evrfhfgls

January 7, 2025

[15]:
```
import pandas as pd
import numpy as np
df_small = pd.read_csv("Data/small.csv")
df_small
```

[15]:

	state	county	repub_percent_08	repub_percent_12
0	Texas	Red River	68.507522	69.944817
1	Texas	Walker	60.707197	64.971903
2	Kentucky	Powell	57.059533	61.727293
3	Texas	Schleicher	74.386503	77.384464
4	West Virginia	Morgan	60.857614	64.068711

[17]:
```
df_small['diff'] = df_small['repub_percent_08'] - df_small['repub_percent_12']
df_small
```

[17]:

	state	county	repub_percent_08	repub_percent_12	diff
0	Texas	Red River	68.507522	69.944817	-1.437295
1	Texas	Walker	60.707197	64.971903	-4.264706
2	Kentucky	Powell	57.059533	61.727293	-4.667760
3	Texas	Schleicher	74.386503	77.384464	-2.997961
4	West Virginia	Morgan	60.857614	64.068711	-3.211097

[19]:
```
df_small['abs_diff'] = df_small['diff'].abs()
df_small
#mutlak değerlerini aldık
```

[19]:

	state	county	repub_percent_08	repub_percent_12	diff	\
0	Texas	Red River	68.507522	69.944817	-1.437295	
1	Texas	Walker	60.707197	64.971903	-4.264706	
2	Kentucky	Powell	57.059533	61.727293	-4.667760	
3	Texas	Schleicher	74.386503	77.384464	-2.997961	
4	West Virginia	Morgan	60.857614	64.068711	-3.211097	

	abs_diff
0	1.437295
1	4.264706
2	4.667760

```
3  2.997961
4  3.211097
```

[21]:
```
from scipy.stats import rankdata

df_small['rank_abs_diff'] = rankdata(df_small['abs_diff'])
df_small
```

[21]:
```
           state       county  repub_percent_08  repub_percent_12      diff  \
0          Texas    Red River         68.507522         69.944817 -1.437295
1          Texas       Walker         60.707197         64.971903 -4.264706
2       Kentucky       Powell         57.059533         61.727293 -4.667760
3          Texas   Schleicher         74.386503         77.384464 -2.997961
4  West Virginia       Morgan         60.857614         64.068711 -3.211097

   abs_diff  rank_abs_diff
0  1.437295            1.0
1  4.264706            4.0
2  4.667760            5.0
3  2.997961            2.0
4  3.211097            3.0
```

[23]:
```
T_minus = 1 + 4 + 5 + 2 + 3
T_plus = 0
W = np.min([T_minus, T_plus])
W
```

[23]:
```
0
```

0.0.1 Wilcoxon-Mann-Whitney Testi

Anlamlılık düzeyi 0.01 olarak belirlensin. `pingouin`'den **mwu** kullanarak bir Wilcoxon-Mann-Whitney testi çalıştırılabilir. Karşılaştırmak istenilen iki sayı sütununa karşılık gelen **x** ve **y** argümanlarını kabul eder, bu durumda **çocuk** ve **yetişkin**.

`alternative`, alternatif hipotezin türünü belirler, bu durumda, önce **çocuk** olarak kodlayanların önce **yetişkin** olarak kodlayanlardan daha yüksek bir gelire sahip olduğu, ki bu sağ kuyruklu bir testtir. Burada, **p-değeri yaklaşık 10 üzeri negatif 19. kuvvet** olarak gösterilmektedir, bu da anlamlılık düzeyinden önemli ölçüde daha küçüktür.

[38]:
```
import pingouin
alpha = 0.01
pingouin.wilcoxon(x=df_small['repub_percent_08'],
                  y=df_small['repub_percent_12'],
                  alternative='less')
```

[38]:
```
          W-val alternative    p-val  RBC  CLES
Wilcoxon    0.0        less  0.03125 -1.0  0.72
```

[40]:
```
import pandas as pd

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

[42]:
```
age_vs_comp = df_stck[['converted_comp', 'age_first_code_cut']]
age_vs_comp
```

[42]:

	converted_comp	age_first_code_cut
0	77556.0	adult
1	74970.0	child
2	594539.0	child
3	2000000.0	adult
4	37816.0	adult
…	…	…
2256	145000.0	child
2257	33972.0	child
2258	97284.0	child
2259	72000.0	child
2260	180000.0	child

[2261 rows x 2 columns]

[44]:
```
age_vs_comp_wide = age_vs_comp.pivot(columns='age_first_code_cut',
                                     values='converted_comp')
age_vs_comp_wide
```

[44]:

age_first_code_cut	adult	child
0	77556.0	NaN
1	NaN	74970.0
2	NaN	594539.0
3	2000000.0	NaN
4	37816.0	NaN
…	…	…
2256	NaN	145000.0
2257	NaN	33972.0
2258	NaN	97284.0
2259	NaN	72000.0
2260	NaN	180000.0

[2261 rows x 2 columns]

[46]:
```
import pingouin

alpha = 0.01

pingouin.mwu(x=age_vs_comp_wide['child'],
             y=age_vs_comp_wide['adult'],
```

```
                alternative='greater')
```

[46]:

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

0.0.2 Kruskal-Wallis Testi

ANOVA'nın t-testlerini ikiden fazla gruba genişletmesi gibi, Kruskal-Wallis testi de Wilcoxon-Mann-Whitney testini ikiden fazla gruba genişletir. Yani, Kruskal-Wallis testi ANOVA'nın parametrik olmayan bir versiyonudur.

İş tatmini grupları arasında **converted_comp** açısından bir fark olup olmadığını araştırmak üzere Kruskal-Wallis testi yapmak için **pingouin**'in **kruskal** yöntemini kullanıyoruz. Wilcoxon-Mann-Whitney testinin aksine, kruskal yöntemi uzun veriler üzerinde çalıştığı için burada verilerimizi pivotlamamıza gerek yoktur.

Veri olarak **stack_overflow**, bağımlı değişken olan **dv**'yi `converted_comp` olarak giriyoruz ve **job_sat** grupları arasında karşılaştırma yapıyoruz.

Sonuç olarak, buradaki **p-değeri çok küçüktür** ve anlamlılık düzeyimizden daha küçüktür. Bu, ortalama tazminat toplamlarından en az birinin bu beş iş memnuniyeti grubunda diğerlerinden farklı olduğuna dair kanıt sağlamaktadır.

[50]:
```
alpha = 0.01

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

[50]:

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

0.0.3 MACHINE LEARNING

[76]:
```
#Sınıflandırmada veriler etiketlidir , kümelemede veriler etiketsizdir
#Features , öznitelik sorarasa bunlar verideki alanlar demek
#büyük X ile giriş verileri, küçük y ile çıkış verileri temsil edilir
#Sayısal verilerin kabul edilmesi için verilerin pandas series , pandas␣
 ↪dataframe yada numpy array şeklinde olmalıdır
#yoksa bu türlere dönüştürmemiz lazım
```

0.0.4 What is Machine Learning?

Makine öğrenimi, bilgisayarların açıkça programlanmadan verilerden karar vermeyi öğrenmesi sürecidir.

0.0.5 Examples of Machine Learning

Örneğin, içeriği ve göndereni göz önüne alındığında bir e-postanın spam olup olmadığını tahmin etmeyi öğrenmek.

Ya da kitapları içerdikleri kelimelere göre farklı kategorilerde kümelemeyi öğrenmek. Ardından herhangi bir yeni kitabı mevcut kümelerden birine atamak.

0.0.6 Unsupervised Learning

Denetimsiz öğrenme, etiketlenmemiş verilerden gizli kalıpları ve yapıları ortaya çıkarma sürecidir. Örneğin, bir işletme müşterilerini, bu kategorilerin ne olduğunu önceden bilmeden satın alma davranışlarına göre farklı kategorilerde gruplamak isteyebilir. Bu, denetimsiz öğrenmenin bir dalı olan **kümeleme** olarak bilinir.

0.0.7 Supervised Learning

Denetimli öğrenme, tahmin edilecek değerlerin zaten bilindiği ve daha önce görülmemiş verilerin değerlerini doğru bir şekilde tahmin etmek amacıyla bir modelin oluşturulduğu bir makine öğrenimi türüdür.
Denetimli öğrenme, hedef değişkenin değerini tahmin etmek için **özellikler** kullanır, örneğin bir basketbol oyuncusunun maç başına attığı sayılara göre pozisyonunu tahmin etmek gibi.

Bu ders yalnızca **denetimli öğrenmeye** odaklanacaktır.

[54]:
```
#  Using scikit-learn to fit a classifier
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

(X.shape, y.shape)
```

[54]:
```
((3333, 2), (3333,))
```

[56]:
```
model_knn = KNeighborsClassifier(n_neighbors=15)
model_knn.fit(X,y)
```

[56]:
```
KNeighborsClassifier(n_neighbors=15)
```

[58]:
```
# Predicting on unlabeled data
import numpy as np
X_new = np.array([
    [56.8, 17.5],
    [24.4, 24.1],
    [50.1, 10.9]
])
X_new.shape
```

[58]:
```
(3, 2)
```

[60]:
```
predictions = model_knn.predict(X_new)
predictions
```

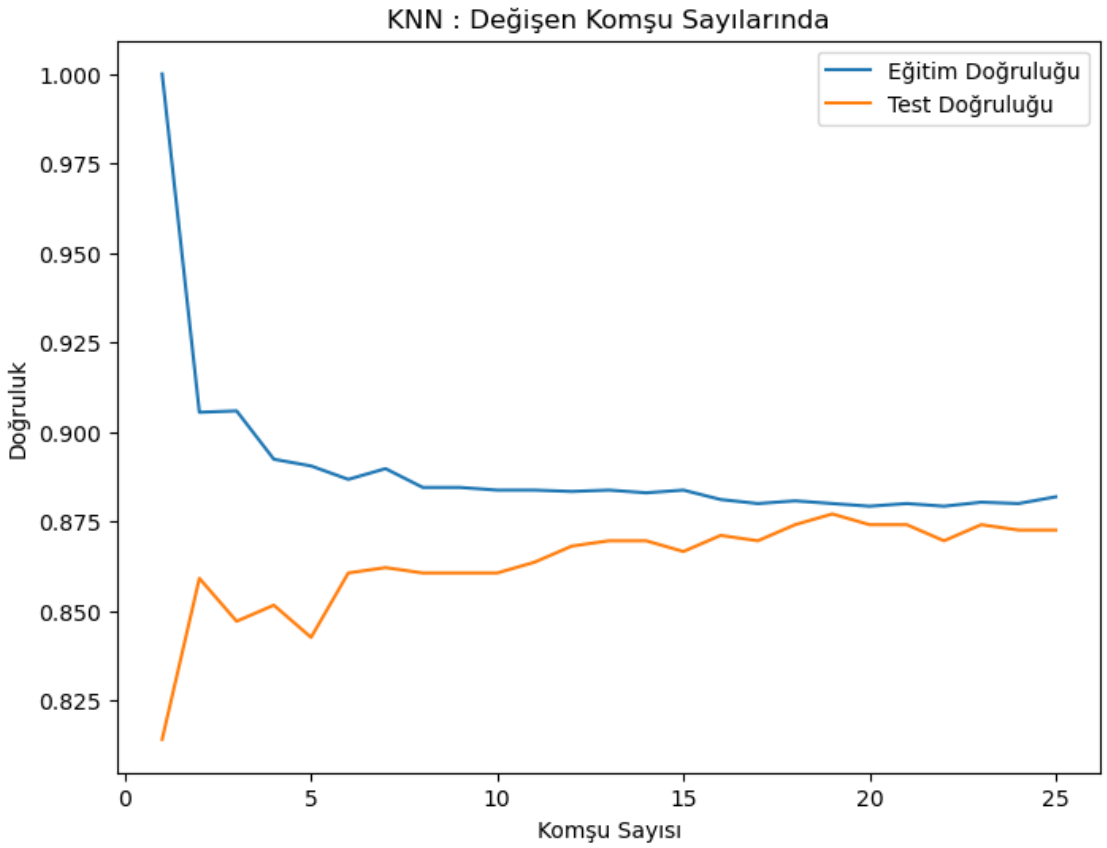
[60]:
```
array([1, 0, 0])
```

[62]:
```
# Train/test split

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)
```

