

## Ödev Sahibi

Büşra Boyacı -20174753032

## Amaç (Problemin Tanımlanması)

Deniz sevk santrallerinin bakım veri kümesini kullandık. Kolonlardaki GT Kompresör Bozulma Durumu katsayısı Çıktı 1'i ve GT Tribün bozulma durumu katsayısını çıktı2'yi temsil ediyor. Verilerdeki basınç değerlerini kullanarak bozulma durumu katsayılarını ne kadar tahmin edebileceğimizi 3 farklı algoritmayla test edeceğiz.

## Veriyi Anlama

Verilerimizi yüklüyoruz.

```
#verinin yüklenmesi
import pandas as pd
veriler = pd.read_csv('grup2.csv')
print(veriler)
```

	lp	knots	GTT	GTn	...	TIC	mf	cikti1	cikti2
0	1.14	3	289.96	1349.49	...	7.14	0.08	0.95	0.97
1	2.09	6	6960.18	1376.17	...	10.65	0.29	0.95	0.97
2	3.14	9	8379.23	1386.76	...	13.09	0.26	0.95	0.97
3	4.16	12	14724.40	1547.46	...	18.11	0.36	0.95	0.97
4	5.14	15	21636.43	1924.31	...	26.37	0.52	0.95	0.97
...	...	...	...	...	...	...	...	...	...
11929	5.14	15	21624.93	1924.34	...	23.80	0.47	1.00	1.00
11930	6.17	18	29763.21	2306.74	...	32.67	0.65	1.00	1.00
11931	7.15	21	39003.87	2678.05	...	42.10	0.83	1.00	1.00
11932	8.21	24	50992.58	3087.43	...	58.06	1.15	1.00	1.00
11933	9.30	27	72775.13	3560.40	...	86.07	1.70	1.00	1.00

İde olarak spyder kullandığımız için bu tip görüntüleri aşağıdaki gibi dataframe şeklinde görüntülememiz de mümkün.

Index	lp	knots	GTT	GTn	GGn	Ts	Tp	HP	T1	T2	P48	P1	P2	Pexh	TIC	mf	cikti1	cikti2
0	1.138	3	289.964	1349.49	6677.38	7.584	7.584	464.006	288	550.563	1.096	0.998	5.947	1.019	7.137	0.082	0.95	0.975
1	2.088	6	6960.18	1376.17	6828.47	28.204	28.204	635.401	288	581.658	1.331	0.998	7.282	1.019	10.655	0.287	0.95	0.975
2	3.144	9	8379.23	1386.76	7111.81	60.358	60.358	606.002	288	587.587	1.389	0.998	7.574	1.02	13.086	0.259	0.95	0.975
3	4.161	12	14724.4	1547.46	7792.63	113.774	113.774	661.471	288	613.851	1.658	0.998	9.007	1.022	18.109	0.358	0.95	0.975
4	5.14	15	21636.4	1924.31	8494.78	175.306	175.306	731.494	288	645.642	2.078	0.998	11.197	1.026	26.373	0.522	0.95	0.975
5	6.175	18	29792.7	2307.4	8828.36	246.278	246.278	800.434	288	676.397	2.501	0.998	13.356	1.03	35.76	0.708	0.95	0.975
6	7.148	21	38982.2	2678.09	9132.43	332.077	332.077	854.747	288	699.954	2.963	0.998	15.679	1.035	45.881	0.908	0.95	0.975

## Keşifsel Veri Analizi

Veriler hakkında daha ayrıntılı bilgilere sahip olabilmek için ekrana bastırdıktan sonra veri türlerini görüntülüyoruz.

lp	float64
knots	int64
GTT	float64
GTn	float64
GGn	float64
Ts	float64
TP	float64
HP	float64
T1	int64
T2	float64
P48	float64
P1	float64
P2	float64
Pexh	float64
TIC	float64
mf	float64
cikti1	float64
cikti2	float64
dtype:	object

Index	0
lp	float64
knots	int64
GTT	float64
GTn	float64
GGn	float64
Ts	float64
TP	float64
HP	float64
T1	int64
T2	float64
P48	float64
P1	float64
P2	float64
Pexh	float64

```
# veri türlerini görüntüleme
types = veriler.dtypes
print(types)
```

Verilerimiz sayısal ve çıktı değişkenlerimiz de sürekli sayılardan oluşuyor. Verilere normalizasyon dışında bir dönüştürme işlemi uygulamayacağız.

Verilerimizin istatistiksel değerlerini görüntülüyoruz.

```
#istatistik tablosu
pd.set_option('display.width', 60, 'precision', 2)
istatistik = veriler.describe()
print(istatistik)
```

```
İstatistik tablosu:
      lp      knots      GTT      ...      mf      ciktı1      ciktı2
count 11934.00 11934.00 11934.00 ... 11934.00 11934.00 1.19e+04
mean    5.17    15.00 27247.50 ...    0.66    0.97 9.87e-01
std     2.63     7.75 22148.61 ...    0.51    0.01 7.50e-03
min     1.14     3.00  253.55 ...    0.07    0.95 9.75e-01
25%     3.14     9.00  8375.88 ...    0.25    0.96 9.81e-01
50%     5.14    15.00 21630.66 ...    0.50    0.97 9.88e-01
75%     7.15    21.00 39001.43 ...    0.88    0.99 9.94e-01
max     9.30    27.00 72784.87 ...    1.83    1.00 1.00e+00
```

