Doğrusal Regresyon Ve Kuzenleri

- Basit Doğrusal Regresyon
- Çoklu Doğrusal Regresyon
- Temel Birleşen Regresyonu
- Kısmı En Küçük Kareler Regresyonu
- Ridge Regresyon
- Lasso Regresyon
- Elastic Net Regresyonu
- Her Model İçin
  - Model
  - Tahmin
  - Model Optimizasyonu

1.Basit Doğrusal Regresyon (Teori):

- Temel amaç bağımlı-bağımsız değişken arasındaki ilişkiyi ifade eden doğrusal fonksiyonu bulmaktır.

Anakitle teorik gösterim: $Y = \beta_0 + \beta_1 X + \varepsilon$

Örneklem gerçek değerler: $y_i = b_0 + b_1 x_i + e_i$

Tahmin modeli: $\hat{y}_i = b_0 + b_1 x_i$

β_0 = Doğrunun y eksenini kestiği nokta

β_1 = Doğrunun eğimi

ε = Hata terimi

Örneklem teorik gösterim: $y_i = b_0 + b_1 x_i + e_i$

$e_i = y_i - b_0 + b_1 x_i$

Tahmin modeli: $\hat{y}_i = b_0 + b_1 x_i$

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$SSE = \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))^2$$

Hatalar/artıklar: $e_i = y_i - \hat{y}_i$

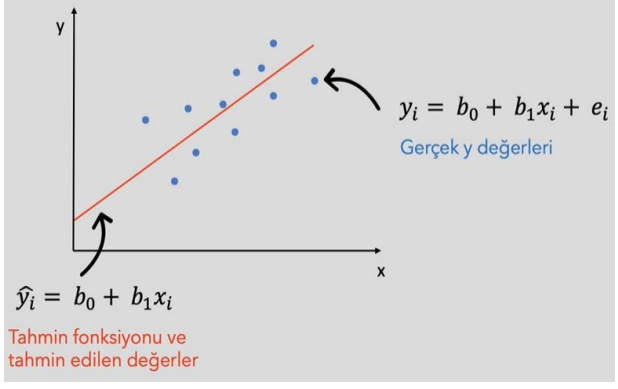
$$SSE = \sum_{i=1}^{n} (y_i - (b_0 + b_1 x_i))^2$$

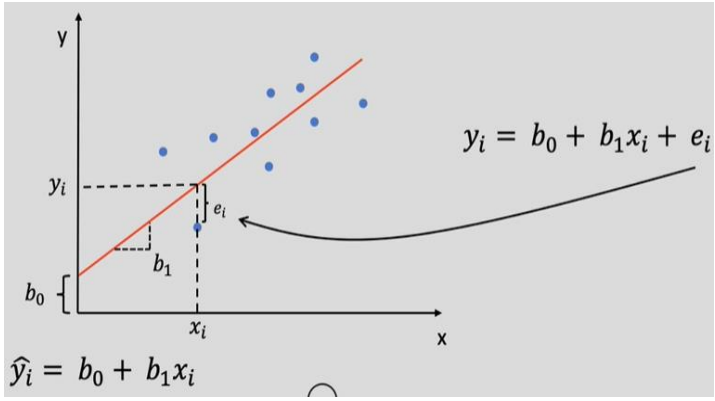
$$b_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$b_0 = \bar{y} - b_1 \bar{x}$$

$\bar{x}$: Bağımsız değişkenin ortalaması

$\bar{y}$: Bağımlı değişkenin ortalaması





- Gerçek değerler mavi noktalar , tahmin değerleri kırmızı ile gösterilen noktalar.

1.Basit Doğrusal Regresyon (Model):

```
import pandas as pd
ad = pd.read_csv("Advertising.csv", usecols = [1,2,3,4])
df = ad.copy()
df.head()
```

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

- Usecols koymamızın sebebi Tv nin sol tarafında unnamed diye bir değişken vardı ve 1 2 3 4 diye aşağıya iniyordu. Onu kaldırdık.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
TV           200 non-null float64
radio        200 non-null float64
newspaper    200 non-null float64
sales        200 non-null float64
dtypes: float64(4)
memory usage: 6.3 KB
```

- 4 tane sayısal değişkenimiz olduğunu gözlemledik.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
TV	200.0	147.0425	85.854236	0.7	74.375	149.75	218.825	296.4
radio	200.0	23.2640	14.846809	0.0	9.975	22.90	36.525	49.6
newspaper	200.0	30.5540	21.778621	0.3	12.750	25.75	45.100	114.0
sales	200.0	14.0225	5.217457	1.6	10.375	12.90	17.400	27.0

- Genel betimsel istatistiklerine baktık.

```
df.isnull().values.any()
```

```
False
```

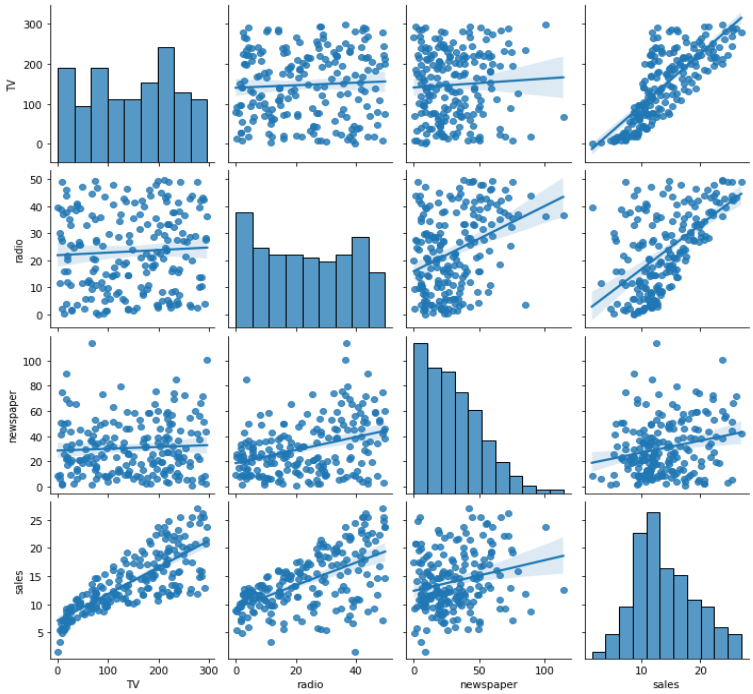
- Veri seti içerisinde eksik değer yokmuş.

```
df.corr()
```

	TV	radio	newspaper	sales
TV	1.000000	0.054809	0.056648	0.782224
radio	0.054809	1.000000	0.354104	0.576223
newspaper	0.056648	0.354104	1.000000	0.228299
sales	0.782224	0.576223	0.228299	1.000000

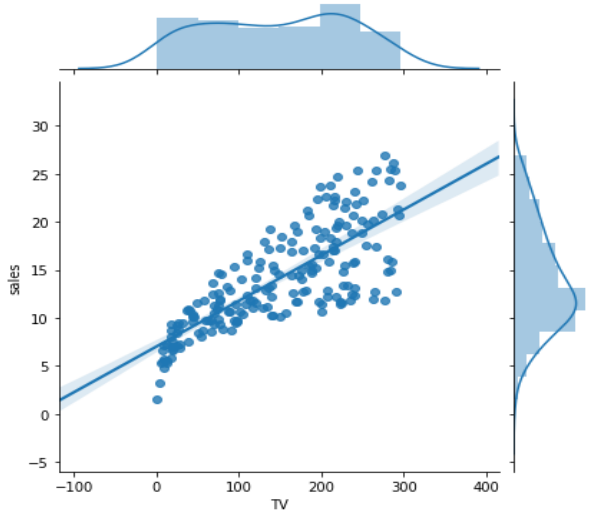
- Birbirleri ile ilişkisi incelendiğinde tv reklamları ile en çok satış gerçekleştiği görülür. Radio içinde tv kadar olmasada benzer durum geçerli.

```
import seaborn as sns
sns.pairplot(df, kind  ="reg");
```



- 4.satır 1. Sütundaki grafiğe baktığımızda TV'nin dağılımı satış üzerinde diğerlerine göre çok daha düzenli.

```
sns.jointplot(x = "TV", y = "sales", data =
df, kind = "reg")
```



- Yukarıdaki ve sağdaki tepelenmelere baktığımızda TV 2 tepeli bir dağılım söz konusuyken satışta tek tepeli bir grafik gözlemledik.

```
import statsmodels.api as sm
X = df[["TV"]]
X[0:5]
```

	TV
0	230.1
1	44.5
2	17.2
3	151.5
4	180.8

- Makine kısmına geçtik. Sadece tv değişkenini üzerinden işlem gerçekleştireceğiz.