[62]:
```
0.8605697151424287
```

[64]:
```
import matplotlib.pyplot as plt


train_accuracies = {}
test_accuracies = {}

neighbours = np.arange(1, 26)

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

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

[64]:
```
Text(0, 0.5, 'Doğruluk')
```



1 KNN Sınıflandırıcısını Kullanarak Model Eğitimi ve Değerlendirme

1.0.1 1. Veri Yükleme ve Hazırlık

- **Veri seti**: `telecom_churn_clean.csv`
- Özellikler (`X`): `total_day_charge` ve `total_eve_charge`
- Hedef değişken (`y`): `churn`
- **Boyutlar**: `(X.shape, y.shape)` ile kontrol edildi.

1.0.2 2. KNN Modeli Eğitimi

- **KNN modeli** oluşturuldu ve `n_neighbors=15` olarak ayarlandı.
- `model_knn.fit(X, y)` ile veri üzerinde eğitildi.

1.0.3 3. Tahminler

- Yeni verilere tahmin yapılması için: `python X_new = np.array([ [56.8, 17.5], [24.4, 24.1], [50.1, 10.9] ]) predictions = model_knn.predict(X_new)`
- **Tahmin edilen sonuçlar**: `predictions`.

1.0.4 4. Eğitim ve Test Seti Ayrımı

- Veriler, `train_test_split` kullanılarak %80 eğitim, %20 test olarak bölündü.
- Bölümleme sırasında sınıfların dengeli kalması için **`stratify=y`** kullanıldı.
- Eğitim ve test doğruluğu ölçülmek üzere yeni bir KNN modeli (`n_neighbors=6`) eğitildi.

1.0.5 5. Komşu Sayısının Doğruluk Üzerindeki Etkisi

- **Farklı komşu sayılarında modelin performansı** incelendi.
- Eğitim ve test doğrulukları hesaplandı ve grafikte gösterildi.
  - Eğitim doğruluğu genelde komşu sayısı arttıkça azalır.
  - Test doğruluğu ise genelde bir noktada maksimuma ulaşıp düşebilir.

1.0.6 6. Görselleştirme

- Komşu sayısına göre eğitim ve test doğruluğu çizildi:
  - **X ekseni**: Komşu sayısı.
  - **Y ekseni**: Doğruluk.
  - **Eğri 1**: Eğitim doğruluğu.
  - **Eğri 2**: Test doğruluğu.

[39]:
```
import pandas as pd

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

[39]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
4          1
```

[41]:
```
df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) | (df_diabets['glucose'] ==␣
↪0)]
df_filtered
```

[41]:

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
9	8	125	96	0	0	0.0	0.232	54	1
49	7	105	0	0	0	0.0	0.305	24	0
60	2	84	0	0	0	0.0	0.304	21	0
75	1	0	48	20	0	24.7	0.140	22	0
81	2	74	0	0	0	0.0	0.102	22	0
145	0	102	75	23	0	0.0	0.572	21	0
182	1	0	74	20	23	27.7	0.299	21	0
342	1	0	68	35	0	32.0	0.389	22	0
349	5	0	80	32	0	41.0	0.346	37	1
371	0	118	64	23	89	0.0	1.731	21	0
426	0	94	0	0	0	0.0	0.256	25	0
494	3	80	0	0	0	0.0	0.174	22	0
502	6	0	68	41	0	39.0	0.727	41	1
522	6	114	0	0	0	0.0	0.189	26	0
684	5	136	82	0	0	0.0	0.640	69	0
706	10	115	0	0	0	0.0	0.261	30	1

[43]:
```
df_diabets.drop(df_filtered.index, inplace=True) # inplace = true demek gerçek␣
↪veri üzerindede siler
df_diabets
```

[43]:
```
     pregnancies  glucose  diastolic  triceps  insulin   bmi    dpf  age  \
0              6      148         72       35        0  33.6  0.627   50
1              1       85         66       29        0  26.6  0.351   31
2              8      183         64        0        0  23.3  0.672   32
3              1       89         66       23       94  28.1  0.167   21
4              0      137         40       35      168  43.1  2.288   33
..           ...      ...        ...      ...      ...  ...    ...  ...
763           10      101         76       48      180  32.9  0.171   63
764            2      122         70       27        0  36.8  0.340   27
765            5      121         72       23      112  26.2  0.245   30
766            1      126         60        0        0  30.1  0.349   47
767            1       93         70       31        0  30.4  0.315   23

     diabetes
0           1
1           0
2           1
3           0
4           1
..        ...
763         0
764         0
765         0
766         1
767         0

[752 rows x 9 columns]
```

[67]:
```
X = df_diabets.drop('glucose', axis=1).values #axis=1 sütün olduğunu gösterir
y = df_diabets['glucose'].values #.values ilede verileri numpy arraya çevirmiş␣
 ↪oluyoruz

(type(X), type(y))
```

[67]:
```
(numpy.ndarray, numpy.ndarray)
```

[47]:
```
X_bmi = X[:, 4]
(X_bmi.shape, y.shape)
```

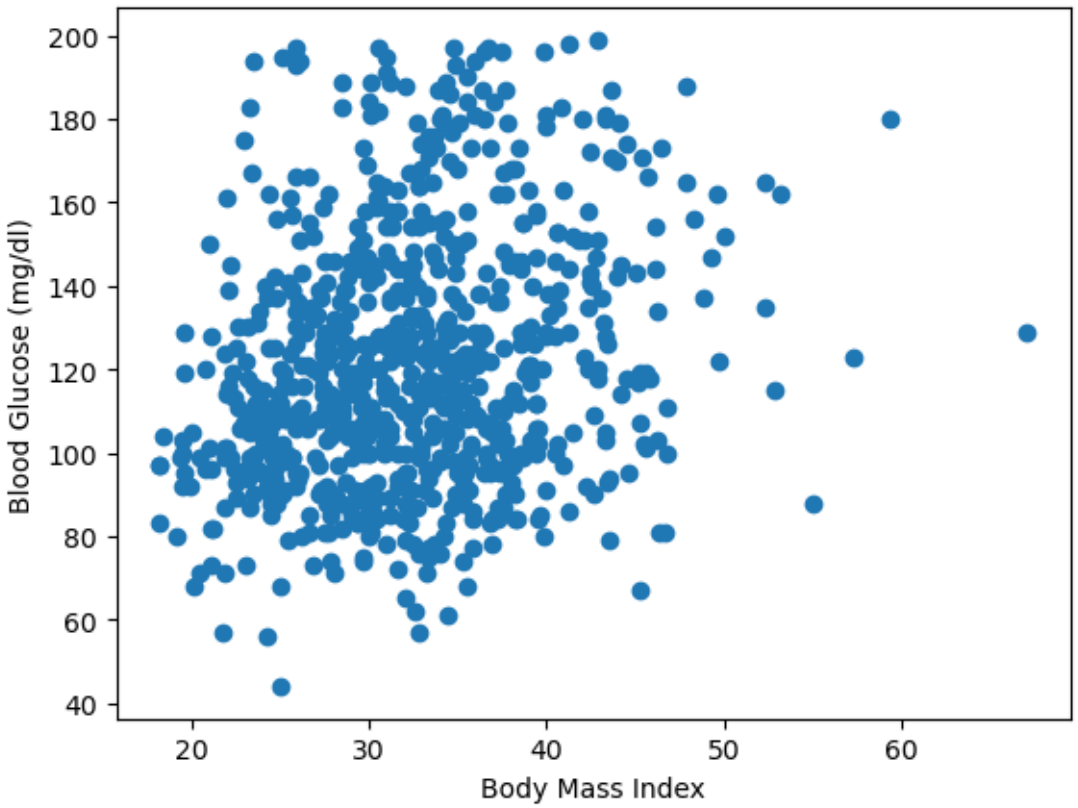
[47]:
```
((752,), (752,))
```

[49]:
```
X_bmi = X_bmi.reshape(-1, 1)
X_bmi.shape
```

[49]:
```
(752, 1)
```

[51]:
```
import matplotlib.pyplot as plt

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

[51]:
```
Text(0.5, 0, 'Body Mass Index')
```



1.0.7 The Basics of Linear Regression

Regresyon Mekaniği Verilere bir çizgi uydurmak istenir ve iki boyutta bu, $\mathbf{y = ax + b}$ biçimini alır. Tek bir özellik kullanmak **basit doğrusal regresyon** olarak bilinir. Burada:
- **y**: hedef
- **x**: özellik
- **a** ve **b**: öğrenmek istediğimiz model parametreleridir.

a ve **b** ayrıca model katsayıları veya sırasıyla **slope** (eğim) ve **intercept** (kesişim) olarak da adlandırılır.

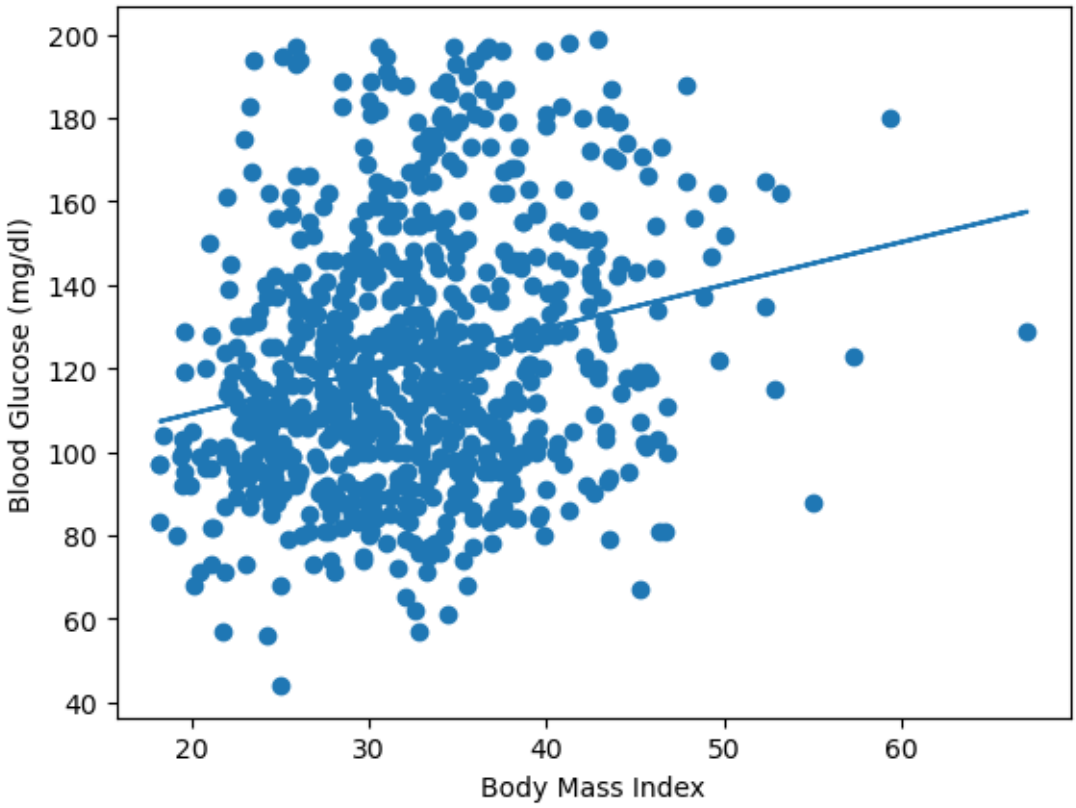
Peki **a** ve **b** için değerler nasıl doğru bir şekilde seçilir?
Herhangi bir verilen çizgi için bir **hata fonksiyonu** tanımlanabilir ve ardından bu fonksiyonu en aza indiren çizgi seçilebilir. Hata fonksiyonlarına ayrıca **loss** veya **cost** fonksiyonları da denir.

