Experiment No.3

Title:Implementation of Polynomial Regression

Part A:Implementation of Polynomial Regression Model on Synthetic data

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Creating and Visualizing Sample Data

```
X=np.array(np.arange(1,20))
Y=X**3+np.random.rand(19)*10

X
```

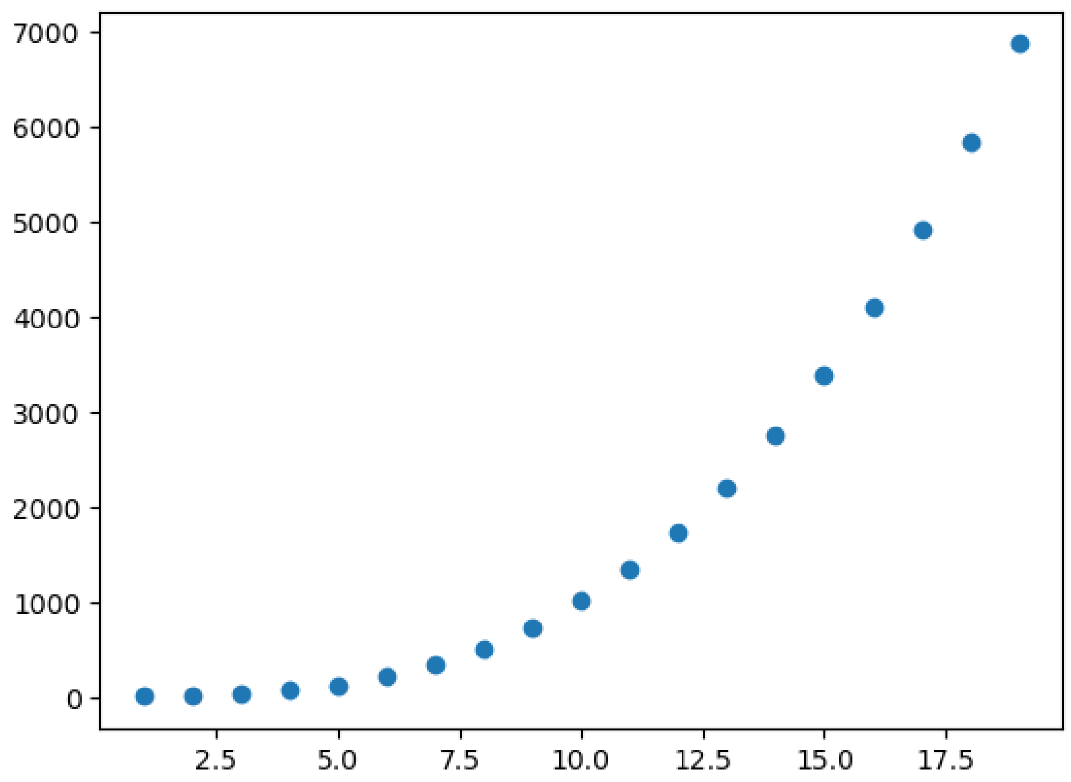
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19])
```

```
Y
```

```
array([   8.75404301,   16.08069724,   34.36633826,   72.08061354,
        125.40025663,  217.18219078,  345.25871733,  512.47385627,
        736.6442064 , 1009.84971831, 1335.62058038, 1734.53885729,
       2204.96372044, 2748.35439452, 3378.97354101, 4101.88486556,
       4922.48194735, 5834.03732789, 6864.68642583])
```