istatistik - DataFrame

— □ ×

Index	lp	knots	GTT	GTn	GGn	Ts	Tp	HP	T1	T2	P48	P1	P2	Pexh	TIC	mf	ciktı1	ciktı2
count	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934	11934
mean	5.16667	15	27247.5	2136.29	8200.95	227.336	227.336	735.495	288	646.215	2.35296	0.998	12.2971	1.02947	33.6413	0.66244	0.975	0.9875
std	2.62639	7.74629	22148.6	774.084	1091.32	200.496	200.496	173.681	0	72.6759	1.08477	2.53364...	5.33745	0.0103903	25.8414	0.507132	0.0147...	0.00750...
min	1.138	3	253.547	1307.67	6589	5.304	5.304	442.364	288	540.442	1.093	0.998	5.828	1.019	0	0.068	0.95	0.975
25%	3.144	9	8375.88	1386.76	7058.32	60.317	60.317	589.873	288	578.092	1.389	0.998	7.44725	1.02	13.6775	0.246	0.962	0.981
50%	5.14	15	21630.7	1924.33	8482.08	175.268	175.268	706.038	288	637.141	2.083	0.998	11.092	1.026	25.2765	0.496	0.975	0.9875
75%	7.148	21	39001.4	2678.08	9132.61	332.365	332.365	834.066	288	693.925	2.981	0.998	15.658	1.036	44.5525	0.882	0.988	0.994
max	9.3	27	72784.9	3560.74	9797.1	645.249	645.249	1115.8	288	789.094	4.56	0.998	23.14	1.052	92.556	1.832	1	1

Girdi kolonlarının birbirlerini ne kadar etkilediğini görüntülemek için korelasyon matrisini oluşturuyoruz.

```
#korelasyon matrisi ve renkli grafiği (keşif devam)
import seaborn as sn
import matplotlib.pyplot as plt
pd.set_option('display.width', 60, 'precision', 2)
corrmatrix = veriler.corr(method='pearson')
print("Korelasyon matrix'i:\n", corrmatrix)
plt.figure(figsize=(10,7))
sn.heatmap(corrmatrix, annot=True)
plt.show()
```

corrmatrix - DataFrame

— □ ×

Index	lp	knots	GTT	GTn	GGn	Ts	Tp	HP	T1	T2	P48	P1	P2	Pexh	TIC	mf	ciktı1	ciktı2
lp	1	0.999915	0.961025	0.96209	0.986017	0.959241	0.959241	0.961242	nan	0.982748	0.963058	-9.6179...	0.969113	0.953361	0.913631	0.931422	-6.016...	5.90381e-20
knots	0.999915	1	0.958243	0.960412	0.986621	0.956371	0.956371	0.958841	nan	0.981161	0.960613	0	0.966975	0.950813	0.910014	0.927815	0	0
GTT	0.961025	0.958243	1	0.989723	0.932993	0.999177	0.999177	0.99113	nan	0.990227	0.9989	-7.2790...	0.997582	0.996046	0.977871	0.995082	0.0029...	0.000357638
GTn	0.96209	0.960412	0.989723	1	0.943038	0.988625	0.988625	0.979638	nan	0.989278	0.995103	3.25187...	0.996035	0.994019	0.962341	0.980223	0.0013...	-1.77656e-05
GGn	0.986017	0.986621	0.932993	0.943038	1	0.929593	0.929593	0.939762	nan	0.966659	0.93746	4.26374...	0.945938	0.924231	0.87912	0.896959	-0.018...	0.0100001
Ts	0.959241	0.956371	0.999177	0.988625	0.929593	1	1	0.985962	nan	0.987369	0.998012	-5.3818...	0.996172	0.996238	0.977468	0.994366	0.0007...	0.000104423
Tp	0.959241	0.956371	0.999177	0.988625	0.929593	1	1	0.985962	nan	0.987369	0.998012	-5.3818...	0.996172	0.996238	0.977468	0.994366	0.0007...	0.000104423
HP	0.961242	0.958841	0.99113	0.979638	0.939762	0.985962	0.985962	1	nan	0.992308	0.989383	-1.2276...	0.990463	0.980071	0.969655	0.986318	-0.039...	-0.0384639
T1	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
T2	0.982748	0.981161	0.990227	0.989278	0.966659	0.987369	0.987369	0.992308	nan	1	0.991704	3.16741...	0.994373	0.983461	0.958711	0.976453	-0.047...	-0.0168551
P48	0.963058	0.960613	0.9989	0.995103	0.93746	0.998012	0.998012	0.989383	nan	0.991704	1	4.25423...	0.999401	0.997941	0.975704	0.992728	0.0081...	-0.00271819
P1	-9.61791...	0	-7.27902...	3.25187e...	4.26374e...	-5.38189...	-5.38189e...	-1.22761...	nan	3.16741e...	4.25423...	1	-3.0175...	-4.7117...	-3.7907...	7.08849e...	3.0865...	2.79276e-12
P2	0.969113	0.966975	0.997582	0.996035	0.945938	0.996172	0.996172	0.990463	nan	0.994373	0.999401	-3.0175...	1	0.996339	0.972109	0.98926	0.0083...	-0.0183034
Pexh	0.953361	0.950813	0.996046	0.994019	0.924231	0.996238	0.996238	0.980071	nan	0.983461	0.997941	-4.7117...	0.996339	1	0.974193	0.990981	0.0352...	0.0117942
TIC	0.913631	0.910014	0.977871	0.962341	0.87912	0.977468	0.977468	0.969655	nan	0.958711	0.975704	-3.7907...	0.972109	0.974193	1	0.985544	-0.032...	-0.0188718
mf	0.931422	0.927815	0.995082	0.980223	0.896959	0.994366	0.994366	0.986318	nan	0.976453	0.992728	7.08849...	0.98926	0.990981	0.985544	1	-0.013...	-0.0173268
ciktı1	-6.01627...	0	0.002978...	0.001369...	-0.018838	0.0007535	0.0007535	-0.03962...	nan	-0.04717...	0.00816...	3.08658...	0.00832...	0.0352852	-0.0320...	-0.01366...	1	4.20521e-17
ciktı2	5.90381e...	0	0.000357...	-1.77656...	0.0100001	0.000104...	0.000104...	-0.03846...	nan	-0.01685...	-0.0027...	2.79276...	-0.0183...	0.0117942	-0.0188...	-0.01732...	4.2052...	1