Loss Function Bu dağılım grafiğini kullanarak bir **kayıp fonksiyonunu** görselleştirelim. Doğrunun gözlemlere mümkün olduğunca yakın olmasını istiyoruz. Bu nedenle, **uyum** ve **veri** arasındaki **dikey mesafeyi** en aza indirmek istiyoruz.

Bu yüzden her gözlem için, gözlem ile doğru arasındaki dikey mesafeyi hesaplıyoruz. Bu mesafeye **residual** denir.

[97]:
```
from sklearn.linear_model import LinearRegression

model_reg = LinearRegression()
model_reg.fit(X_bmi, y)

predictions = model_reg.predict(X_bmi)

plt.scatter(X_bmi, y)
plt.plot(X_bmi, predictions)
plt.ylabel('Blood Glucose (mg/dl)')
plt.xlabel('Body Mass Index')
#Siyah çizgi, dogrusal regresyon modelinin kan glukoz degerlerinin vücut kitle␣
 ↪indeksine uyumunu temsil etmektedir ve
#zayif-orta düzeyde pozitif bir korelasyona sahip oldugu görülmektedir.
```

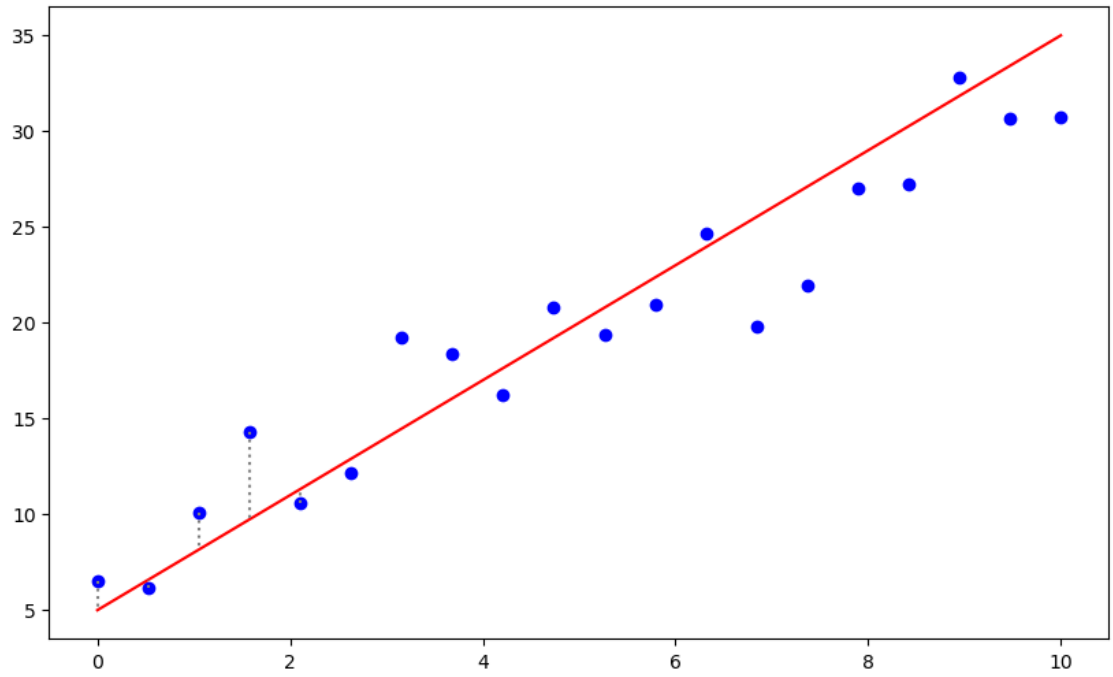
[97]:
```
Text(0.5, 0, 'Body Mass Index')
```



[55]:
```
import numpy as np
import matplotlib.pyplot as plt

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

# Fit a line (y = mx + c) manually
m = 3  # slope
c = 5  # intercept
Y_fit = m * X + c

# Calculate residuals (vertical distances between points and the line)
residuals = Y - Y_fit

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

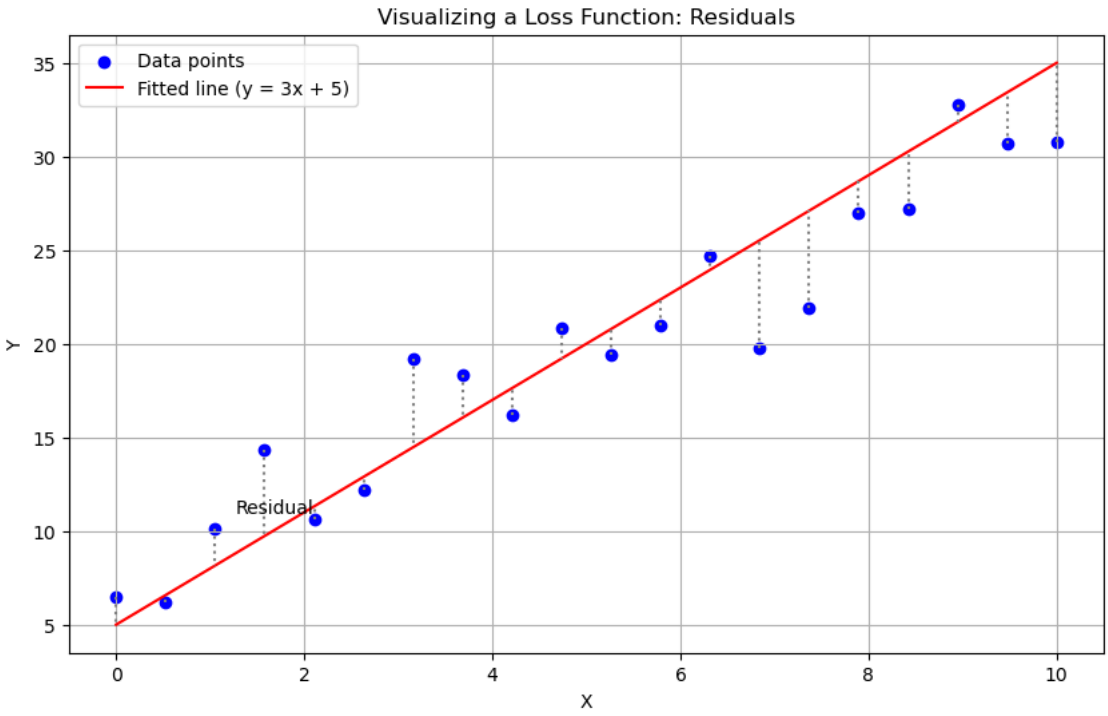
```
# Add vertical lines to represent residuals
for i in range(len(X)):
    plt.plot([X[i], X[i]], [Y[i], Y_fit[i]], color='gray', linestyle='dotted')
    if i == 4: Error
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[55], line 25
     23 for i in range(len(X)):
     24     plt.plot([X[i], X[i]], [Y[i], Y_fit[i]], color='gray',␣
 ↪linestyle='dotted')
---> 25     if i == 4: Error

NameError: name 'Error' is not defined
```



[57]:
```
import numpy as np
import matplotlib.pyplot as plt

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

# Fit a line (y = mx + c) manually
```

```
m = 3  # slope
c = 5  # intercept
Y_fit = m * X + c

# Calculate residuals (vertical distances between points and the line)
residuals = Y - Y_fit

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

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

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

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



[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import pandas as pd

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

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


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


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


model_reg = LinearRegression()
model_reg.fit(X_train, y_train)

y_pred = model_reg.predict(X_test)
```

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

1.0.8 Ordinary Least Squares

Artıkların toplamını en aza indirmeyi deneyebiliriz, ancak bu durumda her pozitif artığın her negatif artığı iptal etmesine neden olur. Bundan kaçınmak için **kalıntıların karesini** alırız.

Tüm kareli kalıntıları toplayarak **kalıntı kareler toplamını** veya **RSS**'yi hesaplarız.

RSS'yi en aza indirmeyi amaçladığımız bu doğrusal regresyon türüne **Sıradan En Küçük Kareler** veya **OLS (Ordinary Least Squares)** denir.

1.0.9 Linear Regression in Higher Dimensions

x1 ve **x2** olmak üzere iki özellik ve **y** olmak üzere bir hedef olduğunda, bir doğrusu şu biçimi alır: **y = a1x1 + a2x2 + b**.

Bu nedenle, doğrusal bir regresyon modeli uydurmak için **a1**, **a2** ve intercept **b** olmak üzere üç değişken belirtilir.

Çoklu doğrusal regresyon modeli kurmak, **n** sayıda özellik için bir katsayı (**an**) ve **b** intercept değerlerini belirtmek anlamına gelir.

Çoklu doğrusal regresyon modelleri için **scikit-learn**, özellik ve hedef değerleri için birer değişken bekler.

[71]:
```
model_reg.score(X_test,y_test)
```

[71]:
```
0.3282802627263198
```

[77]:
```
model_reg.intercept_
```

[77]:
```
75.7242926128128
```

[79]:
```
model_reg.coef_
```

[79]:
```
array([-0.32654116,  0.14686555, -0.27590315,  0.08606826,  0.36160446,
        1.8382773 ,  0.42185562, 25.08247323])
```

1.0.10 Mean Squared Error and Root Mean Squared Error

Bir regresyon modelinin performansını değerlendirmenin bir başka yolu da **artık kareler toplamının** ortalamasını almaktır. Bu, **ortalama karesel hata** veya **MSE** (Mean Squared Error) olarak bilinir.

MSE, hedef değişkenimizin karesi cinsinden ölçülür. Örneğin, bir model bir dolar değerini tahmin ediyorsa, MSE dolar kare cinsinden olacaktır.

Dolara dönüştürmek için, **kök ortalama karesel hata** veya **RMSE** (Root Mean Squared Error) olarak bilinen karekökü alabiliriz.

[89]:
```
from sklearn.metrics import root_mean_squared_error, r2_score, accuracy_score


root_mean_squared_error(y_test, y_pred)
```

[89]:
```
25.695203763480208
```

[ ]:
```

```

jlbxy9wuf

January 7, 2025

[1]:
```
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split

# Veri dosyasını okuyup bir pandas DataFrame'e yüklüyoruz.
df_diabets = pd.read_csv('data/diabetes_clean.csv')

# BMI (vücut kitle indeksi) sıfır olan veya glikoz değeri sıfır olan satırları␣
 ↪filtreliyoruz.
df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) | (df_diabets['glucose'] ==␣
 ↪0)]
# Filtrelenen satırları orijinal veri setinden çıkartıyoruz.
df_diabets.drop(df_filtered.index, inplace=True)

# Bağımsız değişkenler (X) ve bağımlı değişken (y) olarak veriyi ayırıyoruz.
X = df_diabets.drop('glucose', axis=1).values  # Glikoz değerini hedef değişken␣
 ↪olarak çıkarıyoruz.
y = df_diabets['glucose'].values  # Glikoz değerlerini hedef değişken olarak␣
 ↪seçiyoruz.

