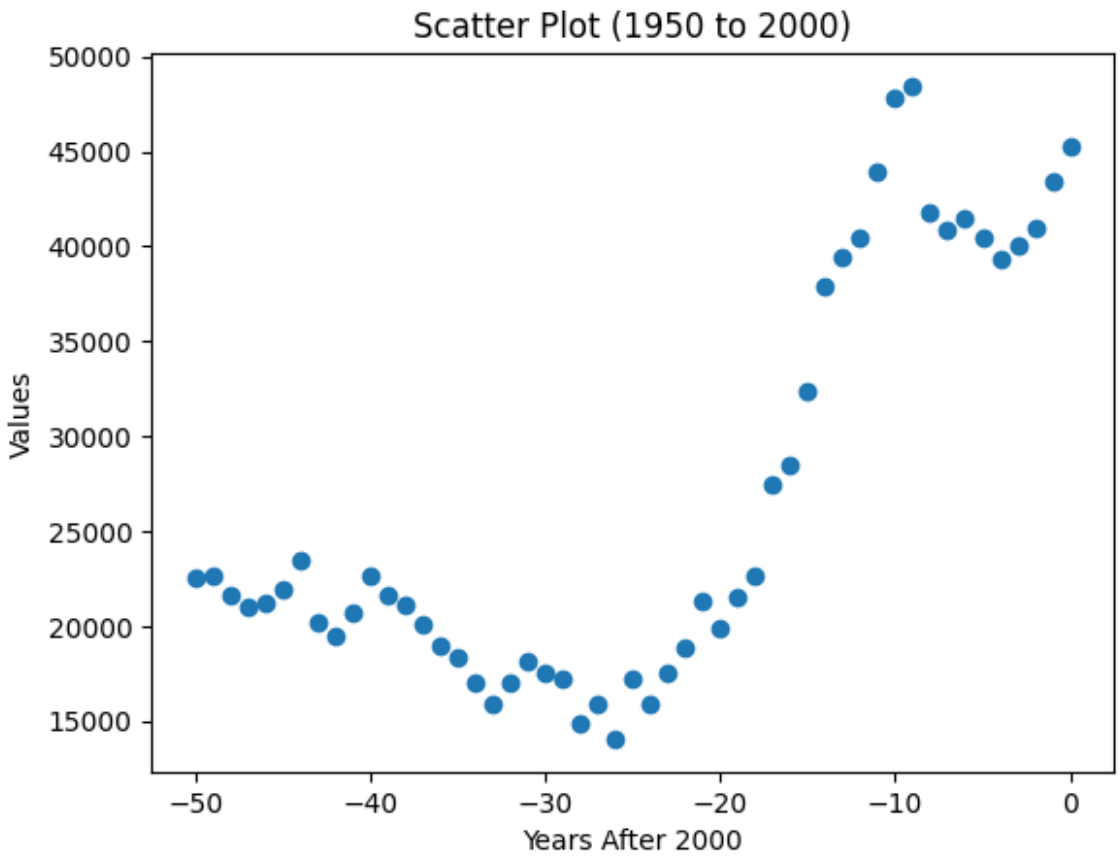
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn import metrics

lob = pd.read_csv('Lobsters.csv')
#print(lob.head())
lob['yearAfter2000'] = lob['year'] - 2000
#print(lob)
t = lob['year']
tlist = t.tolist()
i = tlist.index(2000)
print("2000 in the list: ",i)
lob_train = lob.loc[ t <= 2000 ]
lob_test = lob.loc[ t > 2000 ]
print("lesser than or equal to 2000 in the list: ",len(lob_train))
print("greater than 2000 in the list: ",len(lob_test))
del t
#print(t) - it gives an error because we are trying to print "t" which
has already been deleted
x = lob['yearAfter2000']
x_train = lob_train['yearAfter2000']
y_train = lob_train['tonnes']
x_test = lob_test['yearAfter2000']
y_test = lob_test['tonnes']

X = x.to_numpy().reshape(-1, 1)
X_train = x_train.values.reshape(-1, 1)
X_test = x_test.values.reshape(-1, 1)
```

```
2000 in the list:  50
lesser than or equal to 2000 in the list:  51
greater than 2000 in the list:  18
```

Question: Explain why this value is consistent with the value you obtained for the variable i in the previous step above

**Answer** : The number 50 in the output indicates that the year 2000 is at position 50 in the original list. Now, the length of 'lob_train' is 51, which means it includes all years up to 2000, consistent with the position of 2000. Also, the length of 'lob_test' is 18, representing years after 2000. So, the values are consistent with the position of 2000 in the list.