## Verileri Makine Öğrenmesi Modeli İçin Hazırlama

Modelin girdi ve çıktı değişkenlerini ayırt edebilmesi için x(girdi değişkenlerini temsil eden) ve y(çıkıtı2 değişkenini temsil eden) olarak iki değişken tanımlıyor, hedef kolonlarımızı tanımladığımız değişkenlerin içine aktarıyoruz. Çıktı olarak 2 ayrı değişkenimiz var. Birini tercih ediyoruz ve diğer değişkeni (ciktı1) modelden tamamen çıkarıyoruz.

```
# girdi(x) ve çıktı(y) kolonlarının birbirinden ayrılması,bütün öznitelikler modele dahil
x = veriler.iloc[:, 0:16]
y = veriler.iloc[:, 17:18]
print('\n', x.shape)
print('\n', y.shape, '\n')
```

Ayrıştırmanın shape'le sağlamasını yapıyoruz.

```
(11934, 16)
```

```
(11934, 1)
```

Gözlemleri test ve eğitim için ayırıyoruz.

```
# eğitim ve test verisinin ayrıştırılması
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =0.3, random_state=0)
```

## Veri Dönüştürme

Verilerimizin hepsi sayısal ve sürekli değerlerden oluşuyor. Kategorik veri bulunmadığı için herhangi bir encoder kullanmıyoruz. Ama verilerimizi belli bir aralığa indirmek için normalizasyon yapıyoruz.

```
#normalize işlemi
from sklearn.preprocessing import MinMaxScaler
minMaxScaler = MinMaxScaler()
minMaxScaler.fit(x)
x_norm = minMaxScaler.transform(x)
x_train_norm = minMaxScaler.transform(x_train)
x_test_norm = minMaxScaler.transform(x_test)
```

## Modellemeye Geçiş

İlk aşamamızda bütün özniteliklerimizi modele dahil ederek bir tahmin sonucu elde etmeyi amaçlıyoruz.

## Çoklu Regresyon Modeli (Multiple Linear Regression)

Bu model Sürekli verilerden oluşan, birden çok girdi değişkeni olan veri setlerinde kullanmaya elverişli bir modeldir.

```
# tüm özniteliklerle Modelleme işlemleri
# ilk model Linear Regresyon Modeli
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
reg = LinearRegression()
reg.fit(x_train_norm, y_train)
```

Modeli oluşturduk. Test ve Eğitim verileri için tahmin ediyor performans değerlerini yazdırıyoruz.

```
# regreston test tahmin
y_reg_test_pred = reg.predict(x_test_norm)

# modelin test performansı scorlaması r2
from sklearn.metrics import r2_score
reg_test_score = r2_score(y_test, y_reg_test_pred)
print("Regresyon modeli test performansı \nr2: ", reg_test_score)

# Regresyon modelinin eğitim verisini tahmini
y_reg_train_pred = reg.predict(x_train_norm)

# Regresyon modelinin eğitim performansı scorlaması r2
reg_train_score = r2_score(y_train, y_reg_train_pred)
```

```
print("Regresyon modeli eğitim performansı \nr2: ", reg_train_score)
print("Regresyon modeli eğitim performansı \nr2: ", reg_train_score)
print('MSE eğitim : ', mean_squared_error(y_train, y_reg_train_pred))
print('MSE test : ', mean_squared_error(y_test, y_reg_test_pred))
```

```
Regresyon modeli test performansı
r2: 0.909316699061269
Regresyon modeli eğitim performansı
r2: 0.911603413143252
MSE eğitim : 4.9483738336445025e-06
MSE test : 5.1548622791063726e-06
```

Modelin başarısını gösteren r2 oranı oldukça yüksek. Eğitim ve test performansları arasında da pek bir fark yok. Burada Lineer Regresyon modelimizin başarılı tahminler yaptığını

söyleyebilmekteyiz.

Tahmin edilmiş ve gerçek verilerin bir kısmını yan yana getirdiğimizde tahminlerin yakınlığını görebiliyoruz.

y_test - DataFrame		y_regk_test_pred - N		y_train - DataFrame		y_reg_train_pred -	
Index	cikti2		0	Index	cikti2		0
8784	0.989	0	0.9922...	317	0.984	0	0.9817...
5313	0.993	1	0.9927...	2453	0.987	1	0.9871...
5094	0.995	2	0.9929...	137	0.99	2	0.9860...
3652	0.99	3	0.9904...	10535	0.975	3	0.9764...
10324	0.978	4	0.9803...	11350	0.988	4	0.9946...

