5nkmykagg

January 9, 2025

[ ]:
```
import pandas as pd
#30.09.2025
df_animals = pd.read_csv('../data/msleep.csv')
```

[ ]:
```
df_animals.tail(4)
#.tail fonksiyonu dataframe'deki sondan kaç tane satır getiriliceğini␣
 ↪gösteririr.
```

[ ]:
```
df_animals.head(4)
 #.head fonksiyonu dataframe'nin baştan kaç tane satırının getiriliceğini␣
 ↪belirler.
```

[ ]:
```
df_animals.columns
 #.columns dataframe içerisindeki sütunların isimlerini verir.
```

[ ]:
```
df_animals.dtypes
 # dataframe içerisindeki sütunların veri tiplerini gösterir.
```

[ ]:
```
df_animals.shape
 #dataframe içerisindeki (satır, sütun) sayılarını verir.
```

[ ]:
```
df_animals.info()
#dataframe hakkında bilgi verir.
```

[ ]:
```
df_animals.describe()
 #Bir dataframe veya seri için özet istatistikler üretir.
```

[ ]:
```
df_animals.genus
#genus sütunundaki bilgileri verir.
```

[ ]:
```
df_animals[['name', 'genus']]
# sadece name ve genus sütunlarını getirir.
```

[ ]:
```
df_animals['name'].str.len()
#name sütunundaki verilerin metin olarak uzunluğunu verir.
```

```
[ ]: df_animals[df_animals['vore'] == 'herbi' ]
     #vore sütununda 'herbi' verisi bulunan satırları getirir.
```

```
[ ]: df_animals[(df_animals['vore'] == 'herbi') & (df_animals['sleep_total'] > 9)]
     # vore sütununda 'herbi bulunan ve sleep_total sütunundaki veri 9'dan büyük␣
      ↪olanları getirir.
```

```
[ ]: df_animals['vore'].value_counts()
      # vore sütunundaki verilerin çeşitlerini sayıları ile birlikte veirir.
```

```
[ ]: df_animals.groupby('vore')['vore'].count()
      # vore sütunundaki verilerin çeşitlerini sayıları ile birlikte veirir.
```

```
[ ]: df_animals['vore'].unique()
     #eşsiz olan verileri listeler.
     #teknik olarak üsteki 2'si ile aynı tipleri verir.
```

```
[ ]: df_animals['vore'].nunique(dropna=False)
     #nunique() sütundaki eşsiz değerlerin sayısını verir.
     #dropna, eğer false ise NaN değerleri'de sayar.
```

```
[ ]: import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[ ]: sns.histplot(df_animals.sleep_total)
     #histogram(sıklık dağılımı grafiği) çizer.
     #Grafikte yatay eksende sleep_total değişkeninin değer aralıkları gösterilir.
     #DİKey eksende bu aralıklara düşen veri sayıları gösterilir.

     plt.title("Distrubution of sleep Times of Various Mammals") #Başlık verir
     plt.xlabel("Hours of Sleep") #X eksenine bir label ekler ve içeriğini doldurur.
```

```
[ ]: import numpy as np
```

```
[ ]: np.mean(df_animals.sleep_total)
     df_animals['sleep_total'].mean()
     #İkiside aynı sonucu verir. 'sleep_total' sütunundaki değerlerin ortalamasını␣
      ↪hesaplar.
```

```
[ ]: np.median(df_animals.sleep_total)
     # sleep_total sütunundaki değerlerin median'ını bulur.
     # ortanca değer hesabıdır. Veri setinin sıralandıktan sonra ortada kalmış␣
      ↪değerini ifade eder.
     #Verileri küçükten büyüğe doğru sıraladıktan sonra, veri adedi tek sayı ise␣
      ↪medyan ortadaki verinin değeridir. Eğer çift sayı ise
     # ortadaki iki sayının aritmetik ortalaması medyandır.
```

[ ]:
```
import statistics as stat
```

[ ]:
```
stat.mode(df_animals.vore)
#vore sütunundaki değerlerin mod'unu bulmak için kullandık.
#mod; bir veri setindeki en çok tekrar eden değere denir.
```

[ ]:
```

```

ys7m57j7z

January 9, 2025

[ ]:
```
import pandas as pd
#07.10.2024
```

[ ]:
```
df_animal = pd.read_csv("../data/msleep.csv")
```

[ ]:
```
df_animal.sleep_total.value_counts(ascending = True)
#sleep_total sütunundaki değerlerin toplam sayısını verir.
# ascending = true ise sayılar azdan çoğa doğru sıralanır.(adet olarak)
```

[ ]:
```
df_animal[df_animal.vore == "insecti"]
#vore sütunundaki değeri "insecti" olan satırları getirir.
```

[ ]:
```
df_animal[df_animal.vore == "insecti"]["sleep_rem"].agg(["mean","median"])
#vore sütunu değeri "insecti" olan satırların sleep_rem değerlerinin medyan'ını␣
 ↪ve ortalamasını hesaplar.
```

[ ]:
```
df_animal.loc[len(df_animal.index)] = ["New Insect","","insecti","","",0.0,0.
 ↪0,0.0,0.0,0.0,0.0]
#df_animal dataframe'ine yeni bir satır eklemek için kullanılır.
#Yeni satırdaki sütun değerleri atama operatöründen sonra sırayla girlimiştir.
df_animal #dataframe'i yazdırmak.
```

[ ]:
```
#Bir tane grafik verip outline ı sorabilir . ortalama kullanmak aykırı␣
 ↪değerlerden çok etkilenmesine sebep olur bu yüzden ortalama kullanmak yerine␣
 ↪medyanı kullan.
#Dağılıma bağlı olarak olasılıklarda değişir .
#Range maksimumdan minimumu çıkarma ile bulunur.
```

[ ]:
```
df_animal['sleep_total'].min()
#sleep_total sütunundaki en düşük değeri verir.

df_animal['sleep_total'].max()
#sleep_total sütunundaki en büyük değeri verir.

range_sleep_total = df_animal['sleep_total'].max() - df_animal['sleep_total'].
 ↪min()
```

```
#range: en düşük ve en yüksek değerler arasındaki farktır. Maximum - Minimum olarak hesaplanır.
range_sleep_total
```

[ ]:
```
import numpy as np

dists = df_animal.sleep_total - np.mean(df_animal.sleep_total)
#sleep_total sütunundaki her bir değerin, ortalamadan ne kadar uzak olduğunu hesaplar. Burada dists bir liste

np.mean(np.abs(dists))
#np.abs mutlak değer almak için kullanılır.
#np.mean ile mutlak değeri alınan değerlerin ortalamasını alırız.
```

[ ]:
```
#varyans her bir veri noktasının ortalamaya olan uzaklığını hesaplar. Varyans ne kadar büyükse veriler o kadar dağılmış olur.
#Varyansın karakköüne ise standart sapma diyoruz.
#Quartiles görselleştirirken boxplot kullanırız. Quartiels veriyi çeyrek kısımlara bölerek bize bilgi verir.

#Interquartile Range (IQR) q3 yani %75 den q1 i yani %25 i çıkartılarak bulunur.
```

[ ]:
```
np.var(df_animal.sleep_total, ddof= 1)
# varyans hesaplama. ddof varyans hesaplarken serbestlik derecesidir.

np.sqrt(np.var(df_animal.sleep_total, ddof= 1))
#sqrt karakök almaya yarar. Varyansın karakökü standart sapmadır. Bu işlem standart sapmayı verir.

np.std(df_animal.sleep_total, ddof= 1)
#standart sapmayı std kütüphanesi ile direkt almak.
```

[ ]:
```
#Örnek yerine geri yerleştirildiğinden ve tekrar seçilebildiğinden buna değiştirmeli örnekleme denir.
```

[ ]:
```
#Koşullu olasılık. Bir olayın başka bir olayı etkileyen olasılık türüdür. Bağımlı olasılıktan bahsediyor.
#Ayrık dağılımlar --> veriler sayılabilir olmalı. virgüllü olmamalı ama integer olmalı
#Olasılık dağımlı --> bir senaryodaki her olası sonucun olasılığını açık

#bir dağılımda bütün olasılıklar eşitse buna discrete uniform distrubition denir.
#Örnekleme sayısı ne kadar büyürse teorideki sonuca o kadar yaklaşırız. buna law of large numbers nedir.
```

[ ]:
```
import matplotlib.pyplot as plt
```

[ ]:
```
plt.boxplot(df_animal.sleep_total)
#sleep_total sütununun kutu grafiğini oluşturur.
#q1(çeyrek) => verinin alt %25'ini temsil eder.
#q2(medyan) => verinin ortalamasını veya medyanınını temsil eder.
#q3(3.çeyrek) => verinin en üst %25'ini temsil eder.
#Boxplot grafiği hiçbir zaman ortalamayı vermez.
```

[ ]:
```
np.quantile(df_animal.sleep_total, [0,0.2,0.4,0.6,0.8,1])
#quantile fonksiyonu veriyi yüzdelik dilimlere böler.
#Burada 0 = en düşük veri iken 1 = en büyük veriyi temsil eder.
#0.2,0.4,0.6,0.8 ise sırasıyla %20, %40, %60, %80'in altındaki verileri temsil␣
↪eder.
```

[ ]:
```
np.quantile(df_animal.sleep_total, 0.75) - np.quantile(df_animal.sleep_total, 0.
↪25)
#Bu işlem bize veri kümesindeki çeyrekler arası aralığı(Interquartile Range,␣
↪IQR) hesaplar.
#IQR bize veri setinin ortadaki %50'sinin (orta dağılım) ne kadar yayıldığını␣
↪gösterir.,
# IQR = Q3 - Q1
```

[ ]:
```
from scipy.stats import iqr
iqr(df_animal.sleep_total)
#İQR yi hızlı bulmamızı sağlar.
```

[ ]:
```
from scipy.stats import iqr
iqr = iqr(df_animal.bodywt)
# bodywt sütununun IQR'sini alıp iqr değişkenine atadık.
```

[ ]:
```
lower_thereshold = np.quantile(df_animal.bodywt, 0.25) - 1.5 * iqr
#Bu işlem bodywt sütunu için alt eşik değeri hesaplamamızı sağlar.(lower␣
↪thereshold = alt eşikdeğer)
#alt eşikdeğer, aykırı değeri(outliers) belirlemek için kulllanılır.
```

[ ]:
```
upper_thereshold = np.quantile(df_animal.bodywt, 0.75) + 1.5 * iqr
#Bu işlem bodywt sütunu için üst eşikdeğeri hesaplamak için kullanılır.
#büyük eşikdeğer, aykırı değeri belirlemek için kullanılır.
```

[ ]:
```
df_animal[(df_animal.bodywt < lower_thereshold) | (df_animal.bodywt >␣
↪upper_thereshold)]
#Bu işlem bize aykırı değerleri filtreler.
#Aykırı değerler alt eşik değerin altında, üst eşik değerin üstünde kalan␣
↪verilerdir.
```

[ ]:
```
import seaborn as sns
sns.boxplot(data = df_animal, y='bodywt')
#normalde kutu grafiği oluşturması gerekiyor fakat burada biraz farklı sanki
```

[ ]:
```
import plotly.express as px
px.box(df_animal, x='bodywt')
#bodywt sütununun kutu grafiğini oluşturur. Üstekki ile aynı grafik sadece tipi␣
 ↪biraz farklı.
```

[ ]:
```
import pandas as pd
df_sales = pd.read_csv("../data/amir_deals.csv")
```

[ ]:
```
df_sales_users = df_sales.groupby("num_users")["amount"].agg(sum = "sum")
#Bu kod ile df_sales dataframe'ini num_users sütununa göre gruplar.
#her grup için amount sütunundaki verilerin toplamını hesaplar.
#Toplam sonucunu bir sütun olarak ekler.
df_sales_users
```

[ ]:
```
df_sales_users.sample()
#df_sales_users data frame'inden rastgele bir satır döndürür.
#veri setinden rastgele örnek almamız için kullanılır. (Boş işler =))
```

[ ]:
```
import numpy as np
np.random.seed(42)
#rastgele sayı üretme algoritmasında başlangıç değeri vermemizi sağlar.
#Bu sayede aynı rastgele işlemleri tekrar ettiğimiz sürece aynı sonuçları␣
 ↪alırız.
```

[ ]:
```
df_sales_users.sample(5, replace=True)
#Bu işlem df'den rastgele 5 satır seçmemize yarar. replace = true olduğu için␣
 ↪aynı satırlar gelebilir.
```

[ ]:
```
#Ayrık dağılımlar --> veriler sayılabilir olmalı. virgüllü olmamalı ama integer␣
 ↪olmalı
#Olasılık dağımlı --> bir senaryodaki her olası sonucun olasılığını açıklar.

#bir dağılımda bütün olasılıklar eşitse buna discrete uniform distrubition␣
 ↪denir.

#Örnekleme sayısı ne kadar büyürse teorideki sonuca o kadar yaklaşırız. buna␣
 ↪law of large numbers nedir.
```

dzwrucxoh

January 9, 2025

[ ]:
```
#14.10.2024
import scipy
```

[ ]:
```
from scipy.stats import uniform
#0 ile 12 arasındaki uniform dağılımda 7'nin alanının ne kadarı kapladığını␣
 ↪verir.
uniform.cdf(7,0,12)
```

[ ]:
```
#uniform için cdf
#Belirli bir değerin, toplam olasılık alanında ne kadar yer kapladığını␣
 ↪hesaplar.
#cdf(hesaplanmak istenen değer, alanın başlangıç değeri, Alanın bitiş değeri)
```

