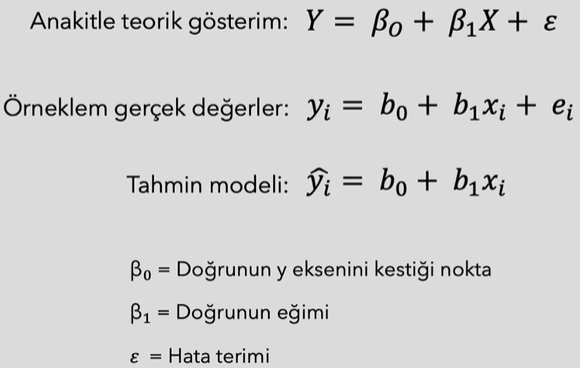
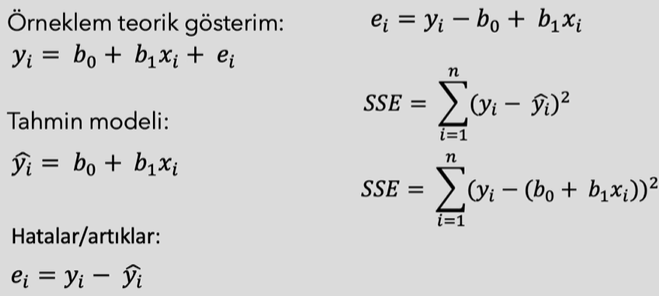
**Doğrusal Regresyon Ve Kuzenleri**

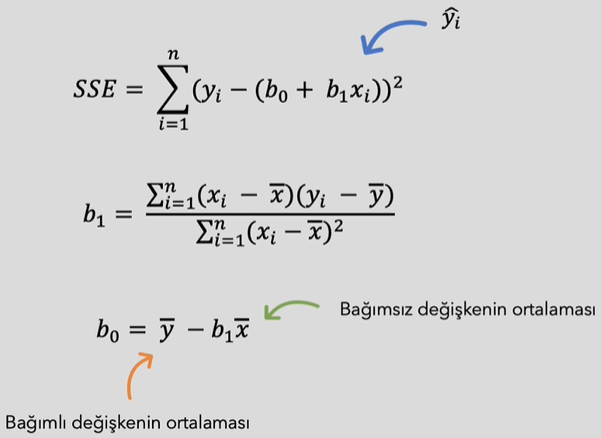
* Basit Doğrusal Regresyon
* Çoklu Doğrusal Regresyon
* Temel Birleşen Regresyonu
* Kısmı En Küçük Kareler Regresyonu
* Ridge Regresyon
* Lasso Regresyon
* Elastic Net Regresyonu
* Her Model İçin
* Model
* Tahmin
* Model Optimizasyonu

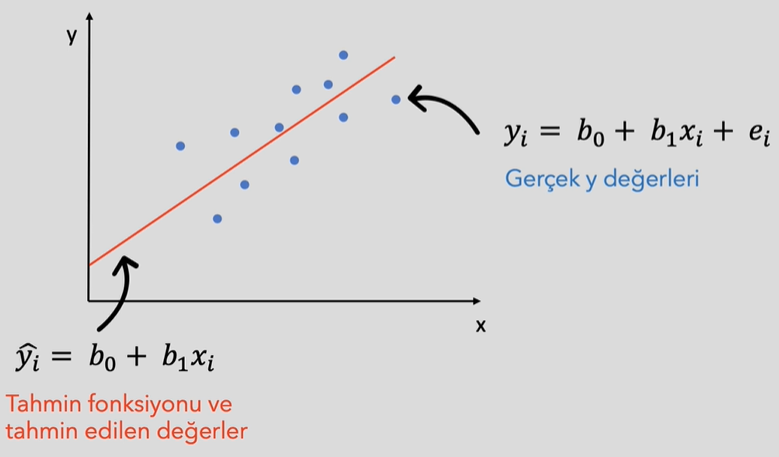
1.Basit Doğrusal Regresyon (Teori):