# Veriyi eğitim ve test seti olarak ikiye bölüyoruz (%80 eğitim, %20 test).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Ridge regresyon modelinin farklı alpha değerleriyle performansını saklamak␣
 ↪için bir liste oluşturuyoruz.
scores = []
# Farklı alpha değerleri için döngü başlatıyoruz.
for alpha in [0.1, 1.0, 10.0, 100.0, 1000.0]:
    # Ridge regresyon modelini tanımlıyoruz ve alpha parametresini belirtiyoruz.
    ridge = Ridge(alpha=alpha)
    # Modeli eğitim setiyle eğitiyoruz.
    ridge.fit(X_train, y_train)
    # Modelin test seti üzerindeki tahminlerini yapıyoruz.
    y_pred = ridge.predict(X_test)
    # Modelin performans skorunu (R^2) test seti üzerinde hesaplayıp listeye␣
 ↪ekliyoruz.
```

```
    scores.append(ridge.score(X_test, y_test))

# Farklı alpha değerleri için hesaplanan skorları gösteriyoruz.
scores
```

[1]:
```
[0.32825526615552425,
 0.3280240795994781,
 0.3252049078298792,
 0.28836032637802334,
 0.20299309688977707]
```

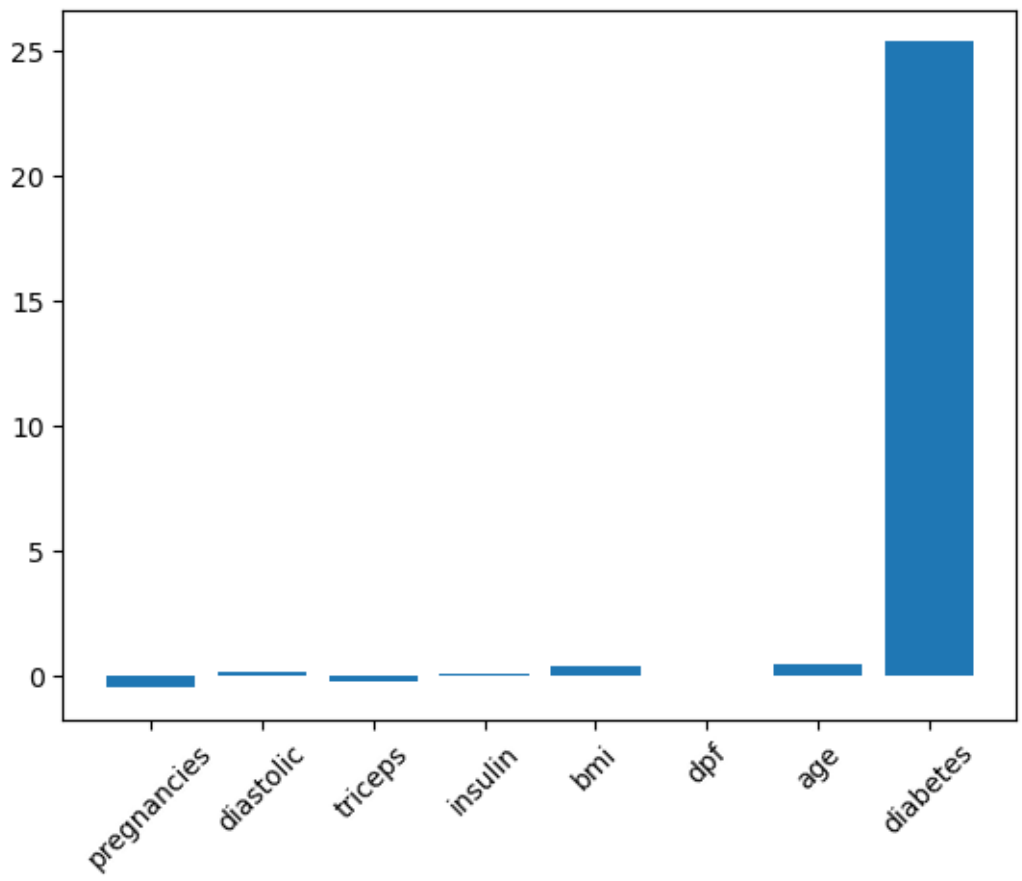
[3]:
```
from sklearn.linear_model import Lasso

# Lasso regresyon modelinin farklı alpha değerleriyle performansını saklamak␣
 ↪için bir liste oluşturuyoruz.
scores = []
# Farklı alpha değerleri için döngü başlatıyoruz.
for alpha in [0.1, 1.0, 10.0, 20.0, 50.0]:
    # Lasso regresyon modelini tanımlıyoruz ve alpha parametresini belirtiyoruz.
    lasso = Lasso(alpha=alpha)
    # Modeli eğitim setiyle eğitiyoruz.
    lasso.fit(X_train, y_train)
    # Modelin test seti üzerindeki tahminlerini yapıyoruz.
    y_pred = lasso.predict(X_test)
    # Modelin performans skorunu (R^2) test seti üzerinde hesaplayıp listeye␣
 ↪ekliyoruz.
    scores.append(lasso.score(X_test, y_test))

# Farklı alpha değerleri için hesaplanan skorları gösteriyoruz.
scores
```

[3]:
```
[0.3284857694292622,
 0.3166121180165745,
 0.17121386697851626,
 0.156847521532139,
 0.11477890284329806]
```

[5]:
```
import matplotlib.pyplot as plt

# Veri dosyasını okuyup bir pandas DataFrame'e yüklüyoruz.
df_diabets = pd.read_csv('data/diabetes_clean.csv')

# BMI (vücut kitle indeksi) sıfır olan veya glikoz değeri sıfır olan satırları␣
 ↪filtreliyoruz.
df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) | (df_diabets['glucose'] ==␣
 ↪0)]
# Filtrelenen satırları orijinal veri setinden çıkartıyoruz.
```

```
df_diabets.drop(df_filtered.index, inplace=True)

# Bağımsız değişkenler (X) ve bağımlı değişken (y) olarak veriyi ayırıyoruz.
X = df_diabets.drop('glucose', axis=1).values  # Glikoz değerini hedef değişken␣
 ↪olarak çıkarıyoruz.
y = df_diabets['glucose'].values  # Glikoz değerlerini hedef değişken olarak␣
 ↪seçiyoruz.

# Özellik isimlerini saklıyoruz (glucose hariç).
names = df_diabets.drop('glucose', axis=1).columns

# Lasso modelini alpha=0.1 ile tanımlıyoruz.
lasso = Lasso(alpha=0.1)
# Modeli tüm veriyi kullanarak eğitiyoruz ve katsayıları alıyoruz.
lasso_coef = lasso.fit(X, y).coef_

# Özellik katsayılarını bir çubuk grafikte görselleştiriyoruz.
plt.bar(names, lasso_coef)  # Özelliklerin katsayılarını çubuk grafik olarak␣
 ↪çiziyoruz.
plt.xticks(rotation=45)  # X eksenindeki etiketlerin yönünü 45 derece␣
 ↪döndürüyoruz.
```

[5]:
```
([0, 1, 2, 3, 4, 5, 6, 7],
 [Text(0, 0, 'pregnancies'),
  Text(1, 0, 'diastolic'),
  Text(2, 0, 'triceps'),
  Text(3, 0, 'insulin'),
  Text(4, 0, 'bmi'),
  Text(5, 0, 'dpf'),
  Text(6, 0, 'age'),
  Text(7, 0, 'diabetes')])
```



[7]:
```
# Gerekli kütüphaneleri yüklüyoruz.
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

# Veri dosyasını okuyup bir pandas DataFrame'e yüklüyoruz.
df_churn = pd.read_csv("data/telecom_churn_clean.csv")

# Bağımsız değişkenler (X) olarak total_day_charge ve total_eve_charge␣
 ↪sütunlarını seçiyoruz.
X = df_churn[['total_day_charge', 'total_eve_charge']].values
# Bağımlı değişken (y) olarak churn sütununu seçiyoruz.
y = df_churn['churn'].values

# Veriyi eğitim ve test seti olarak ikiye bölüyoruz (%80 eğitim, %20 test).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# KNN (K-Nearest Neighbors) sınıflandırıcıyı 7 komşu ile tanımlıyoruz.
knn = KNeighborsClassifier(n_neighbors=7)

# Modeli eğitim setiyle eğitiyoruz.
knn.fit(X_train, y_train)

# Test seti üzerinde tahminler yapıyoruz.
y_pred = knn.predict(X_test)

# Gerçek ve tahmin edilen değerler için karışıklık matrisini (confusion matrix)␣
 ↪hesaplıyoruz.
confusion_matrix(y_test, y_pred)
```

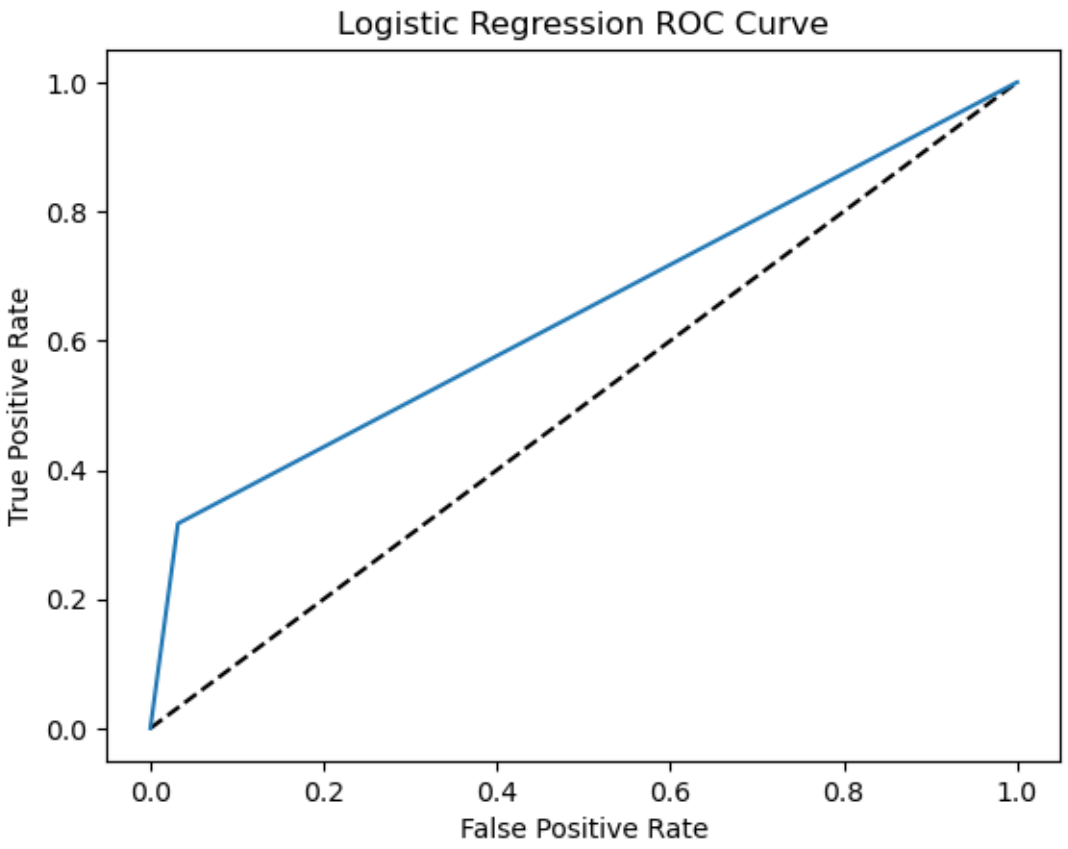
[7]:
```
array([[548,  18],
       [ 69,  32]])
```

[11]:
```
# Gerekli kütüphaneleri yüklüyoruz.
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

# Veri dosyasını okuyup bir pandas DataFrame'e yüklüyoruz.
df_churn = pd.read_csv("data/telecom_churn_clean.csv")

# Bağımsız değişkenler (X) olarak total_day_charge ve total_eve_charge␣
 ↪sütunlarını seçiyoruz.
X = df_churn[['total_day_charge', 'total_eve_charge']].values
# Bağımlı değişken (y) olarak churn sütununu seçiyoruz.
y = df_churn['churn'].values

# Veriyi eğitim ve test seti olarak ikiye bölüyoruz (%80 eğitim, %20 test).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Lojistik regresyon modelini tanımlıyoruz.
model_log = LogisticRegression()
# Modeli eğitim setiyle eğitiyoruz.
model_log.fit(X_train, y_train)

# Test seti üzerinde tahminler yapıyoruz.
y_predict = model_log.predict(X_test)
```

[15]:
```
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

# ROC eğrisini hesaplamak için roc_curve fonksiyonunu kullanıyoruz.
```

```
# y_test: Gerçek sınıf değerleri.
# y_pred_probs: Pozitif sınıfa ait tahmin olasılıkları.
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Rastgele tahmin modelini göstermek için 'y = x' doğrusu çiziyoruz (kırık␣
 ↪çizgi).
plt.plot([0, 1], [0, 1], 'k--')

# ROC eğrisini çiziyoruz (FPR ve TPR değerlerini kullanarak).
plt.plot(fpr, tpr)

# Grafiğe eksen etiketlerini ekliyoruz.
plt.xlabel('False Positive Rate')  # Yanlış Pozitif Oranı (FPR): Negatif sınıfı␣
 ↪pozitif olarak tahmin etme oranı.
plt.ylabel('True Positive Rate')  # Doğru Pozitif Oranı (TPR): Pozitif sınıfı␣
 ↪doğru tahmin etme oranı.
plt.title('Logistic Regression ROC Curve')  # Grafiğe başlık ekliyoruz.