[ ]:
```
#0 ile 12 arasındaki uniform dağılımda tamamından 7'nin alanının farkı.
#ya da 7-12 arasındaki alan.
1 - uniform.cdf(7,0,12)
```

[ ]:
```
#0-12 dağılımında 0-7'arasının alanından 0-4 arası alanının çıkarıldığında␣
 ↪kalan alan.
uniform.cdf(7,0,12) - uniform.cdf(4,0,12)
```

[ ]:
```
#0-12 dağılımında 0-12 arasının alanı
uniform.cdf(12,0,12)
```

[ ]:
```
#uniform dağılıma göre rastgele sayı üretmek için rvs fonksiyonunu kullanırız.
scipy.stats.uniform.rvs(loc=0,scale=1,size=1,random_state=None)
#loc-> Başlangıç noktası.
#scale -> Bitiş noktası
#size -> Üretilicek sayı miktarı.
#random_state -> 'none' ise her seferinde farklı sayı verir.
```

[ ]:
```
from scipy.stats import binom
#Belirli bir sayıda deneme ve başarı olasılığına bağlı olarak rastgele başarı␣
 ↪sayısı üretir.
binom.rvs(8,0.5,size = 1)
# rvs(tek seferde atış miktarı,her denemenin başarı olasılığı, rastgele örnek␣
 ↪sayısı)
```

[ ]:
```
binom.rvs(3,0.5,size=10) #3 parayı 10 defa attık. Her turdaki tura sayısı.
```

[ ]:
```
binom.rvs(3,0.25,size=10)
#3 parayı 10 defa attık.
#%25 yazı, %75 tura gelme ihtimali var.
#her turdaki yazı sayısı.
```

[ ]:
```
# binom için pmf
#pmf => Olasılık Kütle fonksiyonu.
#Belirli bir olasılık dağılımına göre belirli bir olasılığın gerçekleşme␣
 ↪ihtimali.
#.pmf(istenen başarı sayısı, deneme sayısı, başarı olasılığı)
#10 denemede 7'sinin yazı gelme olasılığı.
binom.pmf(7,10,0.5)
```

[ ]:
```
#10 denemede 7 veya daha az yazı gelme olasılığı.
binom.cdf(7,10,0.5)
```

[ ]:
```
#7 den fazla yazı gelme olasılığı.
1 - binom.cdf(7,10,0.5)
```

[ ]:
```
# norm için cdf
#cdf fonksiyonu -> Kümülatif dağılım fonksiyonudur.
#Belirli bir olaslık dağılımı için bir değerin, o değerden  küçük veya eşit␣
 ↪olma olasılığını hesaplar.
#cdf(Hesaplamak istenen değer, Ortalama,
```

[ ]:
```
from scipy.stats import norm
#Ortalaması 161 ve standart sapması 7 olan kadınlara ait bir normal dağılım␣
 ↪olsun.
#Kadınların % kaçının 154'den kısa olduğunu bul.
norm.cdf(154,161,7)
```

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
die = pd.Series([1,2,3,4,5,6])
samp_5 = die.sample(5,replace = True)
#sample methodu bir seriden rastgele örnekler almak için kullanılır.
#sample(örnek sayısı, replace= (true ise geri yerleştirme açık))

samp_5
```

[ ]:
```
import matplotlib.pyplot as plt
sample_means=[]
for i in range(10):
```

```
    samp_5 = die.sample(5,replace=True)
    sample_means.append(samp_5.mean())
plt.title("Örneklem Ortalamasının Örneklem Dağılımı")#Grafiğe başlık verir.
plt.hist(sample_means)#Histogram grafiği oluşturur.
#Böyle bir özet istetistiğinin dağılımına örenklem dağılımı denir.
```

[ ]:
```
import matplotlib.pyplot as plt
sample_means=[]
for i in range(100):
    samp_5 = die.sample(5,replace=True)
    sample_means.append(samp_5.mean())
plt.title("Örneklem Ortalamasının Örneklem Dağılımı")
plt.hist(sample_means)#veri kümesinin histogramını oluşturur.
```

[ ]:
```
import pandas as pd
sales_team = pd.Series(['Amir','Brian','Claire','Damian'])
sales_team.sample(10,replace=True)
```

[ ]:
```
sample_prp= []
for i in range(1000):
    sample_5 = sales_team.sample(10, replace=True)
    try:
        sample_prp.append(samp_5.value_counts()['Claire']/10)
    except:
        sample_prp.append(0)
plt.title("Örneklem oran Dağılımı")
plt.hist(sample_prp)
```

[ ]:
```
#Poisson Dağılımı
#lambda büyüdükçe standart sapma artar
#Örneklem sayısı arttıkça normal dağılıma yaklaşır.
```

[ ]:
```
#Haftada ortalama 8 sahiplenme gerçekleştiği bir sığınakta, bir haftada 5␣
 ↪sahiplenme gerçekleşmesi oranı?
from scipy.stats import poisson
poisson.pmf(5,8)
#poisson.pmf(hesaplanması istenilen olasılık, beklenen değer(lambda, ))
#Poisson dağılımın olasılık kütlesini hesaplar. Belirli bir olayın gerçekleşme␣
 ↪oranıdır.
```

[ ]:
```
#5 veya daha az sahiplenme olasılığı
poisson.cdf(5,8)
#poisson.cdf kümülatif dağılım hesaplar.
#cdf(k, )
#Bir olayın k veya daha az gerçekleşme olasılığıdır.
```

[ ]:
```
#5'den daha fazla gerçekleşme olasılığı.
1 - poisson.cdf(5,8)
```

[ ]:
```
poisson.rvs(8,size=10)
#poisson dağılımına göre 8 ortalamaya sahip rastgele 10 tane sayı üretir.
```

[ ]:
```
#üstel Dağılım
from scipy.stats import expon
#yeni bir istek için 1 dakikadan az bekleme olasılığı
expon.cdf(1,scale=2)
#üstel dağılım için kümülatif dağılım fonksiyonu
#cdf(olasılığını hesaplamak istediğimiz değer,ß üstel dağılımın ortalama değeri)
#cdf(x,scale) belirli bir x değeri için üstel dağılımda o değere kadar olan␣
 ↪olasılık toplamını verir.
```

[ ]:
```
#yeni bir istek için 4 dakikadan fazla bekleme olasılığı
1 - expon.cdf(4,scale=2)
#Tekrar bak !!!

#1 ile 4 dakika arasında bekleme olasılığı
expon.cdf(4,scale=2) - expon.cdf(1,scale=2)
```

[ ]:
```
## ödev
import matplotlib.pyplot as plt
import scipy.stats as stats

x = np.linspace(-5, 5, 100)

# Plot for different degrees of freedom
for df in [1, 5, 10, 30]:
    plt.plot(x, stats.t.pdf(x, df), label=f'df={df}')

plt.plot(x, stats.norm.pdf(x), label='Normal distribution',
         linestyle='--')
plt.legend() #lejant vermek için kullanılır.
plt.title('t-distribution with different degrees of freedom')
plt.xlabel('x')
plt.ylabel('Density')
plt.show()
```

[ ]:
```

```

zjhshhln8

January 9, 2025

null hiptezi (H0) => Yokluk hiptezi olarak bilinir. (İki gurubun arasında fark yoktur gibi) alternatif hipotez (H1) => H0 hipotezine karşı çıkan bir hipotezdir. (İki grup arasında fark vardır)

Öncelikle yokluk hipotezi tanımlanır arkasından ona karşı bir alternatif hipotez tanımlanır.

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

[ ]:
```
df_sleep = pd.read_csv("../data/msleep.csv")
df_sleep
```

[ ]:
```
sns.scatterplot(x='sleep_total', y='sleep_rem',data = df_sleep)
plt.show()#Plot grafiği çizdirdik.
```

[ ]:
```
df_sleep["sleep_total"].corr(df_sleep["sleep_rem"])
#sleep_total ve sleep_rem arasındaki korolasyonu hesapladık.
#1' e yakın olduğu için güçlü pozitif korolasyon.
```

[ ]:
```
sns.lmplot(x='sleep_total',y='sleep_rem', data=df_sleep, ci=None);
#lmplot lineer regresyon'u çizdirmek için kullanılır.
#ci; güven aralığını temsil eder. regesyonun olucağı alanın tahmini␣
 ↪görüntüsüdür.
```

[ ]:
```
sns.lmplot(x='sleep_total',y='sleep_rem', data=df_sleep,);
#Güven aralığına sahip.
```

[ ]:
```
df_sleep.corr(numeric_only= True)
#.corr korolasyon hesabında kullanılır. Fakat string değerlerde patlar.
#numeric only = true parametresi sadece numerik değere sahip sütunları almamızı␣
 ↪sağlar.
```

[ ]:
```
sns.heatmap(df_sleep.corr(numeric_only=True));
#Korolasyonun heatmap(ısı haritası) çizimi
```

[ ]:
```
sns.heatmap(df_sleep.corr(numeric_only=True),annot= True);
#Heatmap'da korolasyonun sayısal katsayılarını gösteriyor. annot = True
#Bir değeri  kendisi ile korolasyonu daima 1'dir.
```

[ ]:
```
sns.heatmap(df_sleep.corr(numeric_only=True), annot=True, cmap='YlGnBu')
#annot = True olduğunda bu map'de korelasyon katsayılarını görmemizi sağlar.
#cmap ise burada renk paleti anlamına gekirç.
```

[ ]:
```
import seaborn as sns
sns.scatterplot(x="bodywt",y="awake",data=df_sleep)
#bodywt'nin awake sütununa göre korolasyonu
print(df_sleep["bodywt"].corr(df_sleep["awake"]))
```

[ ]:
```
plt.hist(df_sleep["bodywt"])
plt.title("Body Weight Distribition")
plt.show()
#Bodywt sütununun histogram grafiği
```

[ ]:
```
sns.histplot(data=df_sleep, x="bodywt", kde=True, bins=10);
# bins = 10 histogramdaki çubuk sayısını belirler.
#kde = True Grafiğin yoğunluk eğrisini gösterir
```

[ ]:
```
# Log Transformation
# Veriler bu sekilde yüksek oranda çarpik oldugunda,
# bir log dönüsümü uygulanabilir.
# Her bir vücut agirlignin logunu tutan log_bodywt adinda # yeni bir sütun␣
 ↪olusturulr. Bu np. log() kullanilarak yapilabilit
# iliski normal vücut agirligi ile uyanik kalma süresi arasindaki # iliskiden␣
 ↪çok daha dogrusal görünür. Vücut agirliginin logaritn
# uyanik kalma süresi arasindaki korelasyon yaklasik 0.57 dir,
# bu da daha önce sahip olunan 0.31'den gok daha yüksektir.
import numpy as np
df_sleep["log_bodywt"] = np.log(df_sleep["bodywt"])
print(df_sleep["log_bodywt"].corr(df_sleep["awake"]))

sns.lmplot(x="log_bodywt", y="awake", data=df_sleep, ci=None)
plt.show()
print()
```

[ ]:
```

```

ykrtfxnwi

January 9, 2025

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib as plt
```

[ ]:
```
df_sleep = pd.read_csv("../data/msleep.csv")
df_sleep
```

[ ]:
```
# p değeri <= 0.05 ise H0 red edilir.
# p değeri > 0.05 ise h0 kabul edilir.

#p değeri cdf fonksiyonu ile hesaplanır.
# kesişen 2 grafik arası p değeri arada kalan alandır.(ortak alan)
```

[ ]:
```
#Tip 1 error = h0 doğruyken, h0'ı red edersek karşılaştığımız hatadır.

#Tip 2 error = h0 yanlış iken, h0'ı doğru kabul edersek tip2 error oluşur.
```

[ ]:
```
# * p değeri sorar. cdf fonk sorar.
```

[ ]:
```
df_coffee =  pd.read_feather("../data/coffee_ratings_full.feather")
df_coffee
```

[ ]:
```
# Bir df nin en yüksek en düşük satırı stünu ... gibi getiren bir sorgu sorusu␣
↪sorulabilir.

#total_cup_points sütunun en yüksek değerlerini taşıyan satırı seçer.
df_coffee[df_coffee['total_cup_points'] == df_coffee['total_cup_points'].max()]
```

[ ]:
```
# Temelde üstteki ile aynı işlevi yapar.
#total_cup_points sütnunua göre sıralar ve en yüksek değeri döndürür.
df_coffee.sort_values(by= 'total_cup_points', ascending= False).iloc[0]
# by = neye göre sıralim, ascending = False ise büyükten küçüğe sırala.
```

[ ]:
```
#total_cup_points sütununda maks değeri içeren satırı seçer ve _max' a atar.
total_cup_points_max = df_coffee.loc[df_coffee['total_cup_points'].idxmax()]
total_cup_points_max
```

[ ]:
```
#dataframe'den total_cup_points ve flavor sütunlarının verilerini çeker.
#yeni bir data frame'e atar.
pts_vs_flavor_pop = df_coffee[['total_cup_points','flavor']] #ikili parantez␣
 ↪bir listeyi temsil ettiği için bulunuyor.
pts_vs_flavor_pop
```

[ ]:
```
#rastgele 10 tane satırı çekip yeni bir df'ye atar.
pts_vs_flavor_pop_sample = pts_vs_flavor_pop.sample(n=10)
#10 tane rastgele kayıt seç ve yerine koymadan seç (yerine koyması için␣
 ↪replacement = true)
pts_vs_flavor_pop_sample
```

[ ]:
```
cut_points_samp = df_coffee["total_cup_points"].sample(n=10) # Serilerden de␣
 ↪veri çekilebilir.
cut_points_samp
```

