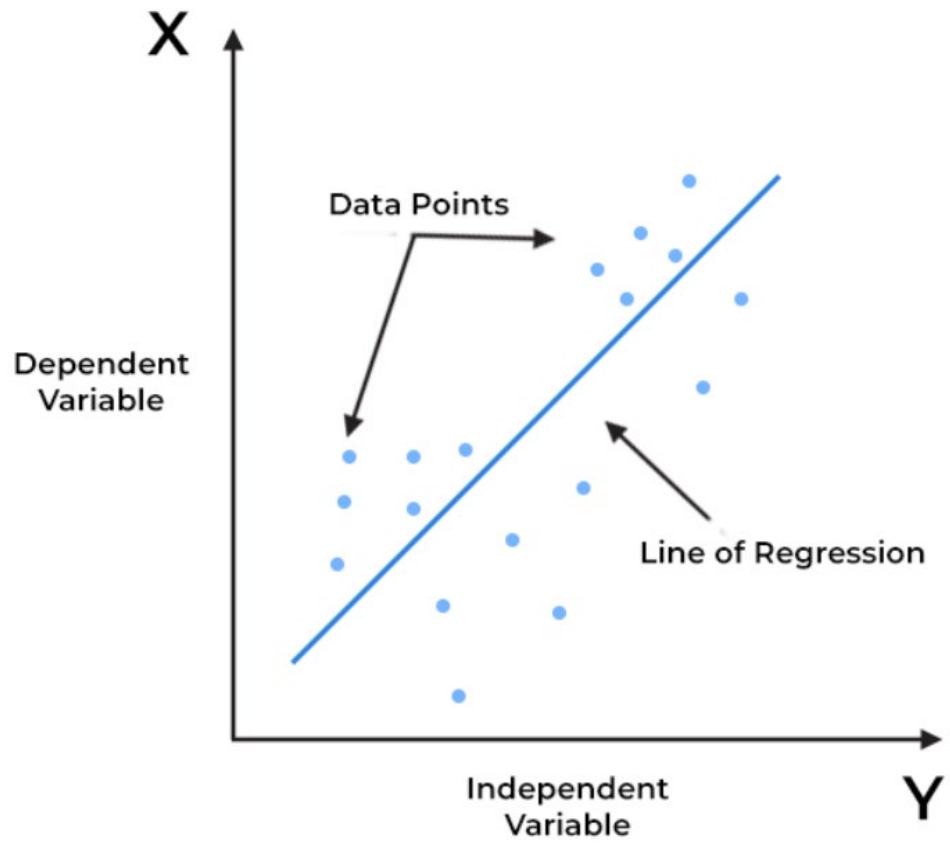
Giriş

Bu proje, Doğrusal regresyonun temel prensiplerini anlamak ve uygulamak üzerine odaklanmaktadır. Doğrusal regresyon, bir bağımlı değişken ile bir veya daha fazla bağımsız değişken arasındaki ilişkiyi modelleme konusunda güçlüdür.[1] Değişkenler arasındaki matematiksel ilişkiyi simüle eden ve satış, maaş, yaş, ürün fiyatı vb. gibi sürekli veya sayısal değişkenler için tahminler yapan denetimli bir öğrenme algoritmasıdır.

Bu analiz yöntemi, borsa tahminlerinde, portföy yönetiminde, bilimsel analizlerde vb. gözlemlendiği gibi verilerde en az iki değişken mevcut olduğunda avantajlıdır. Eğimli bir düz çizgi, doğrusal regresyon modelini temsil eder.[2]



Matematiksel olarak bu eğik çizgiler aşağıdaki denklemi takip eder:

Y = m*X + b X = bağımlı değişken (hedef)

Y = bağımsız değişken

m = çizginin eğimi (eğim 'koşu' üzerindeki 'yükseliş' olarak tanımlanır)[2]

Bu çalışmanın temel amacı, doğrusal regresyonun nasıl çalıştığını anlamak ve bu bilgileri bir veri seti üzerinde uygulayarak gerçek dünya problemlerine nasıl uyarlayabileceğimizi kavramaktır. Projemizde ele alacağımız temel konu verilen Facebook verilerini analiz edip belirlenen bağımlı ve bağımsız özelliklere göre doğrusal regresyon uygulamaktır.