```
plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x7aafbe5d01a0>
```



Polynomial Regression Model

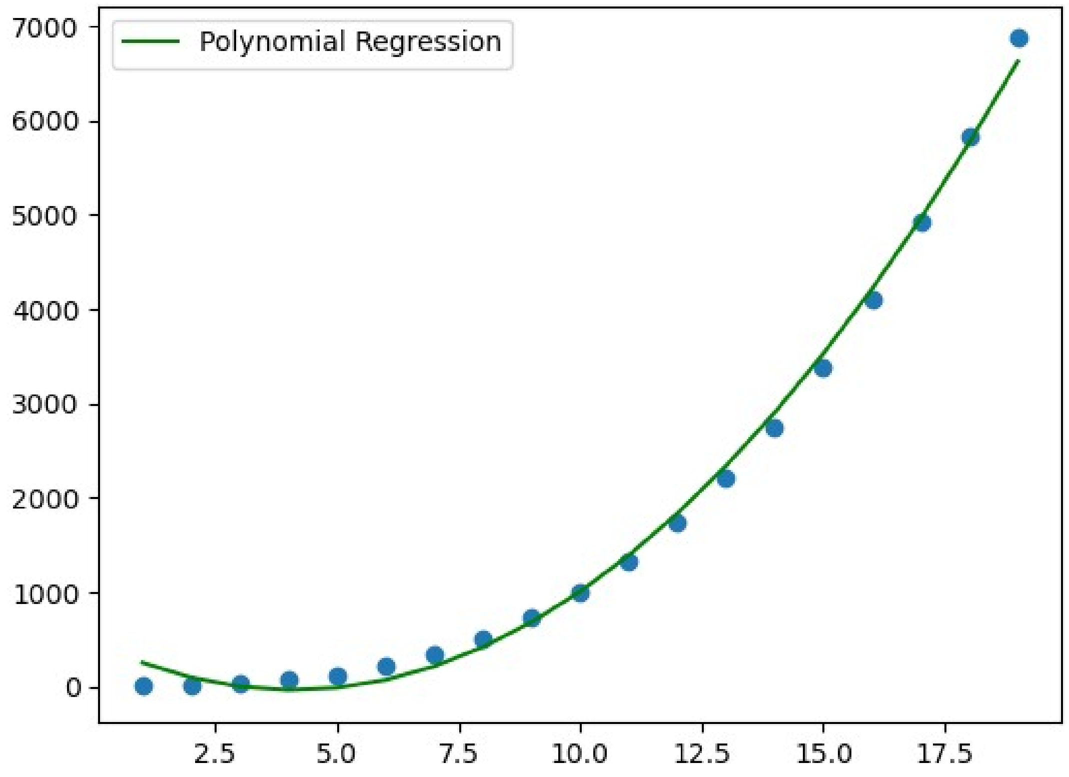
```
# Fit a quadratic polynomial to the data
coefficients = np.polyfit(X, Y, 2)
# Print the coefficients
print("Coefficients (a, b, c):", coefficients)
# Create a polynomial function from the coefficients
polynomial = np.poly1d(coefficients)

Coefficients (a, b, c): [  30.02106986 -246.63638694  469.07911863]
```

Generate predicted values using the polynomial function moddel

```
# Generate predicted values using the polynomial function
Y_pred = polynomial(X)
Y_pred

array([ 2.52463802e+02,  9.58906242e+01, -6.40413428e-01, -
3.71293113e+01,
       -1.35760695e+01,  7.00193121e+01,  2.13656833e+02,
4.17336494e+02,
        6.81058295e+02,  1.00482224e+03,  1.38862832e+03,
1.83247654e+03,
        2.33636690e+03,  2.90029939e+03,  3.52427403e+03,
4.20829081e+03,
        4.95234973e+03,  5.75645079e+03,  6.62059399e+03])
```

Visualizing the Model

```
plt.scatter(X,Y)
plt.plot(X,Y_pred,'Green',label='Polynomial Regression')
plt.legend()
plt.show()
```



Performance Metrics

```
from sklearn.metrics import
mean_absolute_error,mean_squared_error,r2_score
print('Mean Absolute Error :',mean_absolute_error(Y,Y_pred))
print('Mean Squared Error :',mean_squared_error(Y,Y_pred))
print('R2 Value: ',r2_score(Y,Y_pred))
```

