T2_polynomial-regression-VHA

June 3, 2024

1 Introduction to Polynomial Regression

Polynomial Regression is an extension of Linear Regression that models the relationship between the independent variable (x) and the dependent variable (y) as an nth degree polynomial. Unlike linear regression, which fits a straight line, polynomial regression can fit a curve to the data, making it suitable for capturing nonlinear relationships.

1.0.1 Polynomial Regression Intuition

In polynomial regression, the model is represented as:

[y = _0 + _1 x + _2 x^2 + _3 x^3 + …+ _n x^n +]

Where: - (y) is the dependent variable. - (x) is the independent variable. - (_0, _1, …, _n) are the coefficients of the model. - (n) is the degree of the polynomial. - ( ) is the error term.

The choice of the degree (n) determines the flexibility of the model. Higher degrees allow the model to fit more complex curves but can also lead to overfitting.

1.0.2 Polynomial Regression on Ice Cream Selling Data: Code Example

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

[2]:
```
df=pd.read_csv('Ice_cream selling data.csv')
```

[3]:
```
df.head()
```

[3]:

	Temperature (°C)	Ice Cream Sales (units)
0	-4.662263	41.842986
1	-4.316559	34.661120
2	-4.213985	39.383001
3	-3.949661	37.539845
4	-3.578554	32.284531

[4]:
```
df.shape
```

[4]:
```
(49, 2)
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49 entries, 0 to 48
Data columns (total 2 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Temperature (°C)        49 non-null     float64
 1   Ice Cream Sales (units)  49 non-null     float64
dtypes: float64(2)
memory usage: 916.0 bytes
```

[6]:
```
df.describe()
```

[6]:

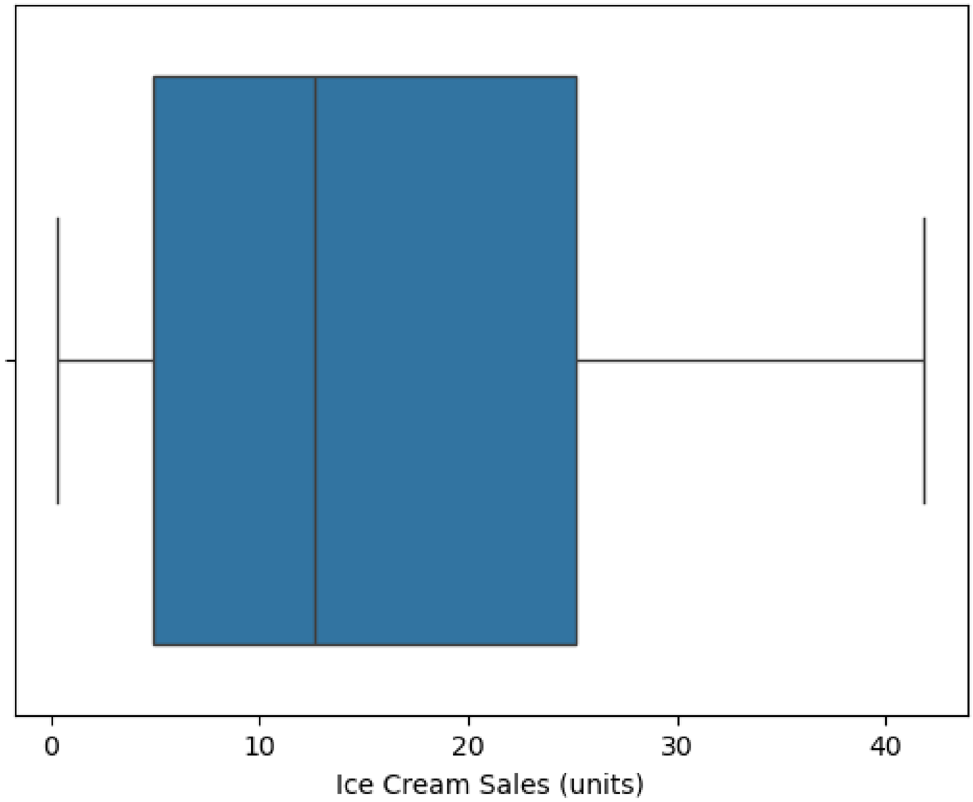
	Temperature (°C)	Ice Cream Sales (units)
count	49.000000	49.000000
mean	0.271755	15.905308
std	2.697672	12.264682
min	-4.662263	0.328626
25%	-2.111870	4.857988
50%	0.688781	12.615181
75%	2.784836	25.142082
max	4.899032	41.842986