# Bu grafik, modelin farklı eşik değerlerinde (thresholds) nasıl bir performans␣
 ↪gösterdiğini görselleştirir.
```

[15]: `Text(0.5, 1.0, 'Logistic Regression ROC Curve')`



[17]:
```
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
import numpy as np

# Veri dosyasını okuyup bir pandas DataFrame'e yüklüyoruz.
df_diabets = pd.read_csv('data/diabetes_clean.csv')

# BMI (vücut kitle indeksi) sıfır olan veya glikoz değeri sıfır olan satırları␣
 ↪filtreliyoruz.
df_filtered = df_diabets[(df_diabets['bmi'] == 0.0) | (df_diabets['glucose'] ==␣
 ↪0)]
# Filtrelenen satırları orijinal veri setinden çıkartıyoruz.
df_diabets.drop(df_filtered.index, inplace=True)

# Bağımsız değişkenler (X) ve bağımlı değişken (y) olarak veriyi ayırıyoruz.
X = df_diabets.drop('glucose', axis=1).values  # Glikoz değerini hedef değişken␣
 ↪olarak çıkarıyoruz.
y = df_diabets['glucose'].values  # Glikoz değerlerini hedef değişken olarak␣
 ↪seçiyoruz.

# Veriyi eğitim ve test seti olarak ikiye bölüyoruz (%80 eğitim, %20 test).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# K-Fold çapraz doğrulama için ayarları yapıyoruz.
kf = KFold(n_splits=5, shuffle=True, random_state=42)  # 5 katmanlı çapraz␣
 ↪doğrulama, verileri karıştırarak.

# Ridge regresyon modeli için hiperparametre aralığını belirtiyoruz.
param_grid = {
    'alpha': np.arange(0.0001, 1, 10),  # 'alpha' (ceza katsayısı) için bir␣
 ↪aralık belirleniyor.
    'solver': ['sag', 'lsqr']  # Ridge modelinin çözüm algoritmaları: 'sag' ve␣
 ↪'lsqr'.
}

# Ridge regresyon modelini oluşturuyoruz.
ridge = Ridge()

# GridSearchCV ile en iyi parametreleri bulmak için Ridge modelini tarıyoruz.
ridge_cv = GridSearchCV(ridge, param_grid, cv=kf)  # GridSearchCV kullanılarak␣
 ↪çapraz doğrulama yapılıyor.

# Eğitim seti üzerinde hiperparametre taramasını gerçekleştiriyoruz.
```

```
ridge_cv.fit(X_train, y_train)

# En iyi parametreler ve en iyi doğrulama skorunu görüntülüyoruz.
(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[17]:
```
({'alpha': 0.0001, 'solver': 'lsqr'}, 0.3404176885506861)
```

[19]:
```
from sklearn.model_selection import RandomizedSearchCV
# İsteğe bağlı olarak, test edilen hiperparametre değerlerinin sayısını␣
↪belirleyen n_iter bağımsız değişkenini ayarlanabilir.
# Böylece n_iter iki olarak ayarlandığında beş katlı çapraz doğrulama 10 fit()␣
↪gerçekleştirir.
# RandomizedSearchCV kullanarak hiperparametre optimizasyonu yapıyoruz.
ridge_cv = RandomizedSearchCV(ridge, param_grid, cv=kf, n_iter=2)  # n_iter ile␣
↪test edilecek hiperparametre kombinasyonu sayısını sınırlandırıyoruz.
ridge_cv.fit(X_train, y_train)  # Eğitim seti üzerinde optimizasyonu␣
↪gerçekleştiriyoruz.

# En iyi hiperparametreler ve çapraz doğrulama skoru
(ridge_cv.best_params_, ridge_cv.best_score_)
```

```
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
```

```
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
/opt/anaconda3/lib/python3.10/site-packages/sklearn/linear_model/_sag.py:350:
ConvergenceWarning: The max_iter was reached which means the coef_ did not
converge
  warnings.warn(
```

[19]:
```
({'solver': 'lsqr', 'alpha': 0.0001}, 0.3404176885506861)
```

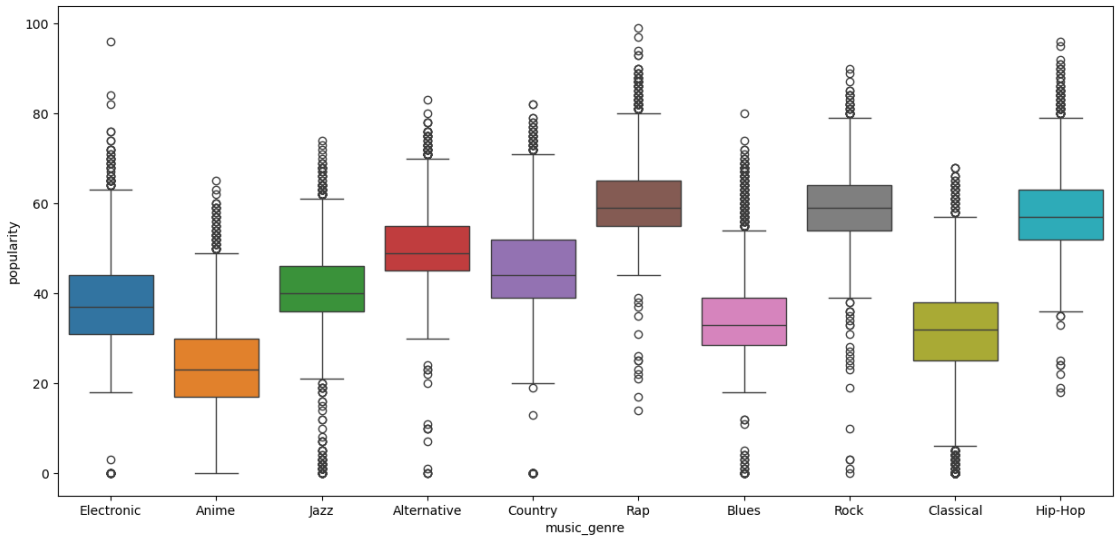
[21]:
```
import pandas as pd

df_music = pd.read_csv('data/music_genre.csv')
df_music.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44996 entries, 0 to 44995
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   instance_id       44996 non-null  int64
 1   popularity        44996 non-null  int64
 2   acousticness      44996 non-null  float64
 3   danceability      44996 non-null  float64
 4   duration_ms       44996 non-null  int64
 5   energy            44996 non-null  float64
 6   instrumentalness  44996 non-null  float64
 7   liveness          44996 non-null  float64
 8   loudness          44996 non-null  float64
 9   speechiness       44996 non-null  float64
 10  tempo             44996 non-null  float64
 11  valence           44996 non-null  float64
 12  music_genre       44996 non-null  object
dtypes: float64(9), int64(3), object(1)
memory usage: 4.5+ MB
```

[23]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 7))
```

```
sns.boxplot(data=df_music, x='music_genre', y='popularity', hue='music_genre')
```

[23]: <Axes: xlabel='music_genre', ylabel='popularity'>



0.0.1 Kutu Grafiği (Boxplot) Açıklaması

Bu kod, Seaborn kütüphanesi ve Matplotlib ile bir **boxplot** (kutu grafiği) oluşturur. Boxplot, bir veri kümesinin dağılımını görselleştirmek için kullanılan bir araçtır. Genellikle medyan, çeyrekler arası aralık (IQR), uç noktalar (outliers) gibi istatistiksel verileri görmek için kullanılır.

Kodun Adım Adım Açıklaması:

1. `plt.figure(figsize=(15, 7))`:
   - Bu satır, grafiğin boyutlarını belirler. `figsize=(15, 7)` parametresi, grafiğin genişliğini 15 birim, yüksekliğini ise 7 birim olarak ayarlar.
2. `sns.boxplot()`:
   - `sns.boxplot`: Seaborn'un boxplot fonksiyonu ile kutu grafiği oluşturuluyor.
   - `data=df_music`: Grafikte kullanılacak veri setini belirtir. Bu örnekte, veri seti `df_music` olarak adlandırılmıştır.
   - `x='music_genre'`: X ekseninde `music_genre` (müzik türü) yer alacak. Bu, her bir müzik türü için bir boxplot oluşturulmasını sağlar.
   - `y='popularity'`: Y ekseninde `popularity` (popülerlik) yer alacak. Bu, müzik türlerinin popülerlik değerlerini gösterir.
   - `hue='music_genre'`: Kutu grafiğinin rengini `music_genre` (müzik türü) kategorilerine göre ayarlamak için `hue` parametresi kullanılır. Böylece her müzik türü için kutu grafiği farklı renklerle görselleştirilir.
3. **Grafiği Görselleştirme**:
   - `plt.show()`: Grafik, belirlenen parametreler ve verilerle ekranda gösterilir.

Grafikle İlgili İpuçları:

- **Kutu**: Verinin %50'sinin, yani 1. çeyrek (Q1) ile 3. çeyrek (Q3) arasında kalan kısmını gösterir. Kutunun içindeki çizgi, verinin medyanını belirtir.
- **Whiskerlar**: Kutu grafiğinin üst ve alt uçlarındaki çizgiler, veri setindeki uç noktaları (outliers hariç) gösterir.
- **Outliers (uç noktalar)**: Kutu grafiğinde kutunun dışında kalan noktalar, uç noktaları temsil eder.

Özet: Bu kod, farklı müzik türlerinin popülerlik değerlerinin dağılımını gösteren bir kutu grafiği oluşturur. Müzik türlerini ve onların popülerlik değerlerini daha iyi anlamak için bu tür görselleştirmeler faydalıdır.

---

[27]:
```
music_dummies = pd.get_dummies(df_music['music_genre'], drop_first=True,␣
↪dtype='int')
music_dummies.head()
```

[27]:

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

0.0.2 `pd.get_dummies` ile Kategorik Verileri Sayısal Hale Getirme

Bu kod, pandas kütüphanesindeki **`get_dummies`** fonksiyonunu kullanarak kategorik bir sütunu (bu örnekte `music_genre` sütunu) sayısal verilere dönüştürür. Bu işlem, **one-hot encoding** olarak adlandırılır ve makine öğrenmesi algoritmalarında kategorik verilerle çalışırken sıklıkla kullanılır.

Kodun Açıklaması:

1. **`music_dummies = pd.get_dummies(df_music['music_genre'], drop_first=True, dtype='int')`**:
   - **`df_music['music_genre']`**: `df_music` veri setindeki **`music_genre`** sütunu, müzik türlerini içeren kategorik bir veri sütunudur.
   - **`pd.get_dummies()`**: Bu fonksiyon, verilen sütunu sayısal verilere dönüştürür. Yani her benzersiz kategori için yeni bir sütun ekler ve bu sütunlarda **1** veya **0** değerleri yer alır (1: ilgili kategori mevcut, 0: ilgili kategori mevcut değil).
   - **`drop_first=True`**: Bu parametre, ilk sütunun düşürülmesini sağlar. Bu, "dummy variable trap" (çoklu doğrusal bağlılık) sorununu önler. İlk sütunun düşürülmesi, modelin gereksiz şekilde yüksek korelasyonlardan etkilenmesini engeller.
   - **`dtype='int'`**: Yeni oluşturulan sütunlar, **tamsayı** türünde olacak şekilde belirtilir.
2. **`music_dummies.head()`**:
   - Bu satır, `music_dummies` veri çerçevesinin ilk 5 satırını görüntüler. Bu, kodun doğru çalışıp çalışmadığını kontrol etmek için faydalıdır.

Özet: Bu kod, **music_genre** sütunundaki kategorik verileri sayısal verilere dönüştürmek için **one-hot encoding** kullanır. Bu dönüşüm, modelin müzik türlerine dayalı öngörüler yapabilmesini sağlar.

---

[31]:
```
music_dummies = pd.concat([df_music, music_dummies], axis=1)
music_dummies
```

[31]:

	instance_id	popularity	acousticness	danceability	duration_ms	\
0	32894	27	0.00468	0.652	-1	
1	46652	31	0.01270	0.622	218293	
2	30097	28	0.00306	0.620	215613	
3	62177	34	0.02540	0.774	166875	
4	24907	32	0.00465	0.638	222369	
…	…	…	…	…	…	
44991	58878	59	0.03340	0.913	-1	
44992	43557	72	0.15700	0.709	251860	
44993	39767	51	0.00597	0.693	189483	
44994	57944	65	0.08310	0.782	262773	
44995	63470	67	0.10200	0.862	267267	

	energy	instrumentalness	liveness	loudness	speechiness	…	\
0	0.941	0.79200	0.115	-5.201	0.0748	…	
1	0.890	0.95000	0.124	-7.043	0.0300	…	
2	0.755	0.01180	0.534	-4.617	0.0345	…	
3	0.700	0.00253	0.157	-4.498	0.2390	…	
4	0.587	0.90900	0.157	-6.266	0.0413	…	
…	…	…	…	…	…	…	
44991	0.574	0.00000	0.119	-7.022	0.2980	…	
44992	0.362	0.00000	0.109	-9.814	0.0550	…	
44993	0.763	0.00000	0.143	-5.443	0.1460	…	
44994	0.472	0.00000	0.106	-5.016	0.0441	…	
44995	0.642	0.00000	0.272	-13.652	0.1010	…	

	music_genre	Anime	Blues	Classical	Country	Electronic	Hip-Hop	\
0	Electronic	0	0	0	0	1	0	
1	Electronic	0	0	0	0	1	0	
2	Electronic	0	0	0	0	1	0	
3	Electronic	0	0	0	0	1	0	
4	Electronic	0	0	0	0	1	0	
…	…	…	…	…	…	…	…	
44991	Hip-Hop	0	0	0	0	0	1	
44992	Hip-Hop	0	0	0	0	0	1	
44993	Hip-Hop	0	0	0	0	0	1	
44994	Hip-Hop	0	0	0	0	0	1	
44995	Hip-Hop	0	0	0	0	0	1	

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

[44996 rows x 22 columns]
```

