ML 08.07.2023

**Evaluation Model**

def eval\_metric(actual, pred):

mae = mean\_absolute\_error(actual, pred)

mse = mean\_squared\_error(actual, pred)

rmse = np.sqrt(mean\_squared\_error(actual, pred))

R2\_score = r2\_score(actual, pred)

print("Model testing performance: ")

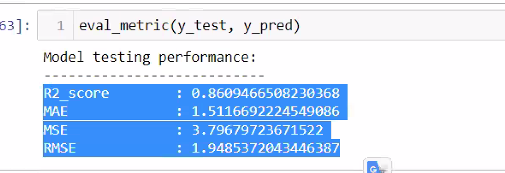
print("---------------------------")

print(f"R2\_score \t: {R2\_score}")

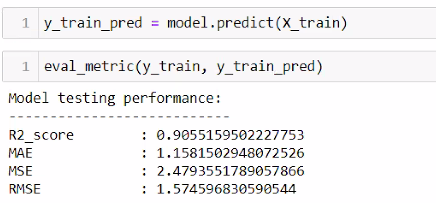
print(f"MAE \t\t: {mae}")

print(f"MSE \t\t: {mse}")

print(f"RMSE \t\t: {rmse}")



Egittigim datayla test datasini karsilastiracagim. Ortaya cikan skorlari yorumlayacagim

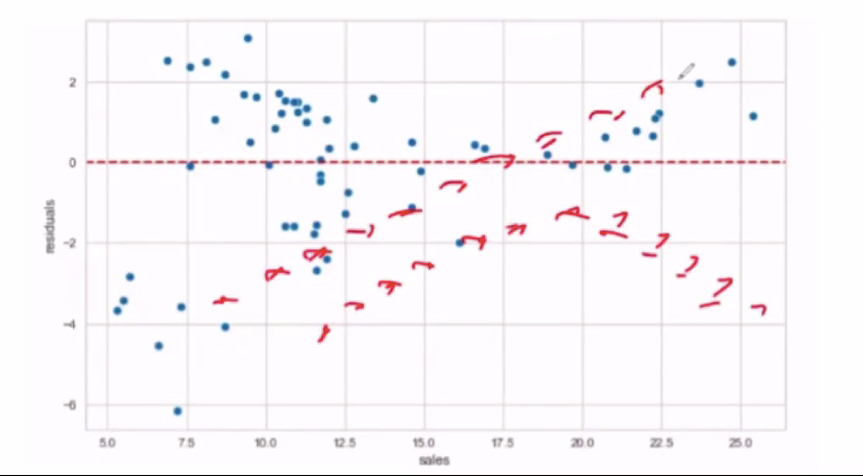


Train setinden aldigi degerle, test setinden aldigi skorlari kiyaslayacagim

Train’de iyi skorlar aliyor ama testte alamiyorsa bunda **overfitting** var demektir

**Is data suitable for linear regression?**

Bir modelin linear regresyona uygun olup olmadigini nasil anlariz?



Residual bir pattern izliyorsa, o zaman datam linear regresyona uygun degildir.

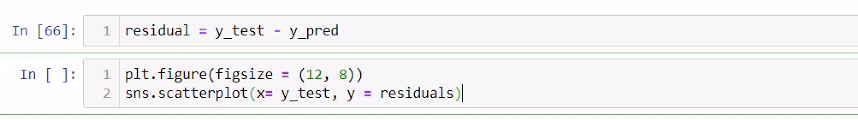
Residual’lar normal dagilmamali!

Scatterplotla buna bakmaliyiz.

Biz ise baslarken su model olsun diye baslamiyoruz.

Tum modelleri deniyoruz.

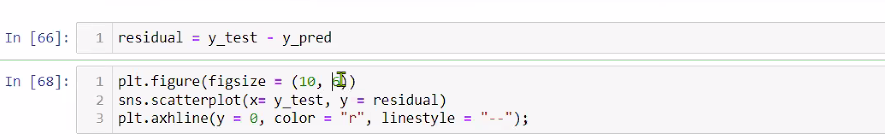
Modele baslarken target label’a gore, continous mu classification mi ona gore onlara uygun modelleri deniyoruz.