[7]:
```
df.duplicated().sum()
```

[7]: 0

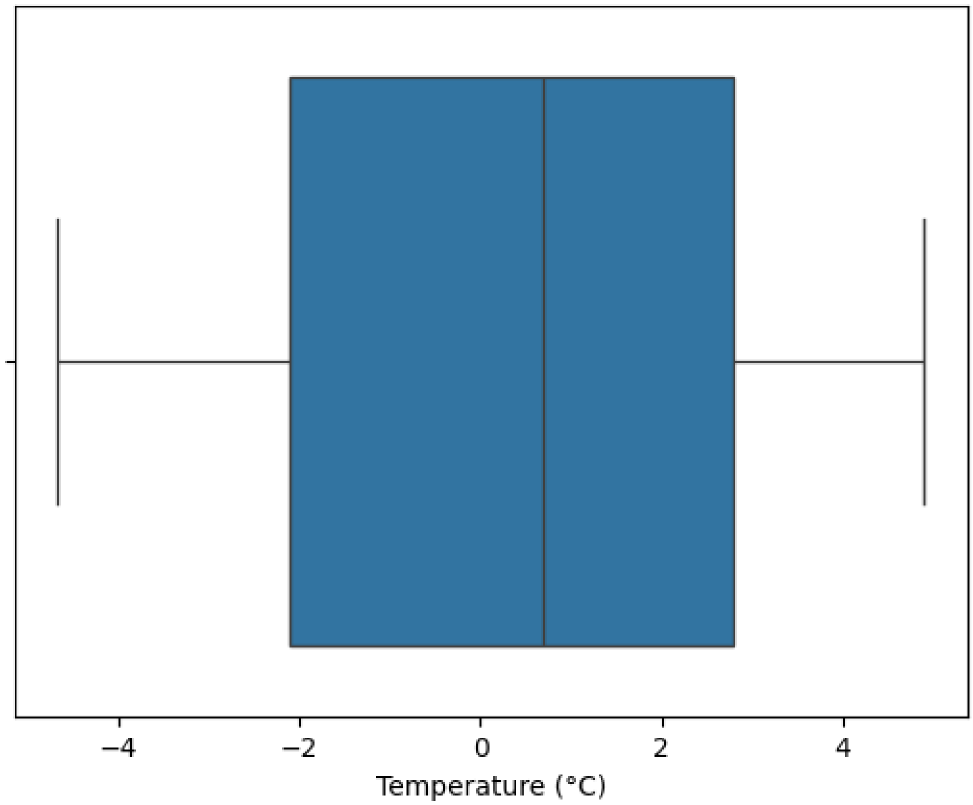
[8]:
```
sns.boxplot(x='Ice Cream Sales (units)',data=df)
```

[8]:
```
<Axes: xlabel='Ice Cream Sales (units)'>
```

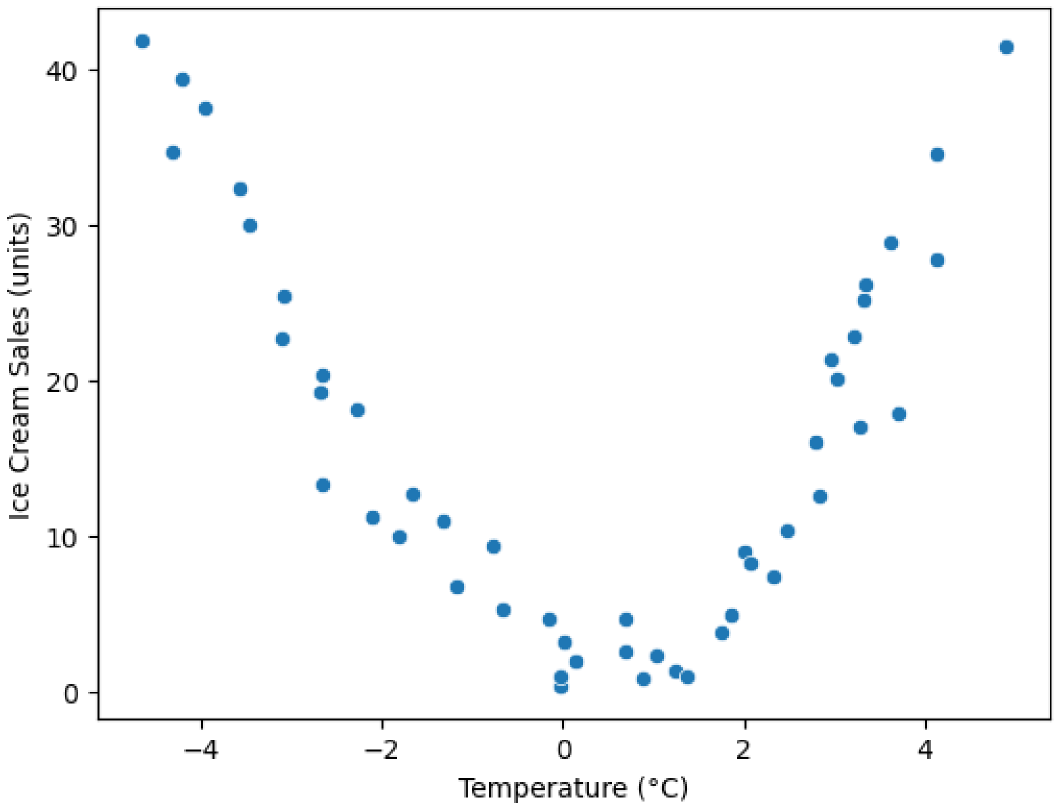


[9]:
```
sns.boxplot(x='Temperature (°C)',data=df)
```

[9]: <Axes: xlabel='Temperature (°C)'>



[10]:
```
sns.scatterplot(x='Temperature (°C)',y='Ice Cream Sales (units)',data=df)
plt.show()
```



[11]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

[12]:
```
x
```

[12]:

	Temperature (°C)
0	-4.662263
1	-4.316559
2	-4.213985
3	-3.949661
4	-3.578554
5	-3.455712
6	-3.108440
7	-3.081303
8	-2.672461
9	-2.652287
10	-2.651498
11	-2.288264
12	-2.111870
13	-1.818938
14	-1.660348

```
15        -1.326379
16        -1.173123
17        -0.773330
18        -0.673753
19        -0.149635
20        -0.036156
21        -0.033895
22         0.008608
23         0.149245
24         0.688781
25         0.693599
26         0.874905
27         1.024181
28         1.240712
29         1.359813
30         1.740000
31         1.850552
32         1.999310
33         2.075101
34         2.318591
35         2.471946
36         2.784836
37         2.831760
38         2.959932
39         3.020874
40         3.211366
41         3.270044
42         3.316073
43         3.335932
44         3.610778
45         3.704057
46         4.130868
47         4.133534
48         4.899032
```

