Day46_Polynomial_Regression_Deployment

July 16, 2025

Today we are learning about **Polynomial Regression**

In this notebook, we will:

- Recap Machine Learning types
- Understand where Polynomial Regression fits
- Learn why and when to use it
- Explore the concept of **degree** in polynomial models
- Build and compare models with degrees from 1 to 7
- Predict salary for unknown levels like 6
- Plot results and analyze which degree fits best

This is especially helpful when your data doesn't follow a straight line pattern — like **salary vs job level**, where growth increases sharply at higher levels.

Recap:

What are the Types of Machine Learning?

Type of ML	Description	Examples
Supervised Learning	Model learns from labeled data	Regression, Classification
Unsupervised Learning	Model finds patterns from unlabeled data	Clustering, Dimensionality Reduction
Reinforcement Learning	Model learns through rewards & penalties	Games, Robotics, Self-driving cars

Where Does Polynomial Regression Fit?

- Polynomial Regression is a part of **Supervised Learning**
- It's used for **Regression tasks**
- It's an **extension of Linear Regression** for **non-linear relationships**

Why Use Polynomial Regression?

We have a dataset of positions and salaries:

Position	Level	Salary
Jr Software Engineer	1	45000
Sr Software Engineer	2	50000
Manager	4	80000

Position	Level	Salary
...	...	...
CEO	10	1000000

Salary increases **non-linearly** with level. If someone comes in at **Level 4.5 or 6.2**, we want to **predict their salary**. That's where **Polynomial Regression** helps — it captures the **curve** in data.

What is Degree in Polynomial Regression?

In **Polynomial Regression**, the model uses not just the input feature x, but also its higher powers like x^2, x^3, ..., up to x^d, where d **is the degree** of the polynomial.

The **degree** decides the **complexity of the curve** the model can fit:

Examples:

- **Degree 1** → fits a **straight line**:
  $$y = b_0 + b_1 x$$
  → Same as **Simple Linear Regression**
- **Degree 2** → fits a **parabolic curve**:
  $$y = b_0 + b_1 x + b_2 x^2$$
  → Can capture basic curvature (U-shape or inverted U)
- **Degree 3** → fits a **cubic curve**:
  $$y = b_0 + b_1 x + b_2 x^2 + b_3 x^3$$
  → Can bend more than once
- **Higher Degrees (4, 5, 6...)** → fit more complex curves with more wiggles

Why Is Degree Important?

- A **low degree** may **underfit** the data (too simple, can't capture real trends)
- A **high degree** may **overfit** the data (fits noise or small fluctuations too much)
- The **goal** is to find a degree that fits the real pattern **without overfitting**

Think of it like adjusting the flexibility of the curve — more degree = more flexibility, but too much = messy predictions

Step-by-Step Polynomial Regression

1 Import Libraries

- pandas: Reading and managing data
- numpy: Numerical operations
- matplotlib: Plotting graphs
- sklearn: Creating and training machine learning models

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.linear_model import LinearRegression
     from sklearn.preprocessing import PolynomialFeatures
```

2 Load the Dataset

```
[3]: dataset = pd.read_csv(r"C:\Users\Lenovo\Downloads\emp_sal.csv")
     dataset.head()
```

```
[3]:               Position  Level  Salary
     0  Jr Software Engineer      1   45000
     1  Sr Software Engineer      2   50000
     2             Team Lead      3   60000
     3               Manager      4   80000
     4            Sr manager      5  110000
```

3 Define Features and Labels

- X = Level (independent variable)
- y = Salary (dependent variable)

```
[4]: X = dataset.iloc[:, 1:2].values  # Level
     y = dataset.iloc[:, 2].values    # Salary
```

4 Train a Linear Regression Model (Degree 1)

