↗️

# REGRESSION

▼ IMPORT

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_absolute_error, mean_squared_error
from joblib import dump, load
import scipy as sp
df = pd.read_csv("Advertising.csv")
```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_absolute_error, mean_squared_error
from joblib import dump, load
import scipy as sp
```
✓ 2.2s

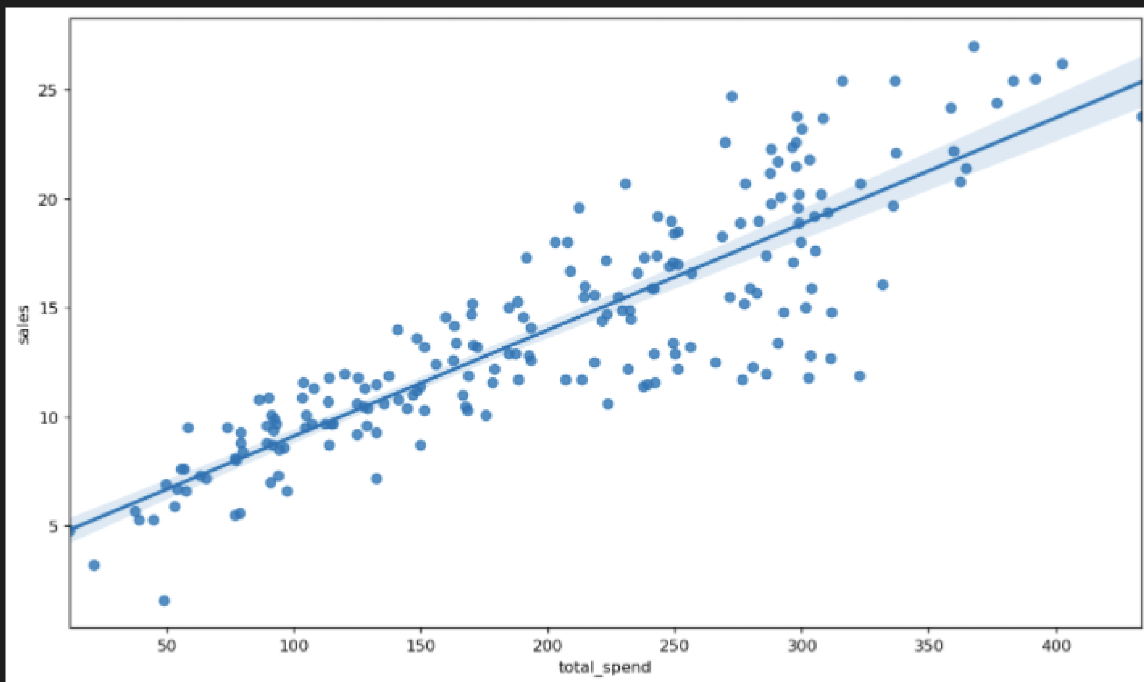
```python
df = pd.read_csv("Advertising.csv")
```
✓ 0.6s

▼ Simple Linear Regr

- sns.regplot(data=df, x="total_spend", y="sales") : regresyon plotu çizer

```python
plt.figure(figsize=(12,7), dpi=120)
sns.regplot(data=df, x="total_spend", y="sales")
```

```
<AxesSubplot:xlabel='total_spend', ylabel='sales'>
```




- Seaborn kodu ile elle hesaplanan poly fit komutunun karşılaştırılması

```python
x = df["total_spend"]
y = df["sales"]
```

✓ 0.1s

```python
np.polyfit(x,y,deg=1)
```

✓ 0.5s

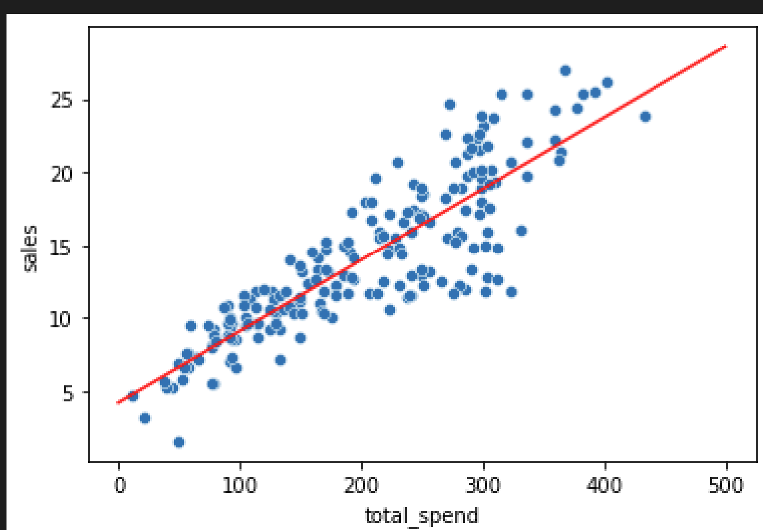
```
array([0.04868788, 4.24302822])
```

```python
potential_spend=np.linspace(0,500,100)
predicted_sales = 0.04868788*potential_spend + 4.24302822

sns.scatterplot(x="total_spend", y="sales", data=df)
plt.plot(potential_spend,predicted_sales, color="red")
```

