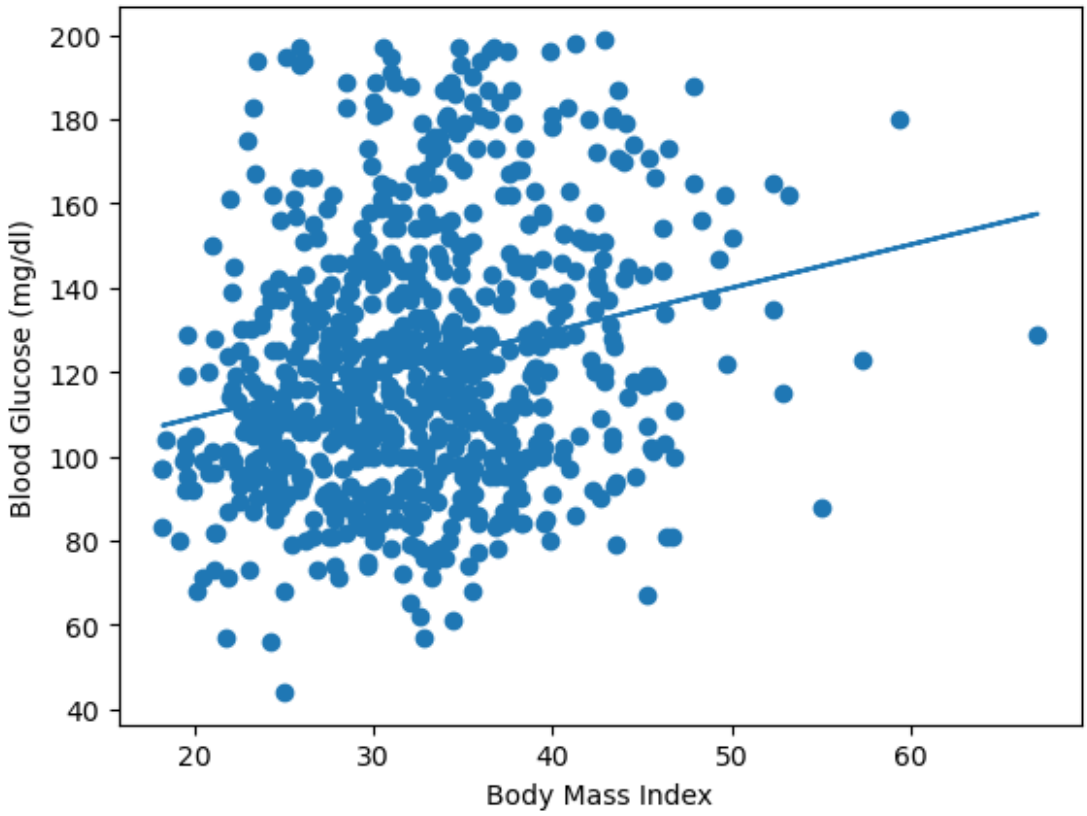
```
array([1, 0, 0])
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# 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 Modeli
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)

# Tahmin
predictions = knn.predict(X_test)
print("Tahminler:", predictions)

# Model Doğruluğu
accuracy = knn.score(X_test, y_test)
print("Doğruluk Skoru:", accuracy)
```

```
Tahminler: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0
 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1
 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
```