```
[5]: lin_reg = LinearRegression()
     lin_reg.fit(X, y)
```

```
[5]: LinearRegression()
```

4.1 Predict Salary for Level 6

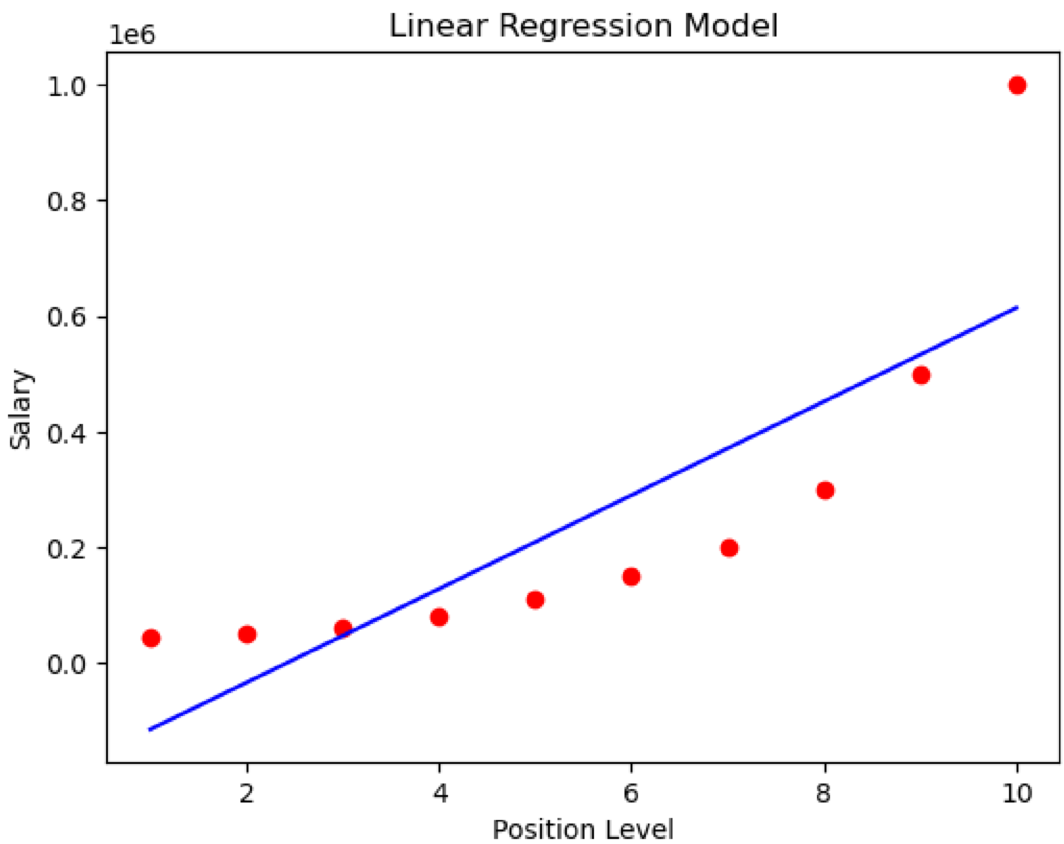
```
[6]: print(lin_reg.predict([[6]]))
```

```
[289939.39393939]
```

4.2 Plot Linear Model

```
[7]: plt.scatter(X, y, color='red')  # Real data
     plt.plot(X, lin_reg.predict(X), color='blue')  # Predicted line
     plt.title("Linear Regression Model")
     plt.xlabel("Position Level")
     plt.ylabel("Salary")
```