[ ]:
```
import numpy as np
print(f"Popülasyon Ortalaması = {np.
 ↪mean(pts_vs_flavor_pop['total_cup_points'])}")
print(f"Örnek Ortalaması = {np.mean(pts_vs_flavor_pop['total_cup_points'])}")
print(f"Örnek Ortalaması = {np.mean(cut_points_samp)}")
```

[ ]:
```
df_coffee['total_cup_points'].mean()
```

[ ]:
```
df_coffee.head()['total_cup_points'].mean()
#.head() fonksiyonu default olarak ilk 5 satırı döndürür.
```

[ ]:
```
#df_coffe dataframe'sinin total_cup_points sütunun histogram grafiği.
df_coffee['total_cup_points'].hist(bins= np.arange(59,93,2))
#arange = 59-93 arasını 2şer 2 şer bölmemizi sağlıyor.
```

[ ]:
```
df_coffee.head()['total_cup_points'].hist(bins=np.arange(59,93,2));
#ilk 5 satırın
```

[ ]:
```
df_coffee.sample(n=10)['total_cup_points'].hist(bins=np.arange(59,93,2));
#rastgele 10 satırın histogramı
```

[ ]:
```
#Beta dağılımına göre rastgele değerler üreten fonksiyondur. Beta dağılımı 0-1␣
 ↪arasındadır.
#a parametresi => dağılımın sol kısımını 0'a yakın tarafını kontrol eder.
#b parametresi => dağılımın sağ tarafını 1'e yakın tarafını kontrol eder.
#size parametresi => kaç rastgele sayı üreteceğini kontrol eder.
#random_state => belirli bir rastgele örneklem tekrarını sağlamak için␣
 ↪kullanılır. (opsiyonel)
randoms = np.random.beta(a=2,b=2,size=5000) # Fonksiyondaki paramatreler nedir␣
 ↪olarak sorular gelebilir.
```

```
randoms
```

[ ]:
```
import matplotlib.pyplot as plt
#datası randoms olan bir histogram.
#np.arange ile (başlangıç aralık, sonr aralık, bölme miktarı)
#0-1 arasındaki aralığı 0.05'lik gruplara ayırır. Burada toplam 20aralık var.
plt.hist(randoms,bins=np.arange(0,1,0.05));
```

[ ]:
```
#sample kullanırken n= rastgele örnek sayısı
#random_state = seed değeridir. Rastgle örneklerin tekrarlanabilir olmasını␣
 ↪sağlar.
#burada random_state = 1900000113 olması her seferinde aynı örnekleri vermesini␣
 ↪sağlar.
df_coffee.sample(n=5,random_state=190000113)
```

[ ]:
```

```

ceb0sosu8

January 9, 2025

[ ]:
```
#Sınavda boşluk doldurma ve kod gelicek.
#Sınav cuma günü saat 9 da
import pandas as pd
df_coffee = pd.read_feather("../data/coffee_ratings_full.feather")
df_coffee

#sistematik örneklenme - aralığın tanımlanması
```

[ ]:
```
sample_size = 5
#len(..) df'deki satır sayısını verir.
pop_size = len(df_coffee) #df_coffee.shape[0] aynı sayıyı verir. (.shape␣
 ↪fonskiyonu hep satır hem sütun çevirir. )
pop_size
```

[ ]:
```
# // ifadesi tam bölme anlamına gelir. Bölme işlemindeki kalan yokmuş gibi␣
 ↪bölen sayısını verir.(Küsürat vermez)
interval = pop_size // sample_size
interval
```

[ ]:
```
#systematic sampling - selecting the rows (sistematik örneklem.)
df_coffee.iloc[interval::interval] # 267. satırdan başla ve 267'şer ilerle␣
 ↪sonuçları döndür
```

[ ]:
```
#the truouble with systematic sampling
df_coffe_id = df_coffee.reset_index()
#plot grafiğinde kind parametresi grafik türünü belirtir.
df_coffe_id.plot(x='index', y='aftertaste',kind='scatter');
#Grafik türleri ; 'line', 'scatter', 'barh', 'hist', 'box', 'area', 'pie ',␣
 ↪'heatmap', 'violin'
```

[ ]:
```
#sample fonk'da frac parametresi tüm verileri rastgele karıştırmak için␣
 ↪kullanılır (frac = 1 ise karıştır.)
shuffled = df_coffee.sample(frac=1) #tüm satırları karıştırarak bir df gönderir.
#.reset_index df' nin indexlerini sıfırlar (dropna= true index sütununu siler.)
shuffled = shuffled.reset_index(drop=True).reset_index();
#önce indexleri sıfırladık ve index sütununu sildik. Ardından yeniden bir index␣
 ↪sütunu ekledik (2. reset ile)
```

```
shuffled.plot(x='index',y='aftertaste',kind='scatter');
```

[ ]:
```
top_counts = df_coffee['country_of_origin'].value_counts()
top_counts
```

##Buradan Sonra işler biraz karışıyor !!!!

[ ]:
```
#El ile istediğimiz şehirleri giriyoruz.Fakat bu yöntem sınavda nanay gibi
top_counted_countries =␣
 ↪{'Mexico','Colombia','Guatemala','Brazil','Taiwan','United States (Hawaii)'}
#list(top_counts.head(6).index) Sanki bu yöntem daha mantıklı...


#Anlamadım burayı...
top_counted_subset = df_coffee['country_of_origin'].isin(top_counted_countries)
coffee_ratings_top = df_coffee[top_counted_subset]
coffee_ratings_top
```

[ ]:
```
#df_coffee df'sini groupby fonksiyonu ile 'country_of_origin' sütununa göre␣
 ↪gurpluyoruz,
#aynı sütünun değerlerine göre
#.agg fonksiyonu ile her grup için aldığımız sütunun eleman sayısını hesaplar,
#.nlargest ile en büyük 6 değerin bulunduğu sütunu alır,
#.index ile değerlerin indexlerini (Burada ülkeleri) alır,
#en son list' e çevirerrek verir.
list(df_coffee.groupby('country_of_origin')['country_of_origin'].agg('count').
 ↪nlargest(6).index)
```

[ ]:
```
list(top_counts.head(6).index) #üsttekinin kısa hali =)
```

[ ]:
```
#Derste birinin (tahminen gpt kullanarak) yaptığı başka bir gruplama yöntemi
import pandas as pd

df_coffee = pd.read_feather("../data/coffee_ratings_full.feather")

top_countries = df_coffee.groupby('country_of_origin')['country_of_origin'].
 ↪count().sort_values(ascending=False)
top_6 = top_countries.head(6)
top_6
```

[ ]:
```
#Basit Rastgele Örneklem
#sample() kullanarak veri setinin yüzde onluk basit rastgele bir örngei alalım␣
 ↪ve frac'ı 0.1 olarak ayarlayalım.
#frac = 0.1 -> DF'nin %10'unu kullan.
#random_stat = 2021 sayesinde her seferinde aynı sonucu elde ederiz.
```

```
coffee_ratings_samp = coffee_ratings_top.sample(frac=0.1,random_state=2021)
coffee_ratings_samp
```

[ ]:
```
#Seçimler arasında seçilme oranını sıralı olarak verir.
#.value_counts fonksiyonu her sütundaki benzersiz değerleri sayar.
#normalize = True parametresi, frekansları orana çevirir. Her ülkenin gözlem␣
 ↪sayısı, toplam gözlem sayısına oranını hesaplar.
coffee_ratings_samp['country_of_origin'].value_counts(normalize=True)
```

[ ]:
```
coffee_ratings_strat = coffee_ratings_top.groupby("country_of_origin").
 ↪sample(frac=0.1,random_state=2021)
coffee_ratings_strat['country_of_origin'].value_counts(normalize=True)
```

[ ]:
```
#katmanlı örne
coffee_rating_eq = coffee_ratings_top.groupby("country_of_origin").
 ↪sample(n=15,random_state=2021)
coffee_rating_eq['country_of_origin'].value_counts()
```

[ ]:
```
import numpy as np
coffee_ratings_weight = coffee_ratings_top
condition = coffee_ratings_weight['country_of_origin'] == 'Taiwan'

#Bu işlem ile taiwan olan satırların diğer satırlara göre iki kat seçilme şansı␣
 ↪olucaktır.
coffee_ratings_weight['weight'] = np.where(condition, 2,1)

coffee_ratings_weight = coffee_ratings_weight.sample(frac= 0.1, weights=␣
 ↪'weight')

coffee_ratings_weight['country_of_origin'].value_counts(normalize=True)



#Uyarının nedeni daha yeni bir yöntem olduğunu belirtiyor. Bizi şu an alakadar␣
 ↪etmez.
```

[ ]:
```
#Cluster Sampling
# cluster sampling vs stratified sampling
```

[ ]:
```
#Kahve veri setindeki kahve çeşitlerine bakalım

#variety sütunundaki her eşsiz değeri alır.
varieties_pop = list(df_coffee['variety'].unique())
varieties_pop
```

[ ]:
```
# Bazı grupları seçmek için basit rastgele alt örnekleme kullanılır
import random
varieties_samp = random.sample(varieties_pop, k=3)
varieties_samp
```

[ ]:
```
#Sadece bu alt gruplarda basit rastgele  alt örnekleme kullanılır

#.isin fonksiyonu df_coffe df'sinin variety sütunundaki değerlerin␣
 ↪varieties_samp listesinde olup olmadığını kontrol eder.
varieties_condition = df_coffee['variety'].isin(varieties_samp)
#df_coffee içerisinde varieties_condition değerlerinin true olduğu satırları␣
 ↪seçer ve yeni df'ye atar.
coffee_ratings_cluster = df_coffee[varieties_condition]
coffee_ratings_cluster
```

[ ]:
```
coffee_ratings_cluster.loc[:,'variety'] = coffee_ratings_cluster['variety'].
 ↪astype('category').cat.remove_unused_categories()

#Üstte alınan uyarıyı bu şekilde düzelterek de kullanabilirdik. loc ile␣
 ↪variety' i kategoriye çevirdik.
```

[ ]:
```
coffee_ratings_cluster.groupby('variety',observed=True).
 ↪sample(n=1,random_state=2021)
```

[ ]:
```
#Multistage sampling(Çok aşamalı örnekleme)
```

[ ]:
```
#Comparing sampling methods
```

[ ]:
```
#review of simple random sampling
coffee_ratings_srs = coffee_ratings_top.sample(frac = 1/3,random_state=2021)
coffee_ratings_srs.shape
```

[ ]:
```
#review of cluster sampling
```

[ ]:
```
coffee_ratings_strat = coffee_ratings_top.groupby('country_of_origin').
 ↪sample(frac=1/3,random_state=2021)
coffee_ratings_strat.shape
```

[ ]:
```
import random

top_countries_samp = random.sample(sorted(top_counted_countries), k=5)

top_condition = coffee_ratings_top['country_of_origin'].isin(top_countries_samp)

coffee_ratings_cls = coffee_ratings_top[top_condition]
```

```
coffee_ratings_cls.loc[:,'country_of_origin'] =␣
 ↪coffee_ratings_cls['country_of_origin'].astype('category').cat.
 ↪remove_unused_categories()

coffee_ratings_clst = coffee_ratings_cls.groupby('country_of_origin').
 ↪sample(n=len(coffee_ratings_top) // 6)

coffee_ratings_clst.shape
```

[ ]:

kknthk3wf

January 7, 2025

[ ]:
```
import pandas as pd
df_coffee = pd.read_feather("Data/coffee_ratings_full.feather")
df_coffee
```

[ ]:
```
sample_size = 5
pop_size=len(df_coffee) ##kaç satır olduğunu verir
##len yerine df_coffee.shape(0) da aynı sonucu verir
pop_size
```

[ ]:
```
interval = pop_size//sample_size
interval #aralık sayısını verir
```

[ ]:
```
df_coffee.iloc[267::267] #267. satırdan başlar ve 267 267 artarak satırları␣
↪dönderir
#systematic sampling - selecting the rows
```

[ ]:
```
df_coffee_id = df_coffee.reset_index()# verilere bir index sütünü oluşturur
df_coffee
```

[ ]:
```
df_coffee_id = df_coffee.reset_index()
df_coffee_id.plot(x="index", y="aftertaste", kind="scatter")
```

[ ]:
```
shuffle = df_coffee.sample(frac=1)
shuffle = shuffle.reset_index(drop=True).reset_index()
shuffle.plot(x="index",y="aftertaste",kind="scatter")
```

1 shuffle = df_coffee.sample(frac=1)

Bu satır, df_coffee adındaki veri çerçevesini tamamen rastgele karıştırır. frac=1 argümanı, veri çerçevesinin tamamının (yani %100'ünün) karıştırılmasını belirtir. Sonuç olarak, shuffle değişkeninde rastgele sıralanmış bir veri çerçevesi elde edilir.

2 shuffle = shuffle.reset_index(drop=True).reset_index()

İlk olarak reset_index(drop=True), önceki indeks değerlerini sıfırlar ve onları veri çerçevesinden kaldırır (artık bir sütun olarak eklenmez). Ardından .reset_index(), yeni bir indeks sütunu oluştu-

rur ve bunu veri çerçevesine ekler. Bu yeni indeks sütunu “index” adıyla eklenir ve 0’dan başlayarak artan bir sayısal sıralama içerir.

3 shuffle.plot(x=“index”, y=“aftertaste”, kind=“scatter”)

Bu satır, veri çerçevesindeki “index” sütununu x ekseni olarak ve “aftertaste” sütununu y ekseni olarak kullanarak bir scatter plot (dağılım grafiği) oluşturur. kind=“scatter”, bu grafiğin türünün bir dağılım grafiği olduğunu belirtir. Bu, “index”e karşılık gelen “aftertaste” değerlerinin rastgele dağılımını görselleştirir.