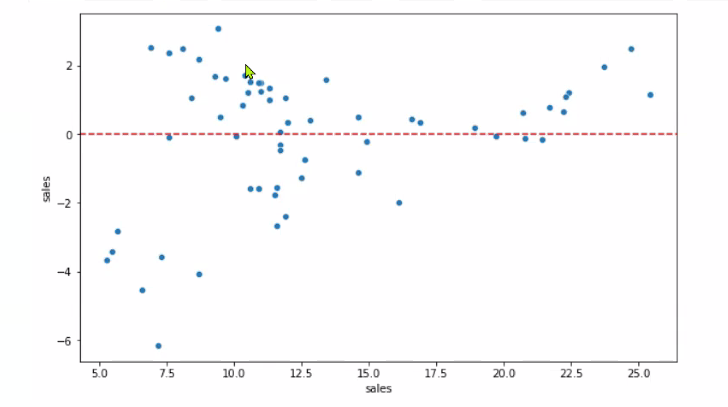


**plt.figure(figsize = (10, 6))**

**sns.scatterplot(x= y\_test, y = residual)**

**plt.axhline(y = 0, color = "r", linestyle = "--");**





ytest gercek degerlerim.

y-test ile y-pred arasindaki farkin 0 oldugu nokta. Yani skorumun mukemmel oldugu nokta.

Linear icin

Feature-targetin normal dagilim gostermesi gerek.

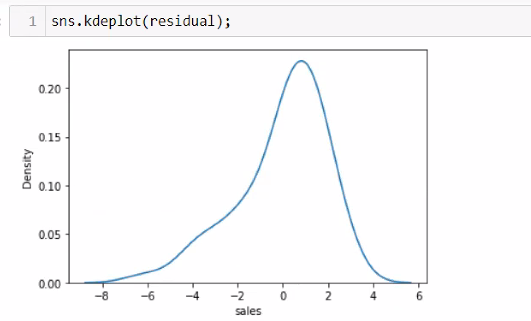
Bir gorus de bunun aksini savunuyor.

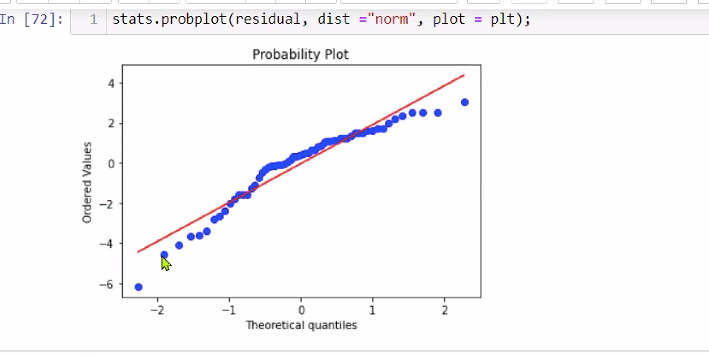
Linear modele karar vermek icin Makale:

<https://stats.stackexchange.com/questions/12262/what-if-residuals-are-normally-distributed-but-y-is-not>

Makalede diyor ki--> ikisi de saga carpiksa sonuc iyilesebilir diyor

|  |
| --- |
| **Fatih hocamdan:**  hata terimleri (residuallar) lineer regresyon modelinin uygunluğunu kontrol etmek için önemli bir özelliktir. Scatter plot üzerinde residualların dağılımını kontrol etmek, **modelin varsayımlarının doğru olup olmadığını değerlendirme**nin önemli bir yoludur. İşte residualların dağılımını değerlendirirken bakılması gereken bazı önemli noktalar:  Rastgele Dağılım: Residualların scatter plot üzerinde rastgele dağılması gerekmektedir. Bu durum, bağımsız değişkenler ve hata terimleri arasında herhangi bir ilişkinin olmadığını gösterir, yani modelin tüm bilgiyi yakaladığını ve hata terimlerinin tamamen rastgele olduğunu gösterir.  Sabit Varyans (Homoskedastisite): Residuallar, scatter plot üzerinde yatay bir çizgi etrafında eşit olarak dağılmalıdır. Bu, hata terimlerinin sabit bir varyansa sahip olduğunu, yani homoskedastik olduğunu gösterir. Eğer residuallar belirli bir modelin etrafında bir 'fan' şekli oluşturuyorsa, bu genellikle heteroskedastisite işaret eder ve bu durum modelin uygun olmadığını gösterir.  Normal Dağılım: Residualların normal bir dağılıma sahip olması beklenir. Bu, genellikle bir histogram veya bir QQ-plot ile kontrol edilir. Normal dağılımdan sapmalar, modelin verileri tam olarak yakalamadığını veya bazı aykırı değerlerin bulunduğunu gösterebilir.  Bu kontroller, modelin doğruluğunu ve güvenilirliğini kontrol etmek için önemlidir. Herhangi bir ihlal durumunda, veri önişleme tekniklerini gözden geçirmek, daha karmaşık bir model kullanmayı düşünmek veya belki de aykırı değerleri kontrol etmek gerekebilir. |





Cizgiye ne kadar yakin geciyorsa o kadar iyi diyoruz.

Bunlar residual’in degerlendirilmesi icin gorseller

Scatter ve kde’ye bakmak yeterli esas olarak.