```
plt.show()
```



4.3 Interpretation:

- This model underestimates salary for Level 6.
- It assumes a straight line, but salaries grow non-linearly in reality.

Lets create Polynomial Regression model to get accurate result (Degrees 2 to 7)

> For each degree: - Create polynomial features - Train model - Predict Level 6 - Plot model - Interpret result

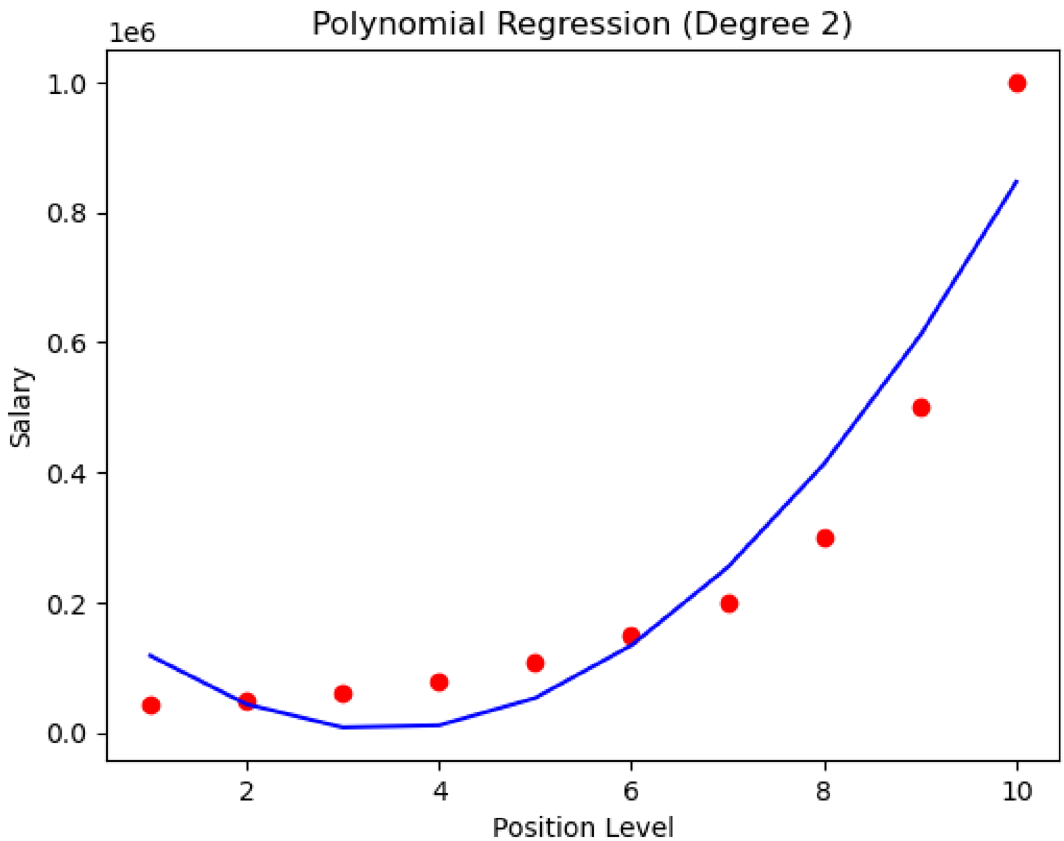
5 Polynomial Regression – Degree 2

[9]:
```
poly_reg2 = PolynomialFeatures(degree=2)
X_poly2 = poly_reg2.fit_transform(X)