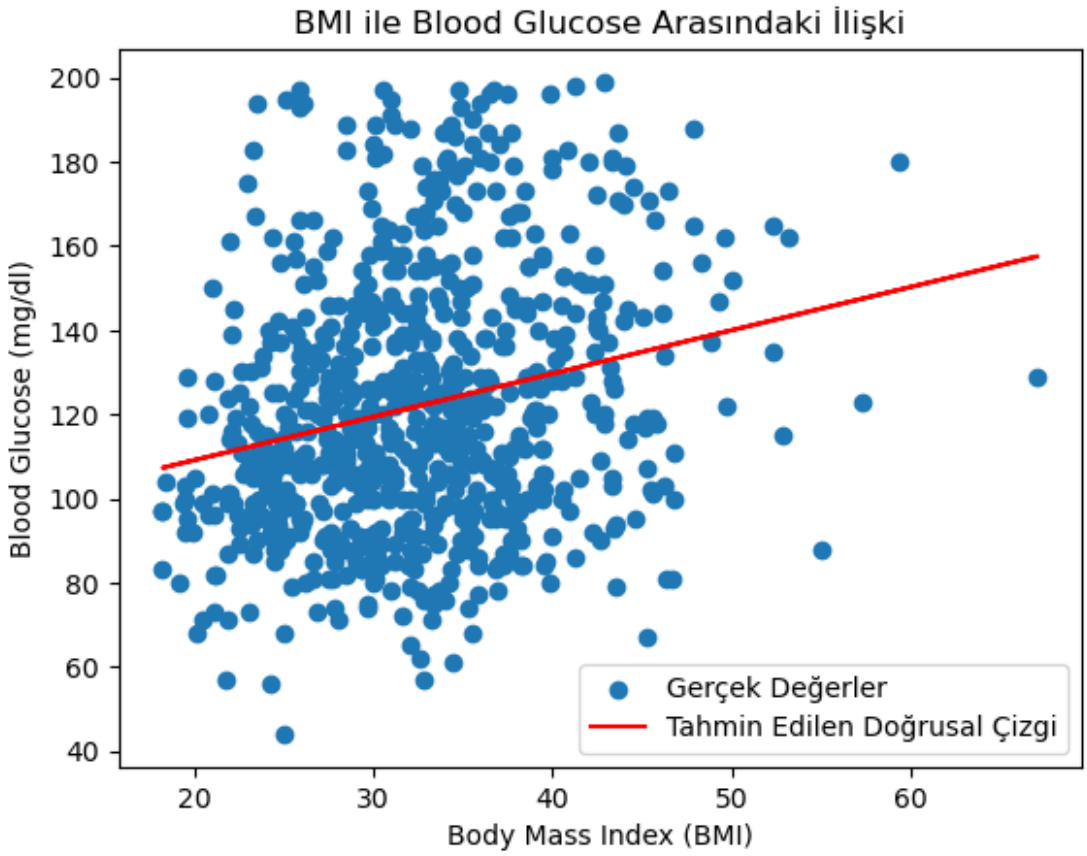
```
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1
 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1
 0]
Doğruluk Skoru: 0.8605697151424287
```

```
import pandas as pd
import matplotlib.pyplot as plt
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).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. 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[:, 4]
X_bmi = X_bmi.reshape(-1, 1) #1d veriyi 2d hale getirmek için


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')
```




```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

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

# Eksik veya hatalı veri temizleme
df_filtered = df_diabetes[(df_diabetes['bmi'] == 0.0) |
(df_diabetes['glucose'] == 0.0)]
df_diabetes.drop(df_filtered.index, inplace=True)

# Bağımsız ve bağımlı değişkenlerin seçimi
X = df_diabetes['bmi'].values.reshape(-1, 1)  # BMI bağımsız değişken
olarak seçildi (2D hale getirildi)
y = df_diabetes['glucose'].values  # Kan şekeri bağımlı değişken

# Model eğitimi
model_reg = LinearRegression()
model_reg.fit(X, y)

# Tahmin
predictions = model_reg.predict(X)

# Görselleştirme
```

```
plt.scatter(X, y, label="Gerçek Değerler")
plt.plot(X, predictions, color='red', label="Tahmin Edilen Doğrusal
Çizgi")
plt.xlabel('Body Mass Index (BMI)')
plt.ylabel('Blood Glucose (mg/dl)')
plt.title('BMI ile Blood Glucose Arasındaki İlişki')
plt.legend()
plt.show()
```



#bu kod, **doğrusal regresyon (Linear Regression)** kullanarak **diabetes veri kümesi** üzerinde bir model oluşturuyor ve bu modelin **kan şekeri seviyelerini** tahmin etmesini sağlıyor. sonrasında ne kadar başarılı olduğunu ölç.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

ddf_filtered = df_diabetes[(df_diabetes['bmi'] == 0.0) |
(df_diabetes['glucose'] == 0)]
```

```
df_diabetes.drop(df_filtered.index, inplace=True)
X = df_diabets.drop('glucose', axis=1).values #bağımsız değişkenler
y=df_diabetes['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
predictions = model_reg.predict(X_test)
model_reg.score(X_test, y_test)
```

```
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, predictions) #modelin tahmin ettiği
değerlerle gerçek değerler arasındaki farkların karelerinin
ortalamasıdır. Daha düşük MSE, modelin daha iyi olduğunu gösterir.
```

```
25.695203763480208
```

```
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).values
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)
cv_results
```

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

```
# 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,
       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
```

```
array([[0.82892316, 0.17107684],
       [0.92986818, 0.07013182],
       [0.98204431, 0.01795569],
       ...,
       [0.91093968, 0.08906032],
```

```
       [0.92723286, 0.07276714],
       [0.9081191 , 0.0918809 ]])
```

```
import pandas as pd
import numpy as np
df = pd.read_csv("data/Generated_Employee_Data.csv")
df
```

```
    EmployeeID Department  MonthlySalary  HoursWorked  RemoteWork  \
0            1      Sales        5087.05        40.07           0
1            2  Marketing        4700.99        47.27           0
2            3         HR        5091.76        38.68           0
3            4         IT        3012.43        53.60           0
4            5      Sales        4780.33        43.13           1
..         ...        ...            ...          ...         ...
95          96         IT        4184.19        32.82           0
96          97         IT        4922.90        45.82           1
97          98         IT        5341.15        40.05           0
98          99      Sales        5276.69        35.09           0
99         100      Sales        5827.18        42.31           1

    SatisfactionScore
0                   3
1                   5
2                   5
3                   2
4                   4
..                ...
95                  2
96                  1
97                  3
98                  1
99                  5

[100 rows x 6 columns]
```

Bir şirkette çalışanların ortalama maaşı 5000 TL olduğuna inanılıyor. 30 çalışanın maaşları incelenerek, şirketin maaş ortalamasının 5000 TL'den farklı olup olmadığı test edilsin.

Hipotezler:

H0: Çalışanların maaş ortalaması 5000 TL. Ha: Çalışanların maaş ortalaması 5000 TL'den farklıdır.

#one t testi

```
sample_data= df['MonthlySalary'].sample(30, random_state=42)
alpha = 0.05
m = 5000
x = np.mean(sample_data)
```

```
s = np.std(sample_data, ddof=1)
n = len(sample_data)
payda = np.sqrt(s**2 / n)
t = (x-m) / payda
dof = n - 1
#çift kuyruk
from scipy import stats
p = 2 * ( 1 - stats.t.cdf(np.abs(t), df=dof))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
sonuc
```

