Polynomial Regression

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. So, Polynomial Regression improves upon linear regression by considering higher order relationships on features.

Polynomial regression adresses two issues

- Non-linear relationships to labels
- Interaction terms between features

For linear: X1, X2
For polynomial (2nd degree): 1(bias), X1, X2, X1^2, X2^2, X1.X2

Imports

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Read CSV

In [2]:
```
df = pd.read_csv('Advertising.csv')
df.head()
```

Out[2]:

	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

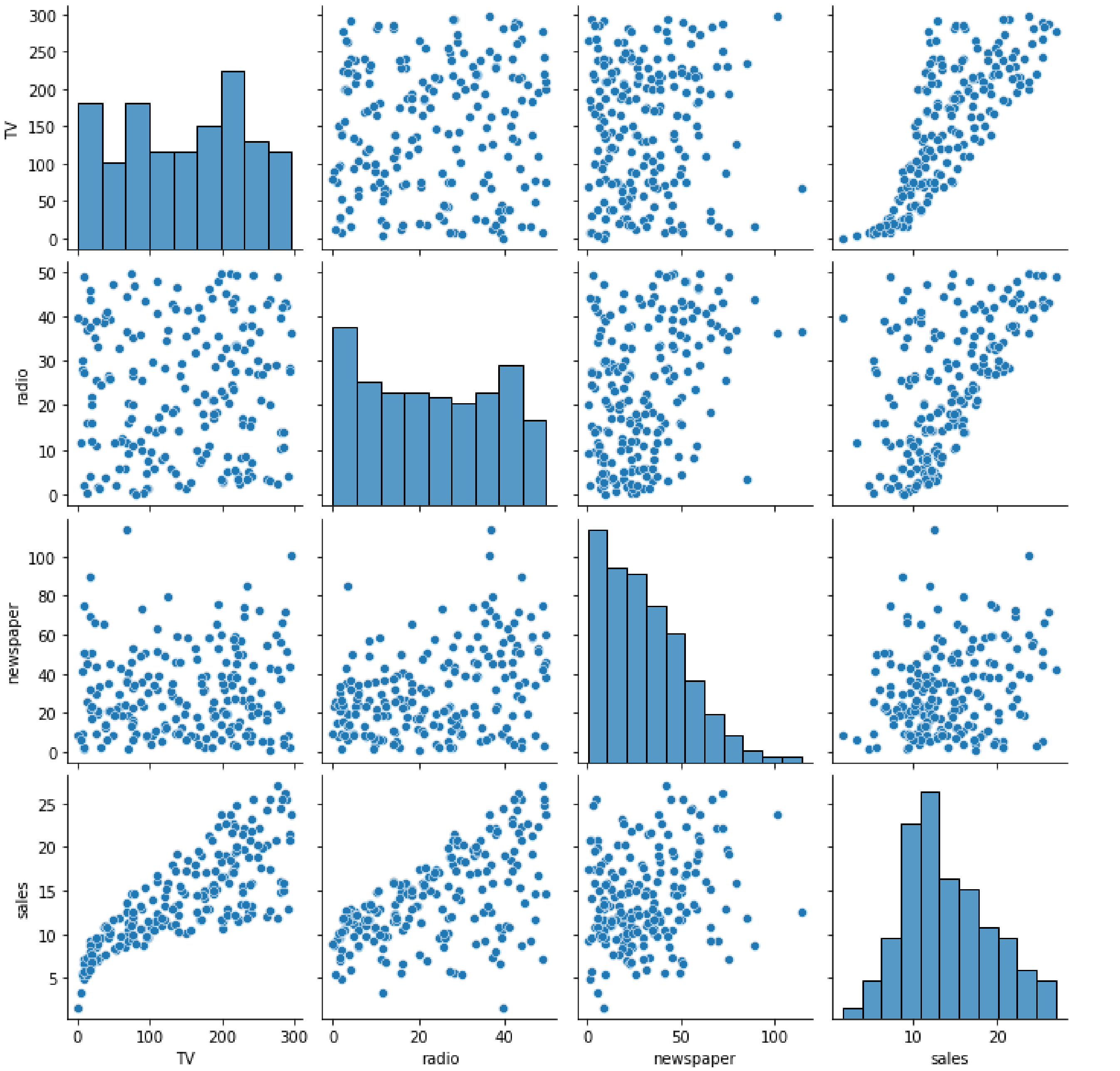
In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   radio      200 non-null    float64
 2   newspaper  200 non-null    float64
 3   sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