[13]:
```
y
```

[13]:
```
0      41.842986
1      34.661120
2      39.383001
3      37.539845
4      32.284531
5      30.001138
6      22.635401
7      25.365022
8      19.226970
9      20.279679
```

```
10    13.275828
11    18.123991
12    11.218294
13    10.012868
14    12.615181
15    10.957731
16     6.689123
17     9.392969
18     5.210163
19     4.673643
20     0.328626
21     0.897603
22     3.165600
23     1.931416
24     2.576782
25     4.625689
26     0.789974
27     2.313806
28     1.292361
29     0.953115
30     3.782570
31     4.857988
32     8.943823
33     8.170735
34     7.412094
35    10.336631
36    15.996620
37    12.568237
38    21.342916
39    20.114413
40    22.839406
41    16.983279
42    25.142082
43    26.104740
44    28.912188
45    17.843957
46    34.530743
47    27.698383
48    41.514822
Name: Ice Cream Sales (units), dtype: float64
```

[14]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪2,random_state=20)
```

[15]:
```
X_train.shape
```

[15]: (39, 1)

[16]:
```
X_test.shape
```

[16]: (10, 1)

[17]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

[18]:
```
# after etration the 4th degree is the best
poly = PolynomialFeatures(degree=4)
x_poly=poly.fit_transform(X_train)
```

[19]:
```
Regression=LinearRegression()
Regression.fit(x_poly,y_train)
```

[19]: LinearRegression()

[20]:
```
# prompt: check  model performance with all metrics
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
y_pred=Regression.predict(poly.fit_transform(X_test))
print('coefficient of determination:', r2_score(y_test, y_pred))
print('mean_squared_error:', mean_squared_error(y_test, y_pred))
print('mean_absolute_error:', mean_absolute_error(y_test, y_pred))
```

```
coefficient of determination: 0.9702708883672491
mean_squared_error: 5.522019917530303
mean_absolute_error: 1.9122529553556515
```

How to Select the Degree

Choosing the appropriate degree for polynomial regression is crucial: - **Cross-Validation**: Use techniques like k-fold cross-validation to evaluate the performance for different degrees and select the one that minimizes the error. - **Visualization**: Plot the polynomial fit for different degrees and visually inspect for underfitting or overfitting. - **Domain Knowledge**: Use knowledge about the data to inform the choice of degree.

1.0.3 Overfitting and Underfitting in Polynomial Regression

- **Overfitting**: When the degree is too high, the model becomes too flexible, capturing noise in the data and failing to generalize to new data.
- **Underfitting**: When the degree is too low, the model is too simplistic and fails to capture the underlying trend in the data.

1.0.4 Limitations of Polynomial Regression

1. **Sensitive to Outliers**: Outliers can disproportionately affect the polynomial fit, leading to poor generalization.

2. **Overfitting**: High-degree polynomials can fit the training data very well but fail to generalize to new data.
3. **Extrapolation**: Polynomial regression performs poorly when extrapolating beyond the range of the training data.
4. **Computational Complexity**: High-degree polynomials can lead to computational inefficiencies and numerical instability.

1.0.5 Outro

Polynomial regression is a powerful tool for modeling nonlinear relationships. By fitting a polynomial curve to the data, it can capture complex trends that linear regression cannot. However, choosing the right degree is critical to avoid overfitting or underfitting. Despite its limitations, polynomial regression remains a fundamental technique in the machine learning toolbox, especially when the relationship between variables is inherently nonlinear.

[ ]:

```

```

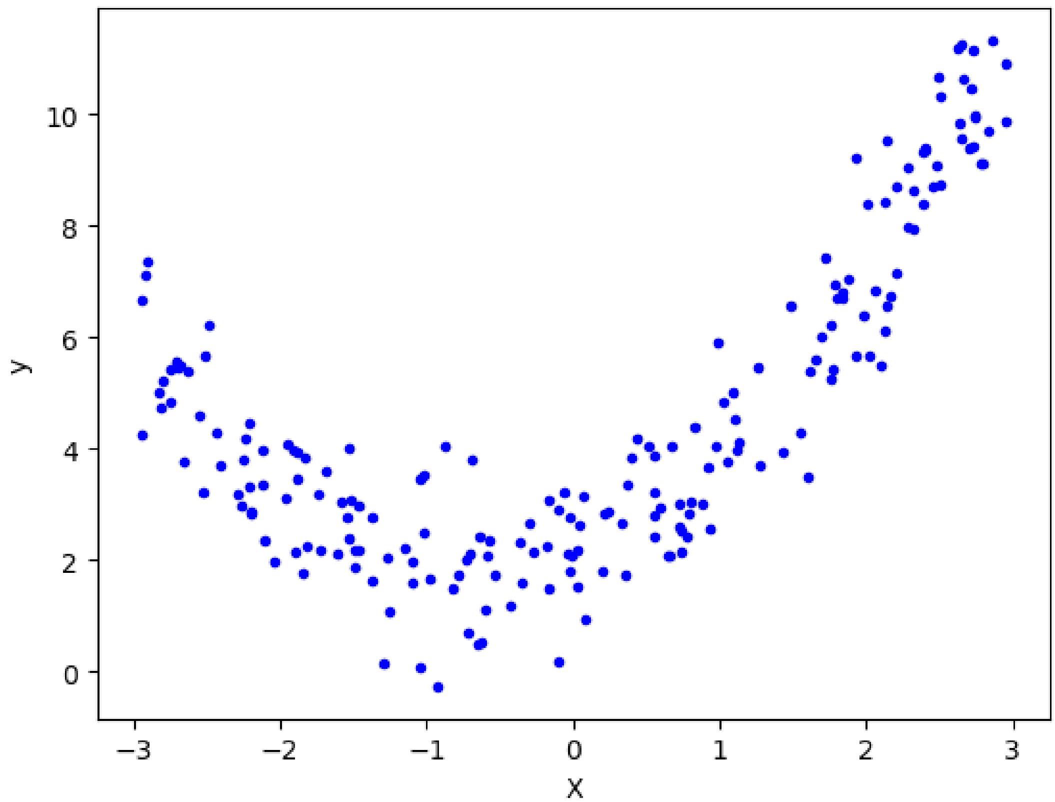
T2_polynomial-regression_EXAMPLE_VHA

June 3, 2024

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression,SGDRegressor

from sklearn.preprocessing import PolynomialFeatures,StandardScaler

from sklearn.metrics import r2_score

from sklearn.pipeline import Pipeline
```