## Polinom Regresyon Modeli (Polynomial Regression)

Girdiler ve çıktı arasındaki ilişkinin doğrusal olmama ihtimalini göz önünde bulunduran modeldir. Polinom derecesine göre modelin karmaşıklığı da artar. Polinom derecesini çok yüksek tutmak modelin kural çıkarıp ezberlemesine ve yeni gözlemler karşısında başarısız olmasına neden olabilir.

```
# ikinci model Polinomal Regresyon modeli
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
poly= PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x_norm)
x_poly_train, x_poly_test, y_poly_train, y_poly_test = train_test_split(x_poly, y, test_size
=0.3, random_state=0)
linreg2 = LinearRegression().fit(x_poly_train, y_poly_train)
```

Modelimizi daha önce normalize etmiş olduğumuz verileri kullanarak oluşturduk. Eğitim ve test olarak ikiye ayırdık. Modelin test ve eğitim tahmin sonuçları ve performansına geçiyoruz.

```
#Polinom model katsayıları ve başarı scorları
print('polinom modelinin katsayıları (w): ', (linreg2.coef_))
print('Polinom Modelinin sabit (b) katsayısı: ', (linreg2.intercept_))
print('Polinom Modelinin R Kare Eğitim performansı: ', (linreg2.score(x_poly_train,
y_poly_train)))
print('Polinom Modelinin R kare Test performansı: ', (linreg2.score(x_poly_test, y_poly_test)))
print('o olmayan öznitelik sayısı: ', np.sum(linreg2.coef_ !=0))
```

```
polinom modelinin katsayıları (w): [[ 2.33591163e-06  7.67875876e+00 -7.60894135e+00  6.34092316e-01
-1.97515401e-01  8.40081249e-01 -2.59877659e-01 -2.59842215e-01
-9.94041847e-02 -6.79092776e-06 -6.99996621e-01  3.64044608e-01
1.58684242e-06  2.80050153e-01  4.05208498e-03  2.73083136e-02
1.10631775e-01  9.92253446e+02 -1.93416147e+03 -5.71854459e+01
1.14332970e+02 -9.95478644e+00 -4.89844696e+01 -4.89844684e+01
4.70500097e+00  2.77944192e-08 -8.27844566e+00  1.63240746e+01]
```

```
Polinom Modelinin sabit (b) katsayısı: [0.98476873]
Polinom Modelinin R Kare Eğitim performansı: 0.9999492653878846
Polinom Modelinin R kare Test performansı: 0.9999487104941605
o olmayan öznitelik sayısı: 135
```



```
# y kolonu test ve eğitim için tahmin sonuçları
pred_y_poly_train = linreg2.predict(x_poly_train)
pred_y_poly_test = linreg2.predict(x_poly_test)
```

Test ve Eğitim gözlemlerini tahmin sonuçlarıyla birlikte görüntülüyoruz.

y_poly_test - DataFrame		pred_y_poly_test - NumPy array		y_poly_train - DataFrame		pred_y_poly_train - NumPy array	
Index	cikti2		0	Index	cikti2		0
8784	0.989	0	0.988...	317	0.984	0	0.984072
5313	0.993	1	0.992...	2453	0.987	1	0.987031
5094	0.995	2	0.994...	137	0.99	2	0.990083
3652	0.99	3	0.989...	10535	0.975	3	0.974958

```
## Modelin hata oranları
```

```
print('MSE eğitim : ', mean_squared_error(y_poly_train, pred_y_poly_train))
print('MSE test : ', mean_squared_error(y_poly_test, pred_y_poly_test))
print('Eğitim performansı (R kare): ', r2_score(y_poly_train, pred_y_poly_train))
print('Test performansı (R kare): ', r2_score(y_poly_test, pred_y_poly_test))
```

```
MSE eğitim : 2.8400850754399835e-09
MSE test : 2.915535012828965e-09
Eğitim performansı (R kare): 0.9999492653878846
Test performansı (R kare): 0.9999487104941606
```

Performans ölçerimiz olan r2 bu modelde oldukça yüksek sonuç vermekte. Test ve eğitim verileri kıyaslandığında hata veya performans oranı olarak aralarındaki farkın yok denecek kadar az olması

modelin daha önce karşılaştığı gözlemler kadar karşılaşmadığı gözlemler karşısında da oldukça başarılı olduğunu göstermekte.

## Karar Ağaçları Regresyon Modeli (Decision Tree Regression)

Karar ağaçları modelleri her ne kadar sınıflandırma ve kategorik yapıları verilerde karşımıza çıksa da regresyon için de kullanılabilir. Aşağıda modelimizi lineer regresyonda oluşturduğumuz test ve eğitim kümesiyle oluşturuyoruz.

```
# üçüncü model Regresyon Karar Ağaçları
```

```
from sklearn.tree import DecisionTreeRegressor
tree = DecisionTreeRegressor(max_leaf_nodes=20, random_state=0, max_depth=10)
tree.fit(x_train, y_train)
```

Modelimizin tahmin sonuçları ve performans sonuçlarını yazdırıyoruz.

```
#tahmin sonuçları
```

```
tree_pred_train = tree.predict(x_train)
tree_pred_test = tree.predict(x_test)
```

tree_pred_test - NumPy array		y_test - DataFrame		tree_pred_train - NumPy array		y_train - DataFrame	
	0	Index	cikti2		0	Index	cikti2
0	0.991935	8784	0.989	0	0.987271	317	0.984
1	0.994894	5313	0.993	1	0.987836	2453	0.987
2	0.985766	5094	0.995	2	0.987271	137	0.99
3	0.994786	3652	0.99	3	0.979003	10535	0.975
4	0.985766	10324	0.978	4	0.985766	11350	0.988
5	0.995681	9314	0.995	5	0.985766	4896	0.999

```
#Performans skorları
```

```
print('Eğitim performansı (R kare): ', r2_score(y_train, tree_pred_train))
print('Test performansı (R kare): ', r2_score(y_test, tree_pred_test))
print('Eğitim hata oranı (MSE): ', mean_squared_error(y_train, tree_pred_train))
print('Test hata oranı (MSE): ', mean_squared_error(y_test, tree_pred_test))
```

```
Eğitim performansı (R kare): 0.665090790530434
Test performansı (R kare): 0.6464692363153517
Eğitim hata oranı (MSE): 1.874796332884944e-05
Test hata oranı (MSE): 2.009633945121765e-05
```

Eğitim, test gözlemlerinin performans sonuçlarının çıktısı yanda görüldüğü gibidir. Başarı oranı düşüktür. Diğer modellere göre başarısız sayılır.