EDA

In [4]:
```
sns.pairplot(df)
```

Out[4]: <seaborn.axisgrid.PairGrid at 0x1a81aa32bb0>



Feature Selection

In [5]:
```
X = df.drop('sales', axis=1)
y = df['sales']
```

Polynomial Features

In [6]:
```
X.head()
```

Out[6]:

	TV	radio	newspaper
0	230.1	37.8	69.2
1	44.5	39.3	45.1
2	17.2	45.9	69.3

	TV	radio	newspaper
3	151.5	41.3	58.5
4	180.8	10.8	58.4

In [7]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [8]:
```
poly = PolynomialFeatures(degree=3, include_bias=False)
```

In [9]:
```
poly_features = poly.fit_transform(X)
```

In [10]:
```
poly_features[0] # One row of data
```

Out[10]:
```
array([2.30100000e+02, 3.78000000e+01, 6.92000000e+01, 5.29460100e+04,
       8.69778000e+03, 1.59229200e+04, 1.42884000e+03, 2.61576000e+03,
       4.78864000e+03, 1.21828769e+07, 2.00135918e+06, 3.66386389e+06,
       3.28776084e+05, 6.01886376e+05, 1.10186606e+06, 5.40101520e+04,
       9.88757280e+04, 1.81010592e+05, 3.31373888e+05])
```

In [11]:
```
len(poly_features[0]) # No. of independent columns
```

Out[11]:
```
19
```

3 columns have been converted to 19 columns for degree = 3.

Train Test Split

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(
    poly_features, # X data
    y, test_size=0.3, random_state=101
)
```

Scaling

In [14]:
```
from sklearn.preprocessing import StandardScaler
```

In [15]:
```
scaler = StandardScaler()
# Standard scaler will scale your data such that every column
# will have a mean of 0 and std of 1

# X_train_new = (X_train_old - X_train_mean) / X_train_std
# X_test_new = (X_test_old - X_train_mean) / X_train_std
```

In [16]:
```
pd.DataFrame(X_train)[0].describe()
```

Out[16]:
```
count    140.000000
mean     151.660714
```

```
std       84.560436
min        0.700000
25%       76.375000
50%      157.400000
75%      220.825000
max      296.400000
Name: 0, dtype: float64
```

In [17]:

```
scaler.fit(X_train)
# We will only use X_train for fitting
# But will transform both X_train anX_testin
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [18]:

```
pd.DataFrame(X_train)[0].describe()
```

Out[18]:

```
count    1.400000e+02
mean    -5.945641e-16
std      1.003591e+00
min     -1.791651e+00
25%     -8.935153e-01
50%      6.811570e-02
75%      8.208642e-01
max      1.717813e+00
Name: 0, dtype: float64
```

In [19]:

```
X_train[0]
```

Out[19]:

```
array([ 0.49300171, -0.33994238,  1.61586707,  0.28407363, -0.02568776,
        1.49677566, -0.59023161,  0.41659155,  1.6137853 ,  0.08057172,
       -0.05392229,  1.01524393, -0.36986163,  0.52457967,  1.48737034,
       -0.66096022, -0.16360242,  0.54694754,  1.37075536])
```

Polynomial Regression Using Linear Model

In [20]:

```
from sklearn.linear_model import LinearRegression
```

In [21]:

```
model = LinearRegression()
```

In [22]:

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

Out[22]:

```
LinearRegression()
```

In [23]:

```
model.coef_
```

Out[23]:

```
array([  7.18845622,   0.46512164,   0.26440021, -10.37072174,
         5.11723059,  -1.73451567,  -1.29762784,   0.8048698 ,
         0.45804867,   4.96674694,  -1.52605419,   1.50990248,
         0.38607921,  -0.32775826,   0.01138305,   0.74094852,
        -0.32771835,  -0.24949952,  -0.28434437])
```