✓ 0.4s

```
[<matplotlib.lines.Line2D at 0x1770cfabf10>]
```




- Scikit-Learn
  - İmport

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

- veri setini düzenleme. sonuçların gideceği yeni bir y parametersi oluşturuldu

```
x= df.drop("sales", axis=1)
y= df["sales"]
print(x)
print(y)
```

✓ 0.4s

```
Output exceeds the size limit. Open the full outp
editor
        TV  radio  newspaper
0    230.1   37.8       69.2
1     44.5   39.3       45.1
2     17.2   45.9       69.3
3    151.5   41.3       58.5
4    180.8   10.8       58.4
..     ...    ...        ...
195   38.2    3.7       13.8
196   94.2    4.9        8.1
197  177.0    9.3        6.4
198  283.6   42.0       66.2
199  232.1    8.6        8.7

[200 rows x 3 columns]
0      22.1
...
197    12.8
198    25.5
199    13.4
Name: sales, Length: 200, dtype: float64
```

- test ve train model sayıları
  test_size=0.3 : veriin %30u test için ayrıldı

random_state=101 : rastgele başlangıç değeri

```
x_train, x_test, y_train, y_test = train_test_split(x,
y, random_state=101, test_size=0.3)
```

✓ 0.5s

```
print(len(df))
print(len(x_train))
print(len(x_test))
```

✓ 0.1s

```
200
140
60
```

- model= LinearRegression() : model oluşturma
  model.fit(x_train, y_train) : modeli yerleştirme
  model.predict(x_test) : öngörülen x değerleri

```python
model= LinearRegression()
```

0.5s Python

```python
model.fit(x_train, y_train)
```

0.8s Python

```
LinearRegression()
```

```python
model.predict(x_test)
```

0.8s Python

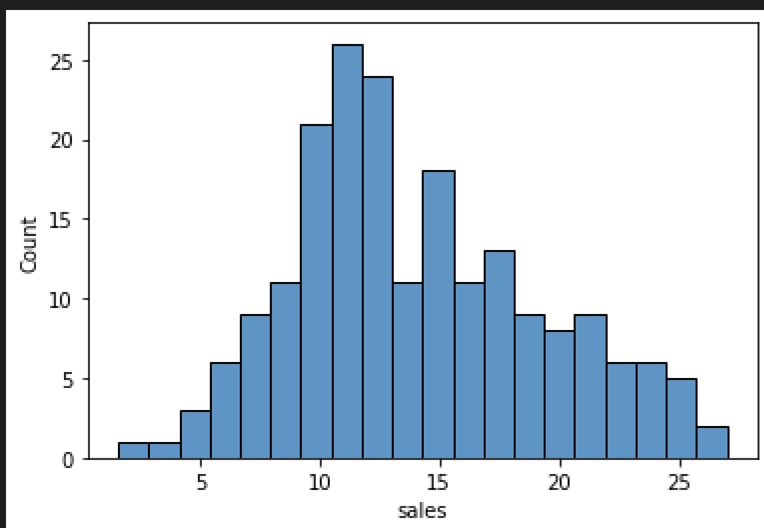
```
array([15.74131332, 19.61062568, 11.44888935, 17.00819787,  9.17285676,
        7.01248287, 20.28992463, 17.29953992,  9.77584467, 19.22194224,
       12.40503154, 13.89234998, 13.72541098, 21.28794031, 18.42456638,
        9.98198406, 15.55228966,  7.68913693,  7.55614992, 20.40311209,
        7.79215204, 18.24214098, 24.68631904, 22.82199068,  7.97962085,
       12.65207264, 21.46925937,  8.05228573, 12.42315981, 12.50719678,
       10.77757812, 19.24460093, 10.070269  ,  6.70779999, 17.31492147,
        7.76764327,  9.25393336,  8.27834697, 10.58105585, 10.63591128,
       13.01002595,  9.77192057, 10.21469861,  8.04572042, 11.5671075 ,
       10.08368001,  8.99806574, 16.25388914, 13.23942315, 20.81493419,
       12.49727439, 13.96615898, 17.56285075, 11.14537013, 12.56261468,
        5.50870279, 23.29465134, 12.62409688, 18.77399978,
15.18785675])
```

- print(df["sales"].mean()) : ortalama mean değeri ile MSE ve RMSE kıyaslanmalı ki hata paylarının ne kadar yüksek olduğu ile alakalı bir sonuç elde edilebilsin. %10 civarı çıktı. fena değil.
  MSE toplam noktaların ne kadar uzakta olduğunu
  RMSE dataların içindeki alakasız, ayrık noktaların ne kadar çok olduğunu gösterir

```python
test_predictions = model.predict(x_test)
```
0.3s

```python
print(df["sales"].mean())
sns.histplot(data=df, x="sales", bins=20);
```
0.3s

```
14.022500000000003
```




```python
mean_absolute_error(y_test, test_predictions)
# Mean value ile kıyaslanmalı
```
0.4s

```
1.2137457736144808
```

```python
np.sqrt(mean_squared_error(y_test, test_predictions))
#root mean square error
```
0.4s

```
1.5161519375993877
```