```
H0 kabul edilir.
```

```
3.385911440783663
```

Soru: Şirketin çalışanlarının %50'sinin uzaktan çalıştığı iddia edilmektedir. Çalışanların %50'sinin uzaktan çalışıp çalışmadığını test edelim.

Hipotezler:

H0: Uzaktan çalışanların oranı %50'dir. Ha: Uzaktan çalışanların oranı %50 değildir.

#one sample pro

```
p = (df['RemoteWork'] == 1).mean()
alpha = 0.05
p0 = 0.5
n=len(df)
se = np.sqrt( p0 * ( 1 - p0 ) / n)
z = (p - p0) / se
#çift kuyruk testi
from scipy import stats
p = 2 * ( 1 - stats.norm.cdf(np.abs(z)))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
p
```

```
H0 reddedilir.
```

```
0.01639507184919231
```

Soru: IT ve HR departmanlarının maaş ortalamaları arasında fark olup olmadığını test edelim.

Hipotezler:

H0: IT ve HR departmanlarının maaş ortalamaları eşittir. Ha: IT ve HR departmanlarının maaş ortalamaları farklıdır

#two sample t test

```
x = df.groupby('Department')['MonthlySalary'].mean()
x1 = x.iloc[1]
x2 = x.iloc[0]
s = df.groupby('Department')['MonthlySalary'].std()
s1 = s.iloc[1]
s2=s.iloc[2]
n = df.groupby('Department')['MonthlySalary'].count()
n1 = n.iloc[0]
n2 = n.iloc[1]
pay = x1 - x2
payda = np.sqrt((s1**2/n1) + (s2**2/n2))
t = pay / payda
dof = n1 + n2 - 2
from scipy import stats
p = 2 * (1 - stats.t.cdf(np.abs(t), df=dof))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
p
```

```
H0 kabul edilir.
```

```
0.1270802192019016
```

```
# 3. İki bağımsız grup - Ortalama (sürekli)
# Soru: IT ve HR departmanlarının maaş ortalamaları arasında fark olup
olmadığını test edelim.
# H0: IT ve HR departmanlarının maaş ortalamaları eşittir.
# Ha: IT ve HR departmanlarının maaş ortalamaları farklıdır.

# IT ve HR maaşlarını ayır
it_salaries = employee_data.loc[employee_data['Department'] == 'IT',
'MonthlySalary']
hr_salaries = employee_data.loc[employee_data['Department'] == 'HR',
'MonthlySalary']

# Gerekli parametreleri hesapla
mean_it = it_salaries.mean()
mean_hr = hr_salaries.mean()
std_it = np.std(it_salaries, ddof=1)
std_hr = np.std(hr_salaries, ddof=1)
n_it = len(it_salaries)
n_hr = len(hr_salaries)

# Standart hata ve T-istatistiği
```

```
se = np.sqrt((std_it**2 / n_it) + (std_hr**2 / n_hr))
t_stat = (mean_it - mean_hr) / se

# Serbestlik derecesi
dof = ((std_it**2 / n_it) + (std_hr**2 / n_hr))**2 / (
    ((std_it**2 / n_it)**2 / (n_it - 1)) + ((std_hr**2 / n_hr)**2 /
(n_hr - 1))
)

# P-Değeri (çift kuyruklu test)
from scipy import stats
p_value = 2 * (1 - stats.t.cdf(np.abs(t_stat), df=dof))

# Karar
alpha = 0.05
if p_value < alpha:
    result = "H0 reddedilir. IT ve HR departmanlarının maaş
ortalamaları farklıdır."
else:
    result = "H0 reddedilemez. IT ve HR departmanlarının maaş
ortalamaları eşit olabilir."

t_stat, p_value, result
```

Bir araştırmada, uzaktan çalışan ve ofiste çalışan çalışanların iş memnuniyet puanlarının farklı olup olmadığı test edilsin.

Hipotezler:

H0: Uzaktan çalışan ve ofiste çalışan çalışanların iş memnuniyet medyanları eşittir. Ha: Uzaktan çalışan ve ofiste çalışan çalışanların iş memnuniyet medyanları farklıdır.

#iki bapımsız ve medyan olduğu için mann whitney

```
import pandas as pd
df = pd.read_csv("data/Generated_Employee_Data.csv")
uzak_vs_yakin = df[['RemoteWork', 'SatisfactionScore']]
uzak_vs_yakin_wide = uzak_vs_yakin.pivot(columns = 'RemoteWork',
values='SatisfactionScore')
import pingouin
pingouin.mwu(x = uzak_vs_yakin_wide[0], y=uzak_vs_yakin_wide[1],
alternative='two-sided')
```

```
      U-val alternative     p-val       RBC     CLES
MWU  1047.5   two-sided  0.344867 -0.110781  0.44461
```

```
Soru: IT ve Sales departmanlarındaki çalışanların memnuniyet
skorlarının medyanları arasında fark olup olmadığını test edelim.

Hipotezler:
```

H0: IT ve Sales departmanlarının memnuniyet skorlarının medyanları eşittir.
Ha: IT ve Sales departmanlarının memnuniyet skorlarının medyanları farklıdır.