**!pip install yellowbrick**

**from yellowbrick.regressor import ResidualsPlot**

Gorsellerle ML modelimizi yorumlayabilecegimiz bir kutuphane yellowbrick

[https://www.scikit-yb.org/en/latest/api/regressor/residuals.html](https://www.scikit-yb.org/en/latest/api/regressor/residuals.html" \t "/Users/merthan/Desktop/PREVENTING EXCHANGE RATE HIKES/x/_blank)

**model = LinearRegression()**

**visualizer = ResidualsPlot(model)**

**visualizer.fit(X\_train, y\_train) # Fit the training data to the visualizer**

**visualizer.score(X\_test, y\_test) # Evaluate the model on the test data**

**visualizer.show()**

**10:18**

**from yellowbrick.regressor import PredictionError**

**10:19**

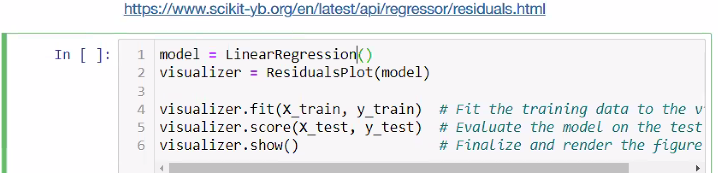
**model = LinearRegression()**

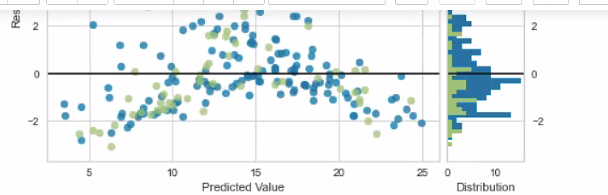
**visualizer = PredictionError(model)**

**visualizer.fit(X\_train, y\_train)**

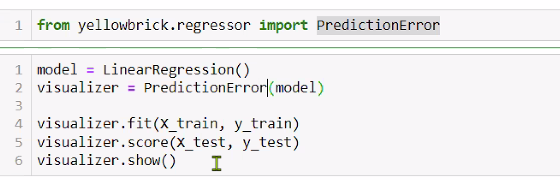
**visualizer.score(X\_test, y\_test)**

**visualizer.show()**

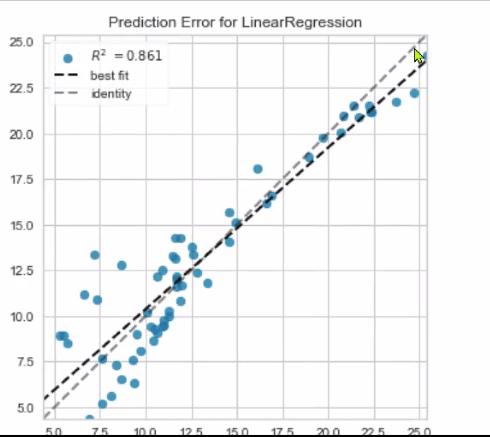




Sadece icine train ve test datalarini yazdim. Yellowbrick kutuphanesi ML gorsellestirmes



Bu da tahminlerin hatasini gosterdigimiz gorsel



Makas ne kadar darsa model o kadar basarili demektir.

Best fit line-->koyu cizgi

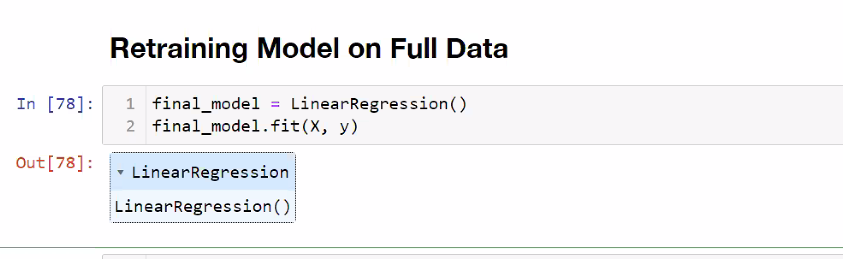
Silik cizgi de tahmin ile gercek degerlerin birbirine en yakin oldugu cizgi

En iyi skoru linear regresyon modelinden aldigimi varsayiyorum. Kendime sectim o regresyonu artik. Tum sorunlarla mucadele etmisim, scale etmisim falan.

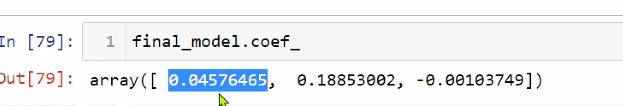
Linear regresyon ile devam edecegiz.

Tum datayi artik final model olarak linear regresyona sokacagiz.