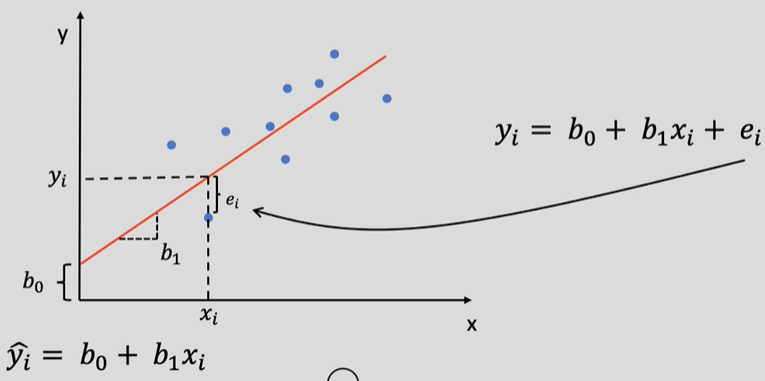
* Temel amaç bağımlı-bağımsız değişken arasındaki ilişkiyi ifade eden doğrusal fonksiyonu bulmaktır.











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

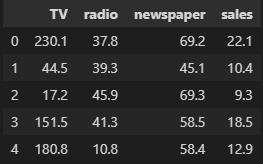
1.Basit Doğrusal Regresyon (Model):

import pandas as pd

ad = pd.read\_csv("Advertising.csv", usecols = [1,2,3,4])

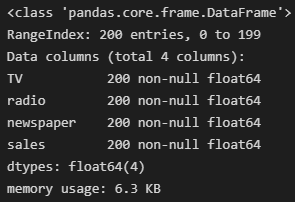
df = ad.copy()

df.head()

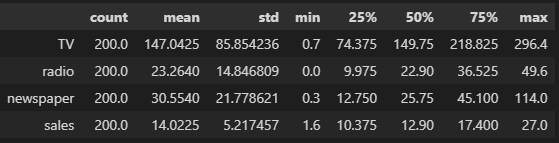


* Usecols koymamızın sebebi Tv nin sol tarafında unnamed diye bir değişken vardı ve 1 2 3 4 diye aşağıya iniyordu. Onu kaldırdık.

df.info()



* 4 tane sayısal değişkenimiz olduğunu gözlemledik.

df.describe().T

* Genel betimsel istatistiklerine baktık.

df.isnull().values.any()



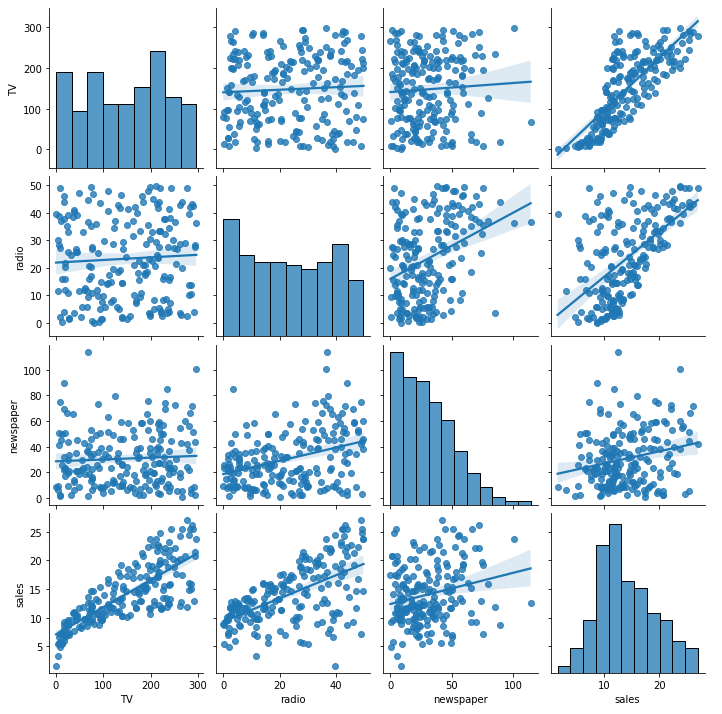
* Veri seti içerisinde eksik değer yokmuş.

df.corr()

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

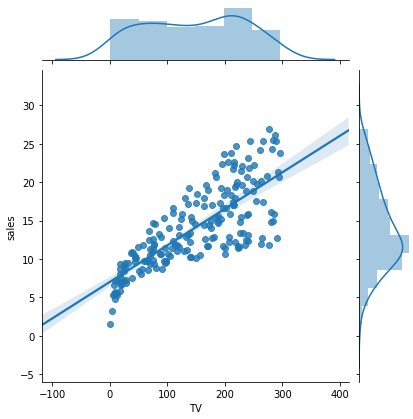
import seaborn as sns

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")