0.0.3 `pd.concat` ile Veri Birleştirme

Bu kod, **`pd.concat()`** fonksiyonunu kullanarak orijinal veri seti **`df_music`** ile one-hot encoding yöntemiyle oluşturulmuş olan **`music_dummies`** veri çerçevesini birleştirir. Bu işlem, müzik türlerinin sayısal temsillerini (dummy sütunları) orijinal veri setine ekler.

Kodun Açıklaması:

1. **`music_dummies = pd.concat([df_music, music_dummies], axis=1)`**:
   - **`pd.concat()`**: Bu fonksiyon, belirtilen veri çerçevelerini belirtilen eksende (satırlar veya sütunlar) birleştirir.
   - **`[df_music, music_dummies]`**: Bu, birleştirilecek veri çerçevelerini içerir. **`df_music`** orijinal veri setini ve **`music_dummies`** dummy sütunlarını içerir.
   - **`axis=1`**: Bu parametre, veri çerçevelerinin **sütunlar (axis=1)** boyunca birleştirileceğini belirtir. Yani, her iki veri çerçevesinin sütunları yan yana eklenir.
2. **`music_dummies`**:
   - Bu satır, birleştirilmiş veri çerçevesinin sonucunu gösterir. Artık **`df_music`** veri seti, müzik türleri için sayısal sütunlara (dummy sütunları) sahip olacaktır.

Özet: Bu kod, **`df_music`** veri seti ile **`music_dummies`** veri çerçevesini birleştirerek müzik türlerini sayısal verilere dönüştürüp, tüm veriyi tek bir veri setinde toplar. Bu işlem, modelin çalışabilmesi için gerekli olan tüm özellikleri aynı veri setinde birleştirir.

---

[35]:
```
music_dummies = music_dummies.drop(['music_genre', 'instance_id'], axis=1)
music_dummies
```

[35]:
```
       popularity  acousticness  danceability  duration_ms  energy  \
0              27       0.00468         0.652           -1   0.941
1              31       0.01270         0.622       218293   0.890
2              28       0.00306         0.620       215613   0.755
3              34       0.02540         0.774       166875   0.700
```

```
4              32       0.00465         0.638       222369   0.587
…           …           …           …          …       …
44991          59       0.03340         0.913           -1   0.574
44992          72       0.15700         0.709       251860   0.362
44993          51       0.00597         0.693       189483   0.763
44994          65       0.08310         0.782       262773   0.472
44995          67       0.10200         0.862       267267   0.642

       instrumentalness  liveness  loudness  speechiness    tempo  valence  \
0               0.79200     0.115    -5.201       0.0748  100.889    0.759
1               0.95000     0.124    -7.043       0.0300  115.002    0.531
2               0.01180     0.534    -4.617       0.0345  127.994    0.333
3               0.00253     0.157    -4.498       0.2390  128.014    0.270
4               0.90900     0.157    -6.266       0.0413  145.036    0.323
…                  …        …        …          …        …       …
44991           0.00000     0.119    -7.022       0.2980   98.028    0.330
44992           0.00000     0.109    -9.814       0.0550  122.043    0.113
44993           0.00000     0.143    -5.443       0.1460  131.079    0.395
44994           0.00000     0.106    -5.016       0.0441   75.886    0.354
44995           0.00000     0.272   -13.652       0.1010   99.201    0.765

       Anime  Blues  Classical  Country  Electronic  Hip-Hop  Jazz  Rap  Rock
0          0      0          0        0           1        0     0    0     0
1          0      0          0        0           1        0     0    0     0
2          0      0          0        0           1        0     0    0     0
3          0      0          0        0           1        0     0    0     0
4          0      0          0        0           1        0     0    0     0
…      …    …       …      …         …      …   …  …   …
44991      0      0          0        0           0        1     0    0     0
44992      0      0          0        0           0        1     0    0     0
44993      0      0          0        0           0        1     0    0     0
44994      0      0          0        0           0        1     0    0     0
44995      0      0          0        0           0        1     0    0     0

[44996 rows x 20 columns]
```

0.0.4 `drop` Fonksiyonu ile Sütun Silme

Bu kod, **`drop()`** fonksiyonu kullanarak **`music_genre`** ve **`instance_id`** sütunlarını **`music_dummies`** veri çerçevesinden siler.

Kodun Açıklaması:

1. **`music_dummies = music_dummies.drop(['music_genre', 'instance_id'], axis=1)`**:
    - **`drop()`**: Bu fonksiyon, belirtilen sütunları veya satırları veri çerçevesinden silmek için kullanılır.
    - **`['music_genre', 'instance_id']`**: Bu, silinmesi istenen sütunları belirtir. Bu örnekte **`music_genre`** ve **`instance_id`** sütunları silinmektedir.

- **axis=1**: Bu parametre, sütunların silineceğini belirtir. axis=0 olsaydı, satırlar silinirdi.

2. **music_dummies**:
   - Bu satır, **music_dummies** veri çerçevesinin, belirli sütunlar (bu durumda **music_genre** ve **instance_id**) silindikten sonraki halini döndüren sonucu gösterir.

Özet: Bu kod, **music_genre** ve **instance_id** sütunlarını **music_dummies** veri setinden kaldırarak sadece gerekli sayısal özellikleri tutar. Bu, model için gereksiz olan sütunların kaldırılması amacıyla yapılan bir işlemdir.

---

[39]:
```
# Linear regression with dummy variables

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

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

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

kf = KFold(n_splits=5, shuffle=True, random_state=42)

model_reg = LinearRegression()
model_reg_cv = cross_val_score(model_reg, X_train, y_train, cv=kf,␣
 ↪scoring='neg_mean_squared_error')

np.sqrt(-model_reg_cv)
```

[39]:
```
array([9.55195316, 9.35056784, 9.426638  , 9.61559878, 9.60648301])
```

0.0.5 Doğrusal Regresyon ve Dummy Değişkenleri ile Modelleme

Bu kod, **dummy değişkenleri** kullanarak bir **doğrusal regresyon modeli** oluşturur ve **cross-validation** (çapraz doğrulama) ile modelin performansını değerlendirir. Ayrıca, modelin **negatif ortalama kare hatası** (negative mean squared error, MSE) üzerinden doğrulama işlemi yapılır ve daha sonra bu hata değeri **karekköüne** alınır.

Kodun Adım Adım Açıklaması:

1. **Veri Seti ve Hedef Değişkenin Belirlenmesi**:
   - **X = music_dummies.drop('popularity', axis=1).values**: Bu satır, **music_dummies** veri setinden **popularity** sütununu çıkararak özellikleri (features) **X**'e atar.
   - **y = music_dummies['popularity'].values**: Bu satır, **popularity** sütununu hedef değişken (target variable) **y** olarak belirler.
2. **Veri Setinin Eğitim ve Test Setlerine Bölünmesi**:

- `train_test_split(X, y, test_size=0.2, random_state=42)`: Veri seti %80 eğitim, %20 test olarak ikiye bölünür. `random_state=42` sabit bir bölme sağlar, böylece her çalıştırmada aynı veri seti bölünmesi elde edilir.

3. **Çapraz Doğrulama İçin KFold Kullanımı**:
   - `kf = KFold(n_splits=5, shuffle=True, random_state=42)`: Bu satırda **KFold** sınıfı ile 5 katmanlı (fold) çapraz doğrulama oluşturulur. `shuffle=True`, verilerin karıştırılmasını sağlar.
4. **Modelin Tanımlanması ve Çapraz Doğrulama**:
   - `model_reg = LinearRegression()`: Doğrusal regresyon modeli oluşturulur.
   - `model_reg_cv = cross_val_score(model_reg, X_train, y_train, cv=kf, scoring='neg_mean_squared_error')`: `cross_val_score` fonksiyonu ile modelin performansı çapraz doğrulama kullanılarak değerlendirilir. Burada `scoring='neg_mean_squared_error'` parametresi, modelin ortalama kare hatasını hesaplamak için kullanılır. Ancak `cross_val_score`, hata değerlerini negatif olarak döndürdüğü için bu değerler **negatif** olur.
5. **Ortalama Kare Hatasının Karekökünün Hesaplanması**:
   - `np.sqrt(-model_reg_cv)`: Çünkü `cross_val_score` fonksiyonu negatif hata değerlerini döndürdü, bunları pozitif yapmak için negatif işaret kaldırılır ve ardından **karekök** alınır. Bu işlem, modelin **kök ortalama kare hatası (RMSE)** değerini verir.