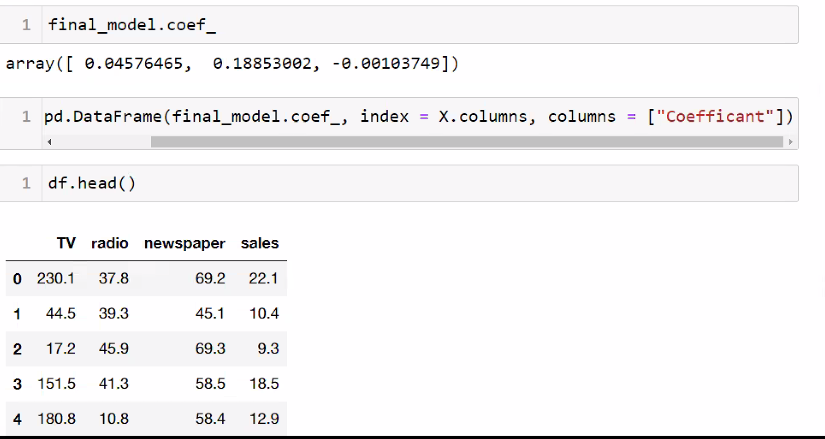
**Retraining Model on Full Data**

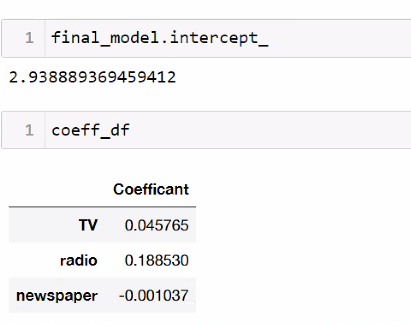


Artik final modelimiz linear regression. Tum datayi artik bu final modele sokuyorum.

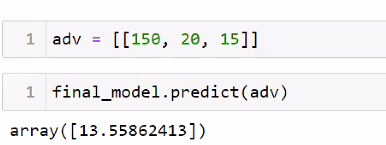


Sadece katsayilara bakarak benim icin radio onemli diyemiyorduk. Scale islemi yapmamiz gerek once.





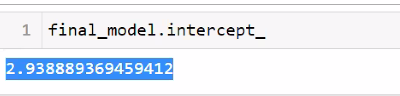
Prediction on new data



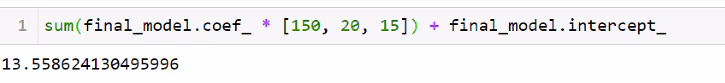
Tv’ye, radio’ya, newspaper’a yaptiginiz harcamayi

Arkasinda calistirdigi kod -->final\_model.predict(adv)

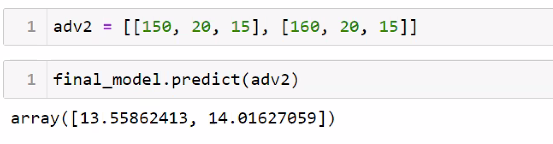
Sonucta 13.55’lik sales olarak bize veriyor.

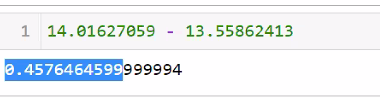


Bu da bize hic reklam harcamasi yapmazsak sales’im kac olarak onu veriyor.



10 birimlik bir artis yapiyorum TV icin





Tv’deki 10 birimlik artisin, TV’nin coefficient’i kadar sales’a etki ettigini goruyoruz. Ama scale etmedik daha ondan.

**Polynomial Regression**

**Overfitting-Underfitting**

Bagimli degiskenle bagimsiz degiskenin n. derece polynomla modellendigi bir regresyon analizi

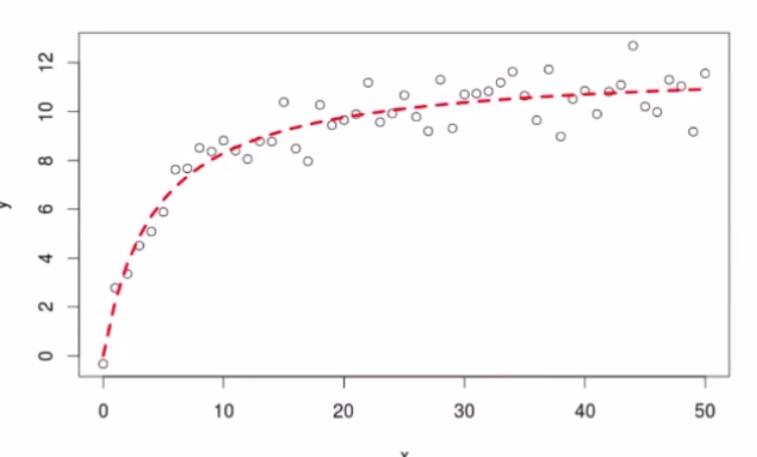
Polinom Regresyon , bağımsız değişkenler ile bağımlı değişkenler arasındaki ilişkinin n'inci derece polinomda modellendiği bir regresyon analizi şeklidir.