- Residuals and residual plot

```
test_residuals = y_test - test_predictions
test_residuals.head()
```

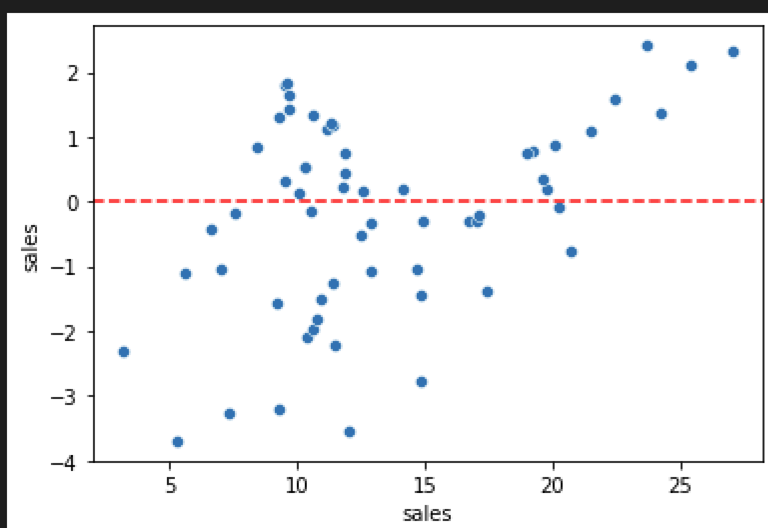
✓ 0.1s

```
37    -1.041313
109    0.189374
31     0.451111
89    -0.308198
66     0.327143
Name: sales, dtype: float64
```

```
sns.scatterplot(x=y_test, y=test_residuals)
plt.axhline(y= 0, color="red", ls="--")
```

✓ 0.2s

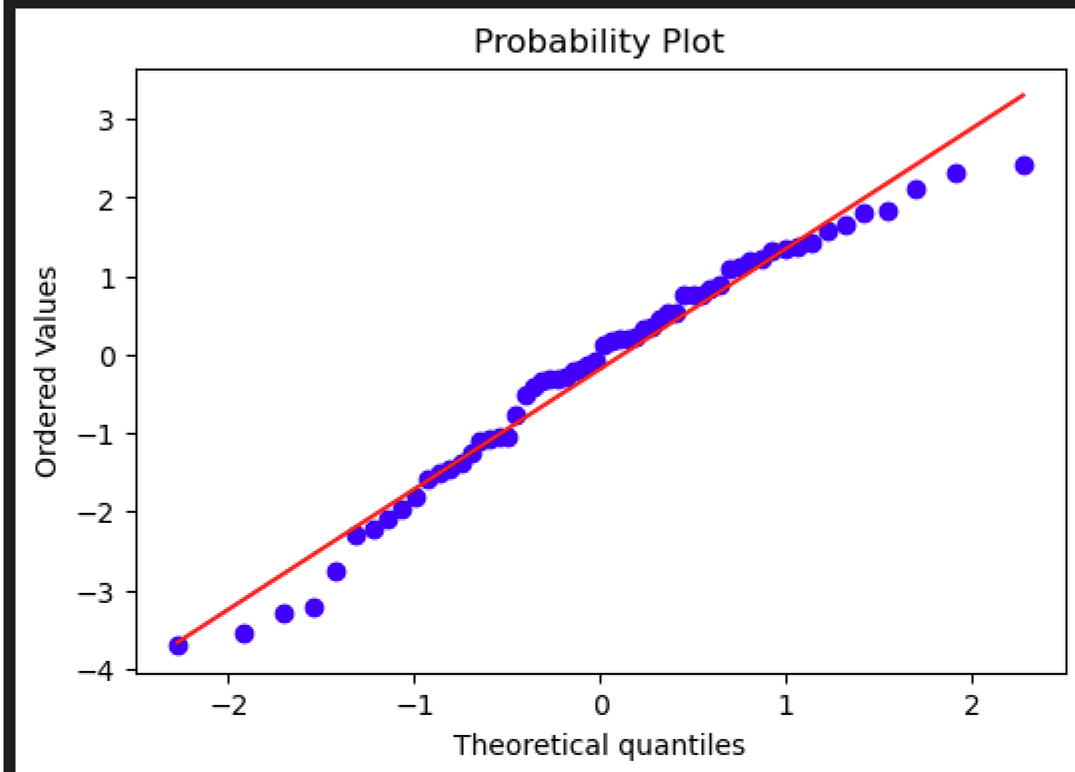
```
<matplotlib.lines.Line2D at 0x1909c787220>
```




- residual plot

```python
# Create a figure and axis to plot on
fig, ax = plt.subplots(figsize=(6,4),dpi=100)
# probplot returns the raw values if needed
# we just want to see the plot, so we assign these values
_ = sp.stats.probplot(test_residuals,plot=ax)
```