Özet: Bu kod, dummy değişkenleriyle yapılan doğrusal regresyon modelinin **kök ortalama kare hatası (RMSE)** değerini çapraz doğrulama kullanarak hesaplar. Çapraz doğrulama, modelin genellenebilirliğini test etmek için veri setini farklı alt gruplara ayırarak doğrulama yapar ve modelin performansını daha güvenilir bir şekilde değerlendirir.

[ ]:

pjpavlzjt

January 7, 2025

[7]:
```
# Handling Missing Data
```

[15]:
```
import pandas as pd

df_house = pd.read_csv('../../AmesHousing.csv') # Elimizde eksik veri olan bir df

df_house.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2930 entries, 0 to 2929
Data columns (total 82 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Order            2930 non-null   int64
 1   PID              2930 non-null   int64
 2   MS SubClass      2930 non-null   int64
 3   MS Zoning        2930 non-null   object
 4   Lot Frontage     2440 non-null   float64
 5   Lot Area         2930 non-null   int64
 6   Street           2930 non-null   object
 7   Alley            198 non-null    object
 8   Lot Shape        2930 non-null   object
 9   Land Contour     2930 non-null   object
 10  Utilities        2930 non-null   object
 11  Lot Config       2930 non-null   object
 12  Land Slope       2930 non-null   object
 13  Neighborhood     2930 non-null   object
 14  Condition 1      2930 non-null   object
 15  Condition 2      2930 non-null   object
 16  Bldg Type        2930 non-null   object
 17  House Style      2930 non-null   object
 18  Overall Qual     2930 non-null   int64
 19  Overall Cond     2930 non-null   int64
 20  Year Built       2930 non-null   int64
 21  Year Remod/Add   2930 non-null   int64
 22  Roof Style       2930 non-null   object
 23  Roof Matl        2930 non-null   object
```

```
24  Exterior 1st     2930 non-null   object
25  Exterior 2nd     2930 non-null   object
26  Mas Vnr Type     1155 non-null   object
27  Mas Vnr Area     2907 non-null   float64
28  Exter Qual       2930 non-null   object
29  Exter Cond       2930 non-null   object
30  Foundation       2930 non-null   object
31  Bsmt Qual        2850 non-null   object
32  Bsmt Cond        2850 non-null   object
33  Bsmt Exposure    2847 non-null   object
34  BsmtFin Type 1   2850 non-null   object
35  BsmtFin SF 1     2929 non-null   float64
36  BsmtFin Type 2   2849 non-null   object
37  BsmtFin SF 2     2929 non-null   float64
38  Bsmt Unf SF      2929 non-null   float64
39  Total Bsmt SF    2929 non-null   float64
40  Heating          2930 non-null   object
41  Heating QC       2930 non-null   object
42  Central Air      2930 non-null   object
43  Electrical       2929 non-null   object
44  1st Flr SF       2930 non-null   int64
45  2nd Flr SF       2930 non-null   int64
46  Low Qual Fin SF  2930 non-null   int64
47  Gr Liv Area      2930 non-null   int64
48  Bsmt Full Bath   2928 non-null   float64
49  Bsmt Half Bath   2928 non-null   float64
50  Full Bath        2930 non-null   int64
51  Half Bath        2930 non-null   int64
52  Bedroom AbvGr    2930 non-null   int64
53  Kitchen AbvGr    2930 non-null   int64
54  Kitchen Qual     2930 non-null   object
55  TotRms AbvGrd    2930 non-null   int64
56  Functional       2930 non-null   object
57  Fireplaces       2930 non-null   int64
58  Fireplace Qu     1508 non-null   object
59  Garage Type      2773 non-null   object
60  Garage Yr Blt    2771 non-null   float64
61  Garage Finish    2771 non-null   object
62  Garage Cars      2929 non-null   float64
63  Garage Area      2929 non-null   float64
64  Garage Qual      2771 non-null   object
65  Garage Cond      2771 non-null   object
66  Paved Drive      2930 non-null   object
67  Wood Deck SF     2930 non-null   int64
68  Open Porch SF    2930 non-null   int64
69  Enclosed Porch   2930 non-null   int64
70  3Ssn Porch       2930 non-null   int64
71  Screen Porch     2930 non-null   int64
```

```
 72  Pool Area        2930 non-null   int64
 73  Pool QC          13 non-null     object
 74  Fence            572 non-null    object
 75  Misc Feature     106 non-null    object
 76  Misc Val         2930 non-null   int64
 77  Mo Sold          2930 non-null   int64
 78  Yr Sold          2930 non-null   int64
 79  Sale Type        2930 non-null   object
 80  Sale Condition   2930 non-null   object
 81  SalePrice        2930 non-null   int64
dtypes: float64(11), int64(28), object(43)
memory usage: 1.8+ MB
```

[21]:
```
df_house.isna().sum().sort_values(ascending = False)
#isna() => Boş verileri gösterir. (?)
```

[21]:
```
Pool QC         2917
Misc Feature    2824
Alley           2732
Fence           2358
Mas Vnr Type    1775
                …
PID                0
Central Air        0
1st Flr SF         0
2nd Flr SF         0
SalePrice          0
Length: 82, dtype: int64
```

[23]:
```
#Dropping missing data
```

[27]:
```
na_series = df_house.isna().sum()
data_len = len(df_house) * 0.05
na_series[(na_series < data_len) & (na_series != 0)]
#dropna (?)
```

[27]:
```
Mas Vnr Area      23
Bsmt Qual         80
Bsmt Cond         80
Bsmt Exposure     83
BsmtFin Type 1    80
BsmtFin SF 1       1
BsmtFin Type 2    81
BsmtFin SF 2       1
Bsmt Unf SF        1
Total Bsmt SF      1
Electrical         1
```

```
Bsmt Full Bath     2
Bsmt Half Bath     2
Garage Cars        1
Garage Area        1
dtype: int64
```

[29]:
```
col_names = list(na_series[(na_series <= data_len) & (na_series != 0)].keys())
```

[31]:
```
df_house = df_house.dropna(subset=col_names)
df_house.shape
```

[31]:
```
(2821, 82)
```

[41]:
```
#Imputing Values => Boşta kalan verileri doldurma işlemi.
#Sayısal veri => Ortalama veya medyanı kullanıcan.
#Kategorik(Sözel) ver => En çok tekrar eden verileri kullanıcan.
```

[43]:
```
#Imputation with sckit-learn (Açıklama yok)
```

[47]:
```
object_cols = list(df_house.select_dtypes(include='object').columns)
```

[51]:
```
X_cat = df_house[object_cols]
X_cat
```

[51]:
```
     MS Zoning Street Alley Lot Shape Land Contour Utilities Lot Config  \
0           RL   Pave   NaN       IR1          Lvl    AllPub     Corner
1           RH   Pave   NaN       Reg          Lvl    AllPub     Inside
2           RL   Pave   NaN       IR1          Lvl    AllPub     Corner
3           RL   Pave   NaN       Reg          Lvl    AllPub     Corner
4           RL   Pave   NaN       IR1          Lvl    AllPub     Inside
…          …    …    …       …           …        …         …
2925        RL   Pave   NaN       IR1          Lvl    AllPub    CulDSac
2926        RL   Pave   NaN       IR1          Low    AllPub     Inside
2927        RL   Pave   NaN       Reg          Lvl    AllPub     Inside
2928        RL   Pave   NaN       Reg          Lvl    AllPub     Inside
2929        RL   Pave   NaN       Reg          Lvl    AllPub     Inside

     Land Slope Neighborhood Condition 1  … Garage Type Garage Finish  \
0           Gtl        NAmes        Norm  …      Attchd           Fin
1           Gtl        NAmes       Feedr  …      Attchd           Unf
2           Gtl        NAmes        Norm  …      Attchd           Unf
3           Gtl        NAmes        Norm  …      Attchd           Fin
4           Gtl      Gilbert        Norm  …      Attchd           Fin
…          …           …          …  …         …           …
2925        Gtl      Mitchel        Norm  …      Detchd           Unf
2926        Mod      Mitchel        Norm  …      Attchd           Unf
2927        Gtl      Mitchel        Norm  …         NaN           NaN
```

```
2928          Mod      Mitchel        Norm  …      Attchd          RFn
2929          Mod      Mitchel        Norm  …      Attchd          Fin

     Garage Qual Garage Cond Paved Drive Pool QC  Fence Misc Feature  \
0             TA          TA           P     NaN    NaN          NaN
1             TA          TA           Y     NaN  MnPrv          NaN
2             TA          TA           Y     NaN    NaN         Gar2
3             TA          TA           Y     NaN    NaN          NaN
4             TA          TA           Y     NaN  MnPrv          NaN
…           …          …          …      …     …      …
2925          TA          TA           Y     NaN  GdPrv          NaN
2926          TA          TA           Y     NaN  MnPrv          NaN
2927         NaN         NaN           Y     NaN  MnPrv         Shed
2928          TA          TA           Y     NaN    NaN          NaN
2929          TA          TA           Y     NaN    NaN          NaN

     Sale Type Sale Condition
0           WD         Normal
1           WD         Normal
2           WD         Normal
3           WD         Normal
4           WD         Normal
…         …            …
2925        WD         Normal
2926        WD         Normal
2927        WD         Normal
2928        WD         Normal
2929        WD         Normal

[2821 rows x 43 columns]
```

[55]:
```
X_nums = df_house.drop(object_cols, axis=1)
X_nums
```

[55]:
```
      Order        PID  MS SubClass  Lot Frontage  Lot Area  Overall Qual  \
0         1  526301100           20         141.0     31770             6
1         2  526350040           20          80.0     11622             5
2         3  526351010           20          81.0     14267             6
3         4  526353030           20          93.0     11160             7
4         5  527105010           60          74.0     13830             5
…       …        …          …           …        …
2925   2926  923275080           80          37.0      7937             6
2926   2927  923276100           20           NaN      8885             5
2927   2928  923400125           85          62.0     10441             5
2928   2929  924100070           20          77.0     10010             5
2929   2930  924151050           60          74.0      9627             7
```

```
      Overall Cond  Year Built  Year Remod/Add  Mas Vnr Area  …  \
0                5        1960            1960         112.0  …
1                6        1961            1961           0.0  …
2                6        1958            1958         108.0  …
3                5        1968            1968           0.0  …
4                5        1997            1998           0.0  …
…              …         …               …            …  …
2925             6        1984            1984           0.0  …
2926             5        1983            1983           0.0  …
2927             5        1992            1992           0.0  …
2928             5        1974            1975           0.0  …
2929             5        1993            1994          94.0  …

      Wood Deck SF  Open Porch SF  Enclosed Porch  3Ssn Porch  Screen Porch  \
0              210             62               0           0             0
1              140              0               0           0           120
2              393             36               0           0             0
3                0              0               0           0             0
4              212             34               0           0             0
…            …              …             …          …           …
2925           120              0               0           0             0
2926           164              0               0           0             0
2927            80             32               0           0             0
2928           240             38               0           0             0
2929           190             48               0           0             0

      Pool Area  Misc Val  Mo Sold  Yr Sold  SalePrice
0             0         0        5     2010     215000
1             0         0        6     2010     105000
2             0     12500        6     2010     172000
3             0         0        4     2010     244000
4             0         0        3     2010     189900
…         …       …       …       …        …
2925          0         0        3     2006     142500
2926          0         0        6     2006     131000
2927          0       700        7     2006     132000
2928          0         0        4     2006     170000
2929          0         0       11     2006     188000

[2821 rows x 39 columns]
```

[57]:
```
y = X_nums['SalePrice'].values.reshape(-1, 1)
X_nums.drop('SalePrice', inplace=True, axis=1)
```