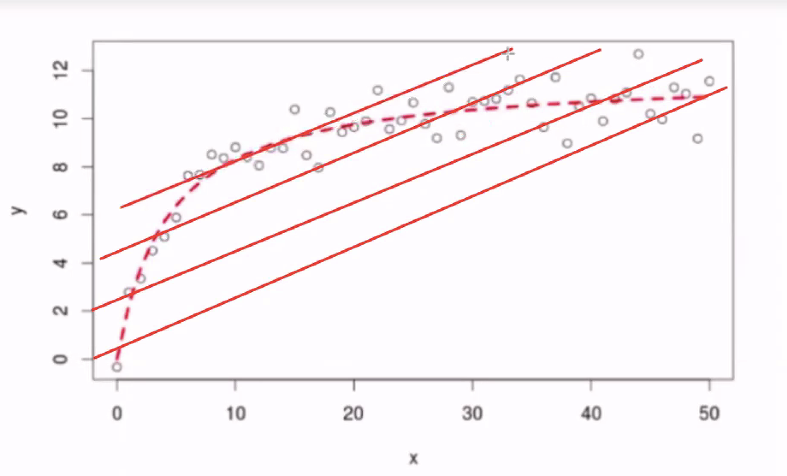
**Types of polinomials**

**1st degree ---> linear b1x + b0**

**2nd degree ---> Quadratic b2x\*\*2 + b1x + b0 2.derece**

**3rd degree ---> Cubic b3x\*\*3 + b2x\*\*2 + b1x + b0 3.derece**





Nerden cizerseniz cizgin dogrusal regresyonla dogru bir tahmin yapamazsiniz. Bu durumlarda polynomial regresyon kullanilir.

Derece arttikca kirilmalar da artiyor. Bir noktadan sonra da overfittinge gider.

Bir optimumu var. Derece arttikca once iyilesir ama sonra overfitting olur.

**Import Libraries**

import numpy as np

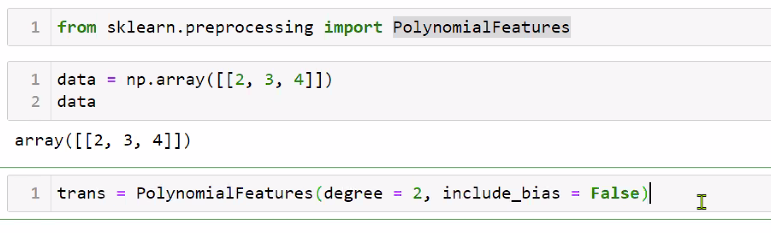
import pandas as pd

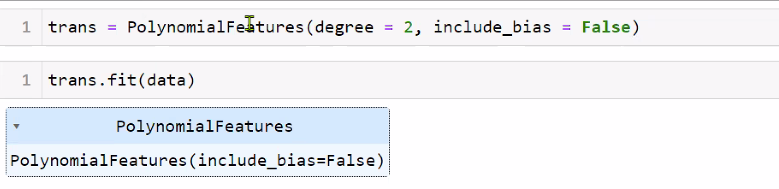
import matplotlib.pyplot as plt

import seaborn as sns

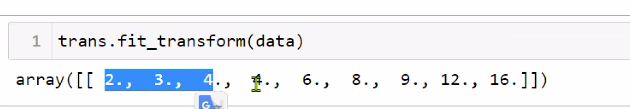
plt.rcParams["figure.figsize"] = (10,6)

from sklearn.preprocessing import PolynomialFeatures



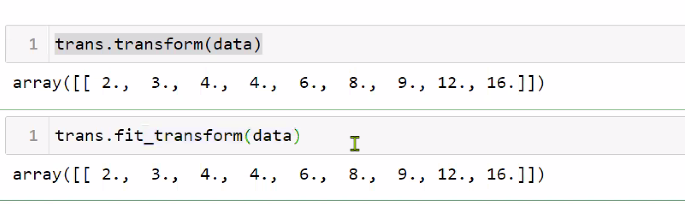


Donusturme emri icin transform emri verdim:



Hepsinin karelerini de alarak bir array uretti.