```
Mean Absolute Error : 109.40921032360683
Mean Squared Error : 15845.049731937492
R2 Value:  0.996410488067558
```

Part B:Implementation of Polynomail Regression Model on a dataset

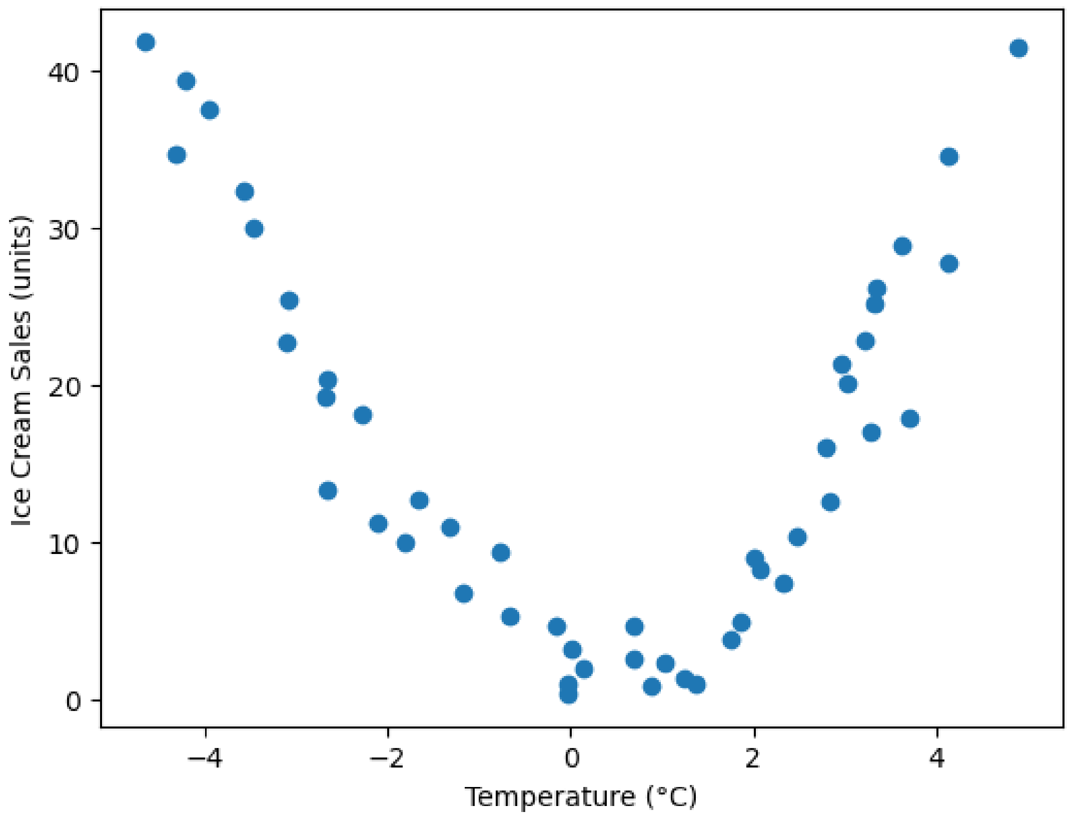
Import dataset

```
df=pd.read_csv('archive/Ice_cream selling data.csv')
df.head()
```

```
   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

df.shape

(49, 2)
```

Visualizing the data

```
plt.scatter(df['Temperature (°C)'],df['Ice Cream Sales (units)'])
plt.xlabel('Temperature (°C)')
plt.ylabel('Ice Cream Sales (units)')

Text(0, 0.5, 'Ice Cream Sales (units)')
```



Separating data into Independent/Features Variable and Dependent/Target Variable

```
X=np.array(df['Temperature (°C)'])
Y=df['Ice Cream Sales (units)']
```

Splitting the data into train and test set

```
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,rando
m_state=42)
```

Polynomial Regression model

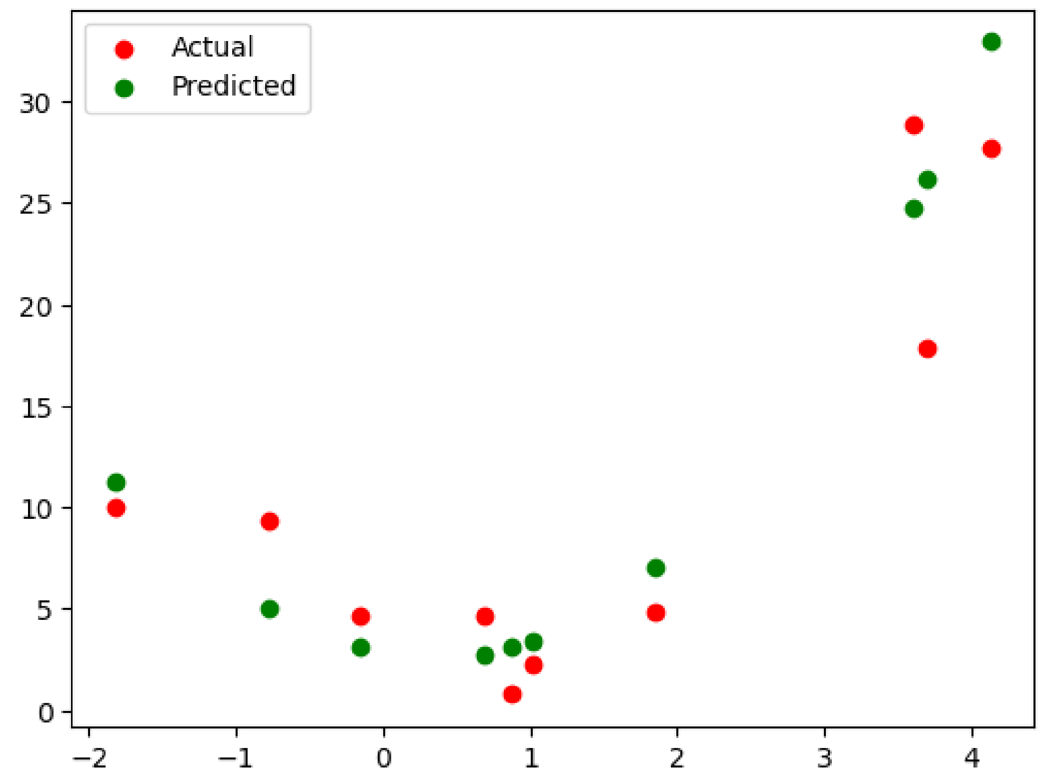
```
# Fit a cubic polynomial to the data
coefficients = np.polyfit(X_train, Y_train, 3)
# Print the coefficients
print("Coefficients (a, b, c, d):", coefficients)
# Create a polynomial function from the coefficients
polynomial = np.poly1d(coefficients)