```
plt.scatter(x_train,y_train)
plt.xlabel('Years After 2000')
plt.ylabel('Values')
plt.title('Scatter Plot (1950 to 2000)')
plt.show()
```




```
#Linear regression with trained data
deg = 1
linear =  LinearRegression()
linear.fit( X_train, y_train )
linear.score( X_train, y_train )
```

```
0.5791322161721189
```

```
deg=2
X_train_poly = PolynomialFeatures(deg).fit_transform(X_train)
print(x_train[0:3])
print(X_train_poly[0:3,:])
```

```
0   -50
1   -49
2   -48
Name: yearAfter2000, dtype: int64
[[ 1.000e+00 -5.000e+01  2.500e+03]
 [ 1.000e+00 -4.900e+01  2.401e+03]
 [ 1.000e+00 -4.800e+01  2.304e+03]]
```

```
model_poly = LinearRegression()
model_poly.fit( X_train_poly, y_train )
model_poly.score( X_train_poly,y_train )
```

```
0.8104938621639016
```

So when the Degree is 2 then the R^2 training data value would be 0.8104938621639016

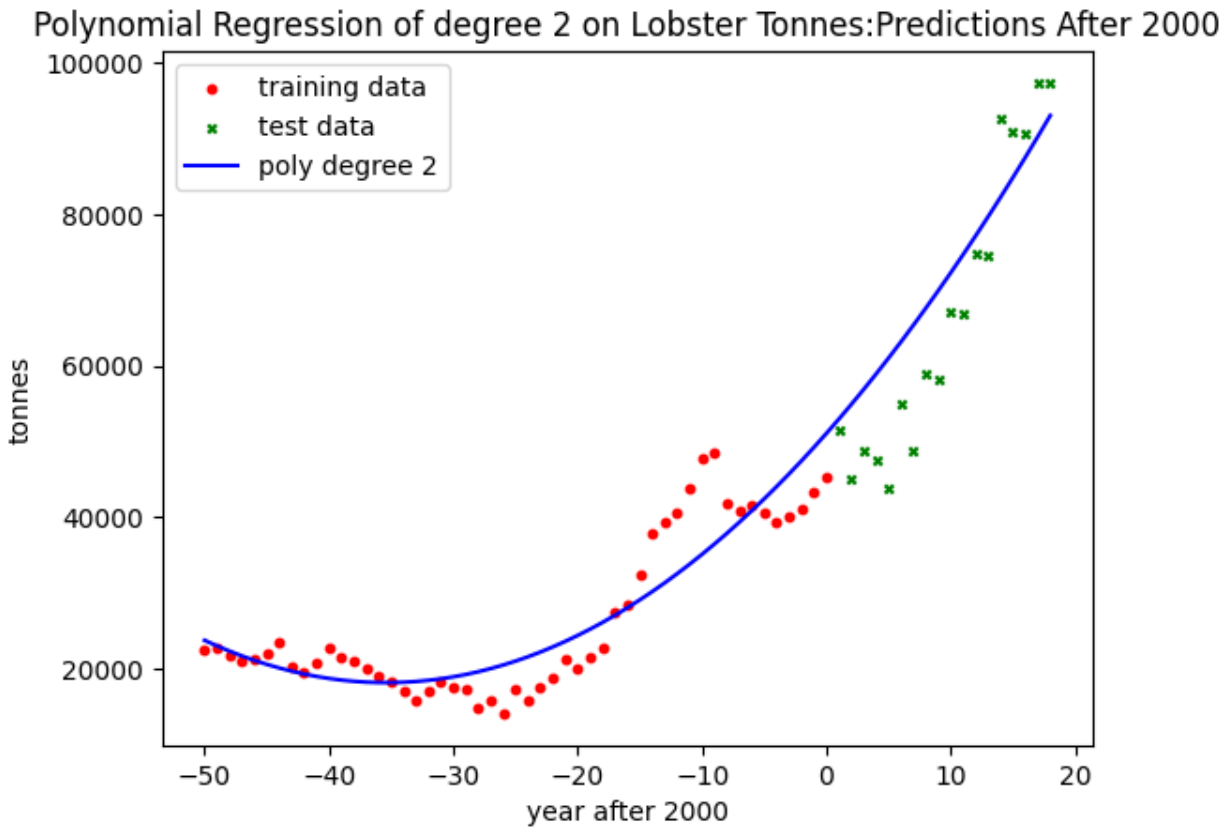
```
metrics.r2_score(y_train,model_poly.predict(X_train_poly))