Her ikisini ayni anda kullanmak icin fit transform data kullandik



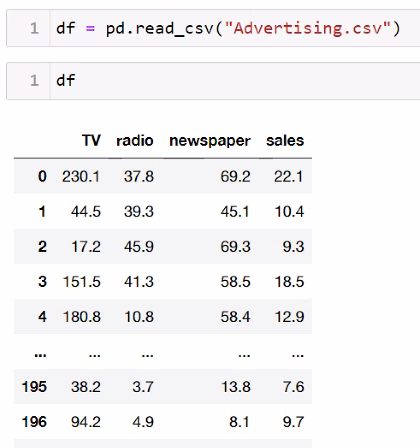
Traine ikisini ayni anda verebilecegiz

Test datasina sadece transform verebilecegiz.

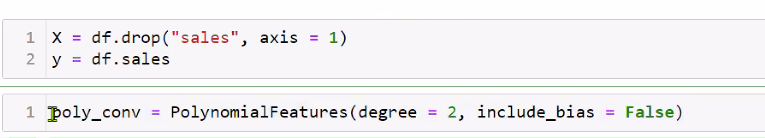
# degree 3 yapınca hem feature sayısı artıyor hemde işlemin yorumlanması zorlaşıyor işler karmaşıklaşıyor

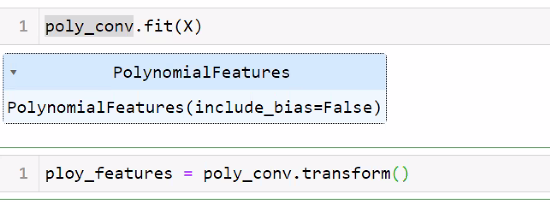
# 2, 3, 4, 2x3, 2x4, 3x4, 2\*\*2, 3\*\*2, 4\*\*2, 2x3x4, 3x2\*\*2, 4x2\*\*2, 2x3\*\*2, 4x3\*\*2, 2x4\*\*2, 3x4\*\*2, 2\*\*3, 3\*\*3, 4\*\*3

**Read Dataset**

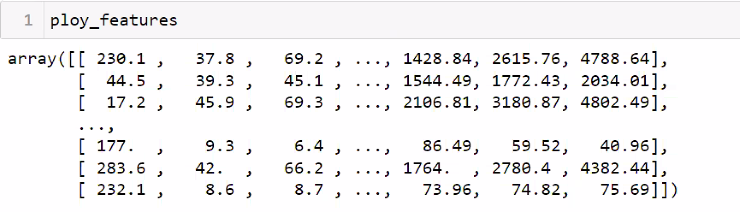


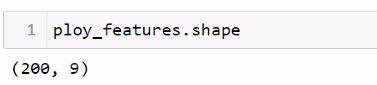
**Polynomial Converter**





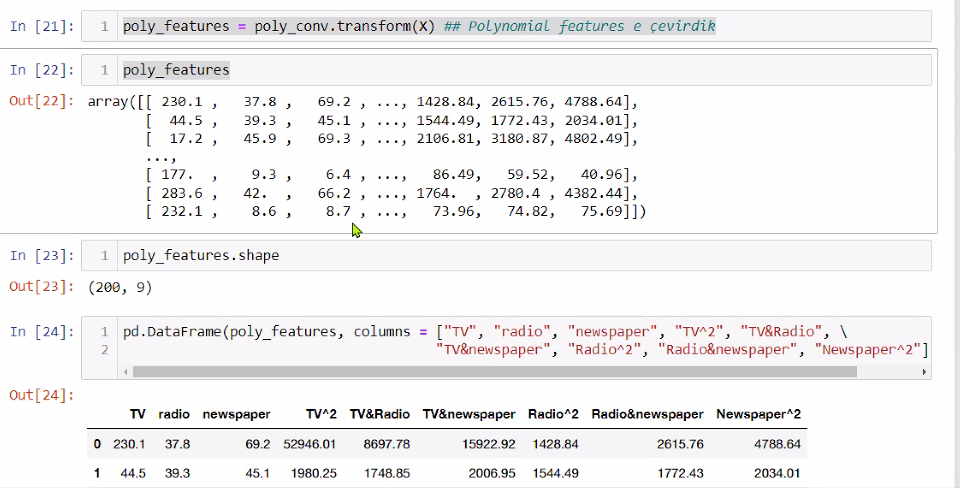
Polynomial features’a cevirdik





3 features’dan 9 features’a cikti

pd.DataFrame(ploy\_features, columns = ["TV", "radio", "newspaper", "TV^2", "TV&Radio",\"TV&newspaper", "Radio^2", "Radio&newspaper", "Newspaper^2"]).head()



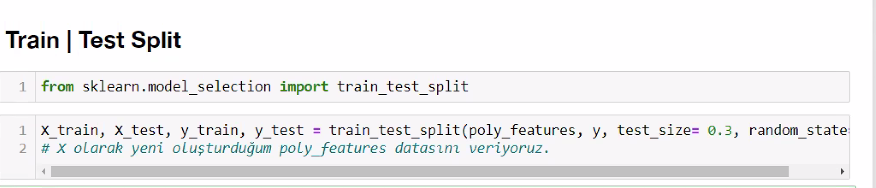
Datasetimi artik 9 feature haline getirdim.