```
import pandas as pd
it_vs_sales = df[['Department','SatisfactionScore']]
it_vs_sales_wide = it_vs_sales.pivot(columns='Department',
values='SatisfactionScore')
import pingouin
pingouin.mwu(x = it_vs_sales_wide['IT'], y= it_vs_sales_wide['Sales'],
alternative='two-sided')
```

```
     U-val alternative     p-val       RBC      CLES
MWU  560.5   two-sided  0.919236  0.015399  0.507699
```

Soru: Çalışanların ortalama maaşının 5000 TL olup olmadığını test edin.

Hipotez:
Null hipotezi (H0): Çalışanların maaşlarının ortalaması 5000 TL'dir.
Alternatif hipotez (H1): Çalışanların maaşlarının ortalaması 5000 TL değildir.

```
import pandas as pd
import numpy as np
df = pd.read_csv("data/Generated_Employee_Data.csv")
m = 5000
alpha = 0.05
x = df['MonthlySalary'].mean()
s = np.std(df['MonthlySalary'], ddof=1)
n = len(df)
payda = np.sqrt( s**2 / n)
t = (x - m) / payda
dof = n - 1
#çift kuyruk
p = 2 * ( 1 - stats.t.cdf(np.abs(t), df=dof))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
t
```

```
H0 kabul edilir.
```

```
-0.011837728780229389
```

```
from scipy import stats

# Maaşların ortalamasının 5000 TL olup olmadığını test et
```

```
t_stat, p_value = stats.ttest_1samp(df['MonthlySalary'], 5000)
t_stat, p_value
```

```
(-0.011837728780229392, 0.9905789020985529)
```

Soru: Çalışanların %60'ının uzaktan çalışıp çalışmadığını test edin.

Hipotez: Null hipotezi (H0): Çalışanların %60'ı uzaktan çalışmaktadır. Alternatif hipotez (H1): Çalışanların %60'ı uzaktan çalışmamaktadır.

```
import pandas as pd
from scipy import stats
import numpy as np

df = pd.read_csv("data/Generated_Employee_Data.csv")

# Beklenen oran
p0 = 0.6

# Gözlemlenen oran (uzaktan çalışanların oranı)
p = (df['RemoteWork'] == 1).mean()

# Veri sayısı
n = len(df['RemoteWork'])

# Standart hata
payda = np.sqrt(p0 * (1 - p0) / n)

# Z istatistiği
z = (p - p0) / payda

# Çift kuyruk testi için p-değeri
p_d = 2 * (1 - stats.norm.cdf(np.abs(z)))

# Anlamlılık seviyesi
alpha = 0.05

if p_d < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 reddedilir.
```

```
import numpy as np
from scipy import stats

# Maaş ve memnuniyet puanlarını iki gruba ayırın
high_salary = df[df['MonthlySalary'] > 5000]['SatisfactionScore']
low_salary = df[df['MonthlySalary'] <= 5000]['SatisfactionScore']
```

```
u_stat, p_value = stats.mannwhitneyu(____, ____)  # Buraya iki grup parametresini yazın

alpha = 0.05

if p_value < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 kabul edilir.
```

```
sample_data = df.sample(n=15, random_state=42)
av = sample_data['MonthlySalary'].mean()
av
```

```
5300.754000000001
```

```
sample = df.groupby('Department').sample(frac=0.1, random_state=42)
mem = sample['SatisfactionScore'].mean()
mem
```

```
2.7
```

```
# 1. Adım: Departmana göre gruplama ve her gruptan 5 örnek seçme (örneğin)
clusters = df.groupby('Department').sample(n=5, random_state=42)

# 2. Adım: Seçilen kümelerden 10 çalışan seçme (bu artık mümkün olacak çünkü her departmandan 5 çalışan var)
clusters_rast = clusters.sample(n=10, random_state=42)

# 3. Adım: Saatlik çalışma ortalamasını hesaplayın
saat = clusters_rast['HoursWorked'].mean()
print(saat)
```

```
39.653000000000006
```

```
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()
        )
    )
```

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

```
import numpy as np
from scipy import stats

# Gözlemlenen oranı hesapla
p = (df_stck['age_first_code_cut'] == 'child').mean()

# Beklenen oran
hyp = 0.35

# Anlamlılık seviyesi
alpha = 0.05

# Veri sayısı
n = len(df_stck)

# Standart hata (se) hesaplama
se = np.sqrt(hyp * (1 - hyp) / n)

# Z istatistiği hesaplama
z = (p - hyp) / se

# Sağ kuyruk için p-değeri
p_value = 1 - stats.norm.cdf(np.abs(z))

# Hipotez testi sonucu
if p_value < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 reddedilir.
```

'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.

```
alpha= 0.05
x = df_stck.groupby('age_first_code_cut')['converted_comp'].mean()
x1 = x.iloc[0]
x2 = x.iloc[1]
s = df_stck.groupby('age_first_code_cut')['converted_comp'].std()
s1 = s.iloc[0]
s2 = s.iloc[1]
n = df_stck.groupby('age_first_code_cut')['converted_comp'].count()
n1 = n.iloc[0]
n2 = n.iloc[1]
pay = x1 - x2
se = np.sqrt((s1**2 / n1) + (s2**2 / n2))
t = pay / se
dof = n1 + n2 - 2
```

```
#sağ kuyruk testi
p = 1 - (stats.t.cdf(np.abs(t), df=dof))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 reddedilir.
```

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 küçüktür.