✓ 0.3s




- Coef

  1 birim TV artışı 0.045 birim satış artışı

  1 birim RADYO artışı 0.188birim satış artışı

  1 birim GAZETE artışı 0 birim satış artışı

  Gazetenin satışa etkisi yok

```
final_model = LinearRegression()
final_model.fit(x,y)
print(final_model.coef_)
```

```
[ 0.04576465  0.18853002 -0.00103749]
```

```
x.head()
```

| | TV | radio | newspaper |
|---|---|---|---|
| 0 | 230.1 | 37.8 | 69.2 |
| 1 | 44.5 | 39.3 | 45.1 |
| 2 | 17.2 | 45.9 | 69.3 |
| 3 | 151.5 | 41.3 | 58.5 |
| 4 | 180.8 | 10.8 | 58.4 |

- dump(final_model, 'sales_model.joblib') : sonuçları kaydeder.
  loaded_model = load('sales_model.joblib') : sonuçları çeker ve gösterilmesine imkan verir

```python
dump(final_model, 'sales_model.joblib')
```
✓ 0.1s

```
['sales_model.joblib']
```

```python
loaded_model = load('sales_model.joblib')
```
✓ 0.9s

```python
loaded_model.coef_
```
✓ 0.8s

```
array([ 0.04576465,  0.18853002, -0.00103749])
```

- campaign = [[149,22,12]] : 149 k TV, 22 k radyo, 12 k gazete rekamı verildiğinde loaded_model.predict(campaign) : sonuçta kaç birimlik satış elde edilir

```python
campaign = [[149,22,12]]
```
✓ 0.8s

```python
loaded_model.predict(campaign)
```
✓ 0.1s

```
array([13.893032])
```

▼ Polynomial Regression

- Dataset düzenleme

```python
x = df.drop("sales", axis=1)
y = df["sales"]
print(x.head())
print(y.head())
```

✓ 0.2s

```
      TV  radio  newspaper
0  230.1   37.8       69.2
1   44.5   39.3       45.1
2   17.2   45.9       69.3
3  151.5   41.3       58.5
4  180.8   10.8       58.4
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: sales, dtype: float64
```

- polynom özellikleri ekleme

```python
polynomial_converter = PolynomialFeatures(degree= 2, include_bias= False)
```

```python
polynomial_converter.fit(x)
```

```
PolynomialFeatures(include_bias=False)
```

```python
poly_features = polynomial_converter.fit_transform(x)
```

```python
print(polynomial_converter.transform(x).shape)
print(x.shape)
```

```
(200, 9)
(200, 3)
```

- poly değerler. 1 2 3 dataframeden gelen. 4 5 6 , ilk 3Ün karesi. 7 8 9 birbirleriyle çarpımı

```python
print(x.iloc[0])
print(poly_features[0])
```

```
TV           230.1
radio         37.8
newspaper     69.2
Name: 0, dtype: float64
[2.301000e+02 3.780000e+01 6.920000e+01 5.294601e+04 8.697780e+03
 1.592292e+04 1.428840e+03 2.615760e+03 4.788640e+03]
```

- polynomial sonuçlar, 2. dereceden

```python
x_train, x_test, y_train, y_test = train_test_split(poly_features,
    y, test_size=0.3, random_state=101)
```

✓ 0.1s

```python
model = LinearRegression()
model.fit(x_train, y_train)
```

✓ 0.2s

```
LinearRegression()
```

```python
test_predictions = model.predict(x_test)
```

✓ 0.2s

```python
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
```

✓ 0.5s

```python
print(f"MAE = {round(MAE,3)}")
print(f"MSE = {round(MSE,3)}")
print(f"RMSE = {round(RMSE,3)}")
```

✓ 0.1s

```
MAE = 0.49
MSE = 0.442
RMSE = 0.665
```

▼ Model Seçimi

- kaçıncı dereceden model seçileceğini belirlemek için min. error rate veren bir derece seçilmeli

```python
# TRAINING ERROR PER DEGREE
train_rmse_errors = []
# TEST ERROR PER DEGREE
test_rmse_errors = []

for d in range(1,10):

    # CREATE POLY DATA SET FOR DEGREE "d"
    polynomial_converter = PolynomialFeatures(degree=d,include_bias=False)
    poly_features = polynomial_converter.fit_transform(x)

    # SPLIT THIS NEW POLY DATA SET
    x_train, x_test, y_train, y_test = train_test_split(poly_features,
     y, test_size=0.3, random_state=101)

    # TRAIN ON THIS NEW POLY SET
    model = LinearRegression(fit_intercept=True)
    model.fit(x_train,y_train)

    # PREDICT ON BOTH TRAIN AND TEST
    train_pred = model.predict(x_train)
    test_pred = model.predict(x_test)

    # Calculate Errors

    # Errors on Train Set
    train_RMSE = np.sqrt(mean_squared_error(y_train,train_pred))

    # Errors on Test Set
    test_RMSE = np.sqrt(mean_squared_error(y_test,test_pred))

    # Append errors to lists for plotting later

    train_rmse_errors.append(train_RMSE)
    test_rmse_errors.append(test_RMSE)
```
✓ 0.1s Python

- 1-9 dereceden polinomların sonuçları

```
train_rmse_errors
```

```
[1.734594124329376,
 0.5879574085292233,
 0.4339344356902067,
 0.35170836883993534,
 0.2509342952029336,
 0.19933332834273104,
 5.4214215994181805,
 0.14237972100695595,
 0.16675080548552418]
```

```
test_rmse_errors
```