**Train Test Split**

**from sklearn.model\_selection import train\_test\_split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(poly\_features, y, test\_size= 0.3, random\_state= 101)**

**# X olarak yeni oluşturduğum poly\_features datasını veriyoruz.**



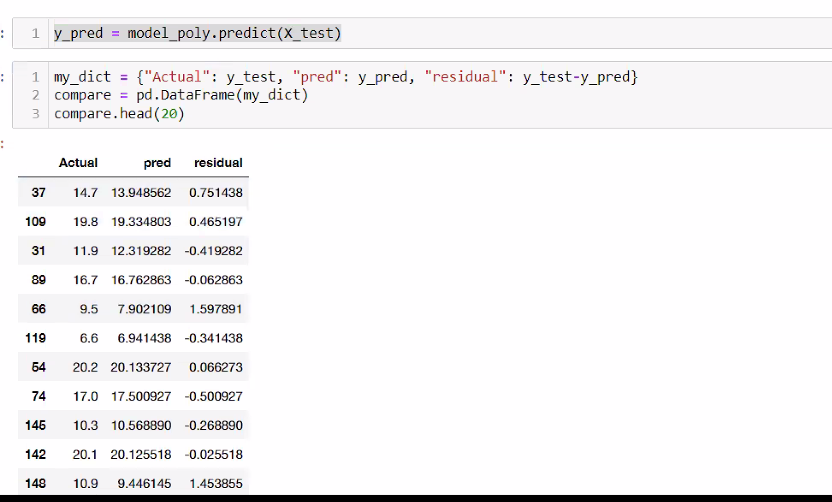
**X\_train, X\_test, y\_train, y\_test = train\_test\_split(poly\_features, y, test\_size= 0.3, random\_state= 101)**

**# X olarak yeni oluşturduğum poly\_features datasını veriyoruz.**

**model\_poly = LinearRegression()**

**model\_poly.fit(X\_train, y\_train)**

**y\_pred = model\_poly.predict(X\_test)**



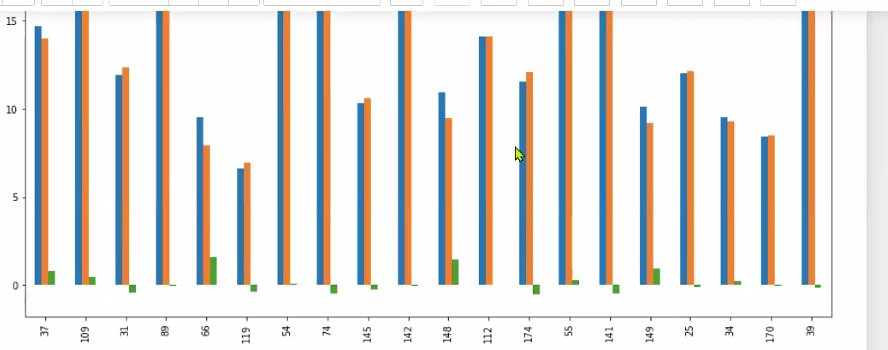
**my\_dict = {"Actual": y\_test, "pred": y\_pred, "residual": y\_test-y\_pred}**