```
X = sm.add_constant(X)
X[0:5]
```

	const	TV
0	1.0	230.1
1	1.0	44.5
2	1.0	17.2
3	1.0	151.5
4	1.0	180.8

- X bağımsız değişkenimiz hazır (TV)

```
y = df["sales"]
y[0:5]
```

```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: sales, dtype: float64
```

- Y bağımlı değişkenimiz hazır (sales-satış)

```
lm = sm.OLS(y,X)
model = lm.fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	sales	R-squared:	0.612
Model:	OLS	Adj. R-squared:	0.610
Method:	Least Squares	F-statistic:	312.1
Date:	Thu, 28 Apr 2022	Prob (F-statistic):	1.47e-42
Time:	17:30:28	Log-Likelihood:	-519.05
No. Observations:	200	AIC:	1042.
Df Residuals:	198	BIC:	1049.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.0326	0.458	15.360	0.000	6.130	7.935
TV	0.0475	0.003	17.668	0.000	0.042	0.053

Omnibus:	0.531	Durbin-Watson:	1.935
Prob(Omnibus):	0.767	Jarque-Bera (JB):	0.669
Skew:	-0.089	Prob(JB):	0.716
Kurtosis:	2.779	Cond. No.	338.

- 1. Satır model kuruldu, 2. Satırda fit edildi, 3.satırla da çıktı.
- R-squared : Bağımsız değişkenin bağımlı değişkendeki değişikliği açıklama başarısıdır. Altındaki ise düzeltilmiş halidir. Değişken sayısı arttıkça r-s artacaktır. Altındaki bunu törpüler.
- F-statistic : Bu modelin anlamlılığının açıklanması için kurulan istatistiktir. Altındaki de ismidir.
- Const kısmında tv 0,0475 olan kat sayışımız istatistiksel olarak anlamlıdır(P>|t|) ve %95 güvenirlik ile en sağdaki kısımlarda yer alacaktır.
- Yani tvde 1 birimlik artışla 0,0475 artış beklenir.

```
import statsmodels.formula.api as smf
lm = smf.ols("sales ~ TV", df)
model = lm.fit()
model.summary()
```

- Yukarıdaki işlemler yerine direk bu kısımla da işlemler gerçekleştirilebilir. Çıktı aynı olacaktır.
- Diğerinden farkı değişkenleri isimlendirerek gerçekleşti.

```
model.params
```

```
Intercept    7.032594
TV           0.047537
dtype: float64
```

- Model parametreleri kısa yollu çağırdık.

```
model.summary().tables[1]
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.0326	0.458	15.360	0.000	6.130	7.935
TV	0.0475	0.003	17.668	0.000	0.042	0.053

- Sadece kat sayılarla ilgili yere yukarıdaki kodla ulaşılabilir.
- En üst kısım 0. Orta kısım 1. En alt kısım 2. İndeks olarak geçer

```
model.conf_int()
```

	0	1
Intercept	6.129719	7.935468
TV	0.042231	0.052843

- Sadece güven aralığı olduğu kısma bu şekilde erişebiriz. [0.025 0.0975] kısmı.

```
print("f_pvalue: ", "%.4f" % model.f_pvalue)
```

```
f_pvalue:  0.0000
```

- Sadece f_pvalue değerine bu şekilde erişilebilir. %.4f kısmı noktadan sonra kaç basamak olduğunu belirtir.

```
print("fvalue: ", "%.2f" % model.fvalue)
```

```
fvalue:  312.14
```

- Sadece fvalue değerine bu şekilde erişilebilir.

```
print("tvalue: ", "%.2f" % model.tvalues[0:1])
```

```
tvalue:  15.36
```

- Sadece parametre değerine bu şekilde erişilebilir. Tahmin değerlerine ulaşılabilmesi için değişkenler katsayılar(parametre) ile çarpılır.

```
model.rsquared_adj
```

```
0.6099148238341623
```

- Açıklana bilirlik değeri.(düzeltilen)
- Sonundaki adj silinirse direk R-squared değerine ulaşılır.

```
model.fittedvalues[0:5]
```

```
0    17.970775
1     9.147974
2     7.850224
3    14.234395
4    15.627218
dtype: float64
```

- Tahmin edilen değerler yan tarafta verilmiştir.

```
y[0:5]
```

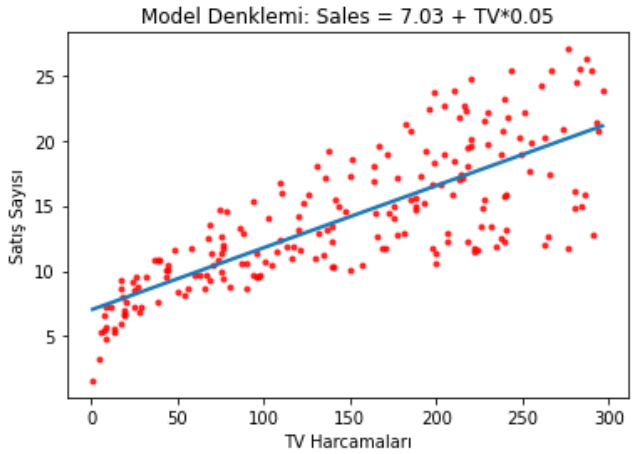
```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: sales, dtype: float64
```

- Gerçek değerlere şekilde ulaşılabilir.

```
print("Sales = " +  str("%.2f" %
model.params[0]) + " + TV" + "*" + str("%.2f"
% model.params[1]))
```

- Bizim makinemizin kullandığı formül yukarıdaki gibidir. Tahmin değerlerini bulurken genel olarak yukarıdaki işlemi gerçekleştirir.

```
g = sns.regplot(df["TV"], df["sales"], ci=None,
scatter_kws={'color':'r', 's':9})
g.set_title("Model Denklemi: Sales = 7.03 + TV*0.05")
g.set_ylabel("Satış Sayısı")
g.set_xlabel("TV Harcamaları")
plt.xlim(-10,310)
plt.ylim(bottom=0);
```



- Bu kodlarla grafiğimizi görselleştirebiliriz.

```
from sklearn.linear_model import
LinearRegression
X = df[["TV"]]
y = df["sales"]
reg = LinearRegression()
model = reg.fit(X, y)
print(model.intercept_,model.coef_)
```

```
7.032593549127695 [0.04753664]
```

- Bu şekilde makine sistemimizde var. Bunda summary çıktısı veremiyoruz.
- İntercept : kat sayı , coef : beta 1 kat sayısı.

```
model.score(X,y)
```

- Bu şekilde r kare değerlerine ulaşılabilir.

```
model.predict(X)[0:10]
```

```
array([17.97077451,  9.14797405,  7.85022376, 14.23439457, 15.62721814,
        7.44616232,  9.76595037, 12.74649773,  7.44140866, 16.53041431])
```

- Bu şekilde tahmin edilen değerlere ulaşılır.

1.Basit Doğrusal Regresyon (Tahmin):

```
Model denklemi:

Sales = 7.03 + TV*0.04

Örneğin 30 birim TV harcaması olduğunda
satışların tahmini değeri ne olur?
```

```
7.03 + 30*0.04
```

: 8.23

- Elle tahminimiz 8.23

```
X = df[["TV"]]
y = df["sales"]
reg = LinearRegression()
model = reg.fit(X, y)
model.predict([[30]])
```

```
array([8.45869276])
```

- Makine tahminimiz 8.45869276
- Predict: Kendisine girdiğimiz değeri denkleme gönderip bir çıktısı varsa onu bize çıkarır.

```
yeni_veri = [[5],[90],[200]]
model.predict(yeni_veri)
```

```
array([ 7.27027675, 11.31089119, 16.53992164])
```

Artıklar Ve Makine Öğrenmesindeki Önemi:

```
from sklearn.metrics importmean_squared_error,
r2_score

lm = smf.ols("sales ~ TV", df)
model = lm.fit()
mse = mean_squared_error(y,model.fittedvalues)
mse
```

- Mse (Hata kareleri ortalaması) elimizde. :10.512652
- 5. Satırda y gerçek değerler model.fittedvalues de tahmini değerlerimiz.

```
rmse = np.sqrt(mse)
```

- Mse karakökü alındığı zaman rmse değeri: 3.2423221486

```
reg.predict(X)[0:10]
```

```
array([17.97077451,  9.14797405,  7.85022376, 14.23439457, 15.62721814,
        7.44616232,  9.76595037, 12.74649773,  7.44140866, 16.53041431])
```

- Tahmin değerlerine ulaştık.

```
y[0:10]
```

```
array([22.1, 10.4,  9.3, 18.5, 12.9,  7.2, 11.8, 13.2,  4.8, 10.6])
```

- Gerçek değerlerimiz. Ben array'e çevirip burada gösterme işlemi yaptım. Normalde aşağıya doğru iniyordu.

```
k_t = pd.DataFrame({"gercek_y": y[0:10],
                    "tahmin_y":reg.predict(X)[0:10]})
k_t
```

	gercek_y	tahmin_y
0	22.1	17.970775
1	10.4	9.147974
2	9.3	7.850224
3	18.5	14.234395
4	12.9	15.627218
5	7.2	7.446162
6	11.8	9.765950
7	13.2	12.746498
8	4.8	7.441409
9	10.6	16.530414

- Karşılaştırma tablosu oluşturuldu. Elimizde gerçek değerler ve tahmini değerlerimiz var. Şimdi bunlar arasındaki farkla hata payımızı bulalım.