Projedeki aşamalar; İlk olarak veri kümesini X ve Y'ye böleceğiz ve sağlanan yüzdesel bölmeyi kullanarak X ve Y'yi eğitim ve test setlerine ayıracağız. Daha sonra, (train_test_split) öğesini çağırarak eğitim ve test setini alacağız. Bir ağırlık (θ) vektörü tanımlayacağız ve yukarıdaki bilgileri kullanarak Gradyan İnişini (Gradient Descent) uygulayacağız. Eğitim ve test verileri için Toplam Kare Hatayı (Sum Squared Error) kaydedeceğiz. Son olarak, ağırlık matrisini, eğitim hatalarını ve test hatalarını döndüreceğiz. Eğitim ve test hatalarını çizeceğiz ve grafik üzerinde yorum yapacağız.

Metodoloji

Veri Setinin Yüklenmesi:

Çalışmanın temelini oluşturan veri seti, Facebook paylaşımlarına ait bilgiler içeren dataset_Facebook.csv dosyasından yükledik.

```
df = pd.read_csv('dataset_Facebook.csv', delimiter=';')
```

```
df.head(3)
```

Veri Ön İşleme Adımları:[3]

Sorunlar ve Çözümler:

Eksik Veriler: 'Paid', 'like', ve 'share' özelliklerindeki eksik değerler, özelliklerin ortalamasıyla doldurularak çözüldü.

```
missing_values = df.isnull().sum()
missing_values
```

```
df['Paid'].fillna(df['Paid'].mean(), inplace=True)
df['like'].fillna(df['like'].mean(), inplace=True)
df['share'].fillna(df['share'].mean(), inplace=True)
```

Kategorik veriyi dönüştürme:

Sorunlar ve Çözümler:

Kategorik olan Type sütununu önce labelencode yapıyoruz 1,2,3,4 gibi değerler atıyor. Fakat bu şekilde dönüşüm yaparsak model aralarında önem sırası olduğunu düşünecek bunu çözmek için OneHotEncoding dönüşü yapıyoruz. Böylelikle aralarında büyüklük ilişkisi ortadan kalkıyor.

```
# Kategorik veri değiştirme
Type = df.iloc[:, 1:2].values
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
Type[:,0] = le.fit_transform(df.iloc[:,1])
#one hot encoding
ohe = preprocessing.OneHotEncoder()
Type = ohe.fit_transform(Type).toarray()
```

```
# Verileri Birleştirme
sonuc = pd.DataFrame(data=Type, index = range(500), columns = ['Link', 'Photo', 'Status
df2 = pd.concat([sonuc,df], axis=1)
df2.drop('Type', axis =1, inplace=True)
```

Veri Setinin Bölünmesi:

Sorunlar ve Çözümler:

Overfitting veya Underfitting: Veri setinin doğru oranda bölünmemesi durumunda, modelin aşırı öğrenme (overfitting) veya yetersiz öğrenme (underfitting) sorunları ortaya çıkabilir. Bu nedenle, veri seti uygun bir oranla bölünmelidir.

```
# Verileri bölme
X = df2.iloc[:,:10]
X.head(1)
```

	Link	Photo	Status	Video	Page total likes	Category	Post Month	Post Weekday	Post Hour	Paid
0	0.0	1.0	0.0	0.0	139441	2	12	4	3	0.0

```
y= df2.iloc[:,-1:]
y.head(1)
```

	Total Interactions
0	100

```
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=4
```

StandardScaler ile özniteliklerimizi ölçeklendirme:

Bu gradient descent algoritmamızın daha hızlı ve iyi karar vermesini sağlamaktadır

```
from sklearn.preprocessing import StandardScaler
sc1 = StandardScaler()
X = sc1.fit_transform(X)
sc2 = StandardScaler()
y= sc2.fit_transform(y)
```

Bias Sütunu Eklenmesi:

Sorunlar ve Çözümler:

Model Basitliği: Bias sütunu, modelin daha genel ve çeşitli durumlar üzerinde etkili olmasını sağlar.

```
def add_bias_feature(X):
    return np.column_stack((np.ones(len(X)), X))
```

X_train'in birinci sütununa bias hesaplamak için 1'ler ekliyoruz

```
X_train = add_bias_feature(X_train)
```

X_trainin satır sayısı kadar sıfırlardan oluşan bir array oluşturuyoruz

```
theta_initial = np.zeros((X_train.shape[1],1))
```