[2]:
```
X = 6 * np.random.rand(200, 1) - 3
y = 0.8 * X**2 + 0.9 * X + 2 + np.random.randn(200, 1)

# y = 0.8x^2 + 0.9x + 2
```

[3]:
```
plt.plot(X, y,'b.')
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```



[4]:
```
# Train test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪2,random_state=2)
```

[5]:
```
# Applying linear regression
lr = LinearRegression()
```

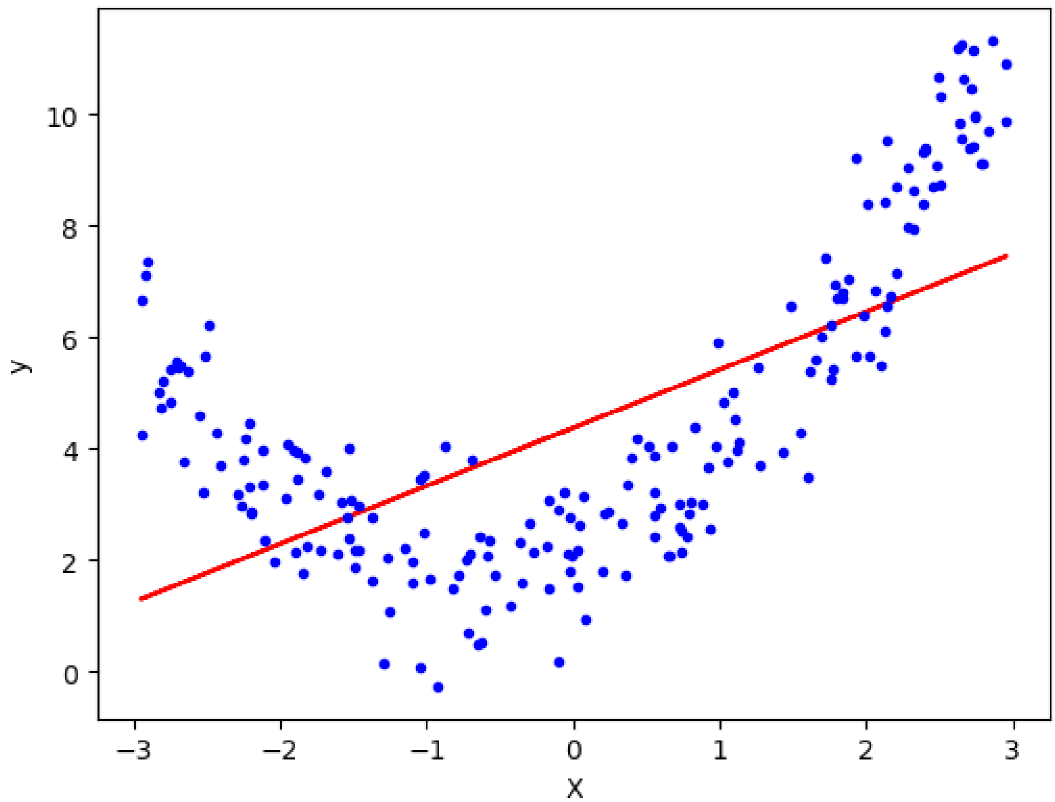
[6]:
```
lr.fit(X_train,y_train)
```

[6]: LinearRegression()

[7]:
```
y_pred = lr.predict(X_test)
r2_score(y_test,y_pred)
```

[7]: 0.31469262792000596

[8]:
```
plt.plot(X_train,lr.predict(X_train),color='r')
plt.plot(X, y, "b.")
plt.xlabel("X")
plt.ylabel("y")
plt.show()
```



[9]:
```
# Applying Polynomial Linear Regression
# degree 2
poly = PolynomialFeatures(degree=2,include_bias=True)

X_train_trans = poly.fit_transform(X_train)
X_test_trans = poly.transform(X_test)
```

[10]:
```
print(X_train[0])
print(X_train_trans[0])
```