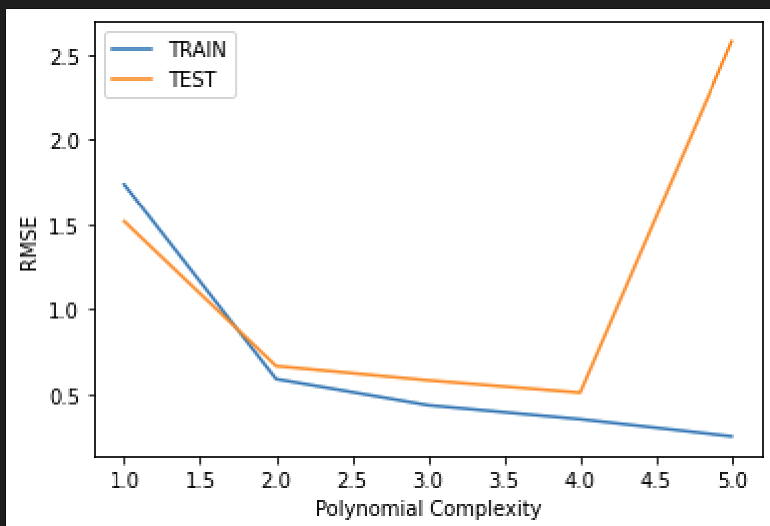
```
[1.5161519375993873,
 0.6646431757269196,
 0.5803286825231453,
 0.5077742624232109,
 2.5758247603435955,
 4.490868529265006,
 1381.404235838588,
 4449.5681972303655,
 95893.0265813161]
```

- NOT : Kırılmanın olduğu noktanın bir öncesinden alırsan daha iyi olur. 4. derecede nerede overfit yaptığını bilemeyebilirsin.
  (Bu örnek için)

```python
plt.plot(range(1,6),train_rmse_errors[:5],label='TRAIN')
plt.plot(range(1,6),test_rmse_errors[:5],label='TEST')
plt.xlabel("Polynomial Complexity")
plt.ylabel("RMSE")
plt.legend()
```

✓ 0.3s

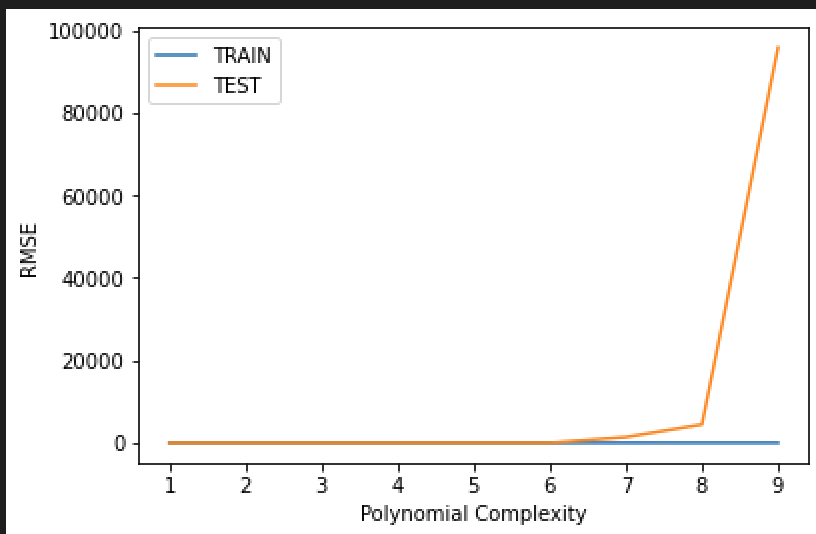
```
<matplotlib.legend.Legend at 0x208a461d730>
```



```python
plt.plot(range(1,10),train_rmse_errors,label='TRAIN')
plt.plot(range(1,10),test_rmse_errors,label='TEST')
plt.xlabel("Polynomial Complexity")
plt.ylabel("RMSE")
plt.legend()
```

✓ 0.3s

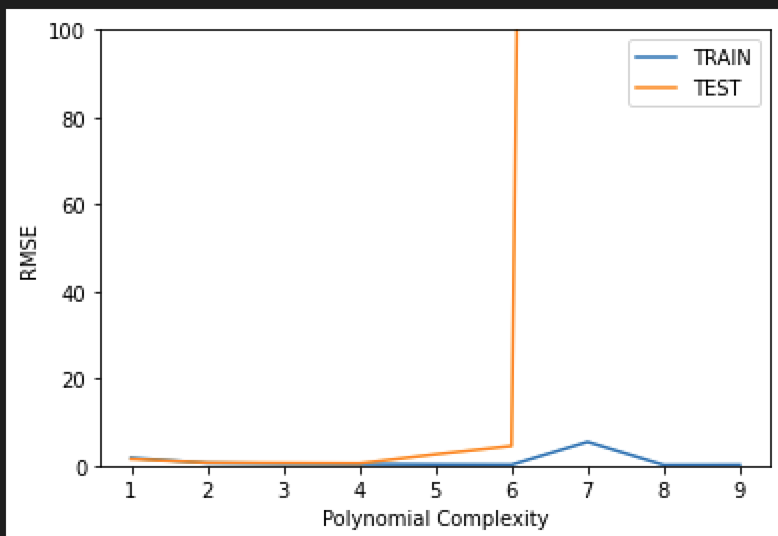
```
<matplotlib.legend.Legend at 0x208a46a4940>
```



```
plt.plot(range(1,10),train_rmse_errors,label='TRAIN')
plt.plot(range(1,10),test_rmse_errors,label='TEST')
plt.xlabel("Polynomial Complexity")
plt.ylabel("RMSE")
plt.ylim(0,100)
plt.legend()
```