Lineer Regresyon Modelinin Oluşturulması:

$$h_w(x) = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n = Xw$$

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_w\left(x^{(i)}\right))^2$$

$$J(w) = \frac{1}{2m}np.sum((y - Xw)^2)$$

$$\phi := y - Xw$$

$$w_j := w_j + \frac{\alpha}{m}\sum_{i=1}^{m}x_j^{(i)}(y^{(i)} - h_w\left(x^{(i)}\right))$$

$$w := w + \frac{\alpha}{m}(X^T(y - Xw))$$

Gradyan iniş algoritması kullanılarak model eğitilir. Yavaş Eğitim olmaması için öğrenme oranı (alpha) ve iterasyon sayısı (num_iterations) gibi hiperparametrelerin uygun seçilmesi, modelin hızlı ve etkili bir şekilde eğitilmesine yardımcı olur.[1]

```
def compute_cost(X, y, theta):
    m = len(y)
    h = X.dot(theta)
    cost = (1 / (2 * m)) * np.sum((h - y) ** 2)
    return cost
```

```
def gradient_descent(X, y, theta, alpha, num_iterations):
    m = len(y)
    cost_history = []

    for _ in range(num_iterations):
        h = X.dot(theta)
        error = h - y
        gradient = (1 / m) * X.T.dot(error)
        theta = theta - alpha * gradient
        cost = compute_cost(X, y, theta)
        cost_history.append(cost)

    return theta, cost_history
```

```
# Hyperparameters
alpha = 0.01
num_iterations = 1000
# Run gradient descent
theta_final, cost_history = gradient_descent(X_train, y_train, theta_initial, alpha, num_iterations)
```

gradient_descent sonucunda oluşan theta_final ilk değeri bias'tır. Diğerleri thetalardır.

```
theta = theta_final[1:]
bias = theta_final[0]
```

Önceden X_traine eklediğimiz eklediğimiz biasları kaldıralım.

```
X_train = X_train[:, 1:]
```

Tahmin

```
def predict(X, weight, bias):
    return np.dot(X, weight) + bias
```

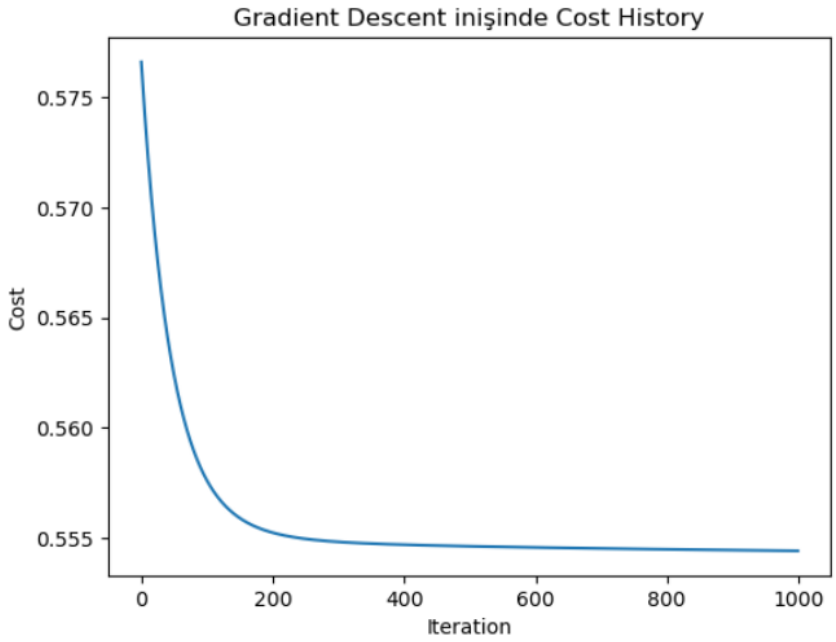
```
y_pred_test = predict(X_test, theta, bias)
y_pred_train = predict(X_train, theta, bias)
```

“predict” fonksiyonu, öğrenilen model parametreleri ve test verileri kullanılarak tahminler yapar. Bu tahminler, modelin genelleme yeteneğini ve performansını değerlendirmek için kullanılır.

Maliyet Fonksiyonunun Çizilmesi:

Gradyan iniş sırasındaki maliyet geçmişi, modelin eğitimi sırasında performansını değerlendirmek adına çizilir.

```
plt.plot(cost_history)
plt.xlabel('Iteration')
plt.ylabel('Cost')
plt.title('Gradient Descent inişinde Cost History')
plt.show()
```



Modelin Değerlendirilmesi ve Test Hatalarının Hesaplanması:

Eğitim ve test hataları hesaplanarak modelin performansı değerlendirilir.