## İlk Aşama Model Performansları Toplu Değerlendirme

Bütün öznitelikleri dahil edip herhangi bir iyileştirme yapmadığımız aşamada bütün modellerin performans değerlerini aşağıdaki tabloda toplu olarak görmektediriz.

Model Adı	Öznitelik	Parametre	R2 Test Performansı	MSE Test Oranı	R2 Eğitim Performansı	MSE Eğitim Oranı
Çoklu Regresyon Modeli	Hepsi dahil	İşlem yapılmadı	0,90	5,15	0,91	4,9
Polinom Regresyon Modeli	Hepsi dahil	İşlem yapılmadı	0,99	2,91	0,99	2,84
Karar Ağaçları Regresyon Modeli	Hepsi dahil	İşlem yapılmadı	0,64	2,00	0,66	1,87

Tabloya göre Lineer regresyon ve Polinom regresyon modelleri başarı oranları Karar ağacı modeline göre oldukça yüksek. Ama hata oranları da göz önünde bulundurulduğunda en başarılı duran modelin Polinom regresyon olduğunu görmekteyiz. Ancak modelin başarısı çok yüksek. Test ve eğitim performansları arasında farklılık gözlenmesi bile kolonların bu derece ilişkili olması modele yeni katılacak gözlemlerin tahminleri açısından tehlikeli olabilir.

## Öznitelik Seçme ve Modelleri Tekrar Kurma

Bu aşama gereksiz görünen özniteliklerin modelden çıkarılmasını veya yeni öznitelikler eklenmesini içerir. Modelin ve veri setinin ihtiyacına göre öznitelik sayıları azaltılıp artırılabilir.

**Sklearn kütüphanesinin** bir özelliği olan **feature selection**'ın 'en iyi öznitelikler' olan **selectkbest** özelliğini kullanıyoruz.

```
#Feature selection (Öznitelik seçme)
from sklearn.feature_selection import SelectKBest, f_regression
Seçilen özniteliklerin regresyon modelleri kuralınca seçilmesini sağlamak için f_regression özelliğini kullandık.
```

```
#normalize verileri kullanıyoruz.
test = SelectKBest(score_func=f_regression, k=10)
fit = test.fit(x_norm, y)
np.set_printoptions(precision=3)
cols = test.get_support(indices=True)
x_kbest = x_norm[:,cols]
print('\nÖzniteliklerin skorları: \n', fit.scores_[0:16], '\n Seçilen öznitelikler: \n',
x_kbest[0:16])
```

Normalize edilmiş verileri kullanarak **selectkbest** modelimizi oluşturduk ve en iyi '10' özniteliği öğrettik.

```
Özniteliklerin skorları:
[5.819e-25 5.941e-25 1.526e-03 3.766e-06 1.193e+00 1.301e-04 1.301e-04
 1.768e+01 nan 3.391e+00 8.816e-02 nan 3.999e+00 1.660e+00
 4.251e+00 3.583e+00]
```

Gh, Ggn, Ts, Hp, T2, P48, P2, Pexh, Tic, Mf öznitelikleri seçilmiştir.

# Makine Öğrenme Modellerinin Yeni Özniteliklerle Tekrar Kurulması

## Çoklu Regresyon Modeli

```
# Seçilen özniteliklerle 3 modelin sırayla tekrar kurulması
# Yeni özniteliklerden Test ve eğitim veri kümesi oluşturulması
x_traink, x_testk, y_traink, y_testk = train_test_split(x_kbest, y, test_size =0.3,
random_state=0)
```

Seçilen özniteliklerden test ve eğitim kümesini oluşturduk.

```
# Linear Regresyon
regk = LinearRegression()
regk.fit(x_traink, y_traink)
```

Regresyon modelini oluşturduk.

```
# regresyon test tahmin
y_regk_test_pred = regk.predict(x_testk)

# modelin test performansı scorlaması r2
regk_test_score = r2_score(y_testk, y_regk_test_pred)
print("Regresyon modeli test performansı \nr2: ", regk_test_score)
```

```
# Regresyon modelinin eğitim verisini tahmini
y_regk_train_pred = regk.predict(x_traink)

# Regresyon modelinin eğitim performansı scorlaması r2
regk_train_score = r2_score(y_traink, y_regk_train_pred)
print("Regresyon modeli eğitim performansı \nr2: ", regk_train_score)
print('MSE eğitim : ', mean_squared_error(y_traink, y_regk_train_pred))
print('MSE test : ', mean_squared_error(y_testk, y_regk_test_pred))
```

y\_testk - DataFrame

Index	cikti2
8784	0.989
5313	0.993
5094	0.995
3652	0.99
10324	0.978
9314	0.995

y\_regk\_test\_pred - DataFrame

0	0.9922...
1	0.9927...
2	0.9929...
3	0.9904...
4	0.9803...

y\_traink - DataFrame

Index	cikti2
317	0.984
2453	0.987
137	0.99
10535	0.975
11350	0.988
4896	0.999

y\_regk\_train\_pred - DataFrame

0	0.983072
1	0.98713
2	0.986308
3	0.97698
4	0.987153

```
Regresyon modeli test performansı
r2: 0.818590162645436
Regresyon modeli eğitim performansı
r2: 0.817664709944071
MSE eğitim : 1.0206991133321827e-05
MSE test : 1.031218226462313e-05
```

Öznitelik seçimi sonrası Regresyon modelimizin başarı oranının düştüğü görmekteyiz. Fakat hata oranında bir düşüş var bu da modelimizi aslında gelecek gözlemlere iyi hazırladığımız anlamına gelebilir.