[ ]:
```
top_counts = df_coffee["country_of_origin"].value_counts()
top_counts
```

[ ]:
```
top_counted_countries =␣
 ↪["Mexico","Colombia","Guatemala","Brazil","Taiwan","United States(Hawai)"]
top_counted_subset = df_coffee["country_of_origin"].isin(top_counted_countries)
coffee_ratings_top = df_coffee[top_counted_subset]
coffee_ratings_top
```

top_counted_countries = [“Mexico”,“Colombia”,“Guatemala”,“Brazil”,“Taiwan”,“United States(Hawai)”] Bu satır, top_counted_countries adlı bir liste oluşturur. Bu liste, filtreleme yapılacak ülkeleri içerir.

top_counted_subset = df_coffee[“country_of_origin”].isin(top_counted_countries) df_coffee[“country_of_origin”].isin(top_counted_countries): Bu ifade, df_coffee veri çerçevesindeki “country_of_origin” sütununun değerlerini top_counted_countries listesindeki ülkelerle karşılaştırır. isin, bir değer bir listedeki elemanlardan biriyle eşleşirse True, aksi takdirde False döndüren bir boolean (mantıksal) dizi oluşturur. Örneğin: Eğer “country_of_origin” sütununda bir satırda “Mexico” varsa, bu satır için True, diğer durumlarda False döner.

coffee_ratings_top = df_coffee[top_counted_subset] Bu satır, top_counted_subset adlı boolean diziyi kullanarak, yalnızca True olan satırları filtreler. Sonuç olarak, “country_of_origin” değeri top_counted_countries listesindeki ülkelerden birine ait olan satırlardan oluşan yeni bir veri çerçevesi olan coffee_ratings_top oluşturulur.

coffee_ratings_top Son satır, oluşturulan filtrelenmiş veri çerçevesini (coffee_ratings_top) döndürür veya gösterir.

.isin(), bir Pandas fonksiyonudur ve bir sütundaki (veya bir Pandas serisindeki) değerlerin belirtilen bir liste veya set içinde olup olmadığını kontrol etmek için kullanılır. Fonksiyon, her bir değer için True veya False döndüren bir boolean (mantıksal) dizi üretir.

[ ]:
```
#Basit Rastgele Örneklem
#sample() kullanarak veri setinin yüzde onluk basit rastgele örnekelme
coffee_ratings_samp = coffee_ratings_top.sample(frac=0.1,random_state=2021)␣
 ↪#frac verilerin % lik seçimini temsil ediyor
#Eğer random_state belirlenirse, aynı kodu tekrar çalıştırdığınızda her␣
 ↪seferinde aynı rastgele satırlar seçilir. Bu, veri bilimi projelerinde␣
 ↪analizlerin tekrarlanabilirliği açısından faydalıdır.
```

```
coffee_ratings_samp
```

[ ]:
```
coffee_ratings_samp["country_of_origin"].value_counts(normalize=True)
```

[ ]:
```
coffee_rating_strat=coffee_ratings_top.groupby("country_of_origin").
 ↪sample(frac=0.1,random_state=2021)
coffee_rating_strat['country_of_origin'].value_counts(normalize=True)
#Bu kod, coffee_ratings_top veri çerçevesindeki verileri "country_of_origin"␣
 ↪sütununa göre gruplandırarak her ülkeden eşit oranlarda (%10) rastgele bir␣
 ↪örneklem alır. Ardından, seçilen örneklemdeki ülkelerin yüzdesel dağılımını␣
 ↪hesaplar.
```

[ ]:
```
coffee_rating_eg=coffee_ratings_top.groupby('country_of_origin').
 ↪sample(n=15,random_state=2021)
coffee_rating_eg['country_of_origin'].value_counts()
#Her ülkeden 15 satır rastgele seçer (sample(n=15)), gruplandırma␣
 ↪"country_of_origin" sütununa göre yapılır.
#Seçilen örneklemin ülke bazlı sayısını hesaplar (value_counts()).
#Sonuç: Her ülke için 15 olacak şekilde eşitlenmiş bir örnekleme ve bu␣
 ↪örneklemin dağılımı.
```

[ ]:
```
import numpy as np
coffee_ratings_weight=coffee_ratings_top
condition=coffee_ratings_weight['country_of_origin']=='Taiwan'
coffee_ratings_weight['weight']=np.where(condition,2,1)
coffee_ratings_weight=coffee_ratings_weight.sample(frac=0.1,weights='weight')
coffee_ratings_weight['country_of_origin'].value_counts(normalize=True)

#"Taiwan" için ağırlık 2, diğer ülkeler için 1 atar (np.where).
#Örneklemi, ağırlıklandırılmış olarak rastgele seçer (weights='weight').
#Seçilen örneklemin ülkelerinin yüzdesel dağılımını hesaplar␣
 ↪(value_counts(normalize=True)).
#Sonuç: "Taiwan" ağırlıklı olarak daha fazla örneklenir.
```

[ ]:
```
varieties_pop= list(df_coffee['variety'].unique())
varieties_pop
```

[ ]:
```
import random
varieties_samp= random.sample(varieties_pop,k=3)
varieties_samp
#bazı grupları seçmek için rastgele alt örnekleme kullanılır
```

[ ]:
```
variety_condition =df_coffee['variety'].isin(varieties_samp)
coffee_ratings_cluster=df_coffee[variety_condition]
coffee_ratings_cluster
```

```
[ ]: coffee_ratings_cluster.loc[:,'variety']=coffee_ratings_cluster['variety'].
      ↪astype('category').cat.remove_unused_categories()
     coffee_ratings_cluster.groupby('variety',observed=True).
      ↪sample(n=1,random_state=2021)
```

astype('category').cat.remove_unused_categories() variety sütununu kategorik türe dönüştürür. Kullanılmayan (veri çerçevesinde bulunmayan) kategorileri kaldırır.

groupby('variety', observed=True).sample(n=1, random_state=2021) variety sütununa göre gruplar oluşturur. Her gruptan 1 rastgele satır seçer (n=1).

Sonuç: Her benzersiz variety kategorisinden birer örnek alınır. Bu örnekler deterministik (aynı sonuçları veren) bir seçim için random_state=2021 ile sabitlenmiştir.

```
[ ]: top_counted_countries=['Mexico','Colombia','Guatemala','Brazil','Taiwan','United␣
      ↪States(Hawai)']
     top_counted_subset=df_coffee['country_of_origin'].isin(top_counted_countries)
     coffee_ratings_top =df_coffee[top_counted_subset]
     coffee_ratings_top.shape
     #Bu kod, sadece belirli ülkelerdeki (top_counted_countries) kahve verilerini␣
      ↪filtreler ve coffee_ratings_top adlı
     #alt kümenin satır ve sütun sayısını döndürür.
     #Sonuç: Veri çerçevesinin boyutları (shape).
```

```
[ ]: coffee_ratings_srs=coffee_ratings_top.sample(frac=1/3,random_state=2021)
     coffee_ratings_srs.shape
     #Bu kod, coffee_ratings_top veri çerçevesinden rastgele olarak 1/3'ünü seçer ve␣
      ↪sonucu coffee_ratings_srs olarak kaydeder.
     #Son olarak, seçilen örneklemin satır ve sütun sayısını döndürür (shape).

     #Sonuç: 1/3'lük örneklemin boyutu.
```

```
[ ]: coffee_ratings_strat=coffee_ratings_top.groupby('country_of_origin').
      ↪sample(frac=1/3,random_state=2021)
     coffee_ratings_strat
     #her ülkeden (her gruptan) verilerin 1/3'ü seçilir ve bu örneklemler␣
      ↪coffee_ratings_strat adlı yeni bir veri çerçevesinde toplanır.
```

```
[ ]: import random
     top_countries_samp=random.sample(top_counted_countries,k=2)
     top_condition=coffee_ratings_top['country_of_origin'].isin(top_countries_samp)
     coffee_ratings_cls = coffee_ratings_top[top_condition]
     coffee_ratings_cls.loc[:,'country_of_origin'] =␣
      ↪coffee_ratings_cls['country_of_origin'].astype('category').cat.
      ↪remove_unused_categories()
     coffee_ratings_clts = coffee_ratings_cls.groupby('country_of_origin').
      ↪sample(n=len(coffee_ratings_top)//6)
     coffee_ratings_clts.shape
```

Bu kod şu işlemleri yapar:

top_countries_samp = random.sample(top_counted_countries, k=2) top_counted_countries listesinden rastgele 2 ülke seçer ve top_countries_samp listesine atar.

top_condition = coffee_ratings_top['country_of_origin'].isin(top_countries_samp) coffee_ratings_top veri çerçevesindeki country_of_origin sütunundaki ülkeleri, top_countries_samp listesindeki rastgele seçilen 2 ülke ile karşılaştırarak, her satır için True veya False değerleri döndürür.

coffee_ratings_cls = coffee_ratings_top[top_condition] top_condition ile oluşturulan maskeyi kullanarak, sadece rastgele seçilen 2 ülkeden gelen verilerden oluşan coffee_ratings_cls alt kümesini oluşturur.

coffee_ratings_cls['country_of_origin'] = coffee_ratings_cls['country_of_origin'].astype('category').cat.remove_ country_of_origin sütununu kategorik veri türüne dönüştürür ve kullanılmayan kategorileri kaldırır.

coffee_ratings_clts = coffee_ratings_cls.groupby('country_of_origin').sample(n=len(coffee_ratings_top)//6) coffee_ratings_cls veri çerçevesini "country_of_origin" sütununa göre gruplandırır. Her grup için toplam veri çerçevesinin 1/6'ı kadar örnekleme alır (n=len(coffee_ratings_top)//6).

coffee_ratings_clts.shape Sonuç olarak, coffee_ratings_clts veri çerçevesinin boyutlarını döndürür.

Sonuç: Bu kod, 2 ülkenin verilerinden rastgele bir örnekleme alır ve bu örneklemenin boyutunu döndürür.

[ ]:
```
import random

top_countries_samp = random.sample(top_counted_countries, k=2)
#top_counted_countries listesinden 2 ülke rastgele seçilir ve␣
 ↪top_countries_samp listesine atanır.
top_condition = coffee_ratings_top['country_of_origin'].\
isin(top_countries_samp)
#isin() fonksiyonu kullanılarak, coffee_ratings_top veri çerçevesindeki␣
 ↪country_of_origin sütununda
#rastgele seçilen 2 ülkeden gelen veriler filtrelenir.

coffee_ratings_cls = coffee_ratings_top[top_condition]

coffee_ratings_cls.loc[:,'country_of_origin'] = \
coffee_ratings_cls['country_of_origin'].\
astype('category').cat.remove_unused_categories()
#country_of_origin sütunu kategorik veri türüne dönüştürülür ve kullanılmayan␣
 ↪kategoriler kaldırılır.


coffee_ratings_clst = coffee_ratings_cls.\
groupby('country_of_origin').\
sample(n=len(coffee_ratings_top) // 6)
#coffee_ratings_cls veri çerçevesi, country_of_origin sütununa göre gruplanır.
```

```
#Her grup için coffee_ratings_top veri çerçevesinin toplam satır sayısının 1/
 ↪6'ı kadar rastgele örnekleme yapılır.

coffee_ratings_clst.shape
```

[ ]:
```
coffee_ratings_top["total_cup_points"].mean() #popülasyon ortalaması
```

[ ]:
```
coffee_ratings_srs['total_cup_points'].mean() #simple random simple ortalamsı
```

[ ]:
```
coffee_ratings_strat['total_cup_points'].mean() # tabakalı örnekleme ,␣
 ↪starified sample mean
```

[ ]:
```
coffee_ratings_clst['total_cup_points'].mean() #kümeleme örnekleme cluster␣
 ↪sample mean
```

[ ]:
```
coffee_ratings_top.groupby("country_of_origin")["total_cup_points"].mean()
#Bu kod, coffee_ratings_top veri çerçevesindeki her ülke için total_cup_points␣
 ↪sütununun ortalamasını hesaplar.
#Sonuç, her ülkenin ortalama total_cup_points değerini gösteren bir seri␣
 ↪olacaktır.
```

[ ]:
```
coffee_ratings_clts.groupby("country_of_origin")["total_cup_points"].mean()
#coffee_ratings_clts veri çerçevesindeki her ülke için total_cup_points␣
 ↪sütununun ortalamasını hesaplar ve her ülkenin ortalama puanını döndürür.
```

[ ]:
```
print(len(df_coffee.sample(n=300)))#Veri çerçevesinden tam olarak 300 rastgele␣
 ↪satır seçilir.

print(len(df_coffee.sample(frac=0.25))) #Veri çerçevesinin %25'lik kısmı␣
 ↪rastgele seçilir.
```

[ ]:
```
df_coffee["total_cup_points"].mean() #kahvelerin ortalama fincan puanı
```

[ ]:
```
df_coffee.sample(n=10)["total_cup_points"].mean()
```

[ ]:
```
df_coffee.sample(n=100)["total_cup_points"].mean()
```

[ ]:
```
df_coffee.sample(n=1000)["total_cup_points"].mean()
```

örneklem sayısı arttırıldıkça relative error sayısı düşecektir daha büyük örneklem daha doğru sonuç verir

relative_error_percantage = 100 * abs(population_mean - sample_mean) / population_mean

[ ]:
```
import numpy as np
df_error= pd.DataFrame(columns=['sample_size','relative_error'])
pop_mean= df_coffee['total_cup_points'].mean()
```

```
for num_of_rows in range(1,len(df_coffee)+1):
    rel_err=np.abs(pop_mean-df_coffee.sample(n=num_of_rows)['total_cup_points'].
 ↪mean())/pop_mean
    df_error.loc[len(df_error.index)]= [num_of_rows,rel_err]
df_error.plot(x="sample_size",y="relative_error",kind='line');
```

df_error = pd.DataFrame(columns=[‘sample_size’, ‘relative_error’]) df_error adında boş bir DataFrame oluşturur, iki sütun içerir: sample_size ve relative_error.

pop_mean = df_coffee[‘total_cup_points’].mean() df_coffee veri çerçevesindeki total_cup_points sütununun ortalamasını hesaplar ve pop_mean değişkenine atar.

for num_of_rows in range(1, len(df_coffee) + 1): 1’den df_coffee’nin toplam satır sayısına kadar her bir satır sayısı için döngü başlatır.

rel_err = np.abs(pop_mean - df_coffee.sample(n=num_of_rows)[‘total_cup_points’].mean()) / pop_mean num_of_rows kadar rastgele seçilen satırdan total_cup_points ortalamasını hesaplar ve bununla pop_mean arasındaki bağıl hatayı hesaplar.

df_error.loc[len(df_error.index)] = [num_of_rows, rel_err] Hesaplanan sample_size (satır sayısı) ve relative_error değerlerini df_error DataFrame’ine ekler.

df_error.plot(x=“sample_size”, y=“relative_error”, kind=‘line’) df_error veri çerçevesini, sample_size (örneklem büyüklüğü) ve relative_error (bağıl hata) değerlerini doğrusal bir grafik olarak çizer.