```
sample_d = df.copy()
alpha = 0.05
sample_d['diff'] = df['repub_percent_08'] - df['repub_percent_12']
m = 0
x = sample_d['diff'].mean()
s = np.std(sample_d['diff'], ddof=1)
n = len(df_stck)
payda = np.sqrt(s**2 / n)
pay = x- m
t = pay / payda
dof = n-1
#sol
stats.t.cdf(np.abs(t), df=dof)
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call
last)
File
/opt/anaconda3/lib/python3.12/site-packages/pandas/core/indexes/base.p
y:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7081, in
pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7089, in
```

```
pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'repub_percent_08'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call
last)
Cell In[372], line 3
      1 sample_d = df.copy()
      2 alpha = 0.05
----> 3 sample_d['diff'] = df['repub_percent_08'] -
df['repub_percent_12']
      4 m = 0
      5 x = sample_d['diff'].mean()

File
/opt/anaconda3/lib/python3.12/site-packages/pandas/core/frame.py:4102,
in DataFrame.__getitem__(self, key)
   4100 if self.columns.nlevels > 1:
   4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
   4103 if is_integer(indexer):
   4104     indexer = [indexer]

File
/opt/anaconda3/lib/python3.12/site-packages/pandas/core/indexes/base.p
y:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will
raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-
raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'repub_percent_08'
```

#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

```
p = (df_stck['age_cat'] == 'Under 30').mean()
n = len(df_stck)

p0 = 0.5
se = np.sqrt(p0 * ( 1 - p0 ) / n)
z = (p - p0) / se
#çift kuyruk testi
p = 1 - (stats.norm.cdf(np.abs(z)))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 reddedilir.
```

Soru: Çalışanların 'RemoteWork' sütunundaki 1 (uzaktan çalışan) ve 0 (uzaktan çalışmayan) kategorilerindeki oran farkını test edin. H0 (Null Hipotezi): Uzaktan çalışanların oranı, uzaktan çalışmayanların oranından farklı değildir.

```
alpha = 0.05
p_hat = df['RemoteWork'].value_counts(normalize=True)
p0 = p_hat.iloc[0]
p1 = p_hat.iloc[1]
n0 = len(df[df['RemoteWork'] == 0])
n1 = len(df[df['RemoteWork'] == 1])
pay = p0 - p1
se = np.sqrt( (p0*( 1-p0) / n0) + (p1 *(1-p1) / n1))
z = pay / se
# H0 (Null Hipotezi): Uzaktan çalışanların oranı, uzaktan
çalışmayanların oranından farklı değildir. -> çift kuyruk testi
yapılmalı
p = 2 * ( 1 - stats.norm.cdf(np.abs(z)))
if p < alpha:
    print("H0 reddedilir.")
else:
    print("H0 kabul edilir.")
```

```
H0 reddedilir.
```

```
import numpy as np
from scipy import stats

alpha = 0.05

# Oranları hesapla
p_hat = df['RemoteWork'].value_counts(normalize=True)
p0 = p_hat.iloc[0]  # Uzaktan çalışmayanların oranı (0)
p1 = p_hat.iloc[1]  # Uzaktan çalışanların oranı (1)

# Grup büyüklüklerini hesapla
```

```
n0 = len(df[df['RemoteWork'] == 0])  # Uzaktan çalışmayanların sayısı
n1 = len(df[df['RemoteWork'] == 1])  # Uzaktan çalışanların sayısı

# Oranlar arasındaki fark
pay = p0 - p1

# Standart hata (se)
se = np.sqrt((p0 * (1 - p0) / n0) + (p1 * (1 - p1) / n1))

# Z istatistiği
z = pay / se

# Çift kuyruklu p-değeri
p_value = 2 * (1 - stats.norm.cdf(np.abs(z)))

# Sonuçları değerlendir
if p_value < alpha:
    print("H0 reddedilir: Uzaktan çalışanların oranı, uzaktan
çalışmayanlardan farklıdır.")
else:
    print("H0 kabul edilir: Uzaktan çalışanların oranı, uzaktan
çalışmayanlardan farklı değildir.")
```

```
H0 reddedilir: Uzaktan çalışanların oranı, uzaktan çalışmayanlardan
farklıdır.
```

"Veri setimdeki müşteri bilgilerini kullanarak, bir müşterinin hizmet iptali (churn) yapıp yapmadığını tahmin etmek istiyorum. Bu problem için hangi modelin daha uygun olduğunu nasıl belirlerim ve hangi adımları izlemeliyim?"

```
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
y = df_churn['churn'].values