```
k_t["hata"] = k_t["gercek_y"] -k_t["tahmin_y"]
k_t
```

	gercek_y	tahmin_y	hata
0	22.1	17.970775	4.129225
1	10.4	9.147974	1.252026
2	9.3	7.850224	1.449776
3	18.5	14.234395	4.265605
4	12.9	15.627218	-2.727218
5	7.2	7.446162	-0.246162
6	11.8	9.765950	2.034050
7	13.2	12.746498	0.453502
8	4.8	7.441409	-2.641409
9	10.6	16.530414	-5.930414

- Hataların karesini veya mutlak değerini alıp negatif (-) değerli hata sonuçlarını pozitife çevireceğiz.
- Negatif (-) olması pek bir şey değiştirmiyor. Yani gerçek- tahmini değerimiz 1 fazla 1 eksik. Sonuç olarak hata olduğu için pozitif (+) yapmalıyız.

```
k_t["hata_kare"] = k_t["hata"]**2
```

	gercek_y	tahmin_y	hata	hata_kare
0	22.1	17.970775	4.129225	17.050503
1	10.4	9.147974	1.252026	1.567569
2	9.3	7.850224	1.449776	2.101851
3	18.5	14.234395	4.265605	18.195390
4	12.9	15.627218	-2.727218	7.437719
5	7.2	7.446162	-0.246162	0.060596
6	11.8	9.765950	2.034050	4.137358
7	13.2	12.746498	0.453502	0.205664
8	4.8	7.441409	-2.641409	6.977040
9	10.6	16.530414	-5.930414	35.169814

```
np.sum(k_t["hata_kare"])
```

- Hata karelerinin toplamı: 92.90350329638102

```
np.mean(k_t["hata_kare"])
```

- Hata karelerinin ortalaması: 9.290350329638102

```
np.sqrt(np.mean(k_t["hata_kare"]))
```

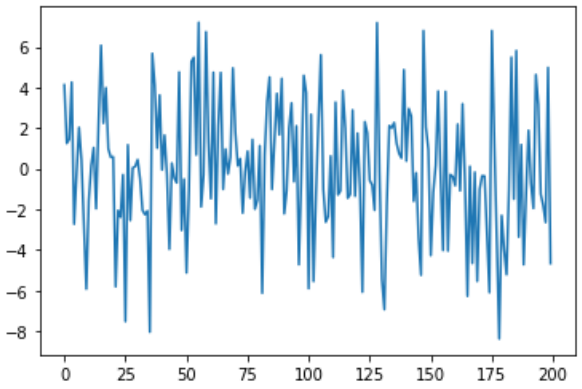
- Hata karelerinin ortalamasının karekökü : 3.0480075

```
model.resid[0:10]
```

```
array([ 4.12922549,  1.25202595,  1.44977624,  4.26560543, -2.72721814,
       -0.24616232,  2.03404963,  0.45350227, -2.64140866, -5.93041431])
```

- Modelin artıkları (Hataları) yukarıdaki gibidir.

```
import matplotlib.pyplot as plt
plt.plot(model.resid)
```



- Modelin artıkları (Hataları) yukarıdaki gibi görselleştirdik.
- 8'e 7'ye vuran değerleri gidip incelemek gerekir (Teknik Detay)

2.Çoklu Doğrusal Regresyon (Teori):

- Temel amaç bağımlı ve bağımsız değişkenler arasındaki ilişkiyi ifade eden doğrusal fonksiyonu bulmaktır. Genelde 2 şekilde amaç var:

+ Bağımlı değişkeni etkilediği belirlenen değişkenler aracılığıyla bağımlı değerlerin tahmin edilmesi.

+ Bağılı değişkeni etkilediği düşünülen bağımsız değişkenlerden hangisinin veya hangilerinin bağımlı değişkenleri ne yönde ne şekilde etkilediğini tespit edebilme, aralarındaki ilişkiyi tanımlayabilmek.

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_j X_{ij} + \cdots + \beta_p X_{ip} + \varepsilon_i$$

$$\sum_{i=1}^{n} e_i{}^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\hat{\beta} = (X^T.X)^{-1} X^T.Y$$

- Genel formülümüz.
- Yi: Teorik model orta: Hata kareler Alt: Genel formül
- Amacımız hata kareler toplamını min kat sayıları bulmak.

```
Call:
lm(formula = Sales ~ TV + Radio, data = caseStudyData)

Residuals:
    Min      1Q  Median      3Q     Max
-8.7977 -0.8752  0.2422  1.1708  2.8328

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.92110    0.29449   9.919   <2e-16 ***
TV           0.04575    0.00139  32.909   <2e-16 ***
Radio        0.18799    0.00804  23.382   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.681 on 197 degrees of freedom
Multiple R-squared:  0.8972,    Adjusted R-squared:  0.8962
F-statistic: 859.6 on 2 and 197 DF,  p-value: < 2.2e-16
```

- Genel çıktımız. Bu şekilde model yorumlanır.

Doğrusal Regresyonun Varsayımları

- Hatalar normal dağılır
- Hatalar birbirinden bağımsızdır ve aralarında otokorelasyon yoktur.
- Her bir gözlem için hata terimlerinin varyansları sabittir.
- Değişkenler ile hata terimi arasında ilişki yoktur.
- Bağımsız değişkenler arasında çoklu doğrusal ilişki problemi yoktur

Regresyonun Modellerin Avantaj ve Dezavantajları

- İyi anlaşılırsa diğer tüm ML ve DL konuları çok rahat kavranır.
- Doğrusallık nedensellik yorumları yapılabilmesini sağlar, bu durum aksiyoner ve stratejik modelleme imkanı sağlar.
- Değişkenlerin etki düzeyleri ve anlamlılıkları değerlendirilebilir
- Bağımlı değişkendeki değişkenliğin açıklanma başarısı ölçülebilir.
- Model anlamlılığı değerlendirilebilir.
  + Varsayım vardır
  + Aykırı gözlemlere duyarlıdır.

2.Çoklu Doğrusal Regresyon (Modelleme):

```
import pandas as pd
ad = pd.read_csv("Advertising.csv", usecols = [1,2,3,4])
df = ad.copy()
df.head()
```

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

- Bir önceki veri setimizi aldık. Sales olmadan bağımsız değişkenlerimizi seçicez.(x)
- Sadece sales ile bağımlı değişkenimizi seçelim(y)

```
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict

X = df.drop("sales", axis = 1)
y = df["sales"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state= 42)
```

- En alttaki kodda bağımlı değişken x, sonra bağımsız değişken y test_size yüzdelik dilimlerimizi ayırdık, 80 eğit 20 test sistemi.
- Her üretmede farklı çıkmasın diye random_state argümanı kullandık.

```
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

```
(160, 3) (160,) (40, 3) (40,)
```

- Boyutları incelenecek olursa testlerin boyutu 40 iken eğitimlerin boyutu 160 oldu. Genel verimizde 200-4 boyut.

```
training = df.copy()
```

- Orijinal verilerimizi copy argümanı ile kopyalayıp training değişkeninin içerisine attık. Lazım oldukça kullanacağız.
- Şimdi kuracağımız iki model var. **Stats** ve **scikit-learn** modeli.
- **Stats modeli** yorumlama ihtiyaçlarımız varsa kullanılır.
- **Scikit-learn** modeli ise yorumlamaya ihtiyacı olmayıp, diğer makine öğrenmesi algoritmaları işlemleri ile aynı model kullanmak isteyenler içindir.

Stats Models

```
lm = sm.OLS(y_train, X_train)
model = lm.fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	sales	R-squared (uncentered):	0.982
Model:	OLS	Adj. R-squared (uncentered):	0.982
Method:	Least Squares	F-statistic:	2935.
Date:	Fri, 29 Apr 2022	Prob (F-statistic):	1.28e-137
Time:	22:32:34	Log-Likelihood:	-336.65
No. Observations:	160	AIC:	679.3
Df Residuals:	157	BIC:	688.5
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
TV	0.0531	0.001	36.467	0.000	0.050	0.056
radio	0.2188	0.011	20.138	0.000	0.197	0.240
newspaper	0.0239	0.008	3.011	0.003	0.008	0.040

Omnibus:	11.405	Durbin-Watson:	1.895
Prob(Omnibus):	0.003	Jarque-Bera (JB):	15.574
Skew:	-0.432	Prob(JB):	0.000415
Kurtosis:	4.261	Cond. No.	13.5

- Yukarıda 2. Sayfa sonunda açıklamaları yer almaktadır.
- Değişken sayısı vs. arttığı için R-squared (açıklana bilirlik) değeri artmıştır.
- Kat sayılar incelendiğinde bütün kat sayılar anlamlı.