[ ]:
```
print(df_coffee.sample(n=30)['total_cup_points'].mean())
print(df_coffee.sample(n=30)['total_cup_points'].mean())
print(df_coffee.sample(n=30)['total_cup_points'].mean())
print(df_coffee.sample(n=30)['total_cup_points'].mean())
```

[ ]:
```
mean_cup_points= []
for i in range(1000):
    mean_cup_points.append(df_coffee.sample(n=30)['total_cup_points'].mean())
mean_cup_points
```

[ ]:
```
import matplotlib.pyplot as plt
plt.hist(mean_cup_points, bins=30)
#yukarıdaki rastgele seçtiğimiz değerlerin kutu grafiği
```

[ ]:
```
mean_cup_points_6= []
for i in range(1000):
    mean_cup_points_6.append(df_coffee.sample(n=6)['total_cup_points'].mean())#␣
 ↪populasyondan 6 örnek yapmışız.
plt.hist(mean_cup_points_6, bins=30)
# df_coffee veri çerçevesinden 1000 kez 6 rastgele örnek alır ve her örneğin␣
 ↪total_cup_points ortalamasını hesaplar.
#Son olarak, bu ortalamaların dağılımını gösteren bir histogram çizer.
```

[ ]:
```
mean_cup_points_30= []
for i in range(1000):
    mean_cup_points_30.append(df_coffee.sample(n=30)['total_cup_points'].
 ↪mean())# populasyondan 30 örnek yapmışız.
plt.hist(mean_cup_points_30, bins=30)
#df_coffee veri çerçevesinden 1000 kez 30 rastgele örnek alır ve her örneğin␣
 ↪total_cup_points ortalamasını hesaplar.
#Son olarak, bu ortalamaların dağılımını gösteren bir histogram çizer.
```

[ ]:
```
mean_cup_points_150= []
for i in range(1000):
    mean_cup_points_150.append(df_coffee.sample(n=150)['total_cup_points'].
 ↪mean())# populasyondan 150 örnek yapmışız.
plt.hist(mean_cup_points_150, bins=30)
#df_coffee veri çerçevesinden 1000 kez 150 rastgele örnek alır ve her örneğin␣
 ↪total_cup_points ortalamasını hesaplar.
#Son olarak, bu ortalamaların dağılımını gösteren bir histogram çizer.
```

örneklem sayısı arttıkça grafiğin daraldığı gözüküyor

[ ]:
```
#Elimizdeki 6 yüzlü 4 zarın tüm kombinasyonları
import itertools as it
dice={
    "dice1":[1,2,3,4,5,6],
    "dice2":[1,2,3,4,5,6],
    "dice3":[1,2,3,4,5,6],
    "dice4":[1,2,3,4,5,6]

}
result = pd.DataFrame(it.product(*dice.values()),columns=dice.keys())
result
#olası 1296 kombinasyonu getirdi
```

[ ]:
```
result["mean_roll"] = (result.dice1 +result.dice2+ result.dice3+ result.dice4)/4
result
#4 zarın ortalamasını alan bir mean_roll adlı sütün ekledik
```

[ ]:
```
result["mean_roll"] = result["mean_roll"].astype("category") # astype tür␣
 ↪dönüşümü yapmamızı sağlar
result["mean_roll"].value_counts(sort=False).plot(kind="bar")
#mean_roll u kategoriye çevirdik ardındanda bar grafiğiyle görselleştirdik
```

[ ]:
```
n_dice = list(range(1, 101))
n_outcomes = []

for n in n_dice:
    n_outcomes.append(6**n)
```

```
outcomes = pd.DataFrame({
    'n_dice': n_dice,
    'n_outcomes': n_outcomes
})

outcomes.plot(x='n_dice', y='n_outcomes', kind='scatter');
#u kod, 1'den 100'e kadar farklı zar sayıları için olasılık sonuçlarını (6^n)␣
 ↪hesaplar ve bunları bir scatter plot olarak görselleştirir.
#x ekseninde zar sayısı, y ekseninde olasılık sonuçları bulunur.
```

[ ]:
```
#4 zar atışının ortalaması simülasyonu
sample_mean_1000=[]
for i in range(1000):
    sample_mean_1000.append(np.random.
 ↪choice(list(range(1,7)),size=4,replace=True).mean())
sample_mean_1000
#Bu kod, 1'den 6'ya kadar olan sayılardan 4 rastgele değer seçip her seferinde␣
 ↪ortalamalarını hesaplar.
#Bu işlemi 1000 kez tekrarlar ve tüm ortalamaları sample_mean_1000 listesine␣
 ↪ekler.
```

Yaklaşık Örnekleme Dağılımı - Approximite Sampling Distribution

[ ]:
```
plt.hist(sample_mean_1000,bins=20);
```

Standart Hatalar ve Merkezi Limit Teoremi

[ ]:
```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(2, 2, figsize=(10, 8))

# Sample Size : 5
mean_cup_points_5 = []

for i in range(5000):
    mean_cup_points_5.append(df_coffee.sample(n=5)['total_cup_points'].mean())

axs[0, 0].hist(mean_cup_points_5)
axs[0, 0].set_title('Sample Size : 5')
axs[0, 0].grid(True)


# Sample Size : 20
```

```
mean_cup_points_20 = []

for i in range(5000):
    mean_cup_points_20.append(df_coffee.sample(n=20)['total_cup_points'].mean())

axs[0, 1].hist(mean_cup_points_20)
axs[0, 1].set_title('Sample Size : 20')
axs[0, 1].grid(True)

# Sample Size : 80
mean_cup_points_80 = []

for i in range(5000):
    mean_cup_points_80.append(df_coffee.sample(n=80)['total_cup_points'].mean())

axs[1, 0].hist(mean_cup_points_80)
axs[1, 0].set_title('Sample Size : 80')
axs[1, 0].grid(True)

# Sample Size : 320
mean_cup_points_320 = []

for i in range(5000):
    mean_cup_points_320.append(df_coffee.sample(n=320)['total_cup_points'].
 ↪mean())

axs[1, 1].hist(mean_cup_points_320)
axs[1, 1].set_title('Sample Size : 320')
axs[1, 1].grid(True)

#df_coffee veri çerçevesinden farklı örneklem büyüklükleri (5, 20, 80, 320) ile␣
 ↪5000 rastgele örnek alır ve
#her örneğin total_cup_points ortalamasını hesaplar.
#Her bir örneklem büyüklüğü için bir histogram oluşturur ve bunları 2x2␣
 ↪düzeninde birleştirir.
#yani bir grafiğin içerisine birden fazla grafik çizebiliriz bunu subplotla␣
 ↪yaparız.


#örneklem arttıkça ortalamaya yaklaşıyor ve genişlik artıyor
```

[ ]:
```
print(f"Population Standart Devilation:{df_coffee['total_cup_points'].
 ↪std(ddof=0)}")
print(f"5 sample Standart Devilation:{np.std(mean_cup_points_5,ddof=1)}")
print(f"20 sample Standart Devilation:{np.std(mean_cup_points_20,ddof=1)}")
print(f"80 sample Standart Devilation:{np.std(mean_cup_points_80,ddof=1)}")
print(f"320 sample Standart Devilation:{np.std(mean_cup_points_320,ddof=1)}")
```

Population Standart Sapması: df_coffee[‘total_cup_points’] sütunundaki tüm verilerin standart sapması hesaplanır. 5, 20, 80, 320 Örneklem Standart Sapmaları: Farklı örneklem büyüklükleri (5, 20, 80, 320) için örneklem ortalamalarının standart sapması hesaplanır. ddof=1 kullanımı, örneklem için bütünleşik standart sapma (bölme işlemi n-1 ile yapılır) hesaplar.

[ ]:
```
#Bir Tablo nasıl oluşturulur kodu
data = [
    [5, np.std(mean_cup_points_5, ddof=1), f"{df_coffee['total_cup_points'].
 ↪std(ddof=0)} / sqrt(5)",
     df_coffee['total_cup_points'].std(ddof=0) / np.sqrt(5)],
    [20, np.std(mean_cup_points_20, ddof=1), f"{df_coffee['total_cup_points'].
 ↪std(ddof=0)} / sqrt(20)",
    df_coffee['total_cup_points'].std(ddof=0) / np.sqrt(20)],
    [80, np.std(mean_cup_points_80, ddof=1), f"{df_coffee['total_cup_points'].
 ↪std(ddof=0)} / sqrt(80)",
    df_coffee['total_cup_points'].std(ddof=0) / np.sqrt(80)],
    [320, np.std(mean_cup_points_320, ddof=1), f"{df_coffee['total_cup_points'].
 ↪std(ddof=0)} / sqrt(320)",
    df_coffee['total_cup_points'].std(ddof=0) / np.sqrt(320)]
]
column_labels = ["Sample Size", "Std dev sample mean", "Calculation", "Result"]

# Boş bir figür oluştur
fig, ax = plt.subplots(figsize=(10, 8))

# Tabloyu oluştur
ax.axis('tight')
ax.axis('off')
ax.table(cellText=data, colLabels=column_labels, cellLoc='center',␣
 ↪loc='center');
```

Bu kod, farklı örneklem büyüklükleri için **standart sapma** hesaplamalarını ve formülleri bir **tablo** şeklinde görselleştirir. Adımlar şu şekilde:

1. `data` listesi, her örneklem büyüklüğü (5, 20, 80, 320) için:
   - **Örneklem standart sapması**,
   - **Hesaplama formülü** (popülasyon standart sapması / √n),
   - **Sonuç** (formülün hesaplanmış değeri) içerir.
2. `column_labels` listesi, tablodaki sütun başlıklarını belirler: **“Sample Size”, “Std dev sample mean”, “Calculation”, “Result”**.
3. `ax.table()` ile bu veriler bir tabloya dönüştürülür ve **figürde** gösterilir. Tablo, eksenler gizlenerek ortalanır.

Sonuç, standart sapma hesaplamaları ve ilgili formüllerin görsel olarak sunulduğu bir tablodur.

3.1 Örneklem Ortalamasının Standart Sapması: Standart Hata (Standard Error)

Örneklem büyüklüğünün örnekleme dağılımının standart sapması üzerindeki etkisini gördük. Örnekleme dağılımının bu standart sapmasının özel bir adı vardır: **standart hata**. Popülasyon standart sapmasını tahmin etmekten, örnekleme sürecinden ne düzeyde değişkenlik bekleyeceğimize dair beklentiler belirlemeye kadar çeşitli bağlamlarda faydalıdır.

Standart hata (SE), bir tahmindeki değişkenliği veya belirsizliği, tipik olarak ortalama gibi bir popülasyon parametresinin tahminini niceleyen istatistiksel bir ölçüdür. Farklı örnekler çekilirse, örnek ortalamasının (veya başka bir istatistiğin) gerçek popülasyon ortalamasından ne kadar farklılaşmasının beklendiğine dair bir gösterge sağlar.

3.1.1 Ortalamanın Standart Hatası (SEM):

Standart hatanın en yaygın kullanımı ortalama bağlamındadır. **Ortalamanın standart hatası (SEM)**, örnek ortalamasının popülasyon ortalamasının bir tahmini olarak kesinliğini niceliksel olarak ifade eder. Örnek ortalamasının popülasyon ortalamasından ne kadar farklılaşma olasılığının olduğunu söyler.

Bu formül, **standart hatanın örneklem büyüklüğü arttıkça azaldığını**, yani daha büyük örneklerin popülasyon ortalamasının daha kesin tahminlerini verdiğini göstermektedir.

3.1.2 Standart Sapmadan Farkı:

Standart sapma, tek bir örneklemdeki veri noktalarının yayılımını ölçerken, **standart hata**, popülasyondan alınan farklı örneklerdeki örnek ortalamalarının değişkenliğini ölçer. Standart sapma, bireysel veri noktalarının ne kadar yayıldığını söylerken, standart hata, örnek istatistiğinizin gerçek popülasyon istatistiğinden ne kadar sapma olasılığının olduğunu söyler.