model_knn = KNeighborsClassifier(n_neighbors=15)
model_knn.fit(X,y) #model eğitimi
X_new = np.array([
    [56.8, 17.5],
    [24.4, 24.1],
    [50.1, 10.9]
])
model_knn.predict(X_new)
```

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

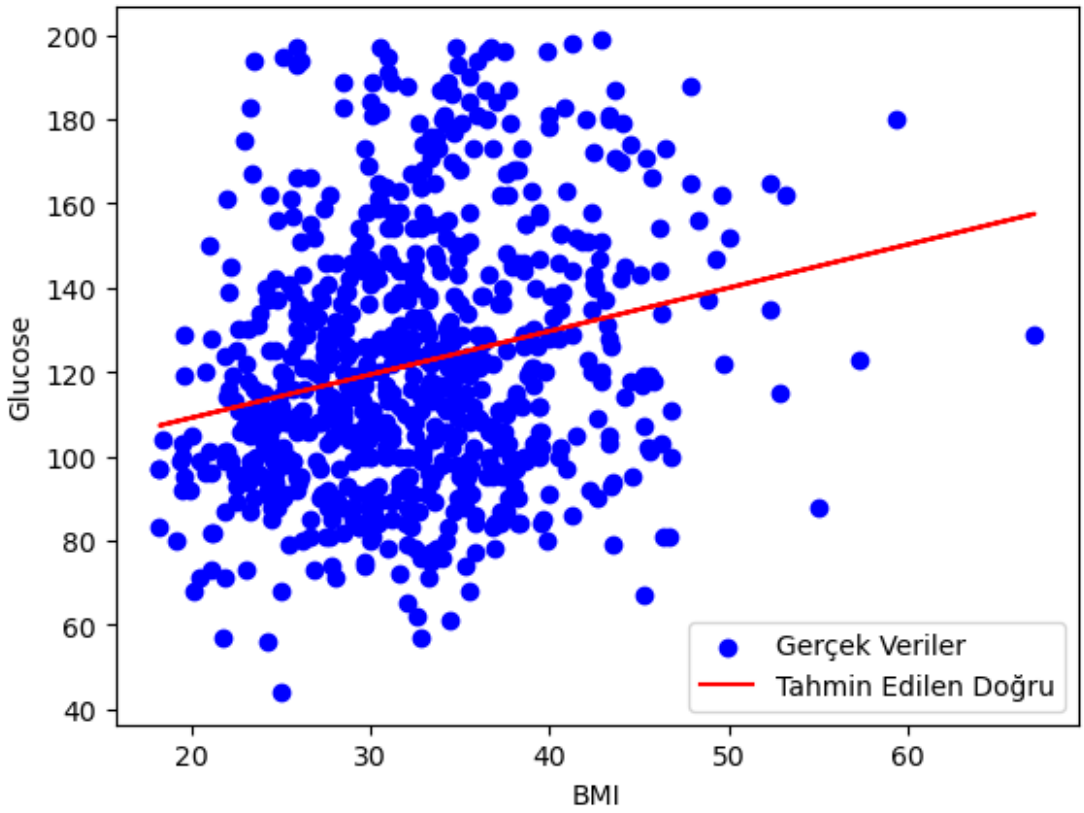
```
test_size=0.2, random_state=42, stratify=y)
knn = KNeighborsClassifier(n_neighbors = 6)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
0.8605697151424287
```

Aşağıdaki Python kodu ile, vücut kitle indeksi (BMI) ile kan şekeri düzeyi (glucose) arasındaki ilişkiyi incelemeye çalışan bir model oluşturulmaktadır. Bu modelin amacı, BMI'yi bağımsız değişken olarak alıp, glucose'u tahmin etmektir.

```
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
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
```



kullanarak **diabetes veri kümesi** üzerinde bir model oluşturuyor ve bu modelin **kan şekeri seviyelerini** tahmin etmesini sağlıyor

```
X = df_diabetes.drop('glucose', axis=1).values
y = df_diabetes['glucose'].values
X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.2, random_state=42)
model_l = LinearRegression()
model_l.fit(X_train, y_train)
pred = model_l.predict(X_test)
model_l.score(X_test, y_test)
```

```
0.3282802627263198
```

```
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).values
y = df_diabets['glucose'].values

kfold = KFold(n_splits=6, shuffle=True, random_state=42)
model_l = LinearRegression()
res = cross_val_score(model_l, X,y,cv=kfold)
res
```

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

Department ve RemoteWork arasındaki bağımsızlık ilişkisini test edin. Bu test, farklı departmanlarda uzaktan çalışma oranlarının birbirinden bağımsız olup olmadığını belirleyecektir.

```
import pandas as pd
import numpy as np
df = pd.read_csv("data/Generated_Employee_Data.csv")
expected , observed, stats = pingouin.chi2_independence(data=df,
x='Department', y='RemoteWork')
stats
```

```
/opt/anaconda3/lib/python3.12/site-packages/pingouin/
contingency.py:151: UserWarning: Low count on observed frequencies.
  warnings.warn(f"Low count on {name} frequencies.")
/opt/anaconda3/lib/python3.12/site-packages/pingouin/contingency.py:15
1: UserWarning: Low count on expected frequencies.
  warnings.warn(f"Low count on {name} frequencies.")
```

	test	lambda	chi2	dof	pval	cramer	power
0	pearson	1.000000	4.447889	3.0	0.216986	0.210900	0.395656
1	cressie-read	0.666667	4.572942	3.0	0.205876	0.213844	0.405915
2	log-likelihood	0.000000	4.951015	3.0	0.175419	0.222509	0.436579
3	freeman-tukey	-0.500000	5.387218	3.0	0.145542	0.232104	0.471176
4	mod-log-likelihood	-1.000000	6.012544	3.0	0.111002	0.245205	0.519016
5	neyman	-2.000000	8.201329	3.0	0.042029	0.286380	0.666212

RemoteWork ve SatisfactionScore kategorik değişkenlerinin beklenen ve gözlenen frekanslarının uyumlu olup olmadığını test edin.

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

X= df_churn[['total_day_charge', 'total_eve_charge']].values
y = df_churn['churn'].values

model_knn = KNeighborsClassifier(n_neighbors=15)
model_knn.fit(X,y)
X_new = np.array([
    [56.8, 17.5],
    [24.4, 24.1],
    [50.1, 10.9]
])
model_knn.predict(X_new)
```