lin_reg2 = LinearRegression()
lin_reg2.fit(X_poly2, y)
```

```
print(lin_reg2.predict(poly_reg2.transform([[6]])))
```

```
[134484.84848485]
```

[10]:
```
plt.scatter(X, y, color='red')
plt.plot(X, lin_reg2.predict(X_poly2), color='blue')
plt.title("Polynomial Regression (Degree 2)")
plt.xlabel("Position Level")
plt.ylabel("Salary")
plt.show()
```



5.1 Interpretation:

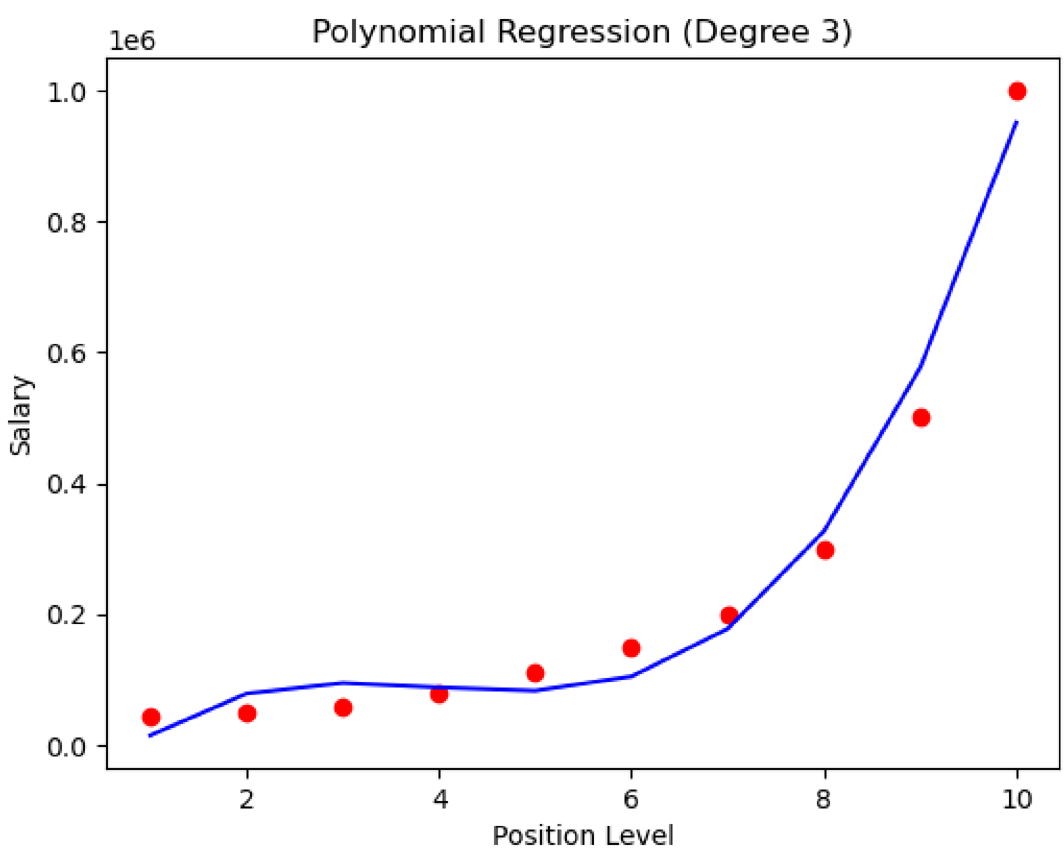
- Prediction: ~134,485
- Better than linear, but still underpredicting.

6 Polynomial Regression – Degree 3

[11]:
```
poly_reg3 = PolynomialFeatures(degree=3)
X_poly3 = poly_reg3.fit_transform(X)

lin_reg3 = LinearRegression()
lin_reg3.fit(X_poly3, y)

print(lin_reg3.predict(poly_reg3.transform([[6]])))
```

```
[104820.51282051]
```

[16]:
```
plt.scatter(X, y, color='red')
plt.plot(X, lin_reg3.predict(X_poly3), color='blue')
plt.title("Polynomial Regression (Degree 3)")
plt.xlabel("Position Level")
plt.ylabel("Salary")
plt.show()
```



6.1 Interpretation:

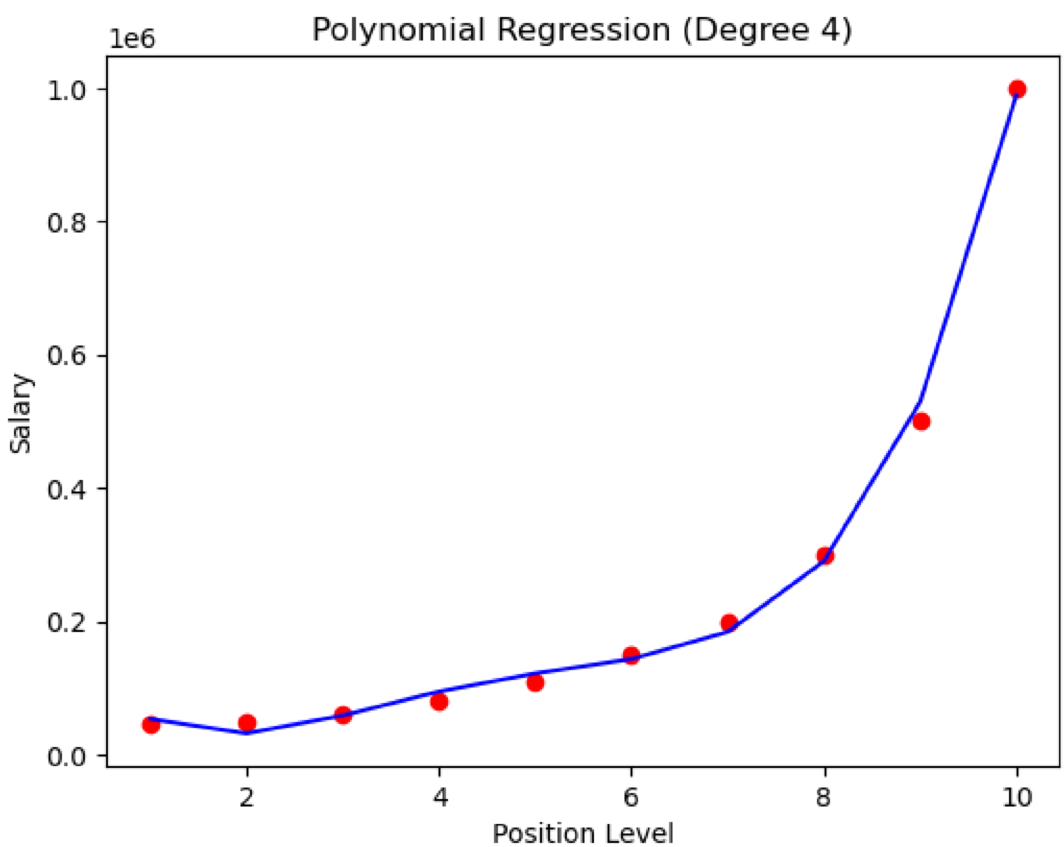
- Worse than degree 2. Slight underfit.

7 Polynomial Regression – Degree 4

[12]:
```
poly_reg4 = PolynomialFeatures(degree=4)
X_poly4 = poly_reg4.fit_transform(X)