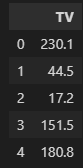


* Yukarıdaki ve sağdaki tepelenmelere baktığımızda TV 2 tepeli bir dağılım söz konusuyken satışta tek tepeli bir grafik gözlemledik.

import statsmodels.api as sm

X = df[["TV"]]

X[0:5]



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

X = sm.add\_constant(X)

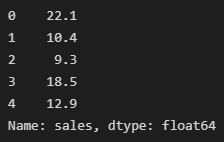
X[0:5]



* X bağımsız değişkenimiz hazır (TV)

y = df["sales"]

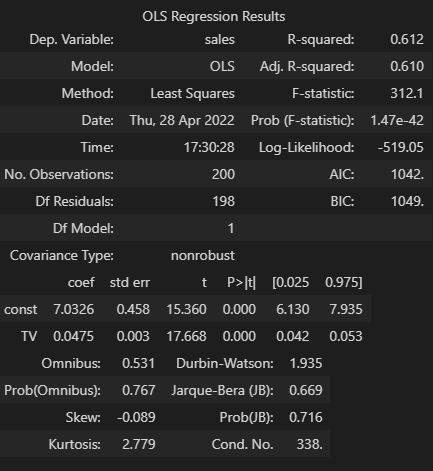
y[0:5]



* Y bağımlı değişkenimiz hazır (sales-satış)

lm = sm.OLS(y,X)

model = lm.fit()

model.summary()

* 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

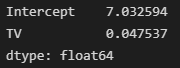
lm = smf.ols("sales ~ TV", df)

model = lm.fit()

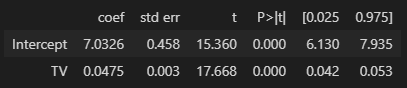
model.summary()

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

model.params

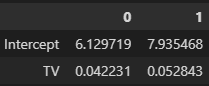


* Model parametreleri kısa yollu çağırdık.

model.summary().tables[1]

* 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()



* Sadece güven aralığı olduğu kısma bu şekilde erişebiriz.

[0.025 0.0975] kısmı.

print("f\_pvalue: ", "%.4f" % model.f\_pvalue)



* Sadece f\_pvalue değerine bu şekilde erişilebilir. %.4f kısmı noktadan sonra kaç basamak olduğunu belirtir.

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



* Sadece fvalue değerine bu şekilde erişilebilir.

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



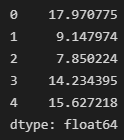
* Sadece parametre değerine bu şekilde erişilebilir. Tahmin değerlerine ulaşılabilmesi için değişkenler katsayılar(parametre) ile çarpılır.

model.rsquared\_adj



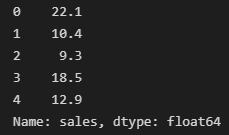
* Açıklana bilirlik değeri.(düzeltilen)
* Sonundaki adj silinirse direk R-squared değerine ulaşılır.

model.fittedvalues[0:5]



* Tahmin edilen değerler yan tarafta verilmiştir.

y[0:5]



* 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})

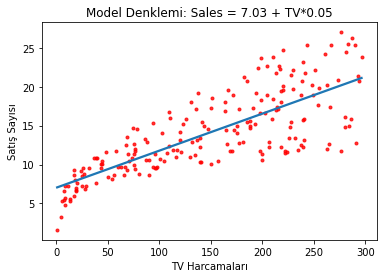
g.set\_title("Model Denklemi: Sales = 7.03 + TV\*0.05")

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\_)



* 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]



* 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]])



* 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)



**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]

* Tahmin değerlerine ulaştık.

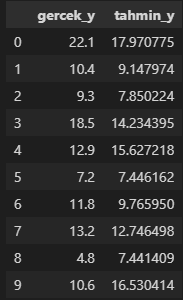
y[0:10]

* 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]})

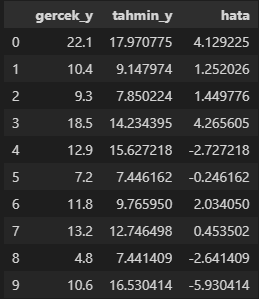
k\_t



* 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



* 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

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"]))