0.8104938621639016
```

the value of R^2 is 0.8104938621639016 from the Alternative way to predict the R^2 value

```
X_test_poly = PolynomialFeatures(deg).fit_transform(X_test)
y_test_pred = model_poly.predict(X_test_poly)
r2_score_test = metrics.r2_score(y_test,y_test_pred)
print("r2 score test value: ",r2_score_test)

r2 score test value:  0.7654356325269248
```

So when the Degree is 2 then the R^2 test data value would be 0.7654356325269248 using sklearn

```
X_poly = PolynomialFeatures(deg).fit_transform(X)
y_pred = model_poly.predict(X_poly)
plt.scatter(x_train,y_train,10,color='r',marker='o',label='training
data')
plt.scatter(x_test ,y_test ,10,color='g',marker='x',label='test data')
plt.plot(x,y_pred,color='b',label='poly degree '+str(deg))
plt.legend(loc='upper left')
plt.xlabel('year after 2000')
plt.ylabel('tonnes')
plt.title("Polynomial Regression of degree 2 on Lobster
Tonnes:Predictions After 2000")

Text(0.5, 1.0, 'Polynomial Regression of degree 2 on Lobster
Tonnes:Predictions After 2000')
```




```
model_poly = np.poly1d(np.polyfit(x_train,y_train,deg))
y_train_pred = model_poly(x_train)
y_pred = model_poly(x)
metrics.r2_score(y_test,y_test_pred)
```

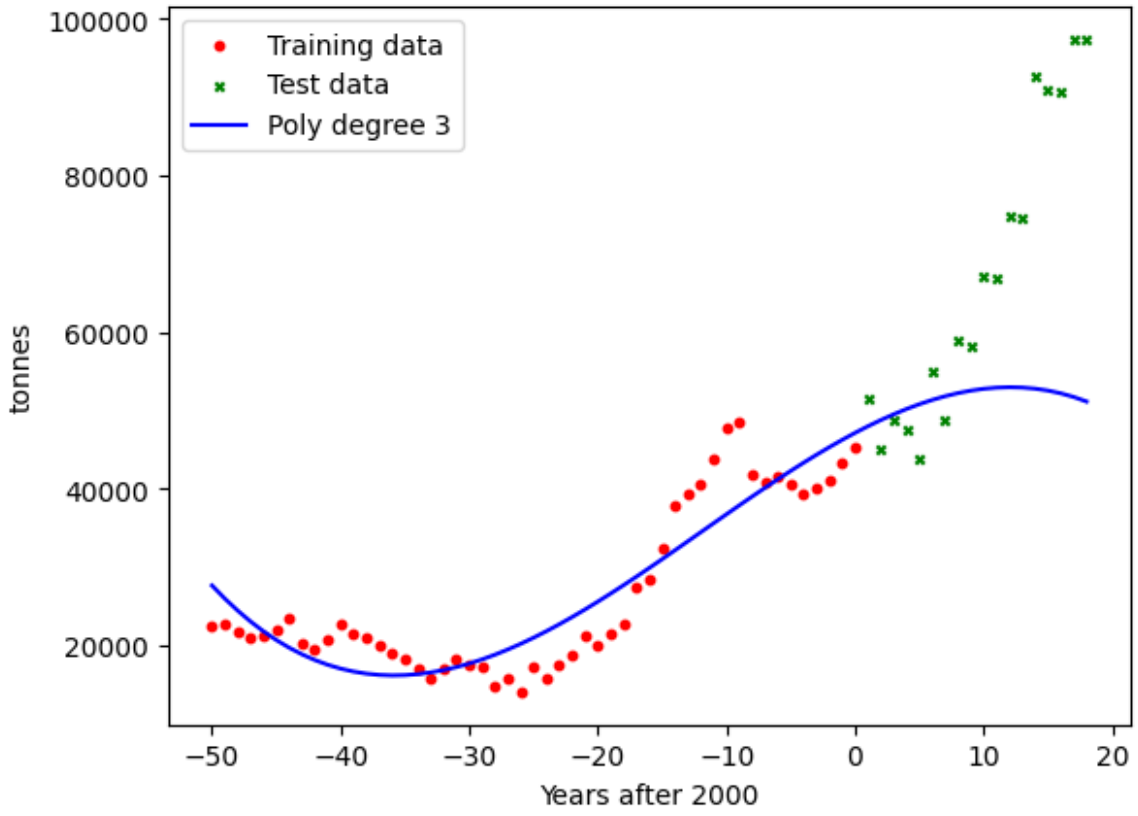
```
0.7654356325269248
```

The output of the r2 test value using numpy is 0.7654356325269248 which is the same as the r2 test value using sklearn.