lin_reg4 = LinearRegression()
lin_reg4.fit(X_poly4, y)

print(lin_reg4.predict(poly_reg4.transform([[6]])))
```

```
[143275.05827508]
```

[17]:
```
plt.scatter(X, y, color='red')
plt.plot(X, lin_reg4.predict(X_poly4), color='blue')
plt.title("Polynomial Regression (Degree 4)")
plt.xlabel("Position Level")
plt.ylabel("Salary")
plt.show()
```



7.1 Interpretation:

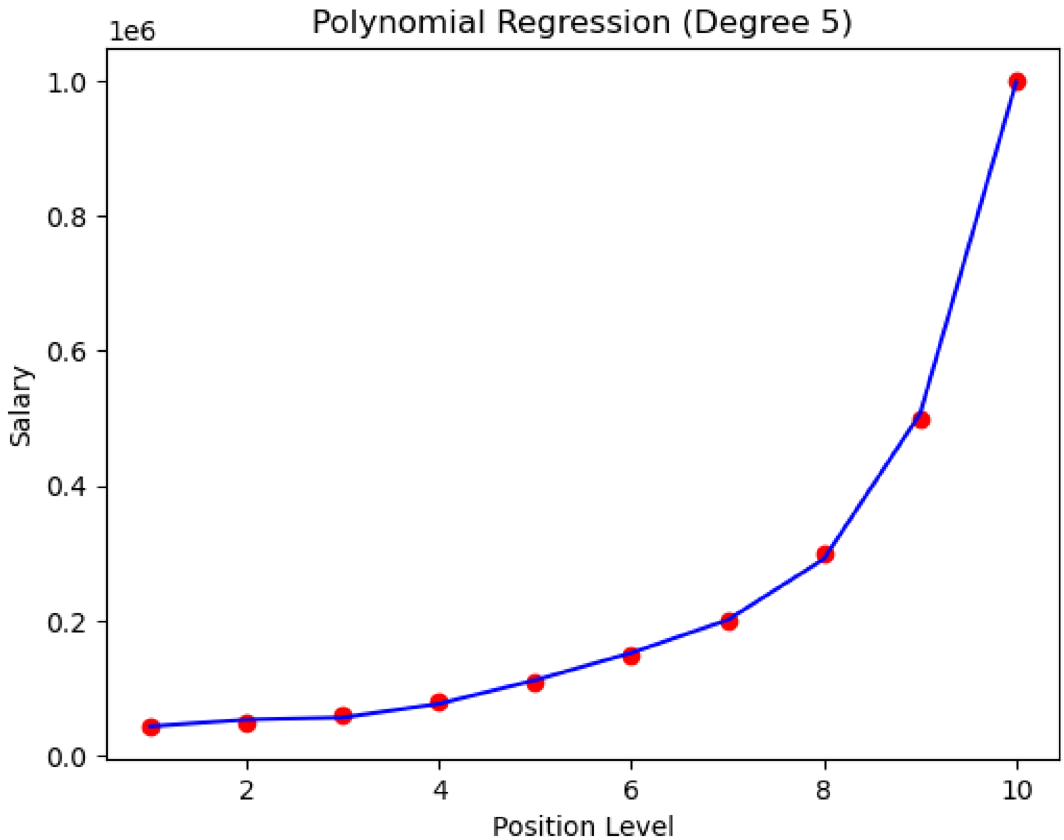
- Much closer to real salary (~150,000).

8 Polynomial Regression – Degree 5

[13]:
```
poly_reg5 = PolynomialFeatures(degree=5)
X_poly5 = poly_reg5.fit_transform(X)