3.1.3 Yorumlama:

- Daha küçük bir **standart hata**, örnek ortalamasının gerçek popülasyon ortalamasına daha yakın olma olasılığını gösterir.
- Daha büyük bir **standart hata**, daha fazla değişkenlik olduğunu, yani örnek ortalamasının popülasyon ortalamasından daha uzak olabileceğini gösterir.

[ ]:

fi9imqew9

January 7, 2025

[ ]:
```
import pandas as pd
```

[ ]:
```
df_coffee = pd.read_feather("Data/coffee_ratings_full.feather")
df_coffee
```

[ ]:
```
coffee_focus = df_coffee[['variety','country_of_origin','flavor']]
coffee_focus = coffee_focus.reset_index()
coffee_focus
#df_coffee veriçerçevesinden sadece variety,country_of_origin,veflavor␣
 ↪sütunlarını seçerek coffee_focus adında yeni bir veri  oluşturur.
#Ardından, bu veri çerçevesinin indeksini sıfırlar ve sonucu görüntüler.
```

[ ]:
```
coffee_resamp = coffee_focus.sample(frac=1 , replace = True)
coffee_resamp
 #offee_focus veri çerçevesinden yeniden örnekleme (resampling) yapar. frac=1␣
 ↪ile tüm veriyi seçip,
#replace=True sayesinde tekrarlı örnekleme yapılır. Sonuç, yeni bir örneklenmiş␣
 ↪veri çerçevesidir.
```

[ ]:
```
coffee_resamp["index"].value_counts() # her bir veriden kaç tane olduğunu␣
 ↪gösterir index koyarak her bir veriyi unique olarak işaretledik
```

[ ]:
```
num_unique_coffees = len(coffee_resamp.drop_duplicates(subset="index"))
num_unique_coffees
#coffee_resamp veri çerçevesindeki tekrarsız (benzersiz) satırların sayısını,␣
 ↪yalnızca index sütununu dikkate alarak hesaplar ve
#bunu num_unique_coffees değişkenine atar.
```

[ ]:
```
#BootStrapping detaylı
import numpy as np
```

[ ]:
```
mean_flavors_1000 = []
for i in range(1000):
    mean_flavors_1000.append(np.mean(coffee_focus.
 ↪sample(frac=1,replace=True)["flavor"]))

mean_flavors_1000
```

[ ]:
```
import matplotlib.pyplot as plt
plt.hist(mean_flavors_1000)

#BootStrap distribution histogram
#işte örnek ortalamasının önyükleme dağılımından bir histogram grafiği
#normal dağılıma yakın olduğuna dikkat edilir
```

[ ]:
```
coffee_sample = df_coffee[['variety','country_of_origin','flavor']].
 ↪reset_index().sample(n=500)
#kahve veri setinden 500 tane veri aldık replace default olarak false geldi
```

[ ]:
```
mean_flavors_500 = []
for i in range(500):
    mean_flavors_500.append(np.mean(coffee_sample.sample(frac=1 ,␣
 ↪replace=True)['flavor']))
```

[ ]:
```
plt.hist(mean_flavors_500)
```

[ ]:
```
coffee_sample['flavor'].mean() #sample mean
```

[ ]:
```
np.mean(mean_flavors_500)
```

[ ]:
```
df_coffee['flavor'].mean() # bu iki ortalama birbirine çok yakın fakat␣
 ↪popülasyon ortalamsı farklıdır
```

[ ]:
```
coffee_sample['flavor'].std()
```

[ ]:
```
np.std(mean_flavors_500,ddof=1) * np.sqrt(500)
#np.std(mean_flavors_500, ddof=1): 500 örnekten hesaplanan ortalamaların␣
 ↪standart sapmasını bulur.
#np.sqrt(500): 500 örneklem boyutunun karekökünü alır.
#Çarpım, örneklem ortalamalarından popülasyon sapması tahmini verir.
```

[ ]:
```
#Sample, bootstrap dist'n, pop Standart Deviations
#Confidence İntervals
#Predicting the weather
```

[ ]:
```
#Bootstrap Distribution of mean flavor
plt.hist(mean_flavors_500)
```

[ ]:
```
plt.hist(mean_flavors_500)
plt.axvline(np.mean(mean_flavors_500), color = 'red',␣
 ↪linestyle='dashed',linewidth=2,label=f'Mean : {np.mean(mean_flavors_500):.
 ↪2f}')
plt.legend()
# mean_flavors_500 verisinin histogramını çizer ve:
#Kırmızı kesikli çizgiyle örneklem ortalamasını gösterir.
```

```
#Çizginin üzerine, ortalama değeri içeren bir etiket (legend) ekler.
```

[ ]:
```
mean = np.mean(mean_flavors_500)
plus_one_std = mean + np.std(mean_flavors_500,ddof=1)
minus_one_std = mean - np.std(mean_flavors_500,ddof=1)

plt.hist(mean_flavors_500)
plt.axvline(mean,color='black',linestyle='dashed',linewidth=2,label=f'Mean :␣
 ↪{mean:.2f}')
plt.
 ↪axvline(plus_one_std,color='red',linestyle='dashed',linewidth=2,label=f'mean+std␣
 ↪: {plus_one_std:.2f}')
plt.
 ↪axvline(minus_one_std,color='yellow',linestyle='dashed',linewidth=2,label=f'mean-std␣
 ↪: {minus_one_std:.2f}')
plt.legend()
```

0.0.1 Ortalama ve Standart Sapma Gösterimi

Bu grafik, **mean_flavors_500** verisinin histogramını çizer ve şu değerleri vurgular:

- **Ortalama (mean)**: Siyah kesikli çizgiyle gösterilir.
- **Ortalama + Standart Sapma (mean + std)**: Kırmızı kesikli çizgiyle gösterilir.
- **Ortalama - Standart Sapma (mean - std)**: Sarı kesikli çizgiyle gösterilir.

Çizgilere açıklama eklemek için bir **legend** kullanılmıştır.

0.0.2 Quantile method for confidence intervals

[ ]:
```
quantile_lower = np.quantile(mean_flavors_500,0.025)
quantile_upper = np.quantile(mean_flavors_500,0.975)

plt.hist(mean_flavors_500)
plt.axvline(mean,color='black',linestyle='dashed',linewidth=2,label=f'Mean :␣
 ↪{mean:.2f}')
plt.
 ↪axvline(quantile_lower,color='red',linestyle='dashed',linewidth=2,label=f'quantile_lower␣
 ↪: {quantile_lower:.2f}')
plt.
 ↪axvline(quantile_upper,color='yellow',linestyle='dashed',linewidth=2,label=f'quantile_upper
 ↪: {quantile_upper:.2f}')
plt.legend()
```

0.0.3 Kuantiller ve Ortalama Gösterimi

Bu grafik, **mean_flavors_500** verisinin histogramını çizer ve şu değerleri vurgular:

- **Ortalama (mean)**: Siyah kesikli çizgiyle gösterilir.
- **Alt %2.5 Kuantili (quantile_lower)**: Kırmızı kesikli çizgiyle gösterilir.
- **Üst %2.5 Kuantili (quantile_upper)**: Sarı kesikli çizgiyle gösterilir.

Bu değerler, dağılımın özelliklerini görselleştirmek ve uç değerleri anlamak için kullanılır. Çizgilere açıklama eklemek için bir **legend** kullanılmıştır.

0.0.4 Inverse cumulative distribution function

[ ]:
```
#Pdf in integrali cdf elde etmemizi sağlar. cdf nin x ve y lerini değiştirerek bunu sağlar
```

[ ]:
```
from scipy.stats import norm
p_values = np.linspace(0,1,100)
quantiles = norm.ppf(p_values,lox=0,scale=1)
plt.plot(p_values,quantiles,label='Inverse CDF(Normal Distribution)')

p1, p2 = 0.025,0.0975
quantile_025 = norm.ppf(p1)
quantile_975 = norm.ppf(p2)

plt.scatter([p1,p2],[quantile_025,quantile_975],color='red')
plt.axvline(p1,color='red',linestyle='dashed',linewidth=1)
plt.axvline(p2,color='red',linestyle='dashed',linewidth=1)

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')
```

[ ]:
```

```

4iio84p9k

January 7, 2025

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
df_stck = pd.read_feather("Data/stack_overflow.feather")
df_stck
```

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

[ ]:
```
prop_child_sample = (df_stck["age_first_code_cut"]=="child").mean()
prop_child_sample
```

[ ]:
```
prop_child_hyp=0.35
```

[ ]:
```
std_err = np.std(first_code_boot_disrn,ddof=1)
std_err
```

[ ]:
```
z_score = (prop_child_sample -prop_child_hyp)/std_err
z_score
```

[ ]:
```
from scipy.stats import norm
1-norm.cdf(z_score,loc=0,scale=1)
```

[ ]:
```
alpha = 0.05
prop_child_samp = (df_stck['age_first_code_cut'] == "child").mean()
prop_child_hyp=0.35
std_err = np.std(first_code_boot_disrn,ddof=1)
z_score = (prop_child_samp - prop_child_hyp) / std_err

p_value = 1- norm.cdf(z_score,loc=0,scale=1)
```

```
p_value
```

[ ]:
```
lower = np.quantile(first_code_boot_disrn,0.025)
upper = np.quantile(first_code_boot_disrn,0.975)
(lower,upper)
```

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

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

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

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

[ ]:
```
df_stck.groupby('age_first_code_cut')['converted_comp'].
 ↪agg(['mean','std','count'])
```

[ ]:
```
pay = xbar.iloc[1]-xbar.iloc[0]
payda = np.sqrt(s.iloc[1]**2 / n.iloc[1] + s.iloc[0]**2 / n.iloc[0])
t_stats = pay/payda
t_stats
```

0.0.1 t-Statistiği Hesaplama

Bu kod, iki grup arasındaki farkın anlamlılığını değerlendiren **t-istatistiği** hesaplamaktadır.
- **pay**: İki grup arasındaki ortalama farkı temsil eder.
- **payda**: Her iki grubun standart sapmalarının karelerinin, örneklem büyüklükleriyle normalleştirilmiş toplamıdır.
- **t_stats**: Ortalama farkın standart hataya bölünmesiyle elde edilen **t-istatistiği** değeridir.

Sonuç olarak, **t_stats** değişkeni, iki grup arasındaki farkın anlamlı olup olmadığını değerlendiren t-istatistiği değerini döndürür.

[ ]:
```
import scipy.stats as stats
x = np.linspace(-4,4,100)
plt.plot(x,stats.norm.pdf(x),label='Normal',linestyle='--',color='blue')
plt.plot(x,stats.t.pdf(x,1),label='t:df=1',color='yellow')
plt.legend()
plt.xlabel('x')
plt.ylabel('PDF(x)')
```

0.0.2 Kod Açıklamaları

1. **`import scipy.stats as stats`**

- **scipy.stats** kütüphanesini **stats** olarak içe aktarır. Bu kütüphane, istatistiksel hesaplamalar ve dağılımlar için birçok fonksiyon sağlar.

2. `x = np.linspace(-4,4,100)`
   - **x** değişkeni, **-4** ile **4** arasında eşit aralıklarla 100 değer üretir. Bu, grafik üzerinde kullanılacak **x** ekseninin değerlerini oluşturur.
3. `plt.plot(x, stats.norm.pdf(x), label='Normal', linestyle='--', color='blue')`
   - **stats.norm.pdf(x)**, **x** üzerindeki normal dağılımın olasılık yoğunluk fonksiyonunu (PDF) hesaplar.
   - **plt.plot()** fonksiyonu, **x** değerleri ile normal dağılımın PDF'sini çizer.
   - **label='Normal'**: Grafikte bu çizgi için etiket belirler.
   - **linestyle='–'**: Çizgiyi kesikli yapar.
   - **color='blue'**: Çizginin rengini mavi yapar.
4. `plt.plot(x, stats.t.pdf(x, 1), label='t:df=1', color='yellow')`
   - **stats.t.pdf(x, 1)**, **x** üzerindeki **t-dağılımı** için PDF'yi hesaplar, burada **df=1** serbestlik derecesi olarak belirtilmiştir.
   - **label='t:df=1'**: Grafikte bu çizgi için etiket belirler.
   - **color='yellow'**: Çizginin rengini sarı yapar.
5. `plt.legend()`
   - **plt.legend()**, çizilen grafiklerde etiketlerin gösterilmesini sağlar. Burada, normal dağılım ve t-dağılımı için etiketler görüntülenir.
6. `plt.xlabel('x')`
   - **plt.xlabel('x')**, **x** eksenine "x" etiketini ekler.
7. `plt.ylabel('PDF(x)')`
   - **plt.ylabel('PDF(x)')**, **y** eksenine "PDF(x)" etiketini ekler, bu da olasılık yoğunluk fonksiyonunu temsil eder.

[ ]:
```
x = np.linspace(-4,4,100)

for df in [1,2,4,8]:
    plt.plot(x,stats.t.pdf(x,df),label=f'df={df}')