Modelin eğitim ve test hataları arasında büyük bir fark varsa, modelin aşırı öğrenme (overfitting) veya yetersiz öğrenme (underfitting) sorunları olabilir. Bu sorunları çözmek için hiperparametrelerin dikkatlice ayarlanması gerekir. [4]

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

```python
# Eğitim ve test hatalarını hesaplama
def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
```

```python
train_mse = mean_squared_error(y_train, y_pred_train)
test_mse = mean_squared_error(y_test, y_pred_test)
print(f"Eğitim seti hatası mse: {train_mse}")
print(f"Test seti hatası mse: {test_mse}")
```

```
Eğitim seti hatası mse: 1.1088337510633506
Test seti hatası mse: 0.36877063352986567
```

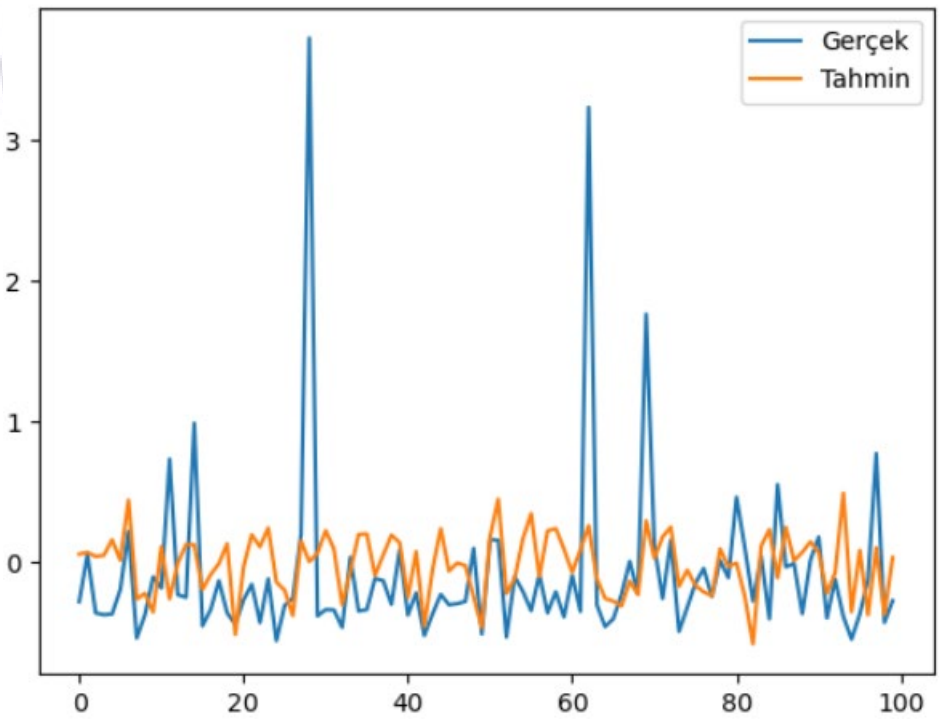
```python
from sklearn.metrics import r2_score
```

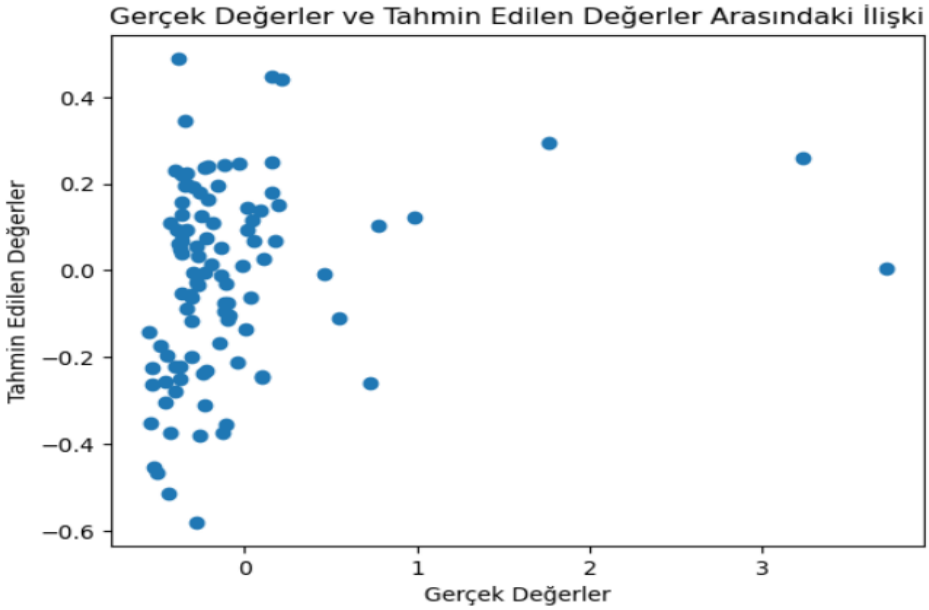
```python
train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)
print(f"Eğitim seti hatası r2: {train_r2}")
print(f"Test seti hatası r2: {test_r2}")
```