lin_reg5 = LinearRegression()
lin_reg5.fit(X_poly5, y)

print(lin_reg5.predict(poly_reg5.transform([[6]])))
```

```
[152736.59673623]
```

[18]:
```
plt.scatter(X, y, color='red')
plt.plot(X, lin_reg5.predict(X_poly5), color='blue')
plt.title("Polynomial Regression (Degree 5)")
plt.xlabel("Position Level")
plt.ylabel("Salary")
plt.show()
```



8.1 Interpretation:

- Very accurate! Near the actual salary.

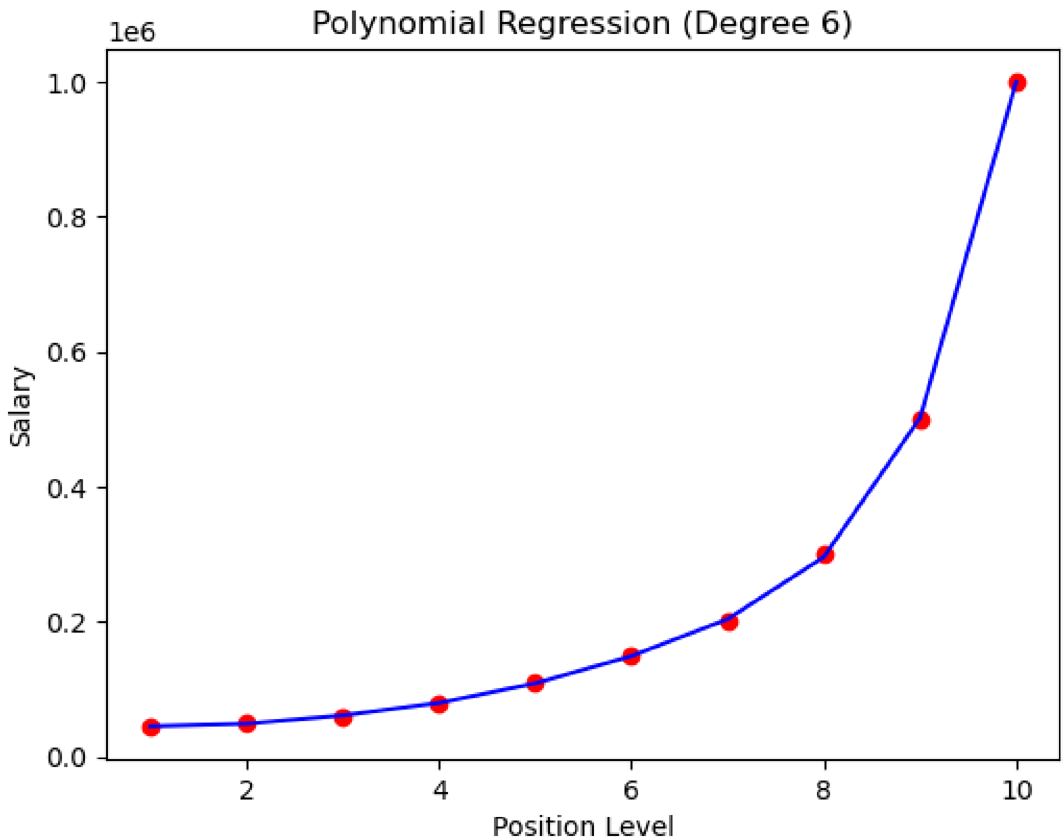
9 Polynomial Regression – Degree 6

[14]:
```
poly_reg6 = PolynomialFeatures(degree=6)
X_poly6 = poly_reg6.fit_transform(X)