✓ 0.1s

```
<matplotlib.legend.Legend at 0x208a466cc70>
```




▼ Regularizations

- regularization

```python
df = pd.read_csv("Advertising.csv")
X = df.drop("sales", axis=1)
y = df["sales"]
print(X)
print(y)
```

✓ 0.6s

```
Output exceeds the size limit. Open the full outp
        TV  radio  newspaper
0    230.1   37.8       69.2
1     44.5   39.3       45.1
2     17.2   45.9       69.3
3    151.5   41.3       58.5
4    180.8   10.8       58.4
..     ...    ...        ...
195   38.2    3.7       13.8
196   94.2    4.9        8.1
197  177.0    9.3        6.4
198  283.6   42.0       66.2
199  232.1    8.6        8.7

[200 rows x 3 columns]
0      22.1
...
197    12.8
198    25.5
199    13.4
Name: sales, Length: 200, dtype: float64
```

```python
polynomial_converter = PolynomialFeatures(degree=3, include_bias=False)
```
✓ 0.7s

```python
poly_features = polynomial_converter.fit_transform(X)
```
✓ 0.5s

```python
X_train, X_test, y_train, y_test = train_test_split(poly_features,
 y, test_size=0.3, random_state=101)
```
✓ 0.7s

```python
scaler = StandardScaler()
scaler.fit(X_train)
```
✓ 0.1s

```
StandardScaler()
```

```python
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```
✓ 0.3s

- Ridge regression

```python
from sklearn.linear_model import Ridge
```
0.6s

```python
ridge_model = Ridge(alpha=10)
ridge_model.fit(X_train,y_train)
```
0.3s

```
Ridge(alpha=10)
```

```python
test_predictions = ridge_model.predict(X_test)
```
0.1s

```python
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
print(f"MAE = {round(MAE,3)}")
print(f"MSE = {round(MSE,3)}")
print(f"RMSE = {round(RMSE,3)}")
```
0.7s

```
MAE = 0.577
MSE = 0.8
RMSE = 0.895
```

- Ridge Cross Validation

```
from sklearn.linear_model import RidgeCV
```
✓ 0.5s

```
ridge_cv_model = RidgeCV(alphas = (0.1, 1, 10), scoring="neg_mean_absolute_error")
```
✓ 0.1s

```
ridge_cv_model.fit(X_train, y_train)
```
✓ 0.7s

```
RidgeCV(alphas=array([ 0.1,  1. , 10. ]), scoring='neg_mean_absolute_error')
```

```
from sklearn.metrics import SCORERS
```
✓ 0.5s

```
test_predictions = ridge_cv_model.predict(X_test)
```
✓ 0.4s

```
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
print(f"MAE = {round(MAE,3)}")
print(f"MSE = {round(MSE,3)}")
print(f"RMSE = {round(RMSE,3)}")
```
✓ 0.1s

```
MAE = 0.427
MSE = 0.382
RMSE = 0.618
```

- Lasso Regression

```python
from sklearn.linear_model import LassoCV
```
✓ 0.7s

```python
Lasso_cv_model = LassoCV(eps=0.1, n_alphas=100, cv=5)
```
✓ 0.9s

```python
Lasso_cv_model.fit(X_train, y_train)
```
✓ 0.3s

```
LassoCV(cv=5, eps=0.1)
```

```python
test_predictions = Lasso_cv_model.predict(X_test)
```
✓ 0.9s

```python
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
print(f"MAE = {round(MAE,3)}")
print(f"MSE = {round(MSE,3)}")
print(f"RMSE = {round(RMSE,3)}")
```
✓ 0.7s

```
MAE = 0.654
MSE = 1.279
RMSE = 1.131
```

- Elastic Net

```python
from sklearn.linear_model import ElasticNetCV
```
✓ 0.5s

```python
elastic_model = ElasticNetCV(l1_ratio=[.1, .5, .7,.9, .95, .99, 1],tol=0.01)
```
✓ 0.8s

```python
elastic_model.fit(X_train,y_train)
```
✓ 0.6s

```
ElasticNetCV(l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1], tol=0.01)
```

```python
test_predictions = elastic_model.predict(X_test)
```
✓ 0.2s

```python
MAE = mean_absolute_error(y_test,test_predictions)
MSE = mean_squared_error(y_test,test_predictions)
RMSE = np.sqrt(MSE)
print(f"MAE = {round(MAE,3)}")
print(f"MSE = {round(MSE,3)}")
print(f"RMSE = {round(RMSE,3)}")
```
✓ 0.2s