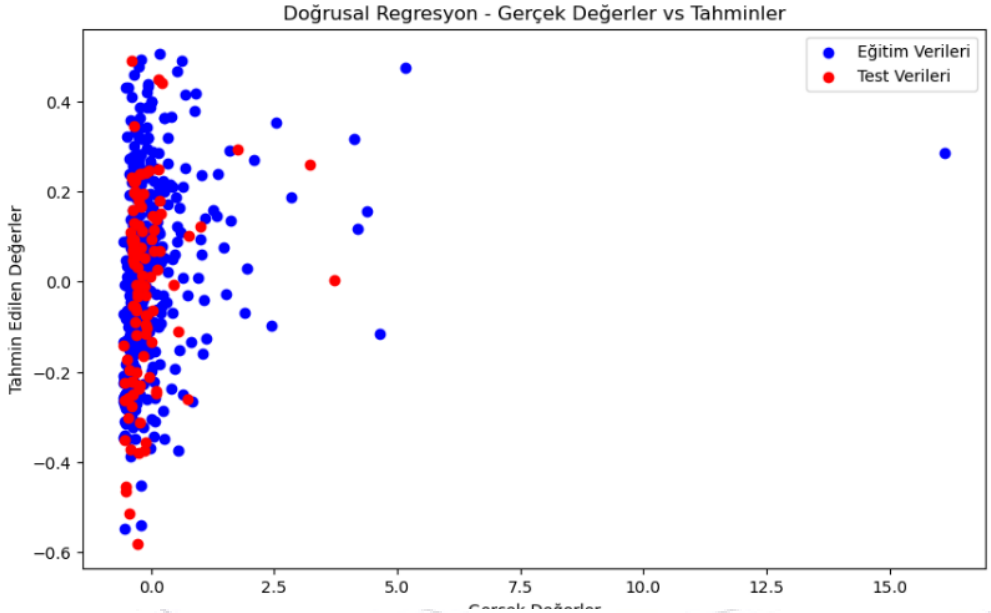
```
Eğitim seti hatası r2: 0.03891202904236302
Test seti hatası r2: 0.01780083624642348
```

```python
data = pd.DataFrame({"Gerçek": y_test.flatten(), "Tahmin": y_pred_test.flatten()})
data.plot(kind="line")
plt.show()
```

SONUÇLAR




Gerçek Değerler ve Tahmin Edilen Değerler Arasındaki İlişki
Tahmin Edilen Değerler
Gerçek Değerler
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
1
2
3


Doğrusal Regresyon - Gerçek Değerler vs Tahminler
Eğitim Verileri
Test Verileri
Tahmin Edilen Değerler
0.4
0.2
0.0
−0.2
−0.4
−0.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0

ÇIKTI PARAMETRELERİ

```
Eğitim seti hatası mse: 1.1088337510633506
Test seti hatası mse: 0.36877063352986567
```

```
Eğitim seti hatası r2: 0.03891202904236302
Test seti hatası r2: 0.01780083624642348
```

```
print("Ağırlık Matrisi:", theta_final[1:])
print("Bias Değeri:", theta_final[0])
```

```
Ağırlık Matrisi: [[-0.03781379]
 [ 0.00523992]
 [ 0.00351081]
 [ 0.04162039]
 [ 0.07994523]
 [ 0.14431995]
 [-0.0349313 ]
 [-0.06703311]
 [ 0.00841553]
 [ 0.10911716]]
Bias Değeri: [0.0139955]
```

ÖZGÜNLÜK

- Projede istenen sütunları kullandığımzda model genel olarak kötü performans sergiliyor o yüzden başka öznitelikler alarak yeni bir model oluşturalım