[61]:
```
from sklearn.model_selection import train_test_split
```

```
X_train_cat, X_test_cat, y_train_cat, y_test_cat =train_test_split(X_cat, y,␣
 ↪test_size=0.2, random_state=42)
X_train_nums, X_test_nums, y_train_nums, y_test_nums =train_test_split(X_nums,␣
 ↪y, test_size=0.2, random_state=42)
#Hem Kategorik hem de numerik olan verileri kendi içerisinde eğitim ve test␣
 ↪içerisinde 2'ye ayırıyoruz.
```

[67]:
```
from sklearn.impute import SimpleImputer

imp_cat = SimpleImputer(strategy='most_frequent')
X_train_cat = imp_cat.fit_transform(X_train_cat)
X_test_cat = imp_cat.fit_transform(X_test_cat)
```

[69]:
```
imp_num = SimpleImputer() #default mean

X_train_nums = imp_num.fit_transform(X_train_nums)
X_test_nums = imp_num.fit_transform(X_test_nums)
```

[71]:
```
#imputers are knwon transformers

import numpy as np
X_train = np.append(X_train_nums,X_train_cat,axis=1)
X_train
```

[71]:
```
array([[963.0, 916403010.0, 20.0, …, 'Shed', 'WD ', 'Normal'],
       [2684.0, 903235020.0, 30.0, …, 'Shed', 'WD ', 'Normal'],
       [2874.0, 910200060.0, 50.0, …, 'Shed', 'WD ', 'Normal'],
       …,
       [1172.0, 533215080.0, 120.0, …, 'Shed', 'CWD', 'Abnorml'],
       [1346.0, 903233080.0, 50.0, …, 'Shed', 'WD ', 'Normal'],
       [893.0, 908186080.0, 180.0, …, 'Shed', 'WD ', 'Normal']],
      dtype=object)
```

[73]:
```
X_test = np.append(X_test_nums, X_test_cat, axis=1)
X_test
```

[73]:
```
array([[1132.0, 531363060.0, 20.0, …, 'Shed', 'WD ', 'Normal'],
       [2433.0, 528235130.0, 60.0, …, 'Shed', 'WD ', 'Normal'],
       [794.0, 905475160.0, 20.0, …, 'Shed', 'WD ', 'Normal'],
       …,
       [2413.0, 528218080.0, 60.0, …, 'Shed', 'New', 'Partial'],
       [967.0, 916460110.0, 60.0, …, 'Shed', 'WD ', 'Abnorml'],
       [1049.0, 527454120.0, 120.0, …, 'Shed', 'WD ', 'Normal']],
      dtype=object)
```

[75]:
```
#Imputing within a pipeline
```

[81]:
```
from sklearn.pipeline import Pipeline
import pandas as pd

df_music = pd.read_csv('../../music_clean.csv')
df_music = df_music.dropna(subset=['genre', 'popularity', 'loudness',␣
 ↪'liveness', 'tempo'])
# df_music['genre'] = np.where(df_music['genre']=='Rock', 1, 0)  -> Sözel veri␣
 ↪türünü, sayısal verilere dönüştürmek
```

[91]:
```
X = df_music.drop('genre', axis=1).values
y = df_music['genre'].values
```

[97]:
```
#Bir ardışık düzende, son adım hariç her adımın bir dönüştürücü olması␣
 ↪gerektiği unutulmamalıdır.

from sklearn.linear_model import LogisticRegression
steps = [
    ('imputation', SimpleImputer()),
    ('logistic_regression', LogisticRegression())
]

pipeline = Pipeline(steps=steps)

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

pipeline.fit(X_train, y_train)
pipeline.score(X_test, y_test) # Acurracy parametresini verir.
```

```
/opt/anaconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[97]:
```
0.775
```

[101]:
```
#Centering and Scaling
#Veriler neden ölçeklendirilir ?
```

[103]:
```
import pandas as pd

df_music = pd.read_csv('../../music_genre.csv')

df_music.describe().T
```

[103]:

	count	mean	std	min	\
instance_id	44996.0	55883.823784	20728.799250	20002.000000	
popularity	44996.0	44.261112	15.556250	0.000000	
acousticness	44996.0	0.306620	0.341413	0.000000	
danceability	44996.0	0.558553	0.178870	0.059600	
duration_ms	44996.0	221163.902236	127706.510819	-1.000000	
energy	44996.0	0.599551	0.264546	0.000792	
instrumentalness	44996.0	0.181912	0.325904	0.000000	
liveness	44996.0	0.193953	0.161733	0.009670	
loudness	44996.0	-9.136920	6.157838	-47.046000	
speechiness	44996.0	0.093810	0.101488	0.022300	
tempo	44996.0	119.952277	30.642777	34.347000	
valence	44996.0	0.456324	0.247154	0.000000	

	25%	50%	75%	max
instance_id	37999.75000	55856.500000	73854.50000	91759.000
popularity	34.00000	45.000000	56.00000	99.000
acousticness	0.02010	0.145000	0.55100	0.996
danceability	0.44200	0.568000	0.68700	0.986
duration_ms	174706.50000	219448.500000	268630.25000	4497994.000
energy	0.43200	0.642000	0.81600	0.999
instrumentalness	0.00000	0.000159	0.15500	0.996
liveness	0.09690	0.126000	0.24400	1.000
loudness	-10.86000	-7.283000	-5.17600	3.744
speechiness	0.03610	0.048900	0.09890	0.942
tempo	94.93975	119.879000	140.48225	220.276
valence	0.25700	0.448000	0.64700	0.992

[105]:
```
#Veriler nasıl ölçeklendirilir ?
```

[107]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np


X = df_music.drop('music_genre', axis=1).values
y = df_music['music_genre'].values

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

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

print(np.mean(X), np.std(X))

print(np.mean(X_train_scaled), np.std(X_train_scaled))
```

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

[109]:
```
# Scaling in a pipeline

from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier

steps = [
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(n_neighbors=6))
]

pipeline = Pipeline(steps)

knn_scaled = pipeline.fit(X_train, y_train)
y_pred = knn_scaled.predict(X_test)

knn_scaled.score(X_test, y_test)
```

[109]:
```
0.48088888888888887
```

[111]:
```
# Comparing performance using unscaled data

knn_unscaled = KNeighborsClassifier(n_neighbors=6).fit(X_train, y_train)
knn_unscaled.score(X_test, y_test)
```

[111]:
```
0.1272222222222222
```

[113]:
```
#CV and scaling in a pipeline
```

[116]:
```
from sklearn.model_selection import GridSearchCV
import numpy as np

steps = [
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
]
```

```
pipeline = Pipeline(steps)

parameters = {"knn__n_neighbors" : np.arange(1, 50)} #Parametreye isim verirken␣
 ↪dikkat edilmelidir yoksa kod çalışmaz.

cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)

(cv.best_score_, cv.best_params_)
```

[116]:
```
(0.5193075581485083, {'knn__n_neighbors': 38})
```

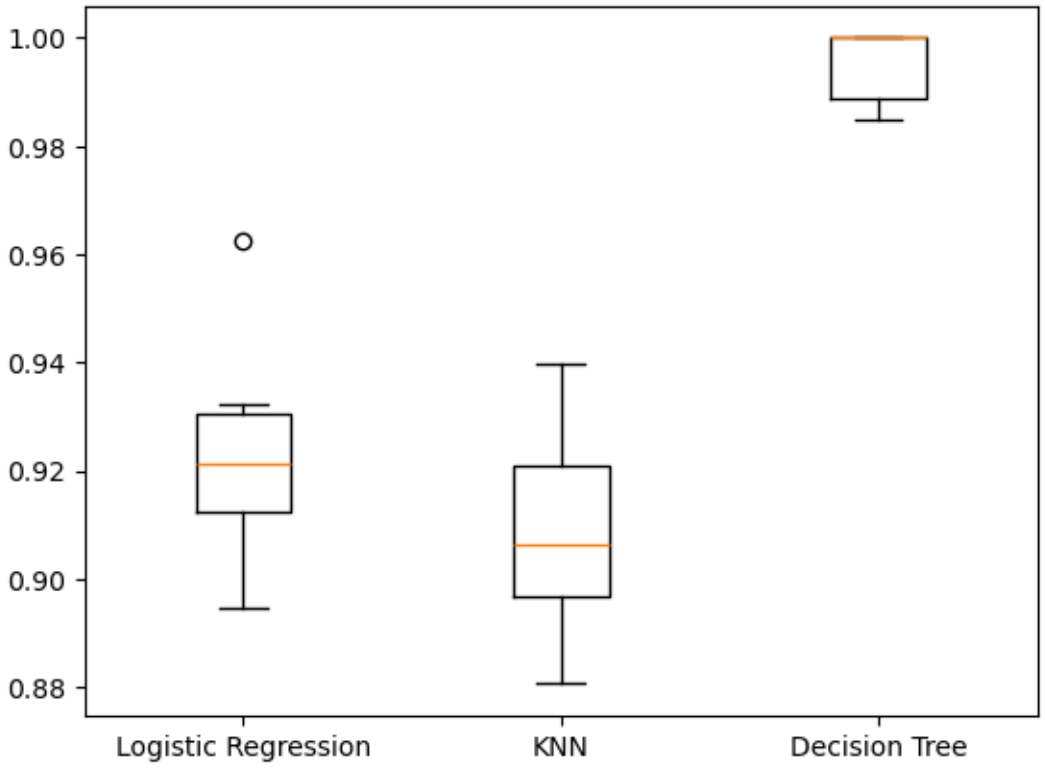
[117]:
```
#Metin var !!
```

[121]:
```
# Sınıflandırma modellerinin değerlendirilmesi

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

df_music = pd.read_csv("../../music_clean.csv")

X = df_music.drop('genre', axis=1).values
y = df_music['genre'].values

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

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

models = {
    "Logistic Regression" : LogisticRegression(),
    "KNN": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier()
}

results = []

for model in models.values():
    kf = KFold(n_splits=6, random_state=True, shuffle=True)
    cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kf)
```

```
    results.append(cv_results)

plt.boxplot(results, labels=models.keys())
```

```
/var/folders/q8/0jm376f10vv50t5b14_31pbr0000gn/T/ipykernel_23958/1007144225.py:3
5: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been
renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be
dropped in 3.11.
  plt.boxplot(results, labels=models.keys())
```

[121]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x17a8a3110>,
  <matplotlib.lines.Line2D at 0x17a901a30>,
  <matplotlib.lines.Line2D at 0x17a902e40>,
  <matplotlib.lines.Line2D at 0x17a9033b0>,
  <matplotlib.lines.Line2D at 0x17a91c320>,
  <matplotlib.lines.Line2D at 0x17a91c920>],
 'caps': [<matplotlib.lines.Line2D at 0x17a902330>,
  <matplotlib.lines.Line2D at 0x17a902210>,
  <matplotlib.lines.Line2D at 0x17a903320>,
  <matplotlib.lines.Line2D at 0x17a903ad0>,
  <matplotlib.lines.Line2D at 0x17a91c3e0>,
  <matplotlib.lines.Line2D at 0x17a91d280>],
 'boxes': [<matplotlib.lines.Line2D at 0x179789a00>,
  <matplotlib.lines.Line2D at 0x17a902ab0>,
  <matplotlib.lines.Line2D at 0x17a91c620>],
 'medians': [<matplotlib.lines.Line2D at 0x17a902e10>,
  <matplotlib.lines.Line2D at 0x17a903bc0>,
  <matplotlib.lines.Line2D at 0x17a91d610>],
 'fliers': [<matplotlib.lines.Line2D at 0x17a902900>,
  <matplotlib.lines.Line2D at 0x17a903fb0>,
  <matplotlib.lines.Line2D at 0x17a91d4c0>],
 'means': []}
```



[ ]:
```
# test performance

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    test_score = model.score(X_test_scaled, y_test)
    print(f"{name} Test Set Accuracy : {test_score} ")
```

[ ]:
```

```