```
# Change the polynomial degree to 3
deg = 3
X_train_poly = PolynomialFeatures(deg).fit_transform(X_train)
model_poly = LinearRegression()
model_poly.fit(X_train_poly, y_train)
r2_train = model_poly.score(X_train_poly, y_train)
print('R2 score for training data (degree 3): ',r2_train)
r2_train_alternative = metrics.r2_score(y_train,
model_poly.predict(X_train_poly))
print('Alternative R2 score for training data (degree 3):
',r2_train_alternative)
X_test_poly = PolynomialFeatures(deg).fit_transform(X_test)
y_test_pred = model_poly.predict(X_test_poly)
```

```
r2_test = metrics.r2_score(y_test, y_test_pred)
print('R2 score for test data (degree 3): ',r2_test)

# Plotting for degree 3
X_poly = PolynomialFeatures(deg).fit_transform(X)
y_pred = model_poly.predict(X_poly)
plt.scatter(x_train, y_train, 10, color='r', marker='o',
label='Training data')
plt.scatter(x_test, y_test, 10, color='g', marker='x', label='Test
data')
plt.plot(x, y_pred, color='b', label=f'Poly degree {deg}')
plt.legend(loc='upper left')
plt.xlabel('Years after 2000')
plt.ylabel('tonnes')
plt.show()
```

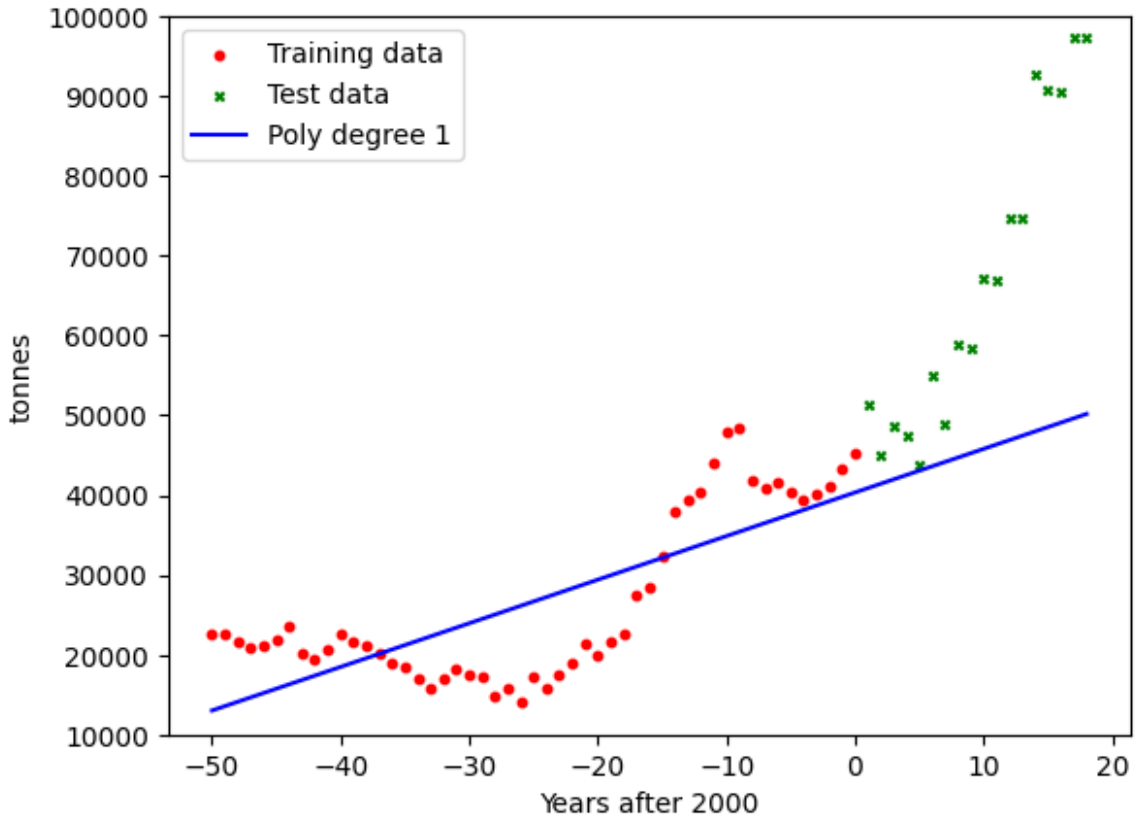
```
R2 score for training data (degree 3):  0.8354414398563609
Alternative R2 score for training data (degree 3):  0.8354414398563609
R2 score for test data (degree 3):  -0.6223872591089696
```