## Polinom Regresyon

```
# Polinomal Regresyon modeli
poly= PolynomialFeatures(degree=2)
x_polyk = poly.fit_transform(x_kbest)
```

```
linreg3k = LinearRegression().fit(x_traink, y_traink)
```

Polinom regresyon modelimizi yeni öznitelikleri kullanarak oluşturduk.

```
#Polinom model katsayıları ve başarı scorları (yeni özniteliklerle)
print('polinom modelinin katsayıları (w): ', (linreg3k.coef_))
print('Polinom Modelinin sabit (b) katsayısı: ', (linreg3k.intercept_))
print('Polinom Modelinin R Kare Eğitim performansı: ', (linreg3k.score(x_traink, y_traink)))
print('Polinom Modelinin R kare Test performansı: ', (linreg3k.score(x_testk, y_testk)))
print('o olmayan öznitelik sayısı: ', np.sum(linreg3k.coef_ !=0))
```



yeni özniteliklerle

```
polinom modelinin katsayıları (w): [[ 0.578  0.032 -0.498 -0.18  0.122
 0.882 -0.939  0.01  0.01 -0.044]]
Polinom Modelinin sabit (b) katsayısı: [0.994]
Polinom Modelinin R Kare Eğitim performansı: 0.817664709944071
Polinom Modelinin R kare Test performansı: 0.818590162645436
o olmayan öznitelik sayısı: 10
```

Test ve eğitim kümesini tahmin ediyor, performans sonuçlarını görüntülüyoruz.

```
# y kolonu test ve eğitim için tahmin sonuçları
y_polyk_train_pred = linreg3k.predict(x_traink)
y_polyk_test_pred = linreg3k.predict(x_testk)

## Modelin hata oranları
print('MSE eğitim : ', mean_squared_error(y_traink, y_polyk_train_pred))
print('MSE test : ', mean_squared_error(y_testk, y_polyk_test_pred))
```

```
MSE eğitim : 1.0206991133321827e-05
MSE test : 1.031218226462313e-05
```

Polinom model'in öznitelik seçimi sonrası performansının düştüğünü görmekteyiz. Bu durum modelimizi başarısız yapmamaktadır.

Index	cikti2
317	0.984
2453	0.987
137	0.99
10535	0.975
11350	0.988
4896	0.999
6760	0.998

0
0.983...
0.987...
0.986...
0.976...
0.987...
0.992...
0.989

Index	cikti2
8784	0.989
5313	0.993
5094	0.995
3652	0.99
10324	0.978
9314	0.995
11060	0.981

0
0.992249
0.992725
0.992933
0.990416
0.980354
0.99989

## Karar Ağaçları Regresyon Modeli

Öznitelikleri dahil ederek bu kez de karar ağacı regresyon modelimizi tekrar oluşturuyoruz.

```
# üçüncü model Regresyon Karar Ağaçları (yeni özniteliklerle)
from sklearn.tree import DecisionTreeRegressor
trek = DecisionTreeRegressor(max_leaf_nodes=20, random_state=0, max_depth=10)
trek.fit(x_traink, y_traink)

#tahmin sonuçları
y_trek_train_pred = trek.predict(x_traink)
y_trek_test_pred = trek.predict(x_testk)

#Performans skorları
print('\nkarar ağaçları yeni özniteliklerle:\n')
print('Eğitim performansı (R kare): ', r2_score(y_traink, y_trek_train_pred))
print('Test performansı (R kare): ', r2_score(y_testk, y_trek_test_pred))
print('Eğitim hata oranı (MSE): ', mean_squared_error(y_traink, y_trek_train_pred))
print('Test hata oranı (MSE): ', mean_squared_error(y_testk, y_trek_test_pred))
```

karar ağaçları yeni özniteliklerle:

```
Eğitim performansı (R kare): 0.5125831377000285
Test performansı (R kare): 0.490834166659465
Eğitim hata oranı (MSE): 2.728522597134827e-05
Test hata oranı (MSE): 2.8943363561906563e-05
```

Modelimiz, öznitelik seçiminden önceki haline göre daha başarısız. Hata oranı da önceki halinden daha fazla. Model eksik öğreniyor olabilir.

Index	cikti2	0	1	2	Index	cikti2	0	1	2
317	0.984	0.987999	0.986486	0.987999	8784	0.989	0.987999	0.987999	0.987999
2453	0.987				5313	0.993			
137	0.99				5094	0.995			
10535	0.975								

## İkinci Aşama Öznitelik Seçimi Model Performansları Toplu Değerlendirme

En iyi 10 öznitelikle modellerimizi tekrar oluşturduktan sonra öznitelik seçiminin modellerimizdeki karşılığını toplu olarak aşağıdaki tabloda görmekteyiz.

Model Adı	Öznitelik	Parametre	R2 Test Performansı	MSE Test Oranı	R2 Eğitim Performansı	MSE Eğitim Oranı
Çoklu Regresyon Modeli	En iyi 10 öznitelik	İşlem yapılmadı	0,81	1,03	0,81	1,02
Polinom Regresyon Modeli	En iyi 10 öznitelik	İşlem yapılmadı	0,81	1,03	0,81	1,02
Karar Ağaçları Regresyon Modeli	En iyi 10 öznitelik	İşlem yapılmadı	0,49	2,89	0,51	2,72

Öznitelik seçme işlemi genel bir performans düşüklüğüne neden olurken **Çoklu regresyon** ve **polinom regresyonu** için hata oranlarında da düşüşe sebep olmuş görünüyor. Bu da ne kadar performansı düşüren bir işlem gibi görünse de birbiriyle çok ilişkili bazı kolonların çıkarılması, eklenecek yeni gözlemler için daha gerçekçi tahmin sonuçlarının hesaplanabileceğini göstermektedir.