```
MAE = 0.566
MSE = 0.56
RMSE = 0.749
```

```python
train_predictions = elastic_model.predict(X_train)
MAE = mean_absolute_error(y_train,train_predictions)
MAE
```
✓ 0.1s

```
0.4307582990472369
```

- 

▼ Appendix

- Transform

```
eighborhood")["Lot Frontage"].transform(lambda value: value.fillna(value.mean()))

✓ 0.7s                                                                        Python
```

- .apply(str8 : Stringe çevirme

```
df["MS SubClass"] = df["MS SubClass"].apply(str)
✓ 0.2s
```

- pd.get_dummies(direction) : string veriyi dummy değişkene çevirir

```
direction = pd.Series(["up","up","down","down"])
direction
✓ 0.1s

0      up
1      up
2    down
3    down
dtype: object
```

```
pd.get_dummies(direction)
✓ 0.5s
```

| | down | up |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 0 |

- pd.get_dummies(direction, drop_first=True) : İlk kolonu atar

```python
pd.get_dummies(direction, drop_first=True)
```

0.7s

| | up |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 2 | 0 |
| 3 | 0 |

- df.select_dtypes(include="object") : Türü obje olan verileri getirir

```python
df.select_dtypes(include="object")
```

0.6s

| | MS SubClass | MS Zoning | Street | Lot Shape | Land Contour | Utilities | Lot Config | Land Slope |
|---|---|---|---|---|---|---|---|---|
| 0 | 20 | RL | Pave | IR1 | Lvl | AllPub | Corner | Gtl |
| 1 | 20 | RH | Pave | Reg | Lvl | AllPub | Inside | Gtl |
| 2 | 20 | RL | Pave | IR1 | Lvl | AllPub | Corner | Gtl |
| 3 | 20 | RL | Pave | Reg | Lvl | AllPub | Corner | Gtl |
| 4 | 60 | RL | Pave | IR1 | Lvl | AllPub | Inside | Gtl |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2925 | 80 | RL | Pave | IR1 | Lvl | AllPub | CulDSac | Gtl |
| 2926 | 20 | RL | Pave | IR1 | Low | AllPub | Inside | Mod |
| 2927 | 85 | RL | Pave | Reg | Lvl | AllPub | Inside | Gtl |
| 2928 | 20 | RL | Pave | Reg | Lvl | AllPub | Inside | Mod |
| 2929 | 60 | RL | Pave | Reg | Lvl | AllPub | Inside | Mod |

- dummy variable oluşturur

```python
my_object_df = df.select_dtypes(include="object")
```

✓ 0.1s

```python
my_numeric_df = df.select_dtypes(exclude="object")
```

✓ 0.8s

```python
df_object_dummies = pd.get_dummies(my_object_df, drop_first=True)
df_object_dummies
```

✓ 0.2s

| | MS SubClass_150 | MS SubClass_160 | MS SubClass_180 | MS SubClass_190 | MS SubClass_20 | Su |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 0 | 0 | 1 | |
| 3 | 0 | 0 | 0 | 0 | 1 | |
| 4 | 0 | 0 | 0 | 0 | 0 | |

- pd.concat : data frameleri birleştirir

```python
final_df = pd.concat([my_numeric_df, df_object_dummies], axis=1)
final_df
```

✓ 0.1s

| | Lot Frontage | Lot Area | Overall Qual | Overall Cond | Year Built | Year Remod/Add | Mas Vnr Area | BsmtFin SF 1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 141.000000 | 31770 | 6 | 5 | 1960 | 1960 | 112.0 | 639.0 |
| 1 | 80.000000 | 11622 | 5 | 6 | 1961 | 1961 | 0.0 | 468.0 |
| 2 | 81.000000 | 14267 | 6 | 6 | 1958 | 1958 | 108.0 | 923.0 |
| 3 | 93.000000 | 11160 | 7 | 5 | 1968 | 1968 | 0.0 | 1065.0 |
| 4 | 74.000000 | 13830 | 5 | 5 | 1997 | 1998 | 0.0 | 791.0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2925 | 37.000000 | 7937 | 6 | 6 | 1984 | 1984 | 0.0 | 819.0 |
| 2926 | 75.144444 | 8885 | 5 | 5 | 1983 | 1983 | 0.0 | 301.0 |
| 2927 | 62.000000 | 10441 | 5 | 5 | 1992 | 1992 | 0.0 | 337.0 |
| 2928 | 77.000000 | 10010 | 5 | 5 | 1974 | 1975 | 0.0 | 1071.0 |
| 2929 | 74.000000 | 9627 | 7 | 5 | 1993 | 1994 | 94.0 | 758.0 |

2925 rows × 274 columns

▼ Grid Search

- İmport and prepare

```python
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=101)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

✓ 0.1s

```python
from sklearn.linear_model import ElasticNet
```

- grid search cv

```python
base_elastic_net_model = ElasticNet()
```

✓ 0.8s

```python
param_grid = {
    "alpha":[0.1,1,5,10,50,100],
    "l1_ratio":[.1,.5,.7,.95,.99,1]
    }
```