**compare = pd.DataFrame(my\_dict)**

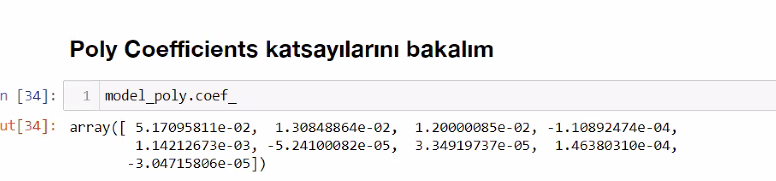
**compare.head(20)**

**compare.head(20).plot(kind="bar", figsize=(15,9))**

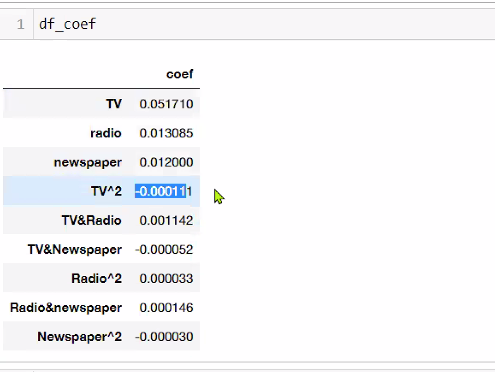
**plt.show();**



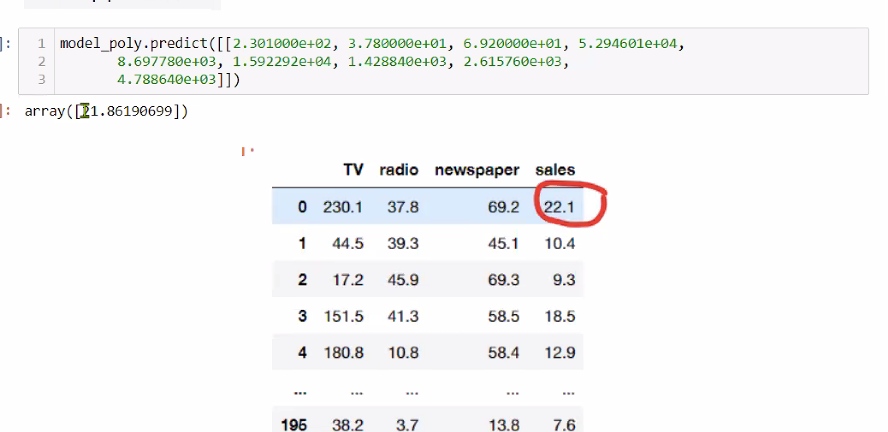
**Poly Coefficients katsayılarını bakalım**



**df\_coef = pd.DataFrame(model\_poly.coef\_, index = ["TV", "radio", "newspaper", "TV^2", "TV&Radio", \"TV&Newspaper", "Radio^2", "Radio&newspaper", "Newspaper^2"], columns = ["coef"])**



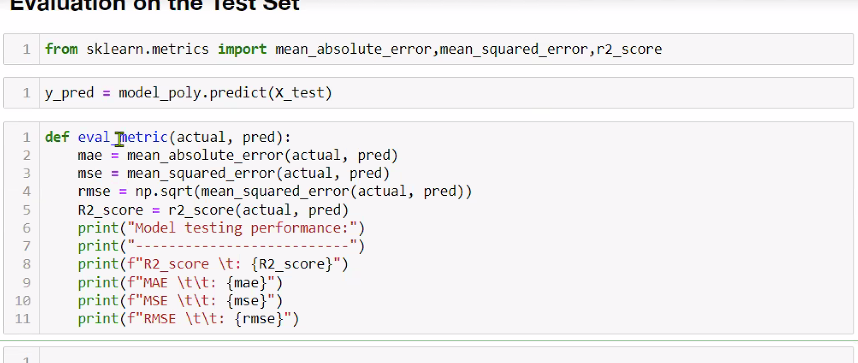
Tv’nin karesi radyodan daha az gorunuyor ama scale asamasinda duzelecek.

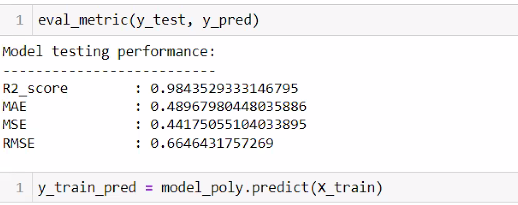


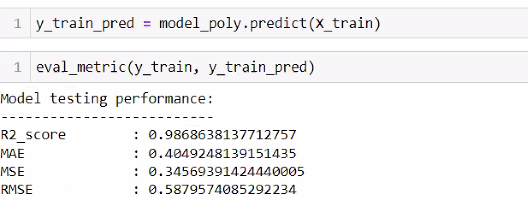
Cok yakin iyi bir skor aldim.

Evaluation test setine bakiyorum.

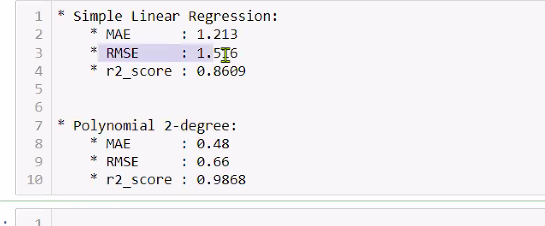
**Evaluation on the Test Set**







Modelim 2. Derecede en optimal degeri verdi. Linear regresyona gore cok daha iyi hata metriclerim. Dolayisiyla polynomial’i tercih ederim.



(X = df.drop(columns = "sales") columns parametresiyle sütun ismi verildiği için axis e gerek kalmıyor

X = df.drop("sales", axis = 1) burda ise belirtmemiz gerekiyor axisi)