```
# Change the polynomial degree to 1
deg = 1
X_train_poly = PolynomialFeatures(deg).fit_transform(X_train)
model_poly = LinearRegression()
```

```
model_poly.fit(X_train_poly, y_train)
#r2_train = model_poly.score(X_train_poly, y_train)
#print('R2 score for training data (degree 1): ',r2_train)
r2_train_alternative = metrics.r2_score(y_train,
model_poly.predict(X_train_poly))
print('Alternative R2 score for training data (degree 1):
',r2_train_alternative)
X_test_poly = PolynomialFeatures(deg).fit_transform(X_test)
y_test_pred = model_poly.predict(X_test_poly)
r2_test = metrics.r2_score(y_test, y_test_pred)
print('R2 score for test data (degree 1): ',r2_test)

# Plotting for degree 1
X_poly = PolynomialFeatures(deg).fit_transform(X)
y_pred = model_poly.predict(X_poly)
plt.scatter(x_train, y_train, 10, color='r', marker='o',
label='Training data')
plt.scatter(x_test, y_test, 10, color='g', marker='x', label='Test
data')
plt.plot(x, y_pred, color='b', label=f'Poly degree {deg}')
plt.ylim(10000, 100000)
plt.legend(loc='upper left')
plt.xlabel('Years after 2000')
plt.ylabel('tonnes')
plt.show()
```

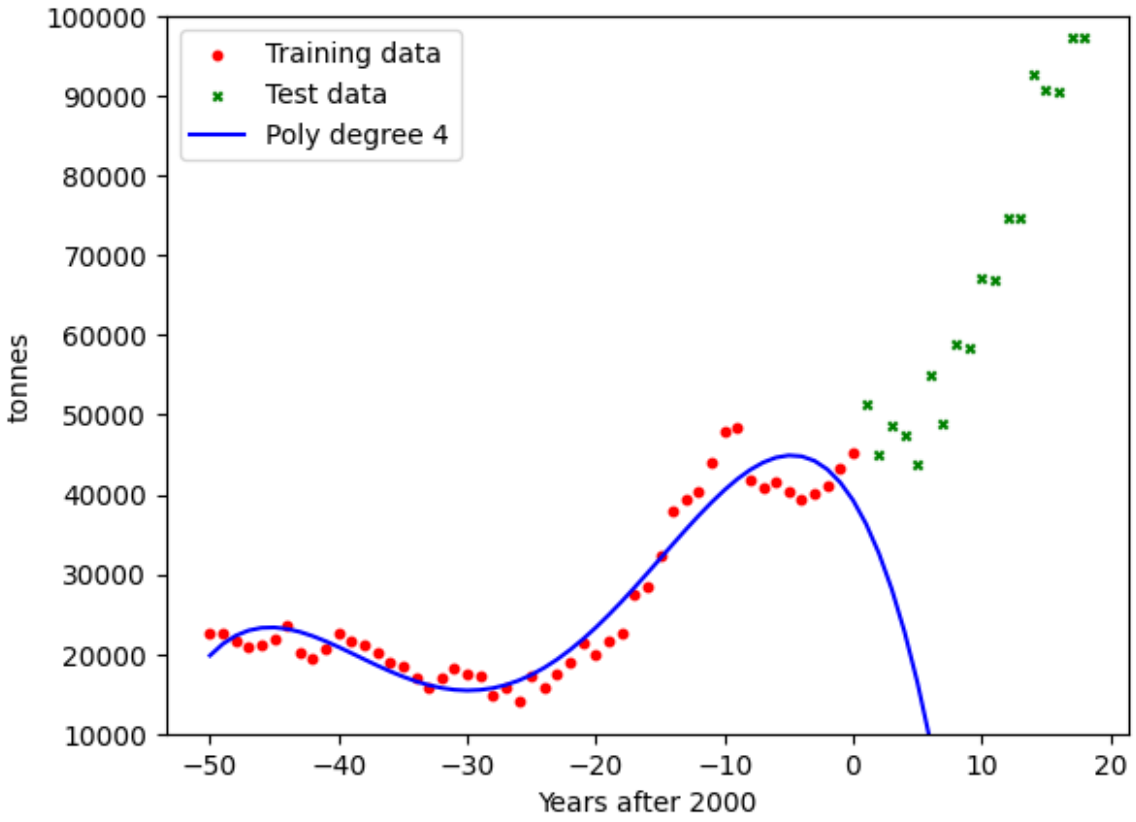