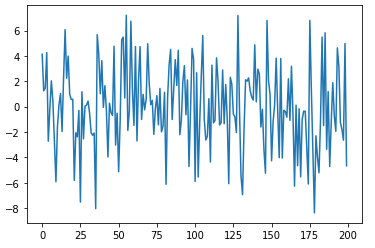
* Hata karelerinin ortalamasının karekökü : 3.0480075

model.resid[0:10]

* Modelin artıkları (Hataları) yukarıdaki gibidir.

import matplotlib.pyplot as plt

plt.plot(model.resid)



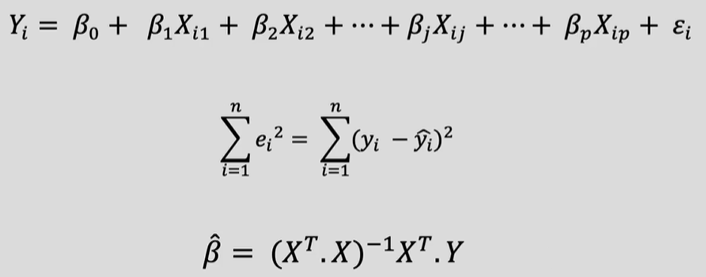
* Modelin artıkları (Hataları) yukarıdaki gibi görselleştirdik.
* 8’e 7’ye vuran değerleri gidip incelemek gerekir (Teknik Detay)

2.Çoklu Doğrusal Regresyon (Teori):

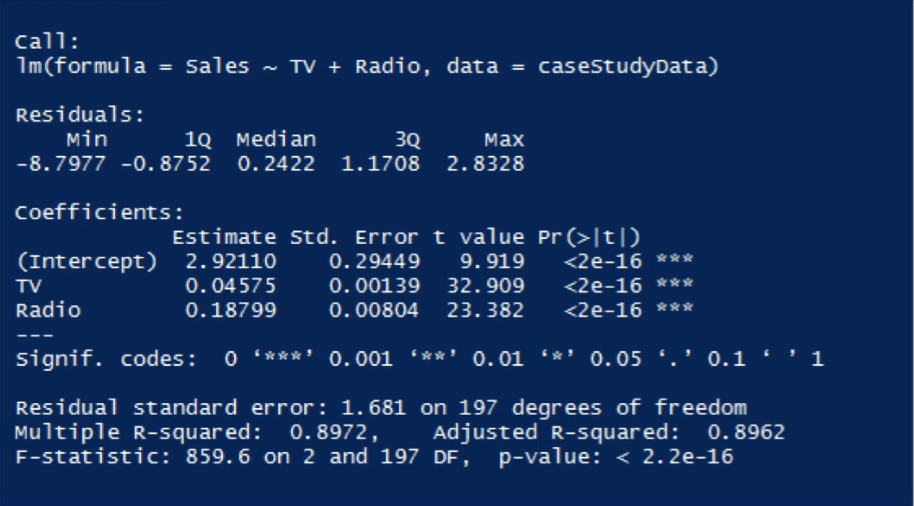
- Temel amaç bağımlı ve bağımsız değişkenler arasındaki ilişkiyi ifade eden doğrusal fonksiyonu bulmaktır. Genelde 2 şekilde amaç var:

+ 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.



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



* 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.

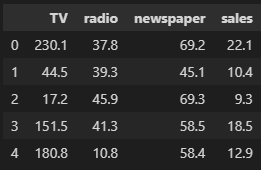
2.Çoklu Doğrusal Regresyon (Modelleme):

import pandas as pd

ad = pd.read\_csv("Advertising.csv", usecols = [1,2,3,4])

df = ad.copy()

df.head()



* 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)



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

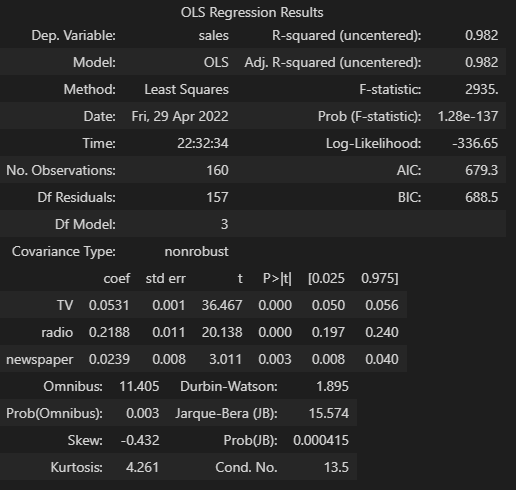
training = df.copy()

* 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.