plt.plot(x,stats.norm.pdf(x),label='Normal',linestyle='--')
plt.legend()
plt.title('t-distribution with different degrees of freedom')
plt.xlabel('x')
plt.ylabel('PDF(x)')
plt.show()
```

0.0.3 Kod Açıklamaları

1. `x = np.linspace(-4,4,100)`
   - **x** değişkeni, **-4** ile **4** arasında eşit aralıklarla 100 değer üretir. Bu, grafik üzerinde kullanılacak **x** ekseninin değerlerini oluşturur.
2. `for df in [1, 2, 4, 8]:`
   - Bu döngü, **df** (serbestlik derecesi) için sırasıyla **1**, **2**, **4** ve **8** değerlerini kullanarak işlemi

tekrarlar. Bu, t-dağılımının farklı serbestlik derecelerine göre grafik çizimini sağlar.

3. `plt.plot(x, stats.t.pdf(x, df), label=f'df={df}')`
   - **stats.t.pdf(x, df)**, her bir **df** değeri için **x** üzerindeki **t-dağılımı** için olasılık yoğunluk fonksiyonunu (PDF) hesaplar.
   - **plt.plot(x, ...)**, **x** ve t-dağılımı PDF'sini çizer.
   - **label=f'df={df}'**: Grafikte bu çizgi için etiket olarak, serbestlik derecesini dinamik olarak ekler. Örneğin, df=1, df=2 vb.
4. `plt.plot(x, stats.norm.pdf(x), label='Normal', linestyle='--')`
   - **stats.norm.pdf(x)**, **x** üzerindeki normal dağılımın olasılık yoğunluk fonksiyonunu hesaplar.
   - **plt.plot(x, ...)**, normal dağılımı çizer.
   - **label='Normal'**: Grafikte bu çizgi için etiket belirler.
   - **linestyle='–'**: Çizgiyi kesikli yapar, normal dağılımı diğerlerinden ayırır.
5. `plt.legend()`
   - **plt.legend()**, çizilen grafiklerde etiketlerin gösterilmesini sağlar. Burada, t-dağılımı için farklı serbestlik dereceleri ve normal dağılımın etiketleri görüntülenir.
6. `plt.title('t-distribution with different degrees of freedom')`
   - **plt.title('t-distribution with different degrees of freedom')**, grafiğe başlık ekler. Burada başlık, "t-dağılımı farklı serbestlik dereceleri ile" olarak belirtilmiştir.
7. `plt.xlabel('x')`
   - **plt.xlabel('x')**, **x** eksenine "x" etiketini ekler.
8. `plt.ylabel('PDF(x)')`
   - **plt.ylabel('PDF(x)')**, **y** eksenine "PDF(x)" etiketini ekler, bu da olasılık yoğunluk fonksiyonunu temsil eder.
9. `plt.show()`
   - **plt.show()**, grafiği ekranda görüntüler. Grafik, belirtilen t-dağılımı ve normal dağılım ile birlikte, her bir serbestlik derecesine ait t-dağılımının farklı görünümlerini gösterir.

[ ]:
```
# Degrees of freedom hesaplaması
# Bu satırda, t-dağılımı için serbestlik derecesi (df) hesaplanmaktadır.
# Serbestlik derecesi, her iki grubun örneklem büyüklüklerinin toplamından 2 çıkarılarak bulunur.
# Bu formül, bağımsız iki örneklem için geçerlidir.

degrees_of_freedom = n.iloc[0] + n.iloc[1] - 2  # İlk ve ikinci grubun örneklem büyüklüklerinin toplamı - 2
degrees_of_freedom  # Serbestlik derecesini döndürür
```

[ ]:
```
# t-dağılımının p-değerini hesaplamak
# Bu satırda, hesaplanan t-istatistiği (t_stats) kullanılarak t-dağılımının
# p-değeri hesaplanmaktadır. `t.cdf()` fonksiyonu, t-dağılımının kümülatif dağılım fonksiyonunu
# hesaplar ve bu fonksiyonun tersini almak için 1 - cdf değeri alınır.
# df = degrees_of_freedom, t-dağılımının serbestlik derecesini belirtir.
```

```
1 - t.cdf(t_stats, df=degrees_of_freedom)  # t-istatistiği için sağ kuyruk␣
 ↪p-değerini hesaplar
```

[ ]:
```
df_election = pd.read_feather('Data/repub_votes_potus_08_12.feather')
df_election
```

[ ]:
```
# Yeni bir 'diff' sütunu oluşturma
# Bu satırda, 'sample_data' adlı yeni bir veri çerçevesi oluşturuluyor ve␣
 ↪'repub_percent_08' ile
# 'repub_percent_12' sütunları arasındaki fark hesaplanarak 'diff' adlı yeni␣
 ↪bir sütun ekleniyor.
# Bu fark, 2008 seçimlerinde Cumhuriyetçi partinin aldığı yüzde ile 2012␣
 ↪seçimlerinde aldığı
# yüzdeler arasındaki farkı gösterir.

sample_data = df_election.copy()  # df_election veri çerçevesinin bir kopyasını␣
 ↪oluştur
sample_data['diff'] = sample_data['repub_percent_08'] -␣
 ↪sample_data['repub_percent_12']  # 'diff' sütunu oluştur
sample_data  # Yeni veri çerçevesini görüntüler
```

[ ]:
```
# 'diff' sütununun histogramını çizer.
# 'sample_data['diff']' verisindeki değerlerin dağılımını görsel olarak␣
 ↪gösterir.
# 'bins=20', histogramda 20 kutu (bin) kullanarak veriyi gruplar.
# Bu, verinin daha detaylı bir şekilde görselleştirilmesini sağlar.

plt.hist(sample_data['diff'], bins=20)
```

[ ]:
```
x_bar_diff = sample_data['diff'].mean()
x_bar_diff
```

[ ]:
```
n_diff = len(sample_data)
s_diff = sample_data['diff'].std()
t_stat = (x_bar_diff - 0)/np.sqrt(s_diff**2/n_diff)
t_stat
```

[ ]:
```
degrees_of_freedom = n_diff-1
p_value = t.cdf(t_stat,df=degrees_of_freedom)
p_value
```

1. **`x_bar_diff = sample_data['diff'].mean()`**
   - Bu satır, **'diff'** sütunundaki değerlerin ortalamasını hesaplar.
2. **`n_diff = len(sample_data)`**
   - Bu satır, **sample_data** veri çerçevesindeki toplam gözlem sayısını hesaplar ve bunu **n_diff** değişkenine atar. Bu, örneklem büyüklüğüdür.

3. `s_diff = sample_data['diff'].std()`
   - Bu satır, **'diff'** sütunundaki değerlerin standart sapmasını hesaplar ve bunu **s_diff** değişkenine atar.
4. `t_stat = (x_bar_diff - 0)/np.sqrt(s_diff**2/n_diff)`
   - Bu satır, **t-istatistiği** hesaplar.
     - **x_bar_diff - 0**: Ortalama fark ile sıfır arasındaki farkı alır (burada sıfır, testin hipotez edilen değeri).
     - **np.sqrt(s_diff**2 / n_diff)**: Standart hata, standart sapmanın örneklem büyüklüğüne bölünmesiyle hesaplanır.
     - Sonuç, **t_stat** değişkeninde saklanır.
5. `degrees_of_freedom = n_diff - 1`
   - Bu satırda, serbestlik derecesi hesaplanır. Bağımsız örneklemler için serbestlik derecesi, **n_diff - 1** şeklinde hesaplanır.
6. `p_value = t.cdf(t_stat, df=degrees_of_freedom)`
   - Bu satırda, **t-dağılımının kümülatif dağılım fonksiyonu** (**cdf**) kullanılarak **p-değeri** hesaplanır. **t_stat**, t-istatistiği ve **degrees_of_freedom**, serbestlik derecesidir. Bu p-değeri, istatistiksel anlamlılık testini gerçekleştirmek için kullanılır.
7. `p_value`
   - Bu satır, hesaplanan **p-değerini** döndürür. Bu değer, t-testinin sonucunu ve hipotezin test edilip edilmediğini belirler.

[ ]:

ea5xrfigf

January 7, 2025

[ ]:
```
import pandas as pd

df_stck = pd.read_feather('Data/stack_overflow.feather')
df_stck['job_sat'].value_counts()
```

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# 'df_stck' veri çerçevesindeki 'converted_comp' ve 'job_sat' sütunları␣
 ↪arasındaki dağılımı görselleştiren bir boxplot çizer.
# 'converted_comp' x eksenine, 'job_sat' y eksenine yerleştirilir.
sns.boxplot(x='converted_comp', y='job_sat', data=df_stck)
```

[ ]:
```
import pingouin
# 'df_stck' veri çerçevesindeki 'converted_comp' ve 'job_sat' sütunları␣
 ↪arasında tek yönlü ANOVA testi yapar.
# 'converted_comp' bağımlı değişken (dv) olarak kullanılırken, 'job_sat'␣
 ↪bağımsız değişken (gruplama) olarak belirlenir.
pingouin.anova(data=df_stck, dv='converted_comp', between='job_sat')
```

[ ]:
```
# 'df_stck' veri çerçevesindeki 'converted_comp' ve 'job_sat' sütunları␣
 ↪arasında ikili karşılaştırmalar yapar.
# 'padjust='none'', p-değerlerinin herhangi bir düzeltme yapılmadan␣
 ↪kullanılacağını belirtir.
pingouin.pairwise_tests(data=df_stck, dv='converted_comp', between='job_sat',␣
 ↪padjust='none')
```

[ ]:
```
# 'df_stck' veri çerçevesindeki 'converted_comp' ve 'job_sat' sütunları␣
 ↪arasında ikili karşılaştırmalar yapar.
# 'padjust='bonf'', Bonferroni düzeltmesini kullanarak p-değerlerini düzeltir.␣
 ↪Bu, çoklu karşılaştırmalarda
# yanlış pozitif sonuçları azaltmayı amaçlar.
pingouin.pairwise_tests(data=df_stck, dv='converted_comp', between='job_sat',␣
 ↪padjust='bonf')
```

[ ]:
```
alpha = 0.01
```

```
# 'df_stck' veri çerçevesindeki 'age_cat' sütununun kategorik değerlerinin yüzdelik dağılımını hesaplar.
# 'normalize=True', her bir kategori için gözlem sayısının toplam gözlem sayısına oranını döndürür,
# yani her bir kategorinin oranını verir.

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

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

[ ]:
```
p_0=0.5
```

[ ]:
```
n=len(df_stck)
n
```

[ ]:
```
import numpy as np
```

[ ]:
```
# 'p_hat' ile 'p_0' arasındaki farkı hesaplar. 'p_hat' örneklem oranını, 'p_0' ise hipotezdeki beklenen oranı temsil eder.
pay = p_hat - p_0

# Paydanın hesaplanmasında, 'p_0' beklenen oran, 'n' ise örneklem büyüklüğüdür.
# Bu formül, z-istatistiği hesaplamada kullanılan standart hatayı verir.
payda = np.sqrt(p_0 * (1 - p_0) / n)

# 'pay' ve 'payda' değerleri kullanılarak z-istatistiği hesaplanır.
z_score = pay / payda

# Hesaplanan z-istatistiği değerini döndürür.
z_score
```

[ ]:
```
from scipy.stats import norm
p_value = 2*(1-norm.cdf(z_score))
p_value

# 'norm.cdf(z_score)' komutu, z-istatistiği için kümülatif dağılım fonksiyonunu (CDF) hesaplar.
# '1 - norm.cdf(z_score)', z-istatistiği kadar veya daha uç bir değeri elde etme olasılığını verir.
# '2 * (1 - norm.cdf(z_score))', çift yönlü test için p-değerini hesaplar. Bu, z-istatistiği kadar veya daha uç bir değer olma olasılığının iki katıdır.
```

[ ]:
```
p_value < alpha
```

[ ]:
```
#h0 ı reddettik yani değerlerin %50 sinden fazlası 30 yaşın altındaymış
```

[ ]:
```
alpha = 0.05
```

[ ]:
```
p_hats = df_stck.groupby('age_cat')['hobbyist'].value_counts(normalize=True)
p_hats
```

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

[ ]:
```
p_hat_at_least_30 = p_hats[('At least 30','Yes')]
p_hat_at_under_30 = p_hats[('Under 30','Yes')]
(p_hat_at_least_30,p_hat_at_under_30)
```

[ ]:
```
n_at_least_30 = n['At least 30']
n_at_under_30 = n['Under 30']
(n_at_least_30,n_at_under_30)
```

[ ]:
```
# 'p_hat', iki grubun örneklem oranlarının ağırlıklı ortalamasını hesaplar.
# 'n_at_least_30' ve 'n_at_under_30' gruplarındaki örneklem büyüklükleri ile␣
 ↪'p_hat_at_least_30' ve 'p_hat_at_under_30' gruplarındaki örneklem oranları␣
 ↪kullanılarak, toplam örneklem oranı hesaplanır.
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', iki grubun örneklem oranlarının birleşik standart hatasını␣
 ↪hesaplar.
# Her iki grubun standart hataları hesaplanır ve bunlar toplam örneklem␣
 ↪büyüklüğü ile birleşir.
std_error = np.sqrt(p_hat * (1 - p_hat) / n_at_least_30 +
                    p_hat * (1 - p_hat) / n_at_under_30)

# 'z_score', gruplar arasındaki farkın standart hataya bölünerek z-istatistiği␣
 ↪hesaplanır.
# Bu, gruplar arasındaki oran farkının anlamlılığını test etmek için kullanılır.
z_score = (p_hat_at_least_30 - p_hat_at_under_30) / std_error

# Hesaplanan z-istatistiği değerini döndürür.
z_score
```

[ ]:
```
p_value = 2*(1-norm.cdf(z_score))
p_value
```

[ ]:
```
p_value < alpha
```

[ ]:
```
# 'n_hobbyist', her grup için gözlemlenen başarı sayısını temsil eder. İlk grup␣
 ↪için 812, ikinci grup için 1021.
n_hobbyist = np.array([812, 1021])

# 'n_rows', her grubun toplam örneklem büyüklüğünü temsil eder. İlk grup için␣
 ↪812 + 238, ikinci grup için 1021 + 190.
n_rows = np.array([812 + 238, 1021 + 190])

# 'proportions_ztest', iki oran arasındaki farkın anlamlı olup olmadığını test␣
 ↪etmek için z-testini uygular.
# 'count' parametresi, her grup için başarı sayısını (n_hobbyist), 'nobs'␣
 ↪parametresi ise toplam örneklem büyüklüklerini (n_rows) belirtir.
# 'alternative='two-sided'', iki yönlü test yapmayı belirtir.
from statsmodels.stats.proportion import proportions_ztest

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

# Hesaplanan z-istatistiği ve p-değerini döndürür.
(z_score, p_value)
```

[ ]:
```
# 'pingouin.chi2_independence', iki kategorik değişkenin bağımsız olup␣
 ↪olmadığını test etmek için Ki-kare bağımsızlık testini uygular.
# 'data' parametresi veri çerçevesini, 'x' ve 'y' parametreleri ise test␣
 ↪edilecek iki kategorik değişkeni belirtir.
# 'correction=False', Yates düzeltmesinin uygulanmayacağını belirtir (küçük␣
 ↪örneklem büyüklüklerinde sıklıkla kullanılır).
expected, observed, stats = pingouin.chi2_independence(data=df_stck,␣
 ↪x='hobbyist',
                                                       y='age_cat',
                                                       correction=False)

# 'stats', Ki-kare testi sonuçlarını (ki-kare istatistiği, p-değeri vb.) içeren␣
 ↪istatistiksel özet bilgilerini döndürür.
stats
```

[ ]:
```
# 'df_stck.groupby('job_sat')', 'job_sat' sütununa göre veri çerçevesini␣
 ↪gruplar.
# Ardından, her grup için 'age_cat' sütununun kategorik değerlerinin␣
 ↪normalleştirilmiş (yüzdelik) dağılımını hesaplar.
# 'value_counts(normalize=True)', her kategori için gözlem sayısının toplam␣
 ↪gözlem sayısına oranını döndürür.