  Burada veri setindeki Total Interactions hariç tüm sutunları X'e atıyoruz. y'ye Total Interactions atıyoruz.

```
# Verileri bölme
X = df3.iloc[:,:-1]
y= df3.iloc[:,-1:]
```

```
import statsmodels.api as sm
# Sabit bir sütun ekleyerek modele eklemek
X = sm.add_constant(X)
# Model oluşturma
model = sm.OLS(y, X).fit()
model.summary()
```

- Geriye Doğru Eleme Yöntemi kullanarak en yüksek p değerine sahip değişkeni modelden çıkararak en sonunda p değerlerini 0.5'in altında bırakarak X özniteliklerimizi seçicez

  Her seferinde tek bir özniteliği çıkarıp tekrar konrol eden bir dögü oluşturalım

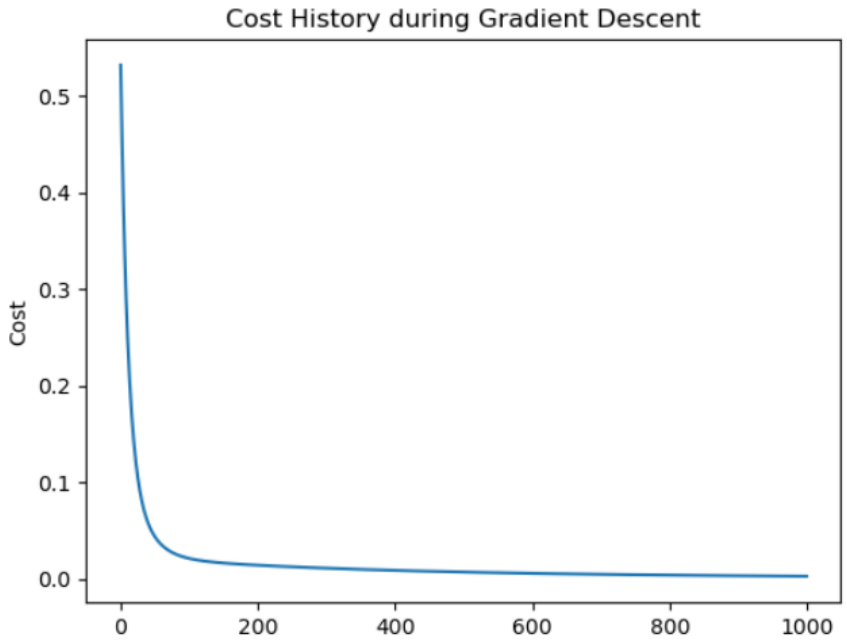
```python
def backward_elimination(x, y, significance_level=0.5):
    num_vars = x.shape[1]
    for i in range(0, num_vars):
        model = sm.OLS(y, x).fit()
        max_pval = max(model.pvalues)
        if max_pval > significance_level:
            max_index = np.where(model.pvalues == max_pval)[0][0]
            x = np.delete(x, max_index, 1)
        else:
            break
    print(model.summary())
    return x
```

- Geriye Doğru Eleme Yöntemi sonucunda oluşan optimal öznitelikleri X'atıyoruz.

```python
X_optimized = backward_elimination(X, y)
X = X_optimized
```

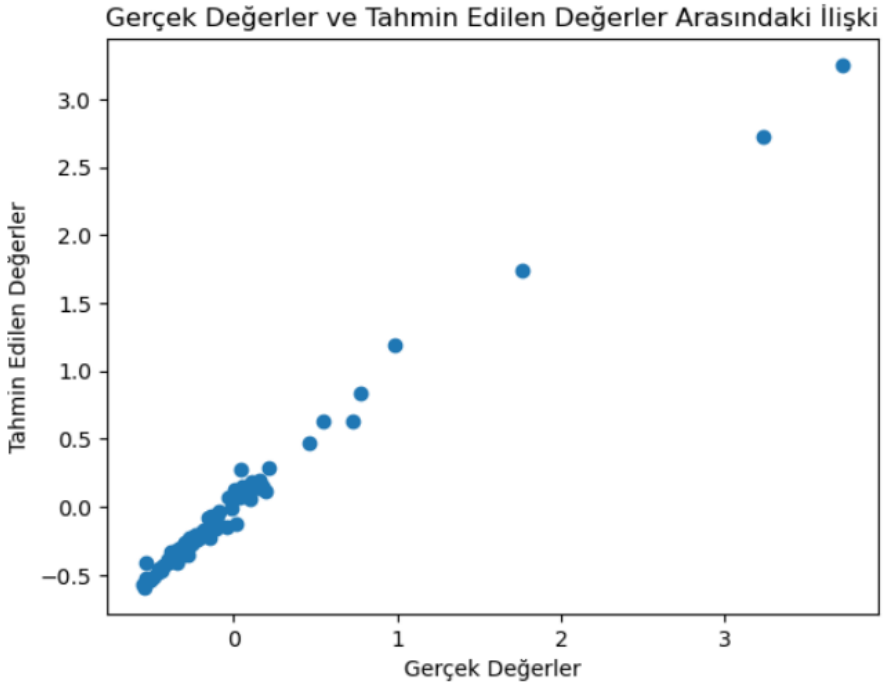
- Optimize öznitelikleri kullanarak modelimiz tekrar eğitelim