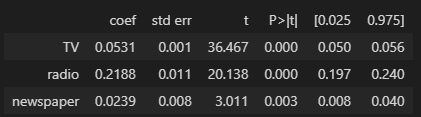
**Stats Models**

lm = sm.OLS(y\_train, X\_train)

model = lm.fit()

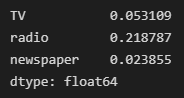
model.summary()

* 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]

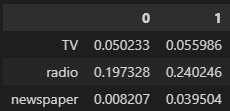
* Sadece kat sayılarla ilgili yere yukarıdaki kodla ulaşılabilir.
* En üst kısım 0. Orta kısım 1. En alt kısım 2. İndeks olarak geçer

model.params



* Model parametreleri kısa yollu çağırdık.

model.conf\_int()



* 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.

model.coef\_

* 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)



* 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,

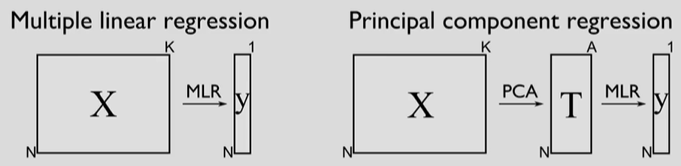
                cv = 10,

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.

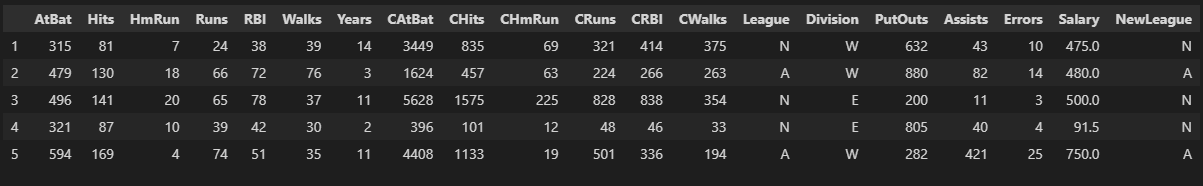
3.Temel Bileşen Regresyon- PCR (Model):

hit = pd.read\_csv("Hitters.csv")

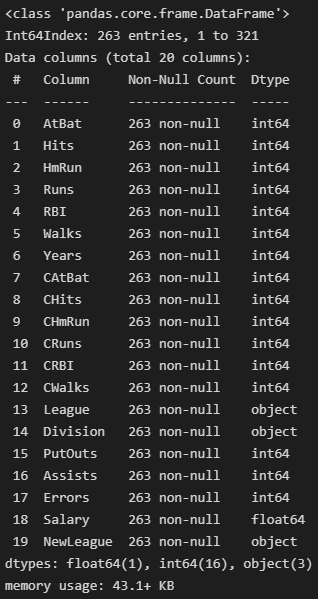
df = hit.copy()

df = df.dropna()

df.head()

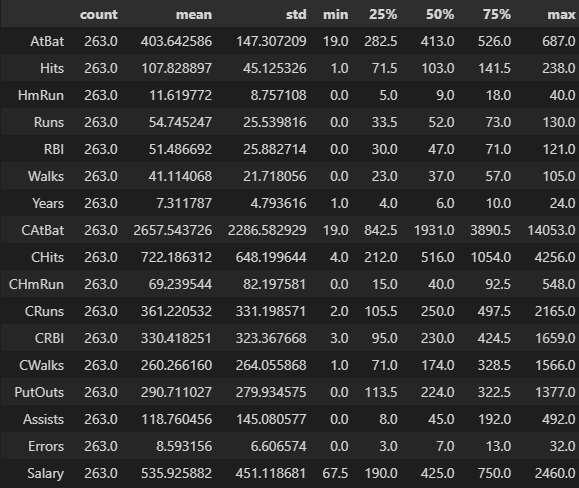


df.info()



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

df.describe().T



* Betimsel istatistikleri yukarıda verilmiştir.

dms = pd.get\_dummies(df[['League', 'Division', 'NewLeague']])

dms.head()



* 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)

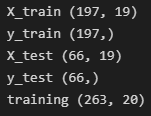
print("y\_train",y\_train.shape)

print("X\_test",X\_test.shape)

print("y\_test",y\_test.shape)

training = df.copy()

print("training", training.shape)