```
model.summary().tables[1]
```

	coef	std err	t	P>|t|	[0.025	0.975]
TV	0.0531	0.001	36.467	0.000	0.050	0.056
radio	0.2188	0.011	20.138	0.000	0.197	0.240
newspaper	0.0239	0.008	3.011	0.003	0.008	0.040

- Sadece kat sayılarla ilgili yere yukarıdaki kodla ulaşılabilir.
- En üst kısım 0. Orta kısım 1. En alt kısım 2. İndeks olarak geçer

```
model.params
```

```
TV           0.053109
radio        0.218787
newspaper    0.023855
dtype: float64
```

- Model parametreleri kısa yollu çağırdık.

```
model.conf_int()
```

	0	1
TV	0.050233	0.055986
radio	0.197328	0.240246
newspaper	0.008207	0.039504

- Sadece güven aralığı olduğu kısma bu şekilde erişebiriz. [0,025 0,0975]

- 3. Sayfadaki gibi çoğu özelliğe bu kısa yollarla ulaşılabilir.

Scikit-Learn Models

```
lm = LinearRegression()
model = lm.fit(X_train, y_train)
model.intercept_
```

- Yukarıdaki kodla sabit kat sayıya ulaşmış olduk.

```
2.979067338122629
```

```
model.coef_
```

```
array([0.04472952, 0.18919505, 0.00276111])
```

- Diğer tüm kat sayılara ulaşmış olduk. Bu modelle ilgili çalışmalar 3. Sayfanın sonunda anlatım gerçekleşmiştir.

2.Çoklu Doğrusal Regresyon (Tahmin):

```
Model denklemi:

Sales = 2.97 + TV0.04 + radio0.18 +
newspaper*0.002

Örneğin 30 birim TV harcaması, 10 birim radio
harcamasi, 40 birimde gazete harcaması
olduğunda satışların tahmini değeri ne olur?
```

```
yeni_veri = [[30], [10],[40]]
yeni_veri = pd.DataFrame(yeni_veri).T
model.predict(yeni_veri)
```

```
array([6.32334798])
```

- 30,10,40 değerlerini girdiğimizde satışların tahmini değerini bu şekilde ulaşabiliriz.

```
rmse = np.sqrt(mean_squared_error(y_train,
model.predict(X_train)))
```

- Modelimizin tahmin başarısı(rmse): 1.64472776564433
- Bu bizim eğitim hatamız. Şimdi makinenin hiç görmediği test değerlerimizi girerek test hatamızı bulalım.

```
rmse = np.sqrt(mean_squared_error(y_test,
model.predict(X_test)))
```

- Modelimizin tahmin başarısı(rmse): 1.78159966153345
- _trein girdiğimiz yerlere test değerlerimizi girerek test hatamızı bulmuş olduk.

2.Çoklu Doğrusal Regresyon (Model Tuning):

```
X = df.drop('sales', axis=1)
y = df["sales"]
X_train, X_test, y_train, y_test =

train_test_split(X, y, test_size=0.20,
random_state=144)

lm = LinearRegression()
model = lm.fit(X_train, y_train)
```

- Model doğrulama yapmamızın sebebi random_state değeri her değiştiğinde çıkan hata değerleri değişecektir.

```
model.score(X_train, y_train)
```

- Model skoru: 0.8971614078663419

```
np.sqrt(mean_squared_error(y_train,
model.predict(X_train)))
```

- Normalde Eğitim hatası: 1.6748559274650712

```
np.sqrt(mean_squared_error(y_test,
model.predict(X_test)))
```

- Normalde test hatası: 1.6640263686701027

```
cross_val_score(model, X_train, y_train, cv =
10, scoring = "r2").mean()
```

- Daha güvenilir r kare(mse) değeri bu kodla çıkar: 0.87337832
- Valude edilmiş skora gittiğimizde daha farklı ve güvenilir skor çıkar. Şimdi rmse değeri için yapalım.

```
np.sqrt(-cross_val_score(model,
                X_train,
                y_train,
                cv = 10,
scoring = "neg_mean_squared_error")).mean()
```

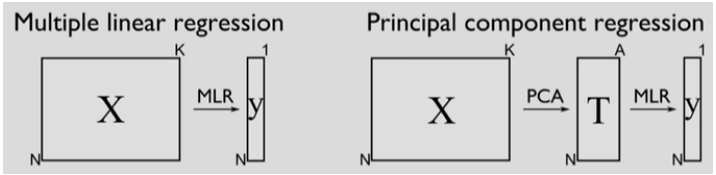
- Gerçek eğitim hatam (rmse) ortalaması: 1.6649345607872932

```
np.sqrt(-cross_val_score(model,
                X_test,
                y_test,
                cv = 10,
scoring = "neg_mean_squared_error")).mean()
```

- Gerçek test hatam (rmse) ortalaması: 1.7399924960346649
- Cross un önüne – yazmamızın sebebi – değerler gelmesini önlemek. Kısacası kafa karışıklılığı olmaması.

3.Temel Bileşen Regresyon- PCR (Teori):

- Değişkenlere boyut indirgeme işlemi uygulandıktan sonra çıkan bileşenlere regresyon modeli kurulması fikrine dayanır.



- Soldaki çoklu doğrusal regresyon modeli genel yapısı. Bağımsız değişkenler üzerinden bir model kurulup tahminler yapılır.
- Sağdaki temel bileşen regresyon modelinde ise değişkenlere ilk başka bir indirgeme uygulanıyor. Daha sonra bu indirgenen değişkenler üzerine regresyon modeli uygulanıyor.

3.Temel Bileşen Regresyon- PCR (Model):

```
hit = pd.read_csv("Hitters.csv")
df = hit.copy()
df = df.dropna()
df.head()
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assists	Errors	Salary	NewLeague
1	315	81	7	24	38	39	14	3449	835	69	321	414	375	N	W	632	43	10	475.0	N
2	479	130	18	66	72	76	3	1624	457	63	224	266	263	A	W	880	82	14	480.0	A
3	496	141	20	65	78	37	11	5628	1575	225	828	838	354	N	E	200	11	3	500.0	N
4	321	87	10	39	42	30	2	396	101	12	48	46	33	N	E	805	40	4	91.5	N
5	594	169	4	74	51	35	11	4408	1133	19	501	336	194	A	W	282	421	25	750.0	A

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 263 entries, 1 to 321
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      263 non-null    int64
 1   Hits       263 non-null    int64
 2   HmRun      263 non-null    int64
 3   Runs       263 non-null    int64
 4   RBI        263 non-null    int64
 5   Walks      263 non-null    int64
 6   Years      263 non-null    int64
 7   CAtBat     263 non-null    int64
 8   CHits      263 non-null    int64
 9   CHmRun     263 non-null    int64
 10  CRuns      263 non-null    int64
 11  CRBI       263 non-null    int64
 12  CWalks     263 non-null    int64
 13  League     263 non-null    object
 14  Division   263 non-null    object
 15  PutOuts    263 non-null    int64
 16  Assists    263 non-null    int64
 17  Errors     263 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  263 non-null    object
dtypes: float64(1), int64(16), object(3)
memory usage: 43.1+ KB
```

- 263 tane değişkenden oluşmakta.
- 1 kesirli , 16 tam sayılı ve 3 tane kategorik değişkenden oluşuyor.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
AtBat	263.0	403.642586	147.307209	19.0	282.5	413.0	526.0	687.0
Hits	263.0	107.828897	45.125326	1.0	71.5	103.0	141.5	238.0
HmRun	263.0	11.619772	8.757108	0.0	5.0	9.0	18.0	40.0
Runs	263.0	54.745247	25.539816	0.0	33.5	52.0	73.0	130.0
RBI	263.0	51.486692	25.882714	0.0	30.0	47.0	71.0	121.0
Walks	263.0	41.114068	21.718056	0.0	23.0	37.0	57.0	105.0
Years	263.0	7.311787	4.793616	1.0	4.0	6.0	10.0	24.0
CAtBat	263.0	2657.543726	2286.582929	19.0	842.5	1931.0	3890.5	14053.0
CHits	263.0	722.186312	648.199644	4.0	212.0	516.0	1054.0	4256.0
CHmRun	263.0	69.239544	82.197581	0.0	15.0	40.0	92.5	548.0
CRuns	263.0	361.220532	331.198571	2.0	105.5	250.0	497.5	2165.0
CRBI	263.0	330.418251	323.367668	3.0	95.0	230.0	424.5	1659.0
CWalks	263.0	260.266160	264.055868	1.0	71.0	174.0	328.5	1566.0
PutOuts	263.0	290.711027	279.934575	0.0	113.5	224.0	322.5	1377.0
Assists	263.0	118.760456	145.080577	0.0	8.0	45.0	192.0	492.0
Errors	263.0	8.593156	6.606574	0.0	3.0	7.0	13.0	32.0
Salary	263.0	535.925882	451.118681	67.5	190.0	425.0	750.0	2460.0

- Betimsel istatistikleri yukarıda verilmiştir.