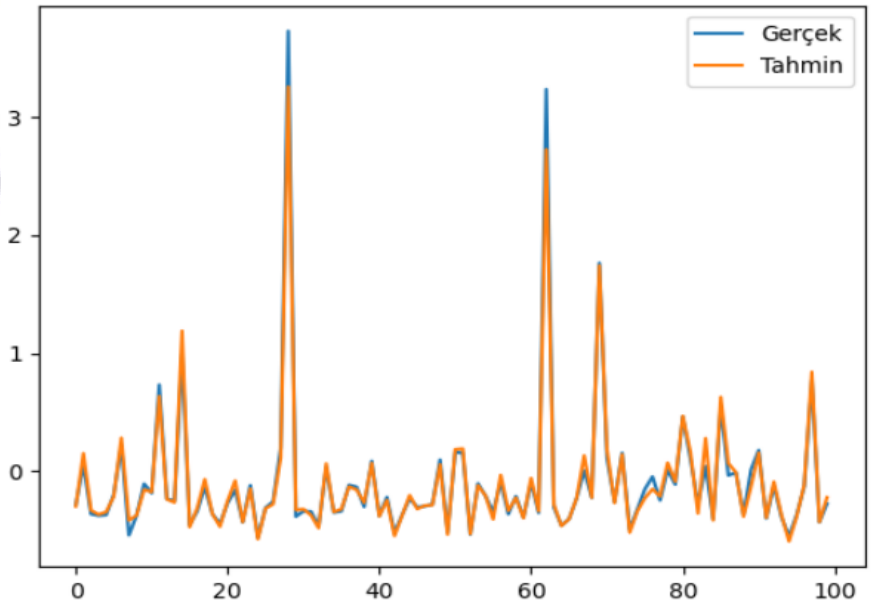
```python
from sklearn.preprocessing import StandardScaler
sc1 = StandardScaler()
X = sc1.fit_transform(X)
sc2 = StandardScaler()
y= sc2.fit_transform(y)

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)

X_train = add_bias_feature(X_train)

theta_initial = np.zeros((X_train.shape[1],1))
alpha = 0.01
num_iterations = 1000
theta_final, cost_history = gradient_descent(X_train, y_train, theta_initial, alpha, num_iterations)

import matplotlib.pyplot as plt
plt.plot(cost_history)
plt.xlabel('Iteration')
plt.ylabel('Cost')
plt.title('Cost History during Gradient Descent')
plt.show()

theta = theta_final[1:]
bias = theta_final[0]

X_train = X_train[:, 1:]

y_pred_test = predict(X_test, theta, bias)
y_pred_train = predict(X_train, theta, bias)

train_mse = mean_squared_error(y_train, y_pred_train)
test_mse = mean_squared_error(y_test, y_pred_test)

print(f"Eğitim seti hatası mse: {train_mse}")
print(f"Test seti hatası mse: {test_mse}")

train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)

print(f"Eğitim seti hatası r2: {train_r2}")
print(f"Test seti hatası r2: {test_r2}")
```