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

Verilen veri setini kullanarak bir KNN modeli oluşturun ve çalışanların uzaktan çalışma durumunu tahmin edin

```
X = df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].values
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)
model_knn = KNeighborsClassifier(n_neighbors=5)
model_knn.fit(X_train, y_train)
y_pred = model_knn.predict(X_test)
print("Tahminler:", y_pred)
model_knn.score(X_test, y_test)
```

```
Tahminler: [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
```

```
0.7
```

Bir çalışanın çalışma saatleri (HoursWorked) ile maaşı (MonthlySalary) arasındaki doğrusal ilişkiyi incelemek için bir model oluşturun

```
from sklearn.linear_model import LinearRegression
X = df['HoursWorked'].values.reshape(-1,1)
y = df['MonthlySalary'].values
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)
model_reg = LinearRegression()
model_reg.fit(X_train,y_train)
model_reg.predict(X_test)
model_reg.score(X_test, y_test)
```

```
-0.006181633738065129
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Bağımsız ve bağımlı değişkenleri ayır
X = df['HoursWorked'].values.reshape(-1, 1)
y = df['MonthlySalary'].values

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

# Modeli oluştur ve eğit
model_reg = LinearRegression()
model_reg.fit(X_train, y_train)

# Test seti ile tahmin yap
y_pred = model_reg.predict(X_test)

# Modelin doğruluğunu (R^2) hesapla
score = model_reg.score(X_test, y_test)

# Sonuçları yazdır
print("Tahminler:", y_pred)
print("Modelin doğruluk skoru (R^2):", score)
```

```
Tahminler: [4828.53590459 4841.70252361 5003.34378265 5198.60194133
4612.8274653
 4830.77703123 4928.82632181 4759.62126035 4870.55702912 4934.98942008
 4912.57815366 4864.67407169 4880.36195818 4397.39916685 4708.35548841
 4849.26632602 4639.720985   4792.39773748 5055.44997708
4864.95421252]
Modelin doğruluk skoru (R^2): -0.351820397734369
```

Verilen veri setinde, çalışanların uzaktan çalışma durumunu tahmin etmek için bir KNN modeli oluşturun ve performansını cross-validation (çapraz doğrulama) yöntemiyle değerlendirin.

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
X= df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].values
kf = KFold(n_splits = 5, shuffle=True, random_state=42)
model_reg=KNeighborsClassifier(n_neighbors=5)
cv_results = cross_val_score(model_reg, X, y, cv = kf)
cv_results.mean()
```

```
0.53
```

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

# Bağımsız ve bağımlı değişkenlerin seçimi
X = df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].values

# KNN Modeli
model_knn = KNeighborsClassifier(n_neighbors=5)

# Cross-validation
cv_scores = cross_val_score(model_knn, X, y, cv=5)  # 5-fold cross-
validation

# Her bir fold için doğruluk skorlarını yazdır
print("Fold doğruluk skorları:", cv_scores)

# Ortalama doğruluk
mean_score = cv_scores.mean()
print(f"Ortalama Doğruluk: {mean_score:.2f}")
```

```
Fold doğruluk skorları: [0.45 0.6  0.45 0.45 0.45]
Ortalama Doğruluk: 0.48
```

Verilen veri setinde çalışanların uzaktan çalışma durumu (RemoteWork) yerine, çalışanların çalışma saatlerini (HoursWorked) tahmin etmek için bir Ridge regresyon modeli oluşturun ve model performansını değerlendirin.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import root_mean_squared_error
X = df[['MonthlySalary', 'RemoteWork', 'SatisfactionScore']].values
y = df['HoursWorked'].values
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =
0.2, random_state=42)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
y_pred = ridge.predict(X_test)
r2 = ridge.score(X_test, y_test)
mse = root_mean_squared_error(y_test,y_pred)
mse
```

```
7.454046068152741
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
import numpy as np

# Bağımsız ve bağımlı değişkenler
```

```
X = df[['MonthlySalary', 'RemoteWork', 'SatisfactionScore']].values
y = df['HoursWorked'].values

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

# Ridge Regresyon Modeli
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)

# Tahmin
y_pred = ridge.predict(X_test)

# Performans Değerlendirme
r2 = ridge.score(X_test, y_test)  # R² Skoru
mse = mean_squared_error(y_test, y_pred)  # Ortalama Kare Hatası (MSE)
rmse = np.sqrt(mse)  # Kök Ortalama Kare Hatası (RMSE)

print("R² Skoru:", r2)
print("Ortalama Kare Hatası (MSE):", mse)
print("Kök Ortalama Kare Hatası (RMSE):", rmse)
```

```
R² Skoru: -0.6101848964304173
Ortalama Kare Hatası (MSE): 55.56280278614334
Kök Ortalama Kare Hatası (RMSE): 7.454046068152741
```

Verilen veri setini kullanarak, çalışanların uzaktan çalışıp çalışmadığını (RemoteWork) tahmin etmek için bir sınıflandırma modeli oluşturun ve modelin performansını confusion matrix ile analiz edin.