```
dms = pd.get_dummies(df[['League', 'Division',
'NewLeague']])
dms.head()
```

	League_A	League_N	Division_E	Division_W	NewLeague_A	NewLeague_N
1	0	1	0	1	0	1
2	1	0	0	1	1	0
3	0	1	1	0	0	1
4	0	1	1	0	0	1
5	1	0	0	1	1	0

- Kategorik değişkenlerimizi 1-0 değerlerine çevirdik. Dummy tuzağı olduğu da görülmekte.

```
X_ = df.drop(
["Salary","League","Division","NewLeague"],
axis = 1).astype("float64")
```

- Bu kodla Bağımlı değişkeni ve kategorik değişkeni df den çıkardık.

```
X = pd.concat([X_, dms[["League_N",
"Division_W","NewLeague_N"]]], axis = 1)
X.head()
```

- Yukarıdan yazdığımız dms(1-0 kodlular) ile X_ değişkenini birleştirdik.
- Böylelikle sol tarafta duran League,Division ve NewLeague yerine 1 ve 0 lardan oluşan değerler durmaktadır.

```
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.25,random_state= 42)

print("X_train", X_train.shape)
print("y_train",y_train.shape)
print("X_test",X_test.shape)
print("y_test",y_test.shape)

training = df.copy()
print("training", training.shape)
```

```
X_train (197, 19)
y_train (197,)
X_test (66, 19)
y_test (66,)
training (263, 20)
```

- Eğitim ve test olarak ayırma işlemlerimizi yaptık. Training genel verinin değişken sayısı.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
pca = PCA()
X_reduced_train =
pca.fit_transform(scale(X_train))
```

- Şimdi değişken kadar elimizde bileşen oluşturuldu.
- Aralarındaki bağlantı ortadan kalkmış , indirgenmiş veri seti var.
- Fit fonksiyonu da kullanılabilir fakat bu durumda model nesnesi oluşturulur. Transform ile hem x göre model oluşacak hemde boyut indirgeme işlemi yapacağız.

```
X_reduced_train[0:1,:]
```

```
array([[-2.49569913e+00, -3.37762397e-01,  7.06391950e-01,
        -1.32791025e+00, -8.21824333e-01, -6.62790677e-01,
        -6.56764789e-01,  3.68093279e-02, -2.03665105e-01,
         1.76134815e-01, -9.20131987e-02,  2.40129020e-01,
        -3.60473661e-03, -3.41246327e-02,  4.32799605e-02,
         1.02996923e-01,  3.70733348e-03,  1.37933445e-03,
        -6.63814471e-03]])
```

- Bileşenleri bu şekilde gözlemleyebiliyoruz.

```
np.cumsum(np.round(pca.explained_variance_rati
o_, decimals = 4)*100)[0:10]
```

```
array([38.18, 59.88, 70.88, 78.88, 84.18, 88.45, 92.05, 94.86, 96.34,
       97.28])
```

- İlk 10 değişkenle açıklanan varyans oranı baktığımız zaman 1. Bileşenin veri setinde bulunan toplam değişkenliğin(varyansın) %38 ini açıklarken 2. Birleşenin 1. Birleşenle birlikte açıkladığı %59.88 ve 10.ya baktığımızda %97 civarlarında açıklanmıştır.

```
lm = LinearRegression()
pcr_model = lm.fit(X_reduced_train, y_train)
pcr_model.intercept_
```

- Modeli kurduk. Dikkat üzerine X_reduced_train değişkenini yani indirgediğimiz değerler girildi. Sabit katsayı: 543.483441

```
pcr_model.coef_
```

```
array([ 111.13977427,  -29.34209502,   26.29799759,  -38.47549852,
        -56.9200785 ,   54.44779423,   40.77493384,  -23.72746012,
          9.31198164,   13.02031672,   45.58357748,   31.97791627,
         18.93930958, -115.60940171,   24.00382778,  415.70806202,
       -449.51779543,  563.07375399,  302.53718462])
```

- 19 değişkene indirgemiştik. Bu şekilde toplam 19 tane kat sayımız geldi.

3.Temel Bileşen Regresyon- PCR (Tahmin):

```
y_pred = pcr_model.predict(X_reduced_train)
y_pred[0:5]
```

```
array([377.44484744, 802.19452124, 495.60987745, 112.53177731,
       426.21613066])
```

- Tahmin edilen değerler yukarıda verilmiştir.

```
np.sqrt(mean_squared_error(y_train, y_pred))
```

- y_pred eğitim seti içerisindeki tahmin değerleri , rmse değeri ise 289.3292825564976

```
df["Salary"].mean()
```

- Maaş ortalaması: 535.9258821292775

```
r2_score(y_train, y_pred)
```

- R2 değeri: 0.5770075250410179

```
pca2 = PCA()
X_reduced_test =
pca2.fit_transform(scale(X_test))
y_pred = pcr_model.predict(X_reduced_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Test hatamız: 405.1575364149965

3.Temel Bileşen Regresyon- PCR (Model Tuning):

- Buraya kadar olan kısımda PCR ile indirgenen tüm bileşenleri kullanıp hesaplamalar yaptık. Fakat farklı bileşen sayılarında farklı sonuçlar elde edebiliriz.

```
lm = LinearRegression()
pcr_model = lm.fit(X_reduced_train[:,0:10],
y_train)
y_pred =
pcr_model.predict(X_reduced_test[:,0:10])
print(np.sqrt(mean_squared_error(y_test,
y_pred)))
```

- Şimdi x_reduced_train kısmında 0:10 arası bileşen aldığımızda 390 çıkarken hepsini aldığımızda 405 çıkıyor.

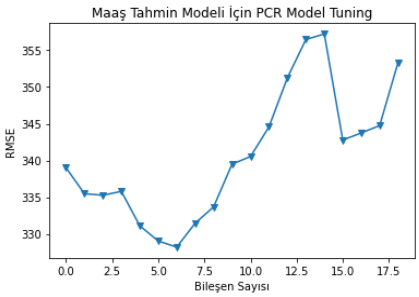
```
from sklearn import model_selection
cv_10 = model_selection.KFold(n_splits = 10,
                              shuffle = True,
                              random_state = 1)
```

- 10 katlı cross validation yapılandırması oluşturduk.
- Shuffle: Gruplara ayrılmadan önce verilerin karıştırılıp karıştırılmayacağı bilgisini taşıyor.

```
lm = LinearRegression()
RMSE = []
for i in np.arange(1, X_reduced_train.shape[1]
+ 1):
score =
np.sqrt(1*model_selection.cross_val_score(lm,
X_reduced_train[:,:i],
y_train.ravel(),  cv=cv_10,
scoring='neg_mean_squared_error').mean())
    RMSE.append(score)
```

- Her bileşen için model kurup, her bileşen için crossvalidation kurup hata değerlerini elde edip hangi bileşen daha az hata.

```
plt.plot(RMSE, '-v')
plt.xlabel('Bileşen Sayısı')
plt.ylabel('RMSE')
plt.title('Maaş Tahmin Modeli İçin PCR Model Tuning');
```



- İncelendiği üzere 6 bileşen alırsak hata sayımız daha az olduğu görülüyor.(Eğitim)

```
lm = LinearRegression()
pcr_model = lm.fit(X_reduced_train[:,0:6], y_train)

y_pred = pcr_model.predict(X_reduced_train[:,0:6])

print(np.sqrt(mean_squared_error(y_train, y_pred)))
```

- Eğitim için gerçek hata değerlerimiz: 308.8265983094501
- Şimdi test içinde yapalım.

```
y_pred = pcr_model.predict(X_reduced_test[:,0:6])
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```

- Test için hata değerimiz : 393.1198700096223
- Tuning öncesi test hatamız 405 küsür bir şey iken tuning sonrası test hatamız 393 çıktı.

- Şimdi genelde ilk olarak ilkel test ve eğitim hatası ölçeceğiz. Bunu daha doğru değerlendirme yolu crossvalidation yöntemi ile bunları incelemektir. İlk olarak bulduğumuz değerleri bu yöntemle daha doğru hatalar gelecektir.

- 2. basamağa geldiğimizde model tuning ile tuning ettiğimiz model için uygun olan hiper parametre değerini bulmak için crossvalidation yöntemi kullanılır. En iyi parametre değeri bulmak amaçlanır. Şimdi elimizde en iyi modelimiz var. Yine odağımızda test setimiz var çünkü dışarıda bırakılan değerleri değerlendirirken crossvalidation yöntemiyle yada direk test hatasına gidebiliriz.

- Bundan sonraki ilerleyişimiz:

1. Model tuining ile modellerin parametre değerlerini bulacağız.

2.Bunlarla final modelleri oluşturulacak.

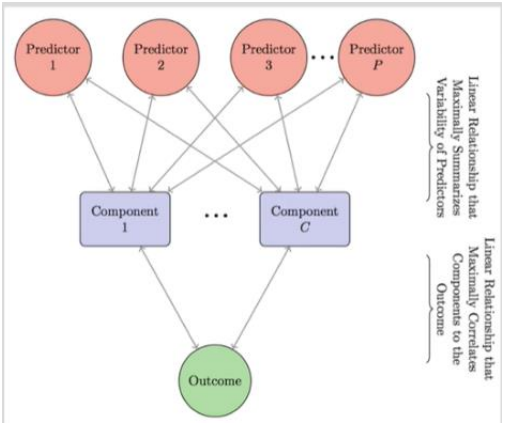
3.Bu final modelleri ile de son test hatalarını hesaplayacağız.