Coefficients (a, b, c, d): [ 0.05451597  1.87501919 -1.39956426
2.84053099]
```

Generate predicted values using the polynomial function

```
# Generate predicted values using the polynomial function
Y_pred = polynomial(X_test)
Y_pred

array([11.26173782, 26.15229744, 32.94239669, 24.79940325,
5.0189787 ,
        3.43248573,  3.08780469,  2.79001857,  7.01713112,
3.09175474])
```

Visualizing the model on test data

```
# plt.scatter(X_train,Y_train)
plt.scatter(X_test,Y_test,c='red',label='Actual')
plt.scatter(X_test,Y_pred,c='green',label='Predicted')
plt.legend()
plt.show()
```



Performance Metrics

```
from sklearn.metrics import
mean_absolute_error,mean_squared_error,r2_score
print('Mean Absolute Error :',mean_absolute_error(Y_test,Y_pred))
print('Mean Squarred Error :',mean_squared_error(Y_test,Y_pred))
print('R2 Value: ',r2_score(Y_test,Y_pred))

Mean Absolute Error : 3.2281211297768864
Mean Squarred Error : 15.120009451229418
R2 Value:  0.8405107685716922
```

Conclusion:

Merits and Demerits of Polynomial Regression

Merits

1. **Flexibility**:
   – Polynomial regression can model complex relationships between independent and dependent variables by fitting higher-degree polynomials, allowing for curvature in the relationship.
2. **Improved Fit for Non-linear Data**:
   – It performs better than linear regression when the underlying relationship is non-linear, as it can capture patterns that linear models cannot.
3. **Easy Interpretation of Coefficients**:

   - Each coefficient in the polynomial can still be interpreted in terms of its effect on the dependent variable, similar to linear regression.
4. **Works Well with Small Datasets**:
   - Polynomial regression can provide reasonable fits with smaller datasets when higher-order terms are included, as it can model non-linearity without needing large amounts of data.
5. **Extensive Applicability**:
   - It is applicable in various fields such as finance, biology, and engineering, where relationships between variables are often non-linear.

Demerits

1. **Overfitting**:
   - Higher-degree polynomials can fit the training data very well, but they may fail to generalize to unseen data, leading to poor predictive performance. This is especially true in the presence of noise.
2. **Increased Complexity**:
   - The model becomes more complex with higher degrees, making it harder to interpret and understand the influence of individual features.
3. **Extrapolation Issues**:
   - Polynomial regression can yield extreme predictions outside the range of the training data, especially with higher-degree polynomials, leading to unreliable results.
4. **Sensitive to Outliers**:
   - Polynomial regression can be heavily influenced by outliers, which may skew the fit and result in misleading interpretations.
5. **Requires Feature Engineering**:
   - Polynomial regression necessitates the creation of polynomial features (e.g., (x^2, x^3)) from the original variables, increasing the dimensionality of the feature space and potentially leading to multicollinearity issues.