```
Alternative R2 score for training data (degree 1):  0.5791322161721189
R2 score for test data (degree 1):  -1.0691716885659526
```




```
# Change the polynomial degree to 4
deg = 4
X_train_poly = PolynomialFeatures(deg).fit_transform(X_train)
model_poly = LinearRegression()
model_poly.fit(X_train_poly, y_train)
#r2_train = model_poly.score(X_train_poly, y_train)
#print('R2 score for training data (degree 4): ',r2_train)
r2_train_alternative = metrics.r2_score(y_train,
model_poly.predict(X_train_poly))
print('Alternative R2 score for training data (degree 4):
',r2_train_alternative)
X_test_poly = PolynomialFeatures(deg).fit_transform(X_test)
y_test_pred = model_poly.predict(X_test_poly)
r2_test = metrics.r2_score(y_test, y_test_pred)
print('R2 score for test data (degree 4): ',r2_test)

# Plotting for degree 4
X_poly = PolynomialFeatures(deg).fit_transform(X)
y_pred = model_poly.predict(X_poly)
plt.scatter(x_train, y_train, 10, color='r', marker='o',
label='Training data')
plt.scatter(x_test, y_test, 10, color='g', marker='x', label='Test
data')
plt.plot(x, y_pred, color='b', label=f'Poly degree {deg}')
```