4.Bunu sınama seti yaklaşımı ile yapacağız ki bütün modellerle değerlendirme imkânı bulabilelim.

5. Bu şekilde elde edeceğimiz değer ise bizim artık en son tuning edilmiş modelimizin değeri olmuş olacak. Yani ulaşmak istediğimiz optimum test hatamız.

4.Kısmi En Küçük Kareler Regresyon- PLS (Teori):

- Değişkenlere daha az sayıda ve aralarında çoklu doğrusal bağlantı problemi olmayan bileşenlere indirgenip regresyon modeli kurulması fikrine dayanır.



- P değişken sayısından c sayısına indirgeyip tahmin işlemi gerçekleşir.

- Çok boyutluluk laneti p>n probleme çözüm sunar.
- Çoklu doğrusal bağlantı problemine çözüm sunar.
- PLS de PCR gibi bağımsız değişkenlerin doğrusal kombinasyonlarını bulur. Bu doğrusal kombinasyonlar bileşen ya da latent değişken olarak adlandırılır.
- PLS NİPALS'in özel bir halidir, iteratif olarak bağımsız değişken ile yüksek korelasyona sahip değişenler arasındaki gizli(latent) ilişkiyi bulmaya çalışır.

PCR İLE PLS ARASINDAKİ FARK

- PCR da doğrusal kombinasyonlar yeni bileşenler **bağımsız değişken uzağındaki değişkenliği** maksimum şekilde özetleyecek şekilde oluşturulur. Bu durum bağımlı değişkeni açıklama yeteneği olmamasına sebep olmaktadır.
- PLS te ise **bileşenler bağımlı değişken ile olan kovaryansı** maksimum şekilde özetleyecek şekilde oluşturulur.
- Değişkenler atılmak istenmiyorsa ve açıklanabilirlik aranıyorsa PLS kullanılır.
- PLS gözetimli boyut indirgeme prosedürü , PCR gözetimsiz boyut indirgeme prosedürü olarak görünür.
- İki yönteminde bir tuning parametresi vardır o da bileşen sayısıdır.
- Optimum bileşen sayısını belirlemek için CV yöntemi kullanılır.

4.Kısmi En Küçük Kareler Regresyon- PLS (Model):

```
hit = pd.read_csv("Hitters.csv")
df = hit.copy()
df = df.dropna()
ms = pd.get_dummies(df[['League', 'Division', 'NewLeague']])

y = df["Salary"]
X_ = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).astype('float64')

X = pd.concat([X_, dms[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

- Yukarıda genel işlemler yapılmıştır. Test, eğitimleri ayırdık vs.

```
from sklearn.cross_decomposition import
PLSRegression, PLSSVD
pls_model = PLSRegression().fit(X_train,
y_train)
pls_model.coef_
```

```
array([[ 35.32916493,  48.83425857,  18.50240933,  39.28117603,
         30.59952998,  40.03398345,  16.85990516,  28.22289896,
         32.73784993,  22.00875744,  33.60903032,  30.39402522,
         25.73279799,  54.98835148,   6.56590871,  -0.90894359,
         17.60903423, -37.24246339,  14.69680385]])
```

- Modelimizin değişken sayısı kadar kat sayılarına erişmiş bulunmaktayız.
- PLSRegression() içerisine n_components=6 argümanı yazılırsa yine aynı sayıda kat sayı gelecektir. Pek takılmayınız.

4.Kısmi En Küçük Kareler Regresyon- PLS (Tahmin):

```
pls_model.predict(X_train)[0:10]
```

```
array([[344.91941493, 848.87070769, 692.93622642, 185.56577984,
        435.49196077, 987.49530026, 120.63097106, 289.9263406 ,
        663.41886918, 817.90486641]])
```

- Tahmin değerlerimiz yukarıda verilmiştir. Bu işlemler bizim veri dizimizdeki her bir değişkenin değeri ile yukarıda bulduğumuz değişkenlerin baş kat sayılarıyla çarpıp toplayıp bize tahmin değerlerimizi buluyor.
- Mesela 344 olan değer 1. Satırın, 848 2. Satırımızın tahmin değeri vs. diye verinin son satırına kadar gider.

```
y_pred = pls_model.predict(X_train)
np.sqrt(mean_squared_error(y_train, y_pred))
```

- Tahmin değerlerimizi y_pred değişkenine atayıp buradan eğitim hatamızı bulduk: 310.1167593109696

```
r2_score(y_train, y_pred)
```

- R kare değerimiz: 0.5140424486535481 (Eğitim)

```
y_pred = pls_model.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- 1. Satırla X_test değerlerini tahmin et yani başkatsayıları ile çarpıp tahmin etmesini istedik.
- Test Hatamız: 398.09956327448526

4.Kısmi En Küçük Kareler Regresyon- PLS (Model Tuning):

```
cv_10 = model_selection.KFold(n_splits=10,
shuffle=True, random_state=1)
```

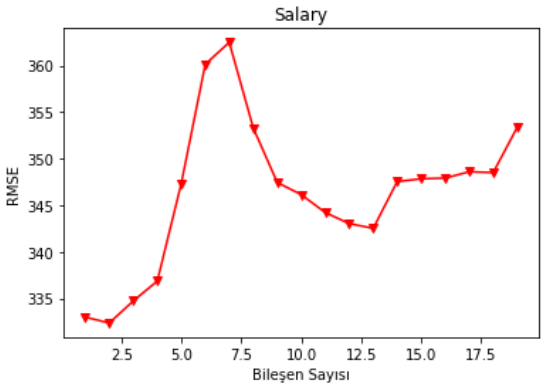
- Cross validation yöntemi oluşturduk 10 katlı.

```
RMSE = []
for i in np.arange(1, X_train.shape[1] + 1):
    pls = PLSRegression(n_components=i)
    score = np.sqrt(-1*cross_val_score(pls,
X_train, y_train, cv=cv_10,
scoring='neg_mean_squared_error').mean())
    RMSE.append(score)
```

- Hata hesaplama işlemleri. 8. Sayfa sonunda detaylı anlatılmıştır.

```
plt.plot(np.arange(1, X_train.shape[1] + 1),
np.array(RMSE), '-v', c = "r")
plt.xlabel('Bileşen Sayısı')
plt.ylabel('RMSE')
plt.title('Salary');
```

- Görselleştirme işlemi yapıldı. En küçük değer 2 de.




```
pls_model = PLSRegression(n_components = 2)
.fit(X_train, y_train)
y_pred = pls_model.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Hata değerimiz: 398.09956327448526
- Final modelimizi kurduk. Test hatası incelendi.
- Önceki ile aynı sonucu verdi. Sanırım n_components'i girmediğimizde otomatik olarak 2 alıyor.
- Grafiğe göre neden aynı sonuç çıkmadı orasını anlamadım.

5.Ridge Regresyon (Teori):

- Amaç hata kareler toplamını minimize eden kat sayılara bir ceza işlemi uygulayarak bulmaktır.

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$SSE_{L_2} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{P}\beta_j{}^2$$

Ayar Parametresi Lambda — Ceza Terimi

- L2 : Düzenlileştirme yöntemleri

Özellikler

- Aşırı öğrenmeye karşı dirençlidir.
- Yanlıdır fakat varyansı düşüktür.(Bazen yanlı modelleri daha çok tercih ederiz.
- Çok fazla parametre olduğunda EKK'ya göre daha iyidir.
- Çok boyutlu lanetine karşı çözüm sunar
- Çoklu doğrusal bağlantı problemi olduğunda etkilidir.
- Tüm değişkenler ile model kurar. İlgisiz değişkenleri modelden çıkarmaz, katsayılarını sıfıra yaklaştırır.
- ƛ kritik roldedir. İki terimin (formüldeki) göreceli etkilerini kontrol etmeyi sağlar.
- ƛ için iyi bir değer bulunması önemlidir. Bunun için CV yöntemi kullanılır.

5.Ridge Regresyon (Model):

```
hit = pd.read_csv("Hitters.csv")
df = hit.copy()
df = df.dropna()
ms = pd.get_dummies(df[['League', 'Division',
'NewLeague']])
y = df["Salary"]
X_ = df.drop(['Salary', 'League', 'Division',
'NewLeague'], axis=1).astype('float64')
X = pd.concat([X_, dms[['League_N',
'Division_W', 'NewLeague_N']]], axis=1)
X_train, X_test, y_train, y_test =
train_test_split(X, y,test_size=0.25,
random_state=42)
```

- Test train işlemleri gerçekleştirildi.

```
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha = 0.1).fit(X_train,
y_train)
```

- Alpha 0.1 olarak ayarladık. Model kuruldu.