✓ 0.5s

```python
from sklearn.model_selection import GridSearchCV
```

✓ 0.8s

```python
grid_model = GridSearchCV(
    estimator = base_elastic_net_model,
    param_grid= param_grid,
    scoring="neg_mean_squared_error",
    cv= 5, verbose=2
    )
```

✓ 0.8s

```python
grid_model.fit(X_train,y_train)
```

✓ 0.6s

- fitted model.

```
1  grid_model.fit(X_train,y_train)
✓ 0.6s

Output exceeds the size limit. Open the full output data in a text editor
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV] END ............................alpha=0.1, l1_ratio=0.1; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.1; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.1; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.1; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.1; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.5; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.5; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.5; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.5; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.5; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.7; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.7; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.7; total time=   0.0s
[CV] END ............................alpha=0.1, l1_ratio=0.7; total time=   0.0s
...
[CV] END ..............................alpha=100, l1_ratio=1; total time=   0.0s
[CV] END ..............................alpha=100, l1_ratio=1; total time=   0.0s
[CV] END ..............................alpha=100, l1_ratio=1; total time=   0.0s
[CV] END ..............................alpha=100, l1_ratio=1; total time=   0.0s

GridSearchCV(cv=5, estimator=ElasticNet(),
             param_grid={'alpha': [0.1, 1, 5, 10, 50, 100],
                         'l1_ratio': [0.1, 0.5, 0.7, 0.95, 0.99, 1]},
             scoring='neg_mean_squared_error', verbose=2)
```

- en iyi modeli seçmek için
  farklı alpha ve l1_ratio değerleri için denendi ve en iyisi bulundu.

```
1  grid_model.best_estimator_
✓ 0.5s

ElasticNet(alpha=0.1, l1_ratio=1)
```

- Aynısı ama dictionary formunda

```python
grid_model.best_params_
```

0.4s

```
{'alpha': 0.1, 'l1_ratio': 1}
```

▼ Project Overview

- project data import

```python
df = pd.read_csv("../00 DATA/AMES_Final_DF.csv")
```

0.1s

```python
df.head()
```

0.9s

| | Lot Frontage | Lot Area | Overall Qual | Overall Cond | Year Built | Year Remod/Add | Mas Vnr Area |
|---|---|---|---|---|---|---|---|
| 0 | 141.0 | 31770 | 6 | 5 | 1960 | 1960 | 112.0 |
| 1 | 80.0 | 11622 | 5 | 6 | 1961 | 1961 | 0.0 |
| 2 | 81.0 | 14267 | 6 | 6 | 1958 | 1958 | 108.0 |
| 3 | 93.0 | 11160 | 7 | 5 | 1968 | 1968 | 0.0 |
| 4 | 74.0 | 13830 | 5 | 5 | 1997 | 1998 | 0.0 |

5 rows × 274 columns

- data and train test set preparing

```python
X = df.drop("SalePrice", axis=1)
y = df["SalePrice"]
```
✓ 0.1s Python

```python
from sklearn.model_selection import train_test_split
```
✓ 0.8s Python

```python
X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=0.1, random_state=101)

```
✓ 0.7s Python

```python
from sklearn.preprocessing import StandardScaler
```
✓ 0.9s Python

```python
scaler = StandardScaler()
scaler.fit(X_train)
scaled_X_train = scaler.transform(X_train)

# scaled_X_train = scaler.fit_transform(X_train)
# Üsteki 2 ve 3. satırın aynısını yapar

scaled_X_test = scaler.transform(X_test)
```
✓ 0.1s Python

- elasticnet and parameters

```python
from sklearn.linear_model import ElasticNet
```
✓ 0.3s

```python
base_elastic_model = ElasticNet()
```
✓ 0.1s

```python
param_grid = {
    "alpha":[0.1,1,5,10,50,100],
    "l1_ratio":[.1,.5,.7,.95,.99,1]
    }
```
✓ 0.4s

- Grid Search model and best parameters

```python
from sklearn.model_selection import GridSearchCV
```

✓ 0.4s

```python
grid_model = GridSearchCV(
    base_elastic_model,
    param_grid= param_grid,
    scoring="neg_mean_squared_error",
    cv=5, verbose=1
)
```

✓ 0.6s

```python
grid_model.fit(scaled_X_train,y_train)
```

✓ 4m 41.6s

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

GridSearchCV(cv=5, estimator=ElasticNet(max_iter=1000000),
             param_grid={'alpha': [0.1, 1, 5, 10, 50, 100],
                         'l1_ratio': [0.1, 0.5, 0.7, 0.95, 0.99, 1]},
             scoring='neg_mean_squared_error', verbose=1)
```

```python
grid_model.best_params_
```

✓ 0.8s

```
{'alpha': 100, 'l1_ratio': 1}
```

- mean squared error & mean absolute error

```
1  y_pred = grid_model.predict(scaled_X_test)
```
✓ 0.4s

```
1  from sklearn.metrics import mean_squared_error, mean_absolute_error
```
✓ 0.4s

```
1  mean_absolute_error(y_test,y_pred)
```
✓ 0.5s

```
14195.354900562168
```

```
1  np.sqrt(mean_squared_error(y_test,y_pred))
```
✓ 0.1s

```
20558.508566893164
```