```
[-0.36010041]
[ 1.         -0.36010041  0.1296723 ]
```

[11]:
```
# include_bias parameter
```

[12]:
```
lr = LinearRegression()
lr.fit(X_train_trans,y_train)
```

[12]: LinearRegression()

[13]:
```
y_pred = lr.predict(X_test_trans)
```

[14]:
```
r2_score(y_test,y_pred)
```

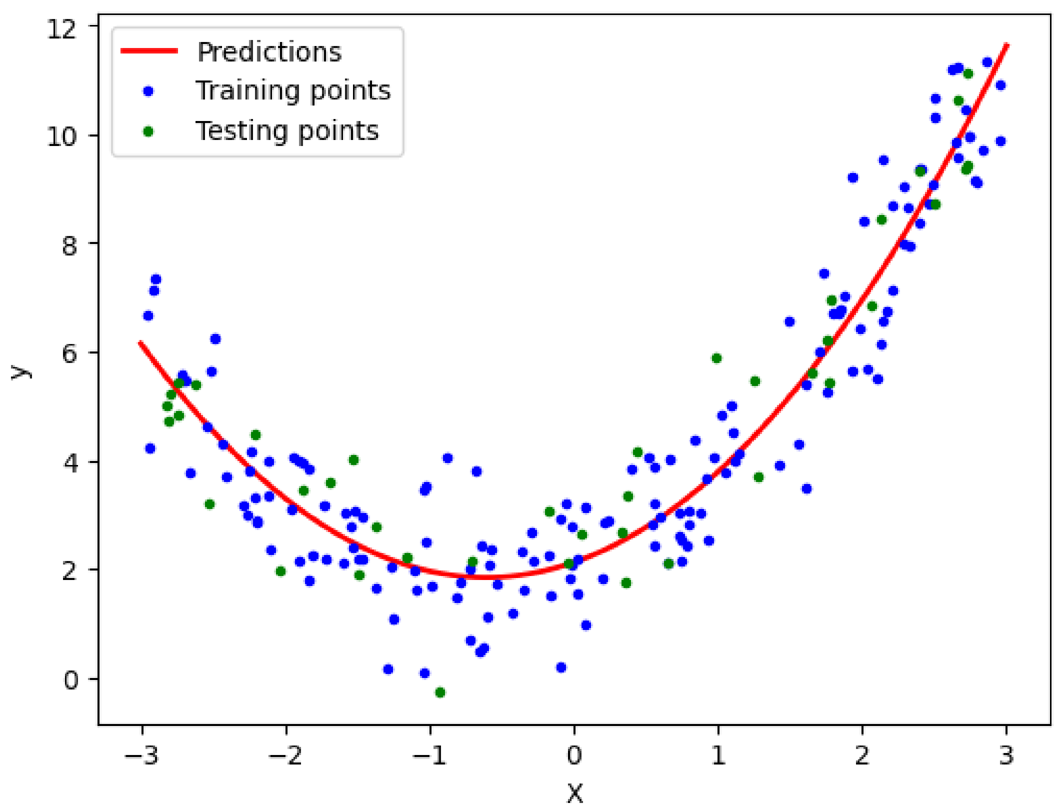
[14]: 0.8878581752899711

[15]:
```
print(lr.coef_)
print(lr.intercept_)
```

```
[[0.         0.91382988 0.75248421]]
[2.11474596]
```

[16]:
```
X_new=np.linspace(-3, 3, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = lr.predict(X_new_poly)
```

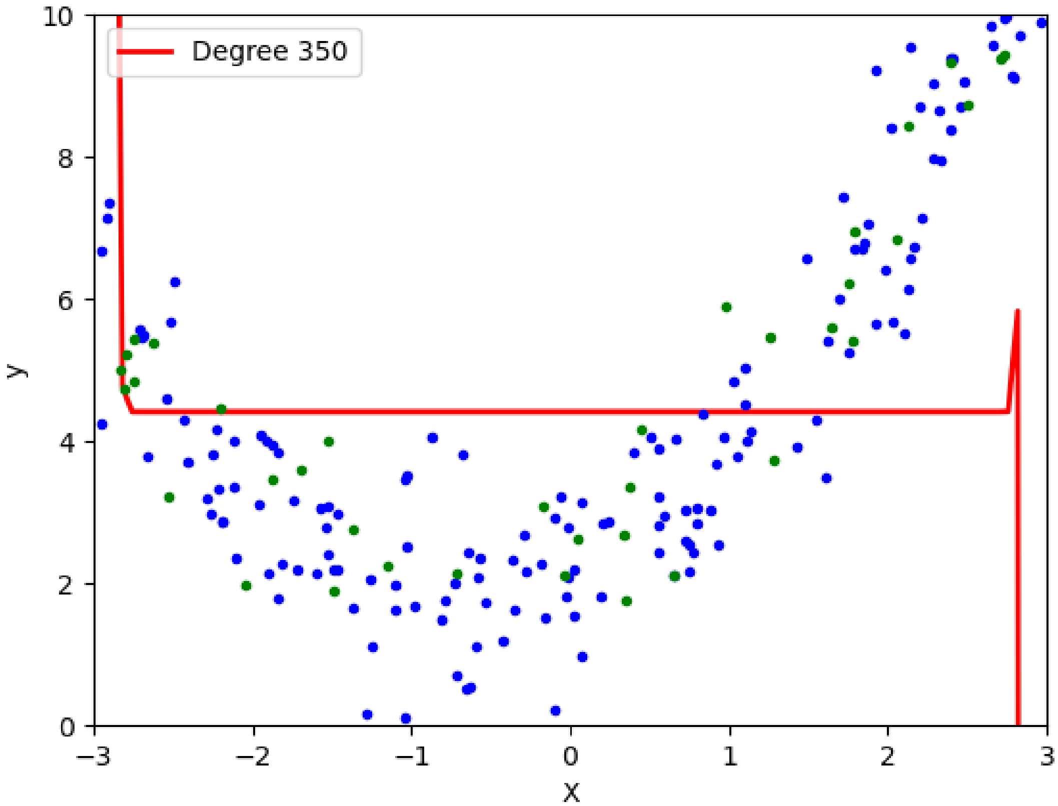
[17]:
```
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")
plt.plot(X_train, y_train, "b.",label='Training points')
plt.plot(X_test, y_test, "g.",label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```