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

```
# 'props', her iş tatmini grubundaki yaş kategorilerinin yüzdelik dağılımını␣
 ↪döndürür.
props
```

[ ]:
```
# 'props', bir Pandas Series nesnesidir ve gruplama sonucu elde edilen yaş␣
 ↪kategorilerinin oranlarını içerir.
# 'unstack()', çok seviyeli indeksleri sütunlara taşır ve veriyi geniş formatta␣
 ↪bir DataFrame'e dönüştürür.

wide_props = props.unstack()

# 'wide_props', 'job_sat' gruplarını satır, 'age_cat' kategorilerini sütun␣
 ↪olarak gösteren bir tablo döner.
# Bu, veriyi daha okunabilir ve analiz edilebilir bir formata dönüştürür.
wide_props
```

[ ]:
```
# 'wide_props.plot(kind='bar', stacked=True)' ifadesi, geniş formatlı␣
 ↪'wide_props' DataFrame'ini kullanarak bir çubuk grafiği oluşturur.
# 'kind='bar'' parametresi, grafik türünün çubuk grafiği olduğunu belirtir.
# 'stacked=True', çubukların üst üste yığılmasını sağlar, böylece her yaş␣
 ↪kategorisinin oranları aynı çubuk üzerinde gösterilir.

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

# Bu grafik, 'job_sat' grupları için yaş kategorilerinin (age_cat) oran␣
 ↪dağılımını karşılaştırmalı olarak görselleştirir.
```

[ ]:
```
# 'pingouin.chi2_independence', iki kategorik değişkenin bağımsız olup␣
 ↪olmadığını test etmek için Ki-kare bağımsızlık testini uygular.
# 'data=df_stck', veri çerçevesini belirtir.
# 'x='job_sat'', birinci kategorik değişkeni temsil eder.
# 'y='age_cat'', ikinci kategorik değişkeni temsil eder.
# Yates düzeltmesi varsayılan olarak 'correction=True' durumundadır.

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

# 'expected': Ki-kare testi için beklenen frekans tablosunu döner.
# 'observed': Gerçek gözlemlenen frekans tablosunu döner.
# 'stats': Test istatistiği, p-değeri ve etki büyüklüğü gibi istatistiksel␣
 ↪sonuçları içeren bir tablo döner.

stats
```

[ ]:
```
# 1. 'df_stck.groupby('age_cat')', veri çerçevesini 'age_cat' sütununa göre␣
 ↪gruplar.
# 2. 'job_sat'.value_counts(normalize=True)' her yaş kategorisi için 'job_sat'␣
 ↪değerlerinin yüzdelik dağılımını hesaplar.
props = df_stck.groupby('age_cat')['job_sat'].value_counts(normalize=True)

# 3. 'unstack()' işlemi, çok seviyeli indeksleri sütunlara taşıyarak veriyi␣
 ↪geniş formatta bir DataFrame'e dönüştürür.
wide_props = props.unstack()

# 4. 'plot(kind='bar', stacked=True)', geniş formatlı veriyi bir çubuk grafiği␣
 ↪olarak görselleştirir.
# 'stacked=True', çubukların üst üste yığılmasını sağlar ve toplamı 1'e eşit␣
 ↪yüzdelik oranları gösterir.
wide_props.plot(kind='bar', stacked=True)

# Bu grafik, her 'age_cat' (yaş kategorisi) için 'job_sat' (iş tatmini)␣
 ↪dağılımını karşılaştırmalı olarak görselleştirir.
```

[ ]:
```
# 1. 'pingouin.chi2_independence' fonksiyonu, iki kategorik değişken ('age_cat'␣
 ↪ve 'job_sat') arasındaki bağımsızlığı test eder.
# 'data=df_stck' → Veri çerçevesi olarak 'df_stck' kullanılır.
# 'x='age_cat'' → Birinci kategorik değişken.
# 'y='job_sat'' → İkinci kategorik değişken.

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

# 2. 'stats' değişkeni, test istatistiği, p-değeri ve etki büyüklüğü gibi␣
 ↪sonuçları içeren bir tablo döner.
# 'stats[stats['test'] == 'pearson']' → Sadece Pearson Ki-kare testine ait␣
 ↪satırları filtreler.

stats[stats['test'] == 'pearson']

# Çıktı: Pearson testi için hesaplanan istatistiksel değerler:
# - Ki-kare testi istatistiği (chi2),
# - p-değeri (pval),
# - Etki büyüklüğü (Cramer's V gibi ölçüler).
```

[ ]:
```
# 1. 'df_stck['purple_link']', veri çerçevesindeki 'purple_link' sütununu seçer.
# Bu sütunun içinde kategorik veya tekrarlı değerler bulunur.

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

```
# 2. 'value_counts()', 'purple_link' sütunundaki her benzersiz değerin sayısını␣
 ↪hesaplar.
# Sonuç, değerlerin frekanslarını içeren bir pandas Series olarak döner.

purple_link_counts

# Çıktı:
# - 'purple_link' sütunundaki her benzersiz değerin kaç kez tekrarlandığını␣
 ↪gösterir.
# - En yüksekten en düşüğe sıralı bir tablo döner.
```

[ ]:
```
purple_link_counts = purple_link_counts.rename_axis('purple_link').\
                                        reset_index(name='n').\
                                        sort_values('purple_link')

purple_link_counts
```

1. `purple_link_counts = purple_link_counts.rename_axis('purple_link')`
   - Bu satırda, `purple_link_counts` veri çerçevesinin index'inin adı `'purple_link'` olarak değiştirilir.
   - Yani, veri çerçevesindeki mevcut index'e `'purple_link'` adı verilir. Bu, index'in anlamlı bir isimle etiketlenmesini sağlar.
2. `.reset_index(name='n')`
   - `reset_index()` fonksiyonu, mevcut index'i bir sütuna dönüştürür ve bu sütuna `'n'` ismini verir.
   - Bu adım, önceki index olan `purple_link` verilerini bir sütun haline getirir ve bu yeni sütuna `'n'` ismini atar.
   - Ayrıca, veri çerçevesine sıfırdan bir index atanır.
3. `.sort_values('purple_link')`
   - Bu satır, `purple_link_counts` veri çerçevesini `'purple_link'` sütunundaki değerlere göre artan sırayla sıralar.
   - Yani, `'purple_link'` sütunundaki değerler artan şekilde düzenlenir.

Sonuçta, bu işlem `purple_link_counts` veri çerçevesinin index'ini `'purple_link'` olarak adlandırır, index'i bir sütun haline getirir ve ardından sıralama işlemi gerçekleştirir.

[ ]:
```
# 1. 'pd.DataFrame({...})' →
# Bir sözlük kullanarak yeni bir pandas DataFrame oluşturur.

hypothesized = pd.DataFrame({
    # 'purple_link' sütunu → Kategorik değerleri içerir.
    'purple_link': ['Amused', 'Annoyed', 'Hello, old friend', 'Indifferent'],

    # 'prop' sütunu → Her kategori için hipotezde belirtilen olasılıkları␣
 ↪(oranları) içerir.
    'prop': [1/6, 1/6, 1/2, 1/6]
})
```

```
# 2. 'hypothesized' →
# Oluşturulan DataFrame şu sütunlara sahiptir:
# - 'purple_link': Değişkenin benzersiz değerleri (kategoriler).
# - 'prop': Her kategori için beklenen (hipotez edilen) olasılık.

hypothesized
```

[ ]:
```
#  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
```

1. `n_total = len(df_stck)`
   - Bu satırda, `df_stck` veri çerçevesindeki toplam satır sayısı `n_total` değişkenine atanır.
   - `len(df_stck)` ifadesi, veri çerçevesindeki toplam gözlem sayısını döndürür.
2. `hypothesized['n'] = hypothesized['prop'] * n_total`
   - `hypothesized` veri çerçevesindeki `'prop'` sütunundaki her bir oranın, toplam gözlem sayısı (`n_total`) ile çarpılması işlemi yapılır.
   - Bu, her bir kategori için varsayılan sayıların hesaplanmasını sağlar. Örneğin, her bir kategorinin beklenen gözlem sayısı hesaplanır.
   - Bu işlem, `'n'` adlı yeni bir sütun ekleyerek sonuçları `hypothesized` veri çerçevesine dahil eder.
3. `hypothesized`
   - Bu satırda, `hypothesized` veri çerçevesinin son hali yazdırılır.
   - Böylece, her bir kategori için beklenen (varsayılan) sayıların hesaplandığı veri çerçevesi görüntülenir.

[ ]:
```
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()
```

1. `import matplotlib.pyplot as plt`
   - Bu satırda, `matplotlib.pyplot` kütüphanesi `plt` adıyla içeri aktarılır. Bu kütüphane, veri görselleştirme (grafik oluşturma) işlemleri için kullanılır.
2. `plt.bar(purple_link_counts['purple_link'], purple_link_counts['n'], color='red', label='Observed')`
   - `plt.bar()` fonksiyonu ile bir çubuk grafik çizilir.
   - `purple_link_counts['purple_link']`: X ekseninde yer alacak kategorik veriler (yani, 'purple_link' sütunundaki kategoriler).

- `purple_link_counts['n']`: Y ekseninde yer alacak değerler (yani, her kategori için hesaplanan gözlem sayıları).
- `color='red'`: Çubukların rengini kırmızı olarak belirler.
- `label='Observed'`: Bu çubukların etiketini 'Observed' olarak ayarlar.

3. `plt.bar(hypothesized['purple_link'], hypothesized['n'], alpha=0.5, color='blue', label='Hypothesized')`
   - Bu satırda, ikinci bir çubuk grafik çizilir.
   - `hypothesized['purple_link']`: X ekseninde yer alacak kategorik veriler (beklenen kategoriler).
   - `hypothesized['n']`: Y ekseninde yer alacak değerler (beklenen gözlem sayıları).
   - `alpha=0.5`: Çubukların şeffaflık değerini %50 olarak ayarlar (yarı şeffaf).
   - `color='blue'`: Çubukların rengini mavi olarak belirler.
   - `label='Hypothesized'`: Bu çubukların etiketini 'Hypothesized' olarak ayarlar.
4. `plt.legend()`
   - `plt.legend()` fonksiyonu, grafikte kullanılan etiketleri gösterir. Bu, çubuk grafiklerdeki 'Observed' ve 'Hypothesized' etiketlerini gösterecektir.

[ ]:
```
from scipy.stats import chisquare

chisquare(f_obs=purple_link_counts['n'], f_exp=hypothesized['n'])
```

1. `from scipy.stats import chisquare`
   - Bu satırda, `scipy.stats` modülünden `chisquare` fonksiyonu içeri aktarılır. `chisquare` fonksiyonu, Ki-kare testi yapmak için kullanılır.
2. `chisquare(f_obs=purple_link_counts['n'], f_exp=hypothesized['n'])`
   - `chisquare()` fonksiyonu, gözlemlenen ve beklenen frekanslar arasındaki farkı test eder.
   - `f_obs=purple_link_counts['n']`: Gözlemlenen frekanslar (örneklem verisi), burada `purple_link_counts['n']` her kategorideki gözlenen sayıları temsil eder.
   - `f_exp=hypothesized['n']`: Beklenen frekanslar (varsayılan oranlar), burada `hypothesized['n']` her kategori için hesaplanan beklenen sayıları temsil eder.
   - Bu fonksiyon, Ki-kare istatistiği ve p-değeri döndürür. p-değeri, gözlemlenen frekanslar ile beklenen frekanslar arasındaki farkın istatistiksel olarak anlamlı olup olmadığını test eder.

[ ]:
```

```