lin_reg6 = LinearRegression()
lin_reg6.fit(X_poly6, y)

print(lin_reg6.predict(poly_reg6.transform([[6]])))
```

```
[149282.05128119]
```

[19]:
```
plt.scatter(X, y, color='red')
plt.plot(X, lin_reg6.predict(X_poly6), color='blue')
plt.title("Polynomial Regression (Degree 6)")
plt.xlabel("Position Level")
```

```
plt.ylabel("Salary")
plt.show()
```



9.1 Interpretation:

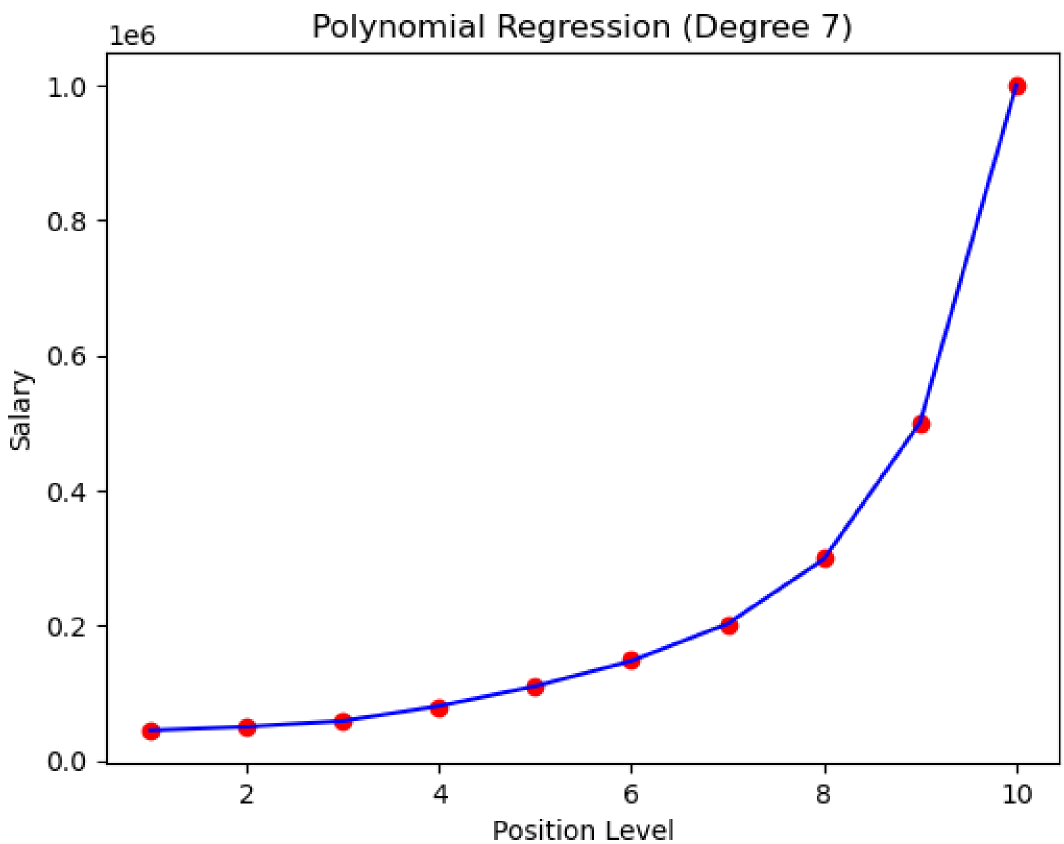
- Still accurate. Curve fits well.

10 Polynomial Regression – Degree 7

[15]:
```
poly_reg7 = PolynomialFeatures(degree=7)
X_poly7 = poly_reg7.fit_transform(X)