## Parametre Optimizasyonu

Uygulamış olduğumuz her üç model için modelin izin verdiği üç parametreyi seçerek optimizasyon işlemlerini yapıyoruz. Modellerimizi optimize edilmiş parametrelerle tekrar oluşturup sonuçları yorumluyoruz.

## Çoklu Regresyon Modeli

Regresyon modeli için **fit\_intercept** parametresi modele bir kesme noktası veren parametredir. Parametre belirlenmediğinde otomatik 'true' olarak belirlenir. '**Normalize**' fit\_intercept değeri 'false' olarak belirlenirse yok sayılır. 'true' olarak belirlenirse model normalize edecektir. '**copy\_x**' varsayılanı 'true' isteğe bağlı belirtilen bir parametredir.

```
# Parametre optimizasyonu Linear model
parameters = {
    'fit_intercept':('True', 'False'),
    'normalize':('True', 'False'),
    'copy_X':('True', 'False')}

lin_grid_model= GridSearchCV(regk, parameters, cv=5 ,n_jobs=-1)
lin_grid_model.fit(x_traink, y_traink)
lin_best_para = lin_grid_model.best_params_
print('Doğrusal regresyon için en iyi parametreler: \n', lin_best_para)
```

```
Doğrusal regresyon için en iyi parametreler:
{'copy_X': 'True', 'fit_intercept': 'True', 'normalize': 'True'}
```

```
# en iyi parametrelerle doğrusal model
y_lin_pred_opt_test = lin_grid_model.predict(x_testk)
y_lin_pred_opt_train = lin_grid_model.predict(x_traink)
print('\nParametre optimizasyonu sonrası model performans ölçümü: \n')
print('MSE Eğitim: ', mean_squared_error(y_traink, y_lin_pred_opt_train))
print('MSE Test: ', mean_squared_error(y_testk, y_lin_pred_opt_test))
print('R Kare Eğitim Skoru: ', r2_score(y_traink, y_lin_pred_opt_train))
print('R Kare Test Skoru: ', r2_score(y_testk, y_lin_pred_opt_test))
```

Parametre optimizasyonu sonrası model performans ölçümü:

```
MSE Eğitim: 1.0206991133321818e-05
MSE Test: 1.0312182264623137e-05
R Kare Eğitim Skoru: 0.8176647099440711
R Kare Test Skoru: 0.8185901626454359
```

Parametre optimizasyonu sonrası modelin performansında fark edilir bir değişim gözlemlenemedik.

y_testk - DataFrame		y_lin_pred_opt_test		y_traink - DataFrame		y_lin_pred_opt_train	
Index	cikti2		0	Index	cikti2		0
8784	0.989	0	0.9922...	317	0.984	0	0.983072
5313	0.993	1	0.9927...	2453	0.987	1	0.98713
5094	0.995	2	0.9929...	137	0.99	2	0.986308
3652	0.99	3	0.9904...	10535	0.975	3	0.97698
10324	0.978	4	0.9803...	11350	0.988	4	0.987153
9314	0.995	5	0.99989	4896	0.999	5	0.992187
11060	0.981						

## Polinom Regresyon Modeli

Polinom regresyon için kullandığımız tek parametre derecelendirme parametresidir. 1 ile 5. Derecen polinom modeller arasındaki en başarılı modeli bulmak için bir döngü kullanarak modeli her parametre değeri için oluşturuyor ve gözlemliyoruz.

```
# Polinom Regresyon modeli Parametre optimizasyonu
print('\nparametre optimizasyonu\n')
print('Polinom Modelinin parametreleri: \n', polyk._get_param_names())
import time
print('Polinom Dereceleri\n')
for a in range(1,5):
    tic=time.time()
    polinom_opt_pred = PolynomialFeatures(degree=a)
    x_poly_opt = polinom_opt_pred.fit_transform(x_kbest)
    x_train1, x_test1, y_train1, y_test1 = train_test_split(x_poly_opt, y, test_size=0.3,
    random_state=0)
    polimodelopt = LinearRegression()
    polimodelopt.fit(x_train1, y_train1)
    pred_poly_y_train1= polimodelopt.predict(x_train1)
    pred_poly_y_test1= polimodelopt.predict(x_test1)
    print('derece ', a, 'için : \n')
    print('MSE Eğitim: ', mean_squared_error(y_train1, pred_poly_y_train1))
    print('MSE Test: ', mean_squared_error(y_test1, pred_poly_y_test1))
    print('R Kare Eğitim Skoru: ', r2_score(y_train1, pred_poly_y_train1))
    print('R Kare Test Skoru: ', r2_score(y_test1, pred_poly_y_test1))
    toc= time.time()
    print(toc-tic, 'saniye Geçti\n')
```

derece 1 için :

MSE Eğitim: 1.0206991133321828e-05  
MSE Test: 1.0312182264623113e-05  
R Kare Eğitim Skoru: 0.817664709944071  
R Kare Test Skoru: 0.8185901626454364  
0.03124070167541504 saniye Geçti

2 için :

MSE Eğitim: 3.127649891702665e-08  
MSE Test: 3.09165502741233e-08  
R Kare Eğitim Skoru: 0.9994412839761022  
R Kare Test Skoru: 0.9994561222626918  
0.06248617172241211 saniye Geçti

derece 3 için :