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

from sklearn.decomposition import PCA

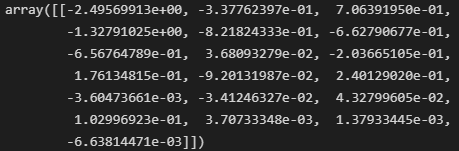
from sklearn.preprocessing import scale

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,:]



* Bileşenleri bu şekilde gözlemleyebiliyoruz.

np.cumsum(np.round(pca.explained\_variance\_ratio\_, decimals = 4)\*100)[0:10]

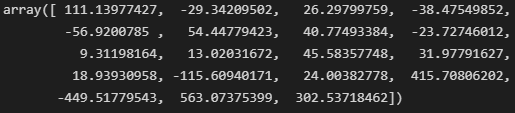
* İlk 10 değişkenle açıklanan varyans oranı baktığımız zaman 1. Bileşenin veri setinde bulunan toplam değişkenliğin(varyansın) %38 ini açıklarken 2. Birleşenin 1. Birleşenle birlikte açıkladığı %59.88 ve 10.ya baktığımızda %97 civarlarında açıklanmıştır.

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\_

* 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]

* 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)

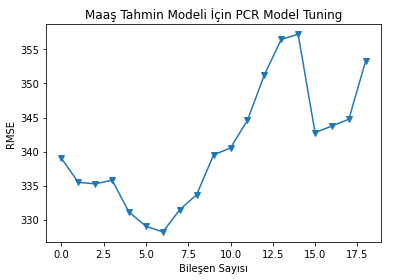
* Her bileşen için model kurup, her bileşen için crossvalidation kurup hata değerlerini elde edip hangi bileşen daha az hata.

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.

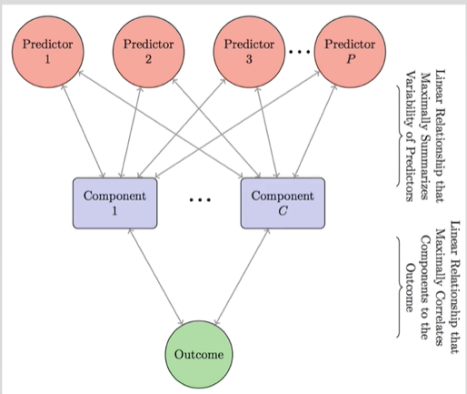
2.Bunlarla final modelleri oluşturulacak.

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')

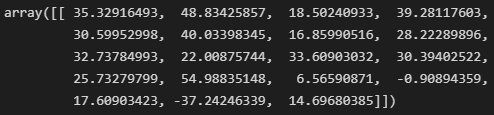
X = pd.concat([X\_, dms[['League\_N', 'Division\_W', 'NewLeague\_N']]], axis=1)

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.

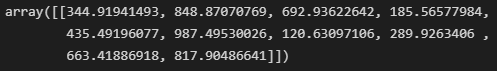
from sklearn.cross\_decomposition import PLSRegression, PLSSVD

pls\_model = PLSRegression().fit(X\_train, y\_train)

pls\_model.coef\_

* 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]

* 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)

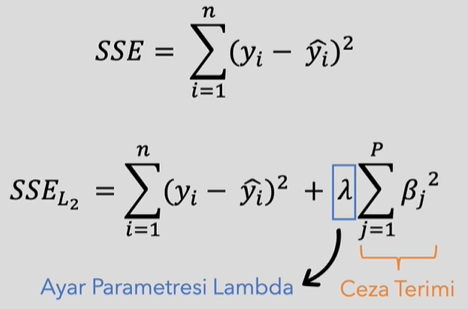
y\_pred = pls\_model.predict(X\_test)

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.



* 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

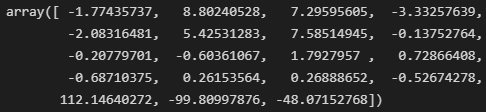
ridge\_model = Ridge(alpha = 0.1).fit(X\_train, y\_train)

* Alpha 0.1 olarak ayarladık. Model kuruldu.

ridge\_model

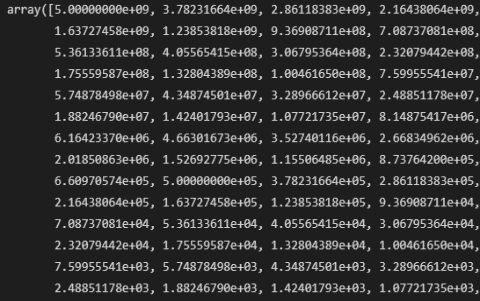
* Bizim ƛ değerimiz. Bazı kaynaklarda alpha olarak da gözükebilir.

ridge\_model.coef\_



* Baş katsayıları getirmiş olduk.( ƛ=1 olduğunda)
* Şimdi çeşitli lamda değerlerinde inceliyelim.