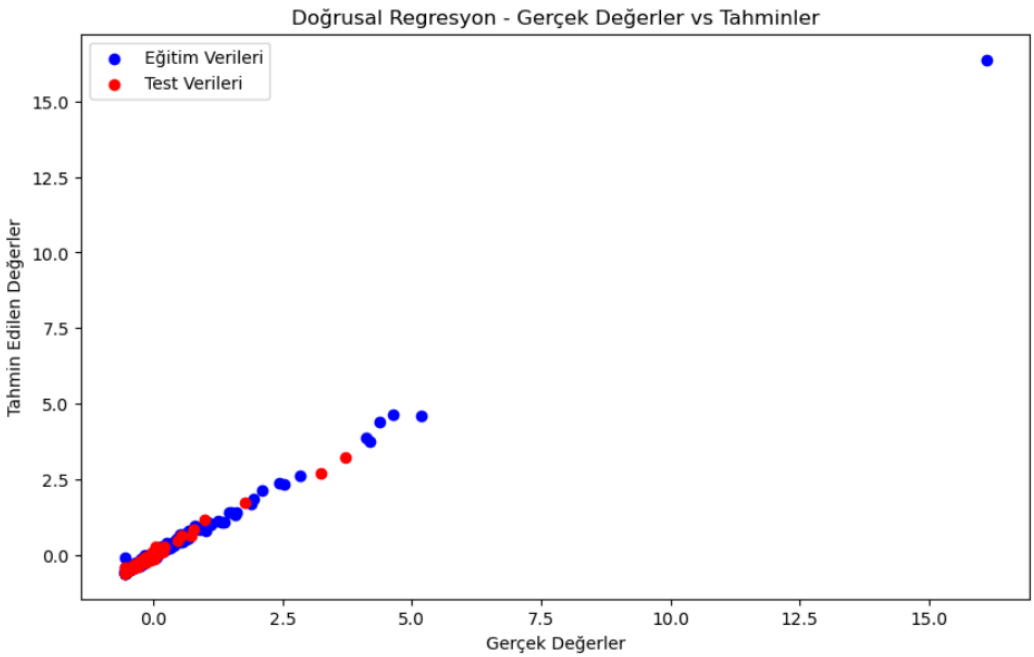



```
Eğitim seti hatası mse: 0.005065699406575469
Test seti hatası mse: 0.007699839886289811
Eğitim seti hatası r2: 0.9956092761791585
Test seti hatası r2: 0.9794919236790643
```

```python
plt.scatter(y_test, y_pred_test)
plt.xlabel("Gerçek Değerler")
plt.ylabel("Tahmin Edilen Değerler")
plt.title("Gerçek Değerler ve Tahmin Edilen Değerler Arasındaki İlişki")
plt.show()


data = pd.DataFrame({"Gerçek": y_test.flatten(), "Tahmin": y_pred_test.flatten()})
data.plot(kind="line")
plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(y_train, y_pred_train, color='blue', label='Eğitim Verileri')
plt.scatter(y_test, y_pred_test, color='red', label='Test Verileri')
plt.xlabel('Gerçek Değerler')
plt.ylabel('Tahmin Edilen Değerler')
plt.title('Doğrusal Regresyon - Gerçek Değerler vs Tahminler')
plt.legend()
plt.show()
```


Gerçek Değerler ve Tahmin Edilen Değerler Arasındaki İlişki
Tahmin Edilen Değerler
Gerçek Değerler
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0
1
2
3


Gerçek
Tahmin
3
2
1
0
0
20
40
60
80
100




```
print("Ağırlık Matrisi:", theta_final[1:])
print("Bias Değeri:", theta_final[0])
```

```
Ağırlık Matrisi: [[-0.03781379]
 [ 0.00523992]
 [ 0.00351081]
 [ 0.04162039]
 [ 0.07994523]
 [ 0.14431995]
 [-0.0349313 ]
 [-0.06703311]
 [ 0.00841553]
 [ 0.10911716]]
Bias Değeri: [0.0139955]
```

2018

KAYNAKÇA

[1] “Everything you need to Know about Linear Regression.” Accessed: Dec. 15, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/

[2] “What is Linear Regression?- Spiceworks - Spiceworks.” Accessed: Dec. 14, 2023. [Online]. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-linear-regression/

[3] “Python ile doğrusal regresyon uygulaması · Miuul Not Defteri.” Accessed: Dec. 15, 2023. [Online]. Available: https://miuul.com/not-defteri/python-ile-dogrusal-regresyon-uygulamasi

[4] “GRADIENT DESCENT (DERECELİ AZALMA) NEDİR ? | by Büşra Akıncı | Medium.” Accessed: Dec. 25, 2023. [Online]. Available: https://medium.com/@bsradogn/gradient-descent-dereceli%CC%87-azalma-nedi%CC%87r-49333e74c0f0