In [24]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [25]:

```
# Testing Accuracy
pred = model.predict(X_test)
```

```
print(f"MAE: {mean_absolute_error(y_test, pred)}")
print(f"MSE: {mean_squared_error(y_test, pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, pred))}")
```

```
MAE: 0.41275160852975146
MSE: 0.33678137975071165
RMSE: 0.5803286825159615
```

In [26]:
```
# Training Accuracy
pred = model.predict(X_train)
print(f"MAE: {mean_absolute_error(y_train, pred)}")
print(f"MSE: {mean_squared_error(y_train, pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, pred))}")
```

```
MAE: 0.2910969680594902
MSE: 0.18829909447777834
RMSE: 0.43393443569020695
```

Higher Degree

In [27]:
```
poly = PolynomialFeatures(degree=5, include_bias=False)
```

In [28]:
```
poly_features = poly.fit_transform(X)
```

In [29]:
```
len(poly_features[0])
```

Out[29]: 55

In [30]:
```
X_train, X_test, y_train, y_test = train_test_split(
    poly_features, # X data
    y, test_size=0.3, random_state=101
)
```

In [31]:
```
# Alternate
# scaler.fit(X_train)
# X_train = scaler.transform(X_train)
# The above two lines can be written as the following

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [32]:
```
model = LinearRegression()
```

In [33]:
```
model.fit(X_train, y_train)
```

Out[33]: LinearRegression()

In [34]:
```
# Testing Accuracy
pred = model.predict(X_test)
print(f"MAE: {mean_absolute_error(y_test, pred)}")
print(f"MSE: {mean_squared_error(y_test, pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, pred))}")
# Worse than simple degree 1 linear regression
```

```
MAE: 0.665959430758744
MSE: 6.634847380747123
RMSE: 2.575819749273447
```

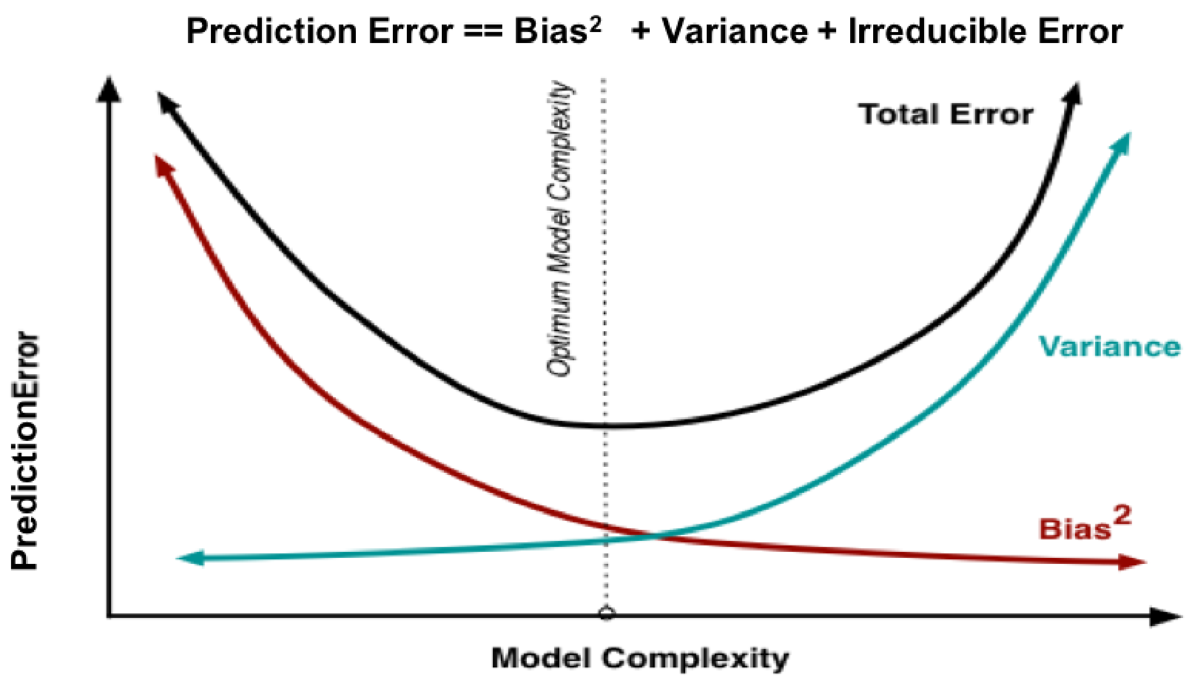
In [35]:

```
# Training Accuracy
pred = model.predict(X_train)
print(f"MAE: {mean_absolute_error(y_train, pred)}")
print(f"MSE: {mean_squared_error(y_train, pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, pred))}")
```

```
MAE: 0.18621293136504619
MSE: 0.06296801889161865
RMSE: 0.25093429198022865
```

Bias Variance Tradeoff

As you keep on increasing the model complexity, the bias decreases but the variance increases. The total error will decrease first but after a certain point it will again increase, so your goal is to find that degree of complexity where both bias and variance are low and the overall validation error is minimum.



If your model is underfitted (high bias and low variance) then increase the degree of the polynomial. On the other hand, if your model is overfitted (high variance and low bias) then either decrease the degree of polynomial or perform regularization.

3 main types of regularization:

- L1 (LASSO) Regularization
- L2 (Ridge) Regularization
- L1 + L2 (Elastic Net) Regularization

Lasso (L1)

In linear regression,
$\hat{y} = \Sigma\beta_i . X_i$

Average Squard Error = $(\Sigma(y - \hat{y})^2)/n$
Loss Function = $\Sigma(y - \Sigma\beta_i.X_i)^2$ (basically cost function = loss / n)

In Lasso,
Add a penalty to the loss function equal to the absolute value of the magnitude of the coefficients multiplied by a parameter. This results in a sparse model where some coefficients can become zero.
Loss Function = $\Sigma(y - \Sigma\beta_i.X_i)^2 + \lambda\Sigma|\beta_i|$

In [36]:
```
from sklearn.linear_model import LassoCV
```

In [37]:
```
l1_model = LassoCV(eps=0.001, n_alphas=100, max_iter=10000)
# eps = alpha_min / alpha_max (here, alpha refers to the lambda parameter)
# n_alphas: No. of alphas along the regularization path
```

In [38]:
```
l1_model.fit(X_train, y_train)
```

Out[38]:
```
LassoCV(max_iter=10000)
```

In [39]:
```
l1_model.coef_
```

Out[39]:
```
array([ 4.95889668,  0.12868624,  0.29785668, -4.16500046,  4.47968636,
       -0.36558799, -0.        ,  0.02476284,  0.        , -0.        ,
       -0.58434023, -0.        ,  0.        , -0.        , -0.        ,
       -0.        ,  0.        , -0.        , -0.        ,  0.        ,
       -0.        ,  0.        , -0.18667812, -0.0204454 , -0.        ,
        0.        , -0.        , -0.        , -0.        , -0.        ,
       -0.        , -0.        , -0.        , -0.        ,  0.86372141,
        0.        ,  0.19401888, -0.        , -0.        ,  0.        ,
       -0.        , -0.04710262, -0.        , -0.        ,  0.16126174,
       -0.        , -0.        , -0.        , -0.04110592, -0.        ,
       -0.        , -0.        , -0.        , -0.        , -0.        ])
```

In [40]:
```
l1_pred = l1_model.predict(X_test)
```

In [41]:
```
print(f"MAE: {mean_absolute_error(y_test, l1_pred)}")
print(f"MSE: {mean_squared_error(y_test, l1_pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, l1_pred))}")
```

```
MAE: 0.41976016404424027
MSE: 0.3525310273027487
RMSE: 0.5937432334795477
```

In [42]:
```
# Training Accuracy
pred = l1_model.predict(X_train)
print(f"MAE: {mean_absolute_error(y_train, pred)}")
print(f"MSE: {mean_squared_error(y_train, pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, pred))}")
```