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



* 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()

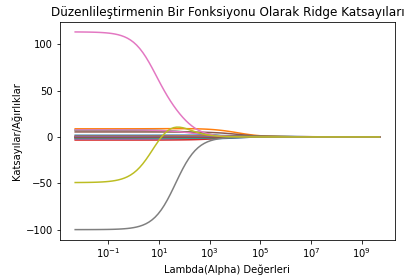
ax.plot(lambdalar, katsayilar)

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)))

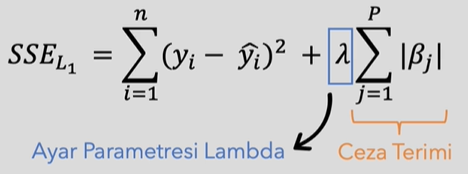
* Test hatamız: 386.6826429756415

- 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.

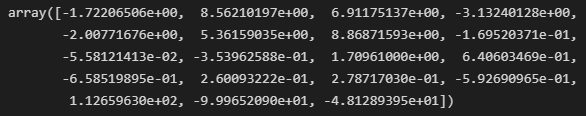
from sklearn.linear\_model import Lasso

lasso\_model = Lasso(alpha = 0.1).fit(X\_train, y\_train)

* Model işlemi hal edildi.

lasso\_model

* Bizim ƛ değerimiz 1 olarak girildi. Göstermiş olduk.

lasso\_model.coef\_

* 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)

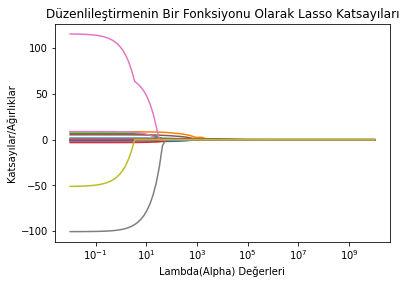
ax.set\_xscale('log')

plt.axis('tight')

plt.xlabel('Lambda(Alpha) Değerleri')

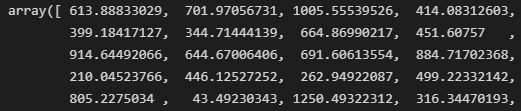
plt.ylabel('Katsayılar/Ağırlıklar')

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)

* 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.

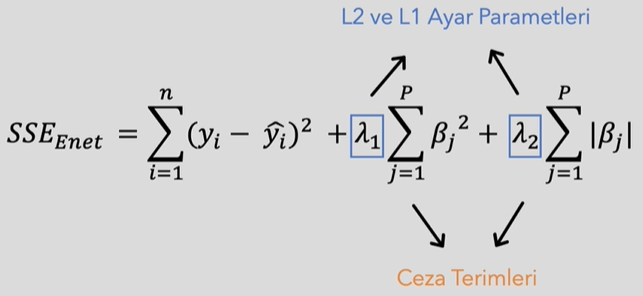
y\_pred = lasso\_tuned.predict(X\_test)

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,

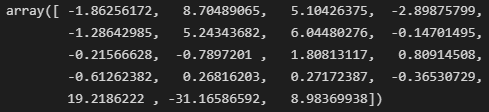
random\_state=42)

* Aynı veri seti ile test train işlemleri gerçekleştirildi.

from sklearn.linear\_model import ElasticNet

enet\_model = ElasticNet().fit(X\_train, y\_train)

* Model kuruldu.

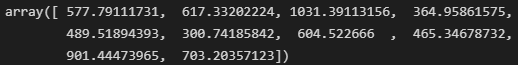
enet\_model.coef\_

* Kat sayılar yukarıda verilmiştir.

enet\_model.intercept\_

* Sabit katsayımız : -6.465955602112331

7.ElasticNet (ENET) Regresyon (Tahmin):

enet\_model.predict(X\_test)[0:10]

* 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.