MSE Eğitim: 1.032431967206964e-09  
MSE Test: 1.0931214924042156e-09  
R Kare Eğitim Skoru: 0.999981556878051  
R Kare Test Skoru: 0.9999807700264545  
0.21869969367980957 saniye Geçti

derece 4 için :

MSE Eğitim: 1.214297830616575e-10  
MSE Test: 3.4729515868913724e-10  
R Kare Eğitim Skoru: 0.9999978308069022  
R Kare Test Skoru: 0.9999938904533846  
1.3573565483093262 saniye Geçti

Döngü sonrası performans değerlerini yukarıda görüyoruz. 2. 3. Ve 4. dereceden modellerin performans sonuçları aşırı farklı durmuyor. Modelin derecesi arttıkça ezberleme riski de artabileceğinden 1. dereceden olan model (Lineer regresyon) veya performansı 2. dereceyle aynı ama hata oranı daha düşük olan 3. dereceden polinom modeli seçilebilir.

## Karar Ağacı Regresyon Modeli

Karar ağaçlarında kullanmayı seçtiğimiz üç parametre; **max\_depth**, **min\_impurity\_decrease** ve **min\_samples\_split**'dir.

```
# Regresyon Karar Ağaçları Parametre optimizasyonu
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    'max_depth': [5, 10, 15, 20, 25],
    'min_impurity_decrease': [0, 0.001, 0.005, 0.01],
    'min_samples_split': [10, 20, 30, 40, 50]}
```

```
grid_model= GridSearchCV(tree, param_grid, cv=5 ,n_jobs=-1)
grid_model.fit(x_traink, y_traink)
best_para= grid_model.best_params_
print('regresyon karar ağaçları için en iyi parametreler: \n', best_para)
```

```
# en iyi parametrelerle model
```

```
y_pred_test_opt= grid_model.predict(x_testk)
y_pred_train_opt= grid_model.predict(x_traink)
print('\nParametre optimizasyonu sonrası model performans ölçümü: \n')
print('MSE Eğitim: ', mean_squared_error(y_traink, y_pred_train_opt))
print('MSE Test: ', mean_squared_error(y_testk, y_pred_test_opt))
print('R Kare Eğitim Skoru: ', r2_score(y_traink, y_pred_train_opt))
print('R Kare Test Skoru: ', r2_score(y_testk, y_pred_test_opt))
```

```
regresyon karar ağaçları için en iyi parametreler:
{'max_depth': 15, 'min_impurity_decrease': 0, 'min_samples_split': 10}
```

Parametre optimizasyonu sonrası model performans ölçümü:

MSE Eğitim: 2.0839971918767545e-05  
MSE Test: 2.257576375403627e-05  
R Kare Eğitim Skoru: 0.627719640888015  
R Kare Test Skoru: 0.6028517024156328

Öznitelik seçimi sonrası düşen performans, parametre optimizasyonu sonrası biraz daha iyileşti. Ama hala Karar ağacı modeli bu gözlemler için en uygun sonucu veren model olarak seçmeye uygun görünmüyor.

best\_para - Dictionary (3 elements)

Key	Type	Size	Value
max_depth	int	1	15
min_impurity_decrease	int	1	0
min_samples_split	int	1	10

## Genel Sonuç Performans Tablosu

### Öznitelikleri Seçmeden Önceki Performans Değerleri

Model Adı	Öznitelik	Parametre	R2 Test Performansı	MSE Test Oranı	R2 Eğitim Performansı	MSE Eğitim Oranı
Çoklu Regresyon Modeli	Hepsi dahil	İşlem yapılmadı	0,90	5,15	0,91	4,9
Polinom Regresyon Modeli	Hepsi dahil	İşlem yapılmadı	0,99	2,91	0,99	2,84
Karar Ağaçları Regresyon Modeli	Hepsi dahil	İşlem yapılmadı	0,64	2,00	0,66	1,87

### Öznitelik Seçiminden Sonraki Performans Değerleri

Model Adı	Öznitelik	Parametre	R2 Test Performansı	MSE Test Oranı	R2 Eğitim Performansı	MSE Eğitim Oranı
Çoklu Regresyon Modeli	En iyi 10 öznitelik	İşlem yapılmadı	0,81	1,03	0,81	1,02
Polinom Regresyon Modeli	En iyi 10 öznitelik	İşlem yapılmadı	0,81	1,03	0,81	1,02
Karar Ağaçları Regresyon Modeli	En iyi 10 öznitelik	İşlem yapılmadı	0,49	2,89	0,51	2,72

### Parametre Optimizasyonundan Sonraki Performans Değerleri

Model Adı	Öznitelik	Parametre	R2 Test Performansı	MSE Test Oranı	R2 Eğitim Performansı	MSE Eğitim Oranı
Çoklu Regresyon Modeli	En iyi 10 öznitelik	fit_intercept Normalize Copy_x	0,81	1,03	0,81	1,02
Polinom Regresyon Modeli	En iyi 10 öznitelik	Degree(3'e göre)	0,99	1,09	0,99	1,03
Karar Ağaçları Regresyon Modeli	En iyi 10 öznitelik	max_depth min_impurity_decrease min_samples_split	0,60	2,57	0,62	2,08

Parametre optimizasyonu sonunda başarı oranı **0,99** olan **Polinom modeli** veya **0,81** olan **Regresyon modeli** seçilebilir. Ancak **polinom modelin** başarı oranının olağanüstü yüksekliği modelin ezberleme riskinin yüksek olduğunu ve yeni gözlemlerde performans düşüklüğüne sebep olabilecek gibi görünmektedir. Regresyon modelinin elimizdeki verilerde ve ileride yeni gözlemler geldiğinde en uygun ve gerçekçi sonuçları tahmin edeceğini düşünmekteyim.