```
ridge_model
```

```
Ridge(alpha=0.1)
```

- Bizim λ değerimiz. Bazı kaynaklarda alpha olarak da gözükebilir.

```
ridge_model.coef_
```

```
array([ -1.77435737,   8.80240528,   7.29595605,  -3.33257639,
        -2.08316481,   5.42531283,   7.58514945,  -0.13752764,
        -0.20779701,  -0.60361067,   1.7927957 ,   0.72866408,
        -0.68710375,   0.26153564,   0.26888652,  -0.52674278,
       112.14640272, -99.80997876, -48.07152768])
```

- Baş katsayıları getirmiş olduk.(λ=1 olduğunda)
- Şimdi çeşitli lamda değerlerinde inceliyelim.

```
10**np.linspace(10,-2,100)*0.5
```

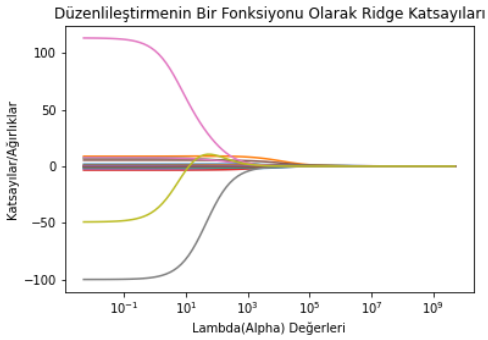
```
array([5.00000000e+09, 3.78231664e+09, 2.86118383e+09, 2.16438064e+09,
       1.63727458e+09, 1.23853818e+09, 9.36908711e+08, 7.08737081e+08,
       5.36133611e+08, 4.05565415e+08, 3.06795364e+08, 2.32079442e+08,
       1.75559587e+08, 1.32804389e+08, 1.00461650e+08, 7.59955541e+07,
       5.74878498e+07, 4.34874501e+07, 3.28966612e+07, 2.48851178e+07,
       1.88246790e+07, 1.42401793e+07, 1.07721735e+07, 8.14875417e+06,
       6.16423370e+06, 4.66301673e+06, 3.52740116e+06, 2.66834962e+06,
       2.01850863e+06, 1.52692775e+06, 1.15506485e+06, 8.73764200e+05,
       6.60970574e+05, 5.00000000e+05, 3.78231664e+05, 2.86118383e+05,
       2.16438064e+05, 1.63727458e+05, 1.23853818e+05, 9.36908711e+04,
       7.08737081e+04, 5.36133611e+04, 4.05565415e+04, 3.06795364e+04,
       2.32079442e+04, 1.75559587e+04, 1.32804389e+04, 1.00461650e+04,
       7.59955541e+03, 5.74878498e+03, 4.34874501e+03, 3.28966612e+03,
       2.48851178e+03, 1.88246790e+03, 1.42401793e+03, 1.07721735e+03,
```

- Bir sürü lamda değeri oluşturduk. Bu liste aşağıda kadar devam ediyor. Ben sadece 1 bölümünü aldım.

```
lambdalar = 10**np.linspace(10,-2,100)*0.5

ridge_model = Ridge()
katsayilar = []

for i in lambdalar:
    ridge_model.set_params(alpha = i)
    ridge_model.fit(X_train, y_train)
    katsayilar.append(ridge_model.coef_)
ax = plt.gca()
ax.plot(lambdalar, katsayilar)
ax.set_xscale('log')
plt.xlabel('Lambda(Alpha) Değerleri')
plt.ylabel('Katsayılar/Ağırlıklar')
plt.title('Düzenlileştirmenin Bir Fonksiyonu
Olarak Ridge Katsayıları');
```



- Her lamda değeri geldiğinde 19 tane değer olduğu için her 1 lamda geldiğinde 19 tane kat sayı olacak.
- Renklerin her biri parametre ifade etmektedir.
- Bu parametreleri göründüğü üzere 0'a yaklaştırmış lakin 0 olmamıştır.
- Katsayılara bir ceza uyguladık ama öyle bir ceza uyguladık ki belirlenecek lamda değeri göre bu kat sayılar bazen etkilerini önemlerine göre kaybetmiş oluyor.
- Tahmin fonksiyonundaki etkilerini biz ayarlayabilmiş oluyoruz. Ama yinede bunlar modelde kalsın 0'a yaklaştıralım ama modelden çıkarmayalım mantığına dayanır.

5.Ridge Regresyon (Tahmin):

```
y_pred = ridge_model.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Test hatamız: 357.0517922282835

5.Ridge Regresyon (Model Tuning):

- Amaç optimum lamda değerlerini ulaşmaktır. Yukarıdaki grafik işleminde çeşitli lamda değerlerinde kat sayılarının neler olabileceğini gözlemledik. Bu lamda değerlerinden hangisinin iyi olduğunu yapcaz.

```
lambdalar = 10**np.linspace(10,-2,100)*0.5
from sklearn.linear_model import RidgeCV
ridge_cv = RidgeCV(alphas = lambdalar,
           scoring = "neg_mean_squared_error",
           normalize = True)
```

- 1. Satır yukarı kısmın işlemi.
- 3. Satırda model oluşturuldu.
- Yukarıdaki gibi döngü vs uğraşmadık.

```
ridge_cv.fit(X_train, y_train)
ridge_cv.alpha_
```

- Modeli fit edip alpha değeri çağırdığımızda: 0.759955541476
- Şimdi final modeli oluşturalım.

```
ridge_tuned = Ridge(alpha = ridge_cv.alpha_,
normalize = True).fit(X_train,y_train)
```

- Final modeli oluşturup fit ettik. Tek satırlık bir kod bu arada.

```
np.sqrt(mean_squared_error(y_test,
ridge_tuned.predict(X_test)))
```

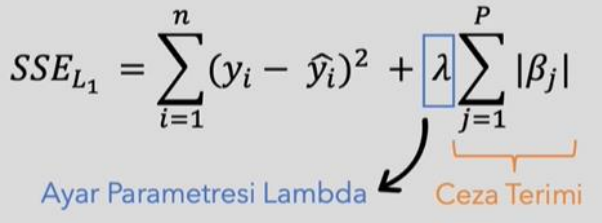
- Test hatamız: 386.6826429756415

- Normalde test hatasının düşmesini bekleriz. Ama validasyon yöntemlerinden dolayı değerler değişecektir. Her zaman küçük olan daha iyidir denemez. Bazen daha doğrusuna erişmek isteriz. Burada elinizdeki daha doğrusudur.

6.Lasso Regresyon (Teori):

- Amaç hata kareler toplamını minimize eden bu kat sayılara bir ceza işlemi uygulayarak bulmaktır

- Ridge den farklı olarak bu ceza işlemlerini biraz daha abartarak katsayıların cezalarını onları 0 yapacak şekilde uygulanır.



- Ridgeden farklı olarak L1 değeri için Bj karesi yerine mutlak değere alınmıştır.

Özellikler

- Ridge regresyonun ilgili-ilgisiz tüm değişkenleri modelde bırakma dezavantajını gidermek için önerilmiştir.
- Lasso'da katsayıları sıfıra yaklaştırır.
- Fakat L1 formu ƛ yeteri kadar büyük olduğunda bazı katsayıları sıfır yapar. Böylece değişken seçimi yapılmış olur.
- ƛ'nın doğru seçilmesi çok önemlidir. Burada da CV kullanılır.
- Ridge ve Lasso yöntemleri birbirinden üstün değildir.

ƛ Ayar Parametresinin Belirlenmesi

- ƛ sıfır olduğu yer EKK'dır. HTK'yi minimum yapan ƛ değeri arıyoruz.
- ƛ için belirli değerleri içeren bir küme seçilir ve her birisi için cross validation test hatası hesaplanır
- En küçük cross validation'ı veren ƛ ayar parametresi olarak seçilir.
- Son olarak seçilen bu ƛ ile model yeniden tüm gözlemlere fit edilir.

6.Lasso Regresyon (Model):

```
hit = pd.read_csv("Hitters.csv")
df = hit.copy()
df = df.dropna()
ms = pd.get_dummies(df[['League', 'Division',
'NewLeague']])
y = df["Salary"]
X_ = df.drop(['Salary', 'League', 'Division',
'NewLeague'], axis=1).astype('float64')
X = pd.concat([X_, dms[['League_N',
'Division_W', 'NewLeague_N']]], axis=1)
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.25,
random_state=42)
```

- Veri setimiz yine aynı. Test train işlemleri gerçekleştirildi.

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha = 0.1).fit(X_train,
y_train)
```

- Model işlemi hal edildi.

```
lasso_model
```

```
Lasso(alpha=0.1)
```

- Bizim ƛ değerimiz 1 olarak girildi. Göstermiş olduk.