```
from sklearn.metrics import classification_report, confusion_matrix
X = df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].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)
y_pred = knn.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
array([[ 7,  0],
       [13,  0]])
```

Logistic Regression Sorusu Bir sınıflandırma problemi için, çalışanların uzaktan çalışıp çalışmadığını (RemoteWork) tahmin etmek amacıyla bir Logistic Regression modeli oluşturun ve modelin performansını değerlendirin.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Bağımsız ve bağımlı değişkenler
X = df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].values

# Veri setini eğitim ve test olarak ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Logistic Regression Modeli
model_log = LogisticRegression()
model_log.fit(X_train, y_train)

# Tahminler
y_pred = model_log.predict(X_test)

# Performans Metrikleri
accuracy = accuracy_score(y_test, y_pred)  # Doğruluk
cm = confusion_matrix(y_test, y_pred)  # Confusion Matrix
classification_rep = classification_report(y_test, y_pred)  #
Precision, Recall, F1-Score

# Sonuçları Yazdırma
print("Doğruluk (Accuracy):", accuracy)
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_rep)
```

```
Doğruluk (Accuracy): 0.35

Confusion Matrix:
 [[ 7  0]
 [13  0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.35      1.00      0.52         7
           1       0.00      0.00      0.00        13

    accuracy                           0.35        20
   macro avg       0.17      0.50      0.26        20
weighted avg       0.12      0.35      0.18        20


/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classific
ation.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classific
ation.py:1531: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

Bir sınıflandırma problemi için, verilen veri setinde KNN (K-Nearest Neighbors) modelini optimize edin ve en iyi hiperparametreleri belirlemek için Grid Search Cross Validation (GS-CV) yöntemini kullanın.

Soru: Hedef Model: KNN. Hedef: En iyi n_neighbors (komşu sayısı) ve weights (ağırlıklandırma metodu) parametrelerini bulmak. Hiperparametre Aralığı: n_neighbors: [3, 5, 7, 9, 11] weights: ['uniform', 'distance'] Grid Search Cross Validation kullanarak en iyi kombinasyonu belirleyin. En iyi modelin doğruluğunu test edin.

```
X = df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].values

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
param_grid = {
    'n_neighbors': [3,5,7,9,11],  # alpha parametresinin farklı
değerlerini deniyoruz
    'weights': ['uniform', 'distance']  # Ridge regresyonu için farklı
çözüm yöntemleri
}

model_knn = KNeighborsClassifier()
grid = GridSearchCV(model_knn, param_grid, cv=5)

from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

# Bağımsız ve bağımlı değişkenleri seçme
X = df[['MonthlySalary', 'HoursWorked', 'SatisfactionScore']].values
y = df['RemoteWork'].values

# Eğitim ve test setine ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42, stratify=y)

# Hiperparametre aralığı belirleme
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance']
}

# KNN Modeli
knn = KNeighborsClassifier()

# Grid Search Cross Validation
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
scoring='accuracy')
grid_search.fit(X_train, y_train)

# En iyi parametreler ve doğruluk
best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Test seti üzerinde performans değerlendirme
y_pred = grid_search.best_estimator_.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

# Sonuçları yazdırma
print("En İyi Parametreler:", best_params)
print("CV Üzerindeki En İyi Doğruluk:", best_score)
print("Test Seti Doğruluğu:", test_accuracy)
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

```
En İyi Parametreler: {'n_neighbors': 9, 'weights': 'uniform'}
CV Üzerindeki En İyi Doğruluk: 0.575
Test Seti Doğruluğu: 0.55
Classification Report:
               precision    recall  f1-score   support

           0       0.58      0.92      0.71        12
           1       0.00      0.00      0.00         8

    accuracy                           0.55        20
   macro avg       0.29      0.46      0.35        20
weighted avg       0.35      0.55      0.43        20
```

Veri setinizde eksik değerleri doldurmak için Simple Imputer'ı kullanın. Aşağıdaki stratejilere göre eksik verileri doldurun: Sayısal sütunlar için ortalama (mean). Kategorik sütunlar için en sık kullanılan değer (most_frequent). Eksik veriler tamamlandıktan sonra bağımlı değişkeni tahmin etmek için bir Linear Regression modeli eğitin. Modelin R^2 ve MSE değerlerini hesaplayın.

```python
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import numpy as np

# Kategorik ve sayısal sütunları ayırma
object_cols = list(df.select_dtypes(include='object').columns)  #
Kategorik sütunlar
X_cat = df[object_cols]
X_nums = df.drop(object_cols, axis=1)  # Sayısal sütunlar

# Kategorik sütunlardaki eksik değerleri doldurma
cat_imputer = SimpleImputer(strategy='most_frequent')
X_cat_imputed = pd.DataFrame(cat_imputer.fit_transform(X_cat),
columns=X_cat.columns)

# Sayısal sütunlardaki eksik değerleri doldurma
num_imputer = SimpleImputer(strategy='mean')
X_nums_imputed = pd.DataFrame(num_imputer.fit_transform(X_nums),
columns=X_nums.columns)

# Sayısal ve kategorik sütunları birleştirme
X_cleaned = pd.concat([X_nums_imputed, X_cat_imputed], axis=1)

# Bağımsız ve bağımlı değişkenleri ayırma
y = X_cleaned['target_column']  # Buraya bağımlı değişken sütununuzu
yazın
X = X_cleaned.drop('target_column', axis=1)

# Veri setini eğitim ve test olarak ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Linear Regression Modeli
model = LinearRegression()
model.fit(X_train, y_train)

# Tahminler
y_pred = model.predict(X_test)

# Performans Metrikleri
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

# Sonuçları yazdırma
print("R² Skoru:", r2)
print("Ortalama Kare Hatası (MSE):", mse)
```