[18]:
```
def polynomial_regression(degree):
    X_new=np.linspace(-3, 3, 100).reshape(100, 1)
    X_new_poly = poly.transform(X_new)

    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
            ("poly_features", polybig_features),
            ("std_scaler", std_scaler),
            ("lin_reg", lin_reg),
        ])
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    plt.plot(X_new, y_newbig,'r', label="Degree " + str(degree), linewidth=2)

    plt.plot(X_train, y_train, "b.", linewidth=3)
    plt.plot(X_test, y_test, "g.", linewidth=3)
    plt.legend(loc="upper left")
    plt.xlabel("X")
    plt.ylabel("y")
    plt.axis([-3, 3, 0, 10])
    plt.show()
```

[19]:
```
polynomial_regression(350)
```

```
C:\Users\VISHAL\anaconda3\Lib\site-packages\sklearn\utils\extmath.py:1066:
RuntimeWarning: overflow encountered in square
  temp **= 2
C:\Users\VISHAL\anaconda3\Lib\site-packages\numpy\core\fromnumeric.py:88:
RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
```



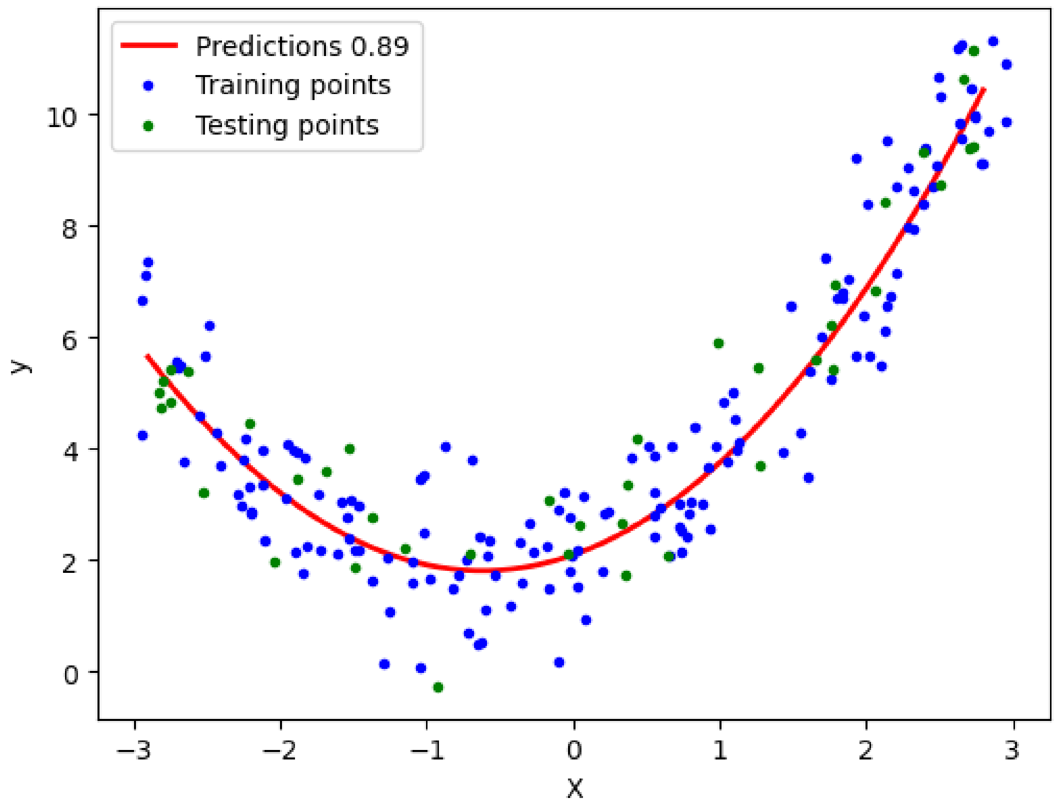
[20]:
```
poly.powers_
```

[20]:
```
array([[0],
       [1],
       [2]], dtype=int64)
```

[21]:
```
# Applying Gradient Descent

poly = PolynomialFeatures(degree=2)

X_train_trans = poly.fit_transform(X_train)
X_test_trans = poly.transform(X_test)

sgd = SGDRegressor(max_iter=100)
sgd.fit(X_train_trans,y_train)

X_new=np.linspace(-2.9, 2.8, 200).reshape(200, 1)
X_new_poly = poly.transform(X_new)
y_new = sgd.predict(X_new_poly)

y_pred = sgd.predict(X_test_trans)
```

```
plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions " +␣
↪str(round(r2_score(y_test,y_pred),2)))
plt.plot(X_train, y_train, "b.",label='Training points')
plt.plot(X_test, y_test, "g.",label='Testing points')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
C:\Users\VISHAL\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```



[22]:
```
# 3D polynomial regression
x = 7 * np.random.rand(100, 1) - 2.8
y = 7 * np.random.rand(100, 1) - 2.8