```
lasso_model.coef_
```

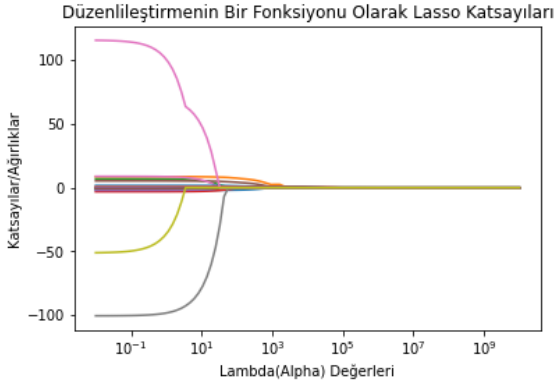
```
array([-1.72206506e+00,  8.56210197e+00,  6.91175137e+00, -3.13240128e+00,
       -2.00771676e+00,  5.36159035e+00,  8.86871593e+00, -1.69520371e-01,
       -5.58121413e-02, -3.53962588e-01,  1.70961000e+00,  6.40603469e-01,
       -6.58519895e-01,  2.60093222e-01,  2.78717030e-01, -5.92690965e-01,
        1.12659630e+02, -9.99652090e+01, -4.81289395e+01])
```

- Model katsayıları yukarıda verilmiştir.

```
lasso = Lasso()
lambdalar = 10**np.linspace(10,-2,100)*0.5
katsayilar = []

for i in lambdalar:
    lasso.set_params(alpha=i)
    lasso.fit(X_train, y_train)
    katsayilar.append(lasso.coef_)

ax = plt.gca()
ax.plot(lambdalar*2, katsayilar)
ax.set_xscale('log')
plt.axis('tight')
plt.xlabel('Lambda(Alpha) Değerleri')
plt.ylabel('Katsayılar/Ağırlıklar')
plt.title('Düzenlileştirmenin Bir Fonksiyonu
Olarak Lasso Katsayıları');
```



- Ridge ile farkı 0 noktasına yaklaştıktan sonraki kısımda kesinlikle 0 olmasıdır.

6.Lasso Regresyon (Tahmin):

```
lasso_model.predict(X_test)
```

```
array([ 613.88833029,  701.97056731, 1005.55539526,  414.08312603,
        399.18417127,  344.71444139,  664.86990217,  451.60757   ,
        914.64492066,  644.67006406,  691.60613554,  884.71702368,
        210.04523766,  446.12527252,  262.94922087,  499.22332142,
        805.2275034 ,   43.49230343, 1250.49322312,  316.34470193,
```

- Yukarıda her bir ƛ değeri için tahmin değerleri verilmiştir.(her biri bağımlı değişken değeri)
- Aşağıya devam etmektedir, ben bir kısmını aldım.

```
y_pred = lasso_model.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Test hatamız: 356.7545270148771

6.Lasso Regresyon (Model Tuning):

```
from sklearn.linear_model import LassoCV
lasso_cv_model = LassoCV(alphas = None,
                         cv = 10,
                         max_iter = 10000,
                         normalize = True)
lasso_cv_model.fit(X_train,y_train)
lasso_cv_model.alpha_
```

- Alpha değerini None girerek kendisi bir alpha değer seçmesini sağladık. Bazen bu fonksiyonların yerel ön tanımlı değerlerine işi bırakıp kendisinin bulması mantıklı olur.
- Alpha değeri kendisi aldığı değer: 0.39406126432470073

```
lasso_tuned = Lasso(alpha =
lasso_cv_model.alpha_)
lasso_tuned.fit(X_train, y_train)
```

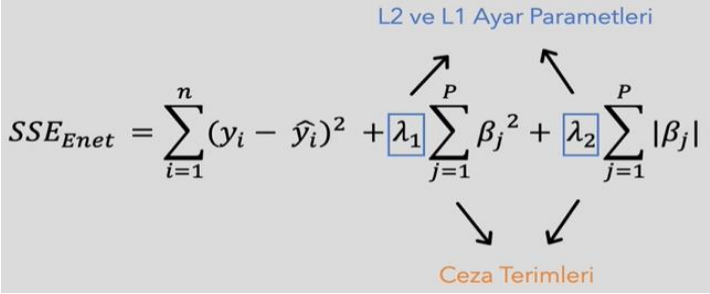
- Final modelimizi(tuning) işlemi gerçekleştirildi.

```
y_pred = lasso_tuned.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Test hatamız: 356.5226376958366

7.ElasticNet (ENET) Regresyon (Teori):

- Amaç hata kareler toplamını minimize eden bu kat sayılara bir ceza işlemi uygulayarak bulmaktır. ElasticNet L1 ve L2 yaklaşımlarını birleştirir.



Özellikler

- Daha etkili düzenlileştirme ya da düzgünleştirme işlemi yapılmasını sağlar.
- EKK'nın problemlerini ortadan kaldırılmasını sağlamaya yarar.
- Ridge tarzı cezalandırma Lasso tarzı değişken seçme işlemini bir araya getirerek iki işlemi birlikte göz önünde bulunduruyor.
- Doğrusal regresyon modelinin en geliştirilmiş halidir.

7.ElasticNet (ENET) Regresyon (Model):

```
hit = pd.read_csv("Hitters.csv")
df = hit.copy()
df = df.dropna()
ms = pd.get_dummies(df[['League', 'Division',
'NewLeague']])
y = df["Salary"]
X_ = df.drop(['Salary', 'League', 'Division',
'NewLeague'], axis=1).astype('float64')
X = pd.concat([X_, dms[['League_N',
'Division_W', 'NewLeague_N']]], axis=1)
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.25,
random_state=42)
```

- Aynı veri seti ile test train işlemleri gerçekleştirildi.

```
from sklearn.linear_model import ElasticNet
enet_model = ElasticNet().fit(X_train, y_train)
```

- Model kuruldu.

```
enet_model.coef_
```

```
array([ -1.86256172,   8.70489065,   5.10426375,  -2.89875799,
        -1.28642985,   5.24343682,   6.04480276,  -0.14701495,
        -0.21566628,  -0.7897201 ,   1.80813117,   0.80914508,
        -0.61262382,   0.26816203,   0.27172387,  -0.36530729,
        19.2186222 , -31.16586592,   8.98369938])
```

- Kat sayılar yukarıda verilmiştir.

```
enet_model.intercept_
```

- Sabit katsayımız : -6.465955602112331

7.ElasticNet (ENET) Regresyon (Tahmin):

```
enet_model.predict(X_test)[0:10]
```

```
array([ 577.79111731,  617.33202224, 1031.39113156,  364.95861575,
        489.51894393,  300.74185842,  604.522666  ,  465.34678732,
        901.44473965,  703.20357123])
```

- Test değerlerimizde tahmin değerlerinin ilk 10 terimi.

```
y_pred = enet_model.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Test hata değerimiz: 357.1676548181246(RMSE)

```
r2_score(y_test, y_pred)
```

- R kare değerimiz: 0.41070222469326867 (Açıklanabilirlik oran)

7.ElasticNet (ENET) Regresyon (Model Tuning):

```
from sklearn.linear_model import ElasticNetCV
enet_cv_model = ElasticNetCV(cv = 10,
random_state = 0).fit(X_train, y_train)
```

- Eğitim seti üzerinden modelimiz kuruldu.

```
enet_cv_model.alpha_
```

- Alpha değeri: 5230.7647364798695
- Şimdi elde edilen alpha değerini final modeli için kullanalım.

```
enet_tuned = ElasticNet(alpha =
enet_cv_model.alpha_).fit(X_train,y_train)
```

- Final modeli kuruldu.

```
y_pred = enet_tuned.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

- Final test hatası: 394.15280563218795

Bölüm Sonu

- ElasticNet test hatası: 394.15 Lasso Test hatamız: 356.75 Ridge test hatası: 386.68

Sonuçlara bakıldığı zaman bu veri seti için çalışan en iyi model Lasso Modeli olduğu görünmektedir.

- Hocanın Bölüm Sonu Notu Aşağıda Verilmiştir.

Önemli Not:

Birinci konu:
"alphas" parametresi için bir liste oluşturmadık ve bunu modele bıraktık. ElasticNetCV fonksiyonu uygun olan değeri buldu. Eğer oluşturulmak istenirse alphas = [.,.,.,] şeklinde bir liste oluşturulabilir ve ElasticNetCV içerisine argüman olarak ifade edilebilir.

İkinci konu:
"l1_ratio" parametresi. Bu parametre değeri 0 olduğunda L2 cezalandırması, 1 olduğunda ise L1 cezalandırması yapar. Dolayısıyla 0-1 arasında değişimi aslında cezalandırma metodlarının göreceli etkilerini ifade eder. Uygulamamızda ikisininde etkisinin aynı düzeyde olmasını istediğimizden dolayı bunun ön tanımlı değerini olduğu gibi bıraktık: 0.5.

Üçüncü konu:
"l1_ratio" ve "alphas" beraber düşünülürse "l1_ratio" için aranacak bir liste verildiğinde yani ceza türlerinin etkilerini ayarlamak için bir liste verildiğinde her bir "l1_ratio" değeri için farklı "alphas" değerleri olacaktır ki bu durumda bu alphalar ceza terimleri etkilerini modele yansıtmış olacak.