```
MAE: 0.3140615779008167
MSE: 0.24484087109782457
RMSE: 0.4948139762555465
```

Ridge (L2)

Loss Function = $\Sigma(y - \Sigma B_i.X_i)^2 + \lambda\Sigma\beta_i^2$

In [43]:
```
from sklearn.linear_model import RidgeCV
```

In [44]:
```
l2_model = RidgeCV(alphas=(0.1, 1.0, 10.0))
```

In [45]:
```
l2_model.fit(X_train, y_train)
```

Out[45]:
```
RidgeCV(alphas=array([ 0.1,  1. , 10. ]))
```

In [46]:
```
l2_model.coef_
```

Out[46]:
```
array([ 4.88247252,  0.79604177,  0.4585615 , -3.67001462,  3.55636016,
       -0.333457  , -0.78142144,  0.57387608, -0.78493528, -0.39855374,
       -1.54418648, -0.78215664,  2.19546759, -0.27219785,  0.09571081,
       -0.92783259,  0.46145128, -0.74755379,  0.72839659,  0.96327349,
        0.68468251,  0.55624967, -1.44959617, -0.13738769,  0.30477433,
        1.21320134, -0.23498898, -0.19700746,  0.24877753, -0.28214241,
        0.27082549, -0.40964296,  0.15327191,  0.73014726, -0.04053588,
        0.46442708,  0.08903263,  0.65866141, -0.08866554,  0.16294276,
       -1.77053136,  0.44506535,  0.41575328,  0.01398179,  0.45781413,
       -0.45113703,  0.18087705,  0.02307267, -0.76774296,  0.65612211,
       -0.13673942, -0.20476947,  0.39668596, -0.00928643, -0.69665206])
```

In [53]:
```
l2_pred = l2_model.predict(X_test)
```

In [54]:
```
print(f"MAE: {mean_absolute_error(y_test, l2_pred)}")
print(f"MSE: {mean_squared_error(y_test, l2_pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, l2_pred))}")
```

```
MAE: 0.4552803132743398
MSE: 0.43757778203062664
RMSE: 0.6614966228414372
```

In [55]:
```
# Training Accuracy
pred = l2_model.predict(X_train)
print(f"MAE: {mean_absolute_error(y_train, pred)}")
print(f"MSE: {mean_squared_error(y_train, pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_train, pred))}")
```

```
MAE: 0.2748310506262111
MSE: 0.18999994866109965
RMSE: 0.4358898354643059
```

Elastic Net Regression (L1 + L2)

Loss Function: $\Sigma(y - \Sigma\beta_i.X_i)^2 + \lambda(0.5 \times (1 - \alpha) \times \Sigma\beta_j^2 + 0.5 \times \alpha \times \Sigma|\beta_j|)$

In [56]:
```
from sklearn.linear_model import ElasticNetCV
```

In [57]:
```
elastic = ElasticNetCV(
    l1_ratio=[.1, .5, .7, .9, .99, 1], # alpha symbol in equation
    eps=0.001, n_alphas=100, # lambda symbol
    max_iter=1000000
)
```

In [58]:
```
elastic.fit(X_train, y_train)
```

Out[58]:
```
ElasticNetCV(l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.99, 1], max_iter=1000000)
```

In [59]:
```python
elastic.coef_
```

Out[59]:
```
array([ 4.95889668,  0.12868624,  0.29785668, -4.16500046,  4.47968636,
       -0.36558799, -0.        ,  0.02476284,  0.        , -0.        ,
       -0.58434023, -0.        ,  0.        , -0.        , -0.        ,
       -0.        ,  0.        , -0.        , -0.        ,  0.        ,
       -0.        ,  0.        , -0.18667812, -0.0204454 , -0.        ,
        0.        , -0.        , -0.        , -0.        , -0.        ,
       -0.        , -0.        , -0.        , -0.        ,  0.86372141,
        0.        ,  0.19401888, -0.        , -0.        ,  0.        ,
       -0.        , -0.04710262, -0.        , -0.        ,  0.16126174,
       -0.        , -0.        , -0.        , -0.04110592, -0.        ,
       -0.        , -0.        , -0.        , -0.        , -0.        ])
```

In [60]:
```python
elastic.l1_ratio_
# Elastic net has determined that l1_ratio=1 gives the best performance
# It means that in this case, L2 regularization is not occurring
```

Out[60]:
```
1.0
```

In [61]:
```python
elastic_pred = elastic.predict(X_test)
```

In [62]:
```python
print(f"MAE: {mean_absolute_error(y_test, elastic_pred)}")
print(f"MSE: {mean_squared_error(y_test, elastic_pred)}")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, elastic_pred))}")
```

```
MAE: 0.41976016404424027
MSE: 0.3525310273027487
RMSE: 0.5937432334795477
```