z = x**2 + y**2 + 0.2*x + 0.2*y + 0.1*x*y +2 + np.random.randn(100, 1)
```

```
# z = x^2 + y^2 + 0.2x + 0.2y + 0.1xy + 2
```

[23]:
```
import plotly.express as px
df = px.data.iris()
fig = px.scatter_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())
fig.show()
```

[24]:
```
lr = LinearRegression()
lr.fit(np.array([x,y]).reshape(100,2),z)

x_input = np.linspace(x.min(), x.max(), 10)
y_input = np.linspace(y.min(), y.max(), 10)
xGrid, yGrid = np.meshgrid(x_input,y_input)

final = np.vstack((xGrid.ravel().reshape(1,100),yGrid.ravel().reshape(1,100))).T

z_final = lr.predict(final).reshape(10,10)
```

[25]:
```
import plotly.graph_objects as go

fig = px.scatter_3d(df, x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.add_trace(go.Surface(x = x_input, y = y_input, z =z_final ))

fig.show()
```

[26]:
```
X_multi = np.array([x,y]).reshape(100,2)
X_multi.shape
```

[26]:
```
(100, 2)
```

[27]:
```
poly = PolynomialFeatures(degree=30)
X_multi_trans = poly.fit_transform(X_multi)
```

[30]:
```
#print("Input",poly.n_input_features_)
print("Ouput",poly.n_output_features_)
print("Powers\n",poly.powers_)
```

```
Ouput 496
Powers
 [[ 0  0]
 [ 1  0]
 [ 0  1]
 [ 2  0]
 [ 1  1]
 [ 0  2]
 [ 3  0]
 [ 2  1]
```

```
[ 1  2]
[ 0  3]
[ 4  0]
[ 3  1]
[ 2  2]
[ 1  3]
[ 0  4]
[ 5  0]
[ 4  1]
[ 3  2]
[ 2  3]
[ 1  4]
[ 0  5]
[ 6  0]
[ 5  1]
[ 4  2]
[ 3  3]
[ 2  4]
[ 1  5]
[ 0  6]
[ 7  0]
[ 6  1]
[ 5  2]
[ 4  3]
[ 3  4]
[ 2  5]
[ 1  6]
[ 0  7]
[ 8  0]
[ 7  1]
[ 6  2]
[ 5  3]
[ 4  4]
[ 3  5]
[ 2  6]
[ 1  7]
[ 0  8]
[ 9  0]
[ 8  1]
[ 7  2]
[ 6  3]
[ 5  4]
[ 4  5]
[ 3  6]
[ 2  7]
[ 1  8]
[ 0  9]
[10  0]
```

```
[ 9  1]
[ 8  2]
[ 7  3]
[ 6  4]
[ 5  5]
[ 4  6]
[ 3  7]
[ 2  8]
[ 1  9]
[ 0 10]
[11  0]
[10  1]
[ 9  2]
[ 8  3]
[ 7  4]
[ 6  5]
[ 5  6]
[ 4  7]
[ 3  8]
[ 2  9]
[ 1 10]
[ 0 11]
[12  0]
[11  1]
[10  2]
[ 9  3]
[ 8  4]
[ 7  5]
[ 6  6]
[ 5  7]
[ 4  8]
[ 3  9]
[ 2 10]
[ 1 11]
[ 0 12]
[13  0]
[12  1]
[11  2]
[10  3]
[ 9  4]
[ 8  5]
[ 7  6]
[ 6  7]
[ 5  8]
[ 4  9]
[ 3 10]
[ 2 11]
[ 1 12]
```

```
[ 0 13]
[14  0]
[13  1]
[12  2]
[11  3]
[10  4]
[ 9  5]
[ 8  6]
[ 7  7]
[ 6  8]
[ 5  9]
[ 4 10]
[ 3 11]
[ 2 12]
[ 1 13]
[ 0 14]
[15  0]
[14  1]
[13  2]
[12  3]
[11  4]
[10  5]
[ 9  6]
[ 8  7]
[ 7  8]
[ 6  9]
[ 5 10]
[ 4 11]
[ 3 12]
[ 2 13]
[ 1 14]
[ 0 15]
[16  0]
[15  1]
[14  2]
[13  3]
[12  4]
[11  5]
[10  6]
[ 9  7]
[ 8  8]
[ 7  9]
[ 6 10]
[ 5 11]
[ 4 12]
[ 3 13]
[ 2 14]
[ 1 15]
```

```
[ 0 16]
[17  0]
[16  1]
[15  2]
[14  3]
[13  4]
[12  5]
[11  6]
[10  7]
[ 9  8]
[ 8  9]
[ 7 10]
[ 6 11]
[ 5 12]
[ 4 13]
[ 3 14]
[ 2 15]
[ 1 16]
[ 0 17]
[18  0]
[17  1]
[16  2]
[15  3]
[14  4]
[13  5]
[12  6]
[11  7]
[10  8]
[ 9  9]
[ 8 10]
[ 7 11]
[ 6 12]
[ 5 13]
[ 4 14]
[ 3 15]
[ 2 16]
[ 1 17]
[ 0 18]
[19  0]
[18  1]
[17  2]
[16  3]
[15  4]
[14  5]
[13  6]
[12  7]
[11  8]
[10  9]
```