lin_reg7 = LinearRegression()
lin_reg7.fit(X_poly7, y)

print(lin_reg7.predict(poly_reg7.transform([[6]])))
```

```
[147736.73383819]
```

```
[20]: plt.scatter(X, y, color='red')
      plt.plot(X, lin_reg7.predict(X_poly7), color='blue')
      plt.title("Polynomial Regression (Degree 7)")
      plt.xlabel("Position Level")
      plt.ylabel("Salary")
      plt.show()
```



10.1 Interpretation:

- Still good, but we are now risking overfitting.

11 Summary Table

This table shows the predicted salary for **Level 6** using each degree of polynomial regression. We are comparing the values to find which model fits best.

= Close to real-world expected salary (around 150,000)

Degree	Predicted Salary for Level 6	Comment
1	~289,939	Too high (overpredicting)
2	~134,485	Slightly under
3	~104,821	Underpredicts
4	~143,275	Very close
5	~152,736	Best match
6	~149,282	Very good
7	~147,736	Still good

Final Thoughts

- Start with Linear Regression (Degree 1)
- Switch to Polynomial Regression for curves
- Try degrees 2 to 7 to find best fit
- Degree 5 or 6 gave most accurate prediction for Level 6
- Watch out for overfitting at high degrees

---

12 Taking It Further: Model Deployment Steps

Once your model is trained and tested, you can deploy it as a frontend app using Streamlit for real-time prediction.

12.1 Save Polynomial Regression Model (e.g., Degree 5)

Save the trained Degree 5 polynomial regression model and the transformer.

[21]:
```
import pickle

# Save both the polynomial feature transformer and the model
with open("poly_transformer.pkl", "wb") as f1:
    pickle.dump(poly_reg5, f1)

with open("poly_model.pkl", "wb") as f2:
    pickle.dump(lin_reg5, f2)

print("Polynomial model and transformer saved successfully!")
```

```
Polynomial model and transformer saved successfully!
```

12.1.1 Confirm Save Location

[22]:
```
import os
print("Saved in directory:", os.getcwd())
```

```
Saved in directory: C:\Users\Lenovo\OneDrive\Desktop\Python Everyday work\Github
work
```

12.2 Create a Streamlit Web App

Save the following code in a file named: poly_app.py

```python
import streamlit as st
import pickle
import numpy as np

# Load the saved transformer and model
with open("poly_transformer.pkl", "rb") as f1:
    poly_transformer = pickle.load(f1)

with open("poly_model.pkl", "rb") as f2:
    poly_model = pickle.load(f2)

# Streamlit UI
st.markdown("<h1 style='color:#4b8bff;'> Polynomial Regression Salary Predictor</h1>", unsafe_a
st.write("Enter the Position Level to predict the corresponding salary using a trained Polynomi

level = st.slider("Select Position Level", min_value=1.0, max_value=10.0, step=0.1)

if st.button(" Predict Salary"):
    input_array = np.array([[level]])
    transformed_input = poly_transformer.transform(input_array)
    prediction = poly_model.predict(transformed_input)
    st.success(f"Predicted Salary for Level {level}:  {prediction[0]:,.2f}")
```

12.3 Run the Streamlit App

In your terminal or CMD, run:

```
streamlit run poly_app.py
```

13 Conclusion

13.1 What We Achieved:

- Trained and evaluated Polynomial Regression models (degrees 1 to 7)
- Found best-fitting model using visualization and prediction
- Saved model and transformer using `pickle`
- Built a Streamlit app to predict salary interactively

13.2 Key Learnings:

- Polynomial Regression handles non-linear relationships effectively
- Model persistence (`pickle`) enables reuse without retraining
- Streamlit turns ML models into user-friendly tools

13.3 Next Steps:

- Add more features like job title, years of experience
- Try different models: Random Forest, XGBoost
- Host your app using Streamlit Cloud or Hugging Face Spaces

We have a real deployable ML project!