```
[ 9 10]
[ 8 11]
[ 7 12]
[ 6 13]
[ 5 14]
[ 4 15]
[ 3 16]
[ 2 17]
[ 1 18]
[ 0 19]
[20  0]
[19  1]
[18  2]
[17  3]
[16  4]
[15  5]
[14  6]
[13  7]
[12  8]
[11  9]
[10 10]
[ 9 11]
[ 8 12]
[ 7 13]
[ 6 14]
[ 5 15]
[ 4 16]
[ 3 17]
[ 2 18]
[ 1 19]
[ 0 20]
[21  0]
[20  1]
[19  2]
[18  3]
[17  4]
[16  5]
[15  6]
[14  7]
[13  8]
[12  9]
[11 10]
[10 11]
[ 9 12]
[ 8 13]
[ 7 14]
[ 6 15]
[ 5 16]
```

```
[ 4 17]
[ 3 18]
[ 2 19]
[ 1 20]
[ 0 21]
[22  0]
[21  1]
[20  2]
[19  3]
[18  4]
[17  5]
[16  6]
[15  7]
[14  8]
[13  9]
[12 10]
[11 11]
[10 12]
[ 9 13]
[ 8 14]
[ 7 15]
[ 6 16]
[ 5 17]
[ 4 18]
[ 3 19]
[ 2 20]
[ 1 21]
[ 0 22]
[23  0]
[22  1]
[21  2]
[20  3]
[19  4]
[18  5]
[17  6]
[16  7]
[15  8]
[14  9]
[13 10]
[12 11]
[11 12]
[10 13]
[ 9 14]
[ 8 15]
[ 7 16]
[ 6 17]
[ 5 18]
[ 4 19]
```

```
[ 3 20]
[ 2 21]
[ 1 22]
[ 0 23]
[24  0]
[23  1]
[22  2]
[21  3]
[20  4]
[19  5]
[18  6]
[17  7]
[16  8]
[15  9]
[14 10]
[13 11]
[12 12]
[11 13]
[10 14]
[ 9 15]
[ 8 16]
[ 7 17]
[ 6 18]
[ 5 19]
[ 4 20]
[ 3 21]
[ 2 22]
[ 1 23]
[ 0 24]
[25  0]
[24  1]
[23  2]
[22  3]
[21  4]
[20  5]
[19  6]
[18  7]
[17  8]
[16  9]
[15 10]
[14 11]
[13 12]
[12 13]
[11 14]
[10 15]
[ 9 16]
[ 8 17]
[ 7 18]
```

```
[ 6 19]
[ 5 20]
[ 4 21]
[ 3 22]
[ 2 23]
[ 1 24]
[ 0 25]
[26  0]
[25  1]
[24  2]
[23  3]
[22  4]
[21  5]
[20  6]
[19  7]
[18  8]
[17  9]
[16 10]
[15 11]
[14 12]
[13 13]
[12 14]
[11 15]
[10 16]
[ 9 17]
[ 8 18]
[ 7 19]
[ 6 20]
[ 5 21]
[ 4 22]
[ 3 23]
[ 2 24]
[ 1 25]
[ 0 26]
[27  0]
[26  1]
[25  2]
[24  3]
[23  4]
[22  5]
[21  6]
[20  7]
[19  8]
[18  9]
[17 10]
[16 11]
[15 12]
[14 13]
```

```
[13 14]
[12 15]
[11 16]
[10 17]
[ 9 18]
[ 8 19]
[ 7 20]
[ 6 21]
[ 5 22]
[ 4 23]
[ 3 24]
[ 2 25]
[ 1 26]
[ 0 27]
[28  0]
[27  1]
[26  2]
[25  3]
[24  4]
[23  5]
[22  6]
[21  7]
[20  8]
[19  9]
[18 10]
[17 11]
[16 12]
[15 13]
[14 14]
[13 15]
[12 16]
[11 17]
[10 18]
[ 9 19]
[ 8 20]
[ 7 21]
[ 6 22]
[ 5 23]
[ 4 24]
[ 3 25]
[ 2 26]
[ 1 27]
[ 0 28]
[29  0]
[28  1]
[27  2]
[26  3]
[25  4]
```

```
[24  5]
[23  6]
[22  7]
[21  8]
[20  9]
[19 10]
[18 11]
[17 12]
[16 13]
[15 14]
[14 15]
[13 16]
[12 17]
[11 18]
[10 19]
[ 9 20]
[ 8 21]
[ 7 22]
[ 6 23]
[ 5 24]
[ 4 25]
[ 3 26]
[ 2 27]
[ 1 28]
[ 0 29]
[30  0]
[29  1]
[28  2]
[27  3]
[26  4]
[25  5]
[24  6]
[23  7]
[22  8]
[21  9]
[20 10]
[19 11]
[18 12]
[17 13]
[16 14]
[15 15]
[14 16]
[13 17]
[12 18]
[11 19]
[10 20]
[ 9 21]
[ 8 22]
```

```
 [ 7 23]
 [ 6 24]
 [ 5 25]
 [ 4 26]
 [ 3 27]
 [ 2 28]
 [ 1 29]
 [ 0 30]]
```

[31]:
```
X_multi_trans.shape
```

[31]: (100, 496)

[32]:
```
lr = LinearRegression()
lr.fit(X_multi_trans,z)
```

[32]: LinearRegression()

[33]:
```
X_test_multi = poly.transform(final)
```

[34]:
```
z_final = lr.predict(X_multi_trans).reshape(10,10)
```

[35]:
```
fig = px.scatter_3d(x=x.ravel(), y=y.ravel(), z=z.ravel())

fig.add_trace(go.Surface(x = x_input, y = y_input, z =z_final))

fig.update_layout(scene = dict(zaxis = dict(range=[0,35])))

fig.show()
```