```
plt.ylim(10000, 100000)
plt.legend(loc='upper left')
plt.xlabel('Years after 2000')
plt.ylabel('tonnes')
plt.show()
```

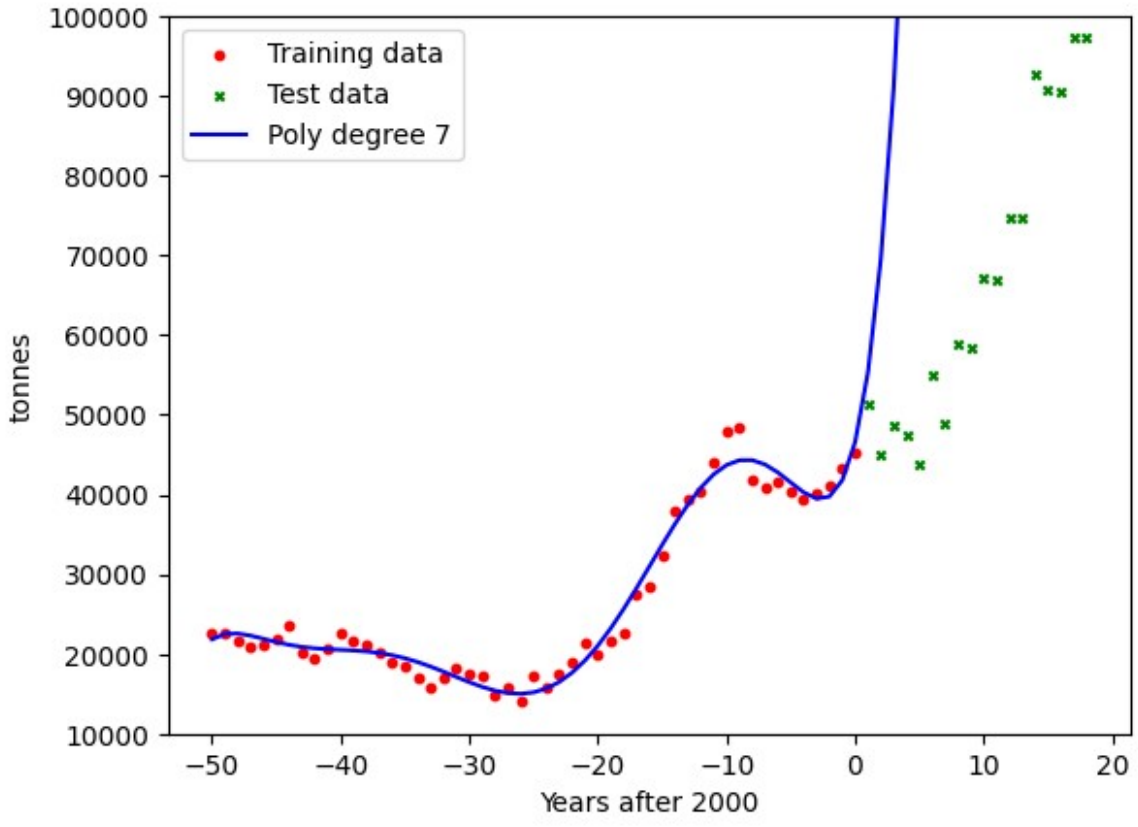
```
Alternative R2 score for training data (degree 4):  0.9262859619872615
R2 score for test data (degree 4):  -56.59525926583845
```




```
# Change the polynomial degree to 7
deg = 7
X_train_poly = PolynomialFeatures(deg).fit_transform(X_train)
model_poly = LinearRegression()
model_poly.fit(X_train_poly, y_train)
#r2_train = model_poly.score(X_train_poly, y_train)
#print('R2 score for training data (degree 7): ',r2_train)
r2_train_alternative = metrics.r2_score(y_train,
model_poly.predict(X_train_poly))
print('Alternative R2 score for training data (degree 7):
',r2_train_alternative)
X_test_poly = PolynomialFeatures(deg).fit_transform(X_test)
y_test_pred = model_poly.predict(X_test_poly)
r2_test = metrics.r2_score(y_test, y_test_pred)
```

```
print('R2 score for test data (degree 7): ',r2_test)

# Plotting for degree 7
X_poly = PolynomialFeatures(deg).fit_transform(X)
y_pred = model_poly.predict(X_poly)
plt.scatter(x_train, y_train, 10, color='r', marker='o',
label='Training data')
plt.scatter(x_test, y_test, 10, color='g', marker='x', label='Test
data')
plt.plot(x, y_pred, color='b', label=f'Poly degree {deg}')
plt.ylim(10000, 100000)
plt.legend(loc='upper left')
plt.xlabel('Years after 2000')
plt.ylabel('tonnes')
plt.show()
```

```
Alternative R2 score for training data (degree 7):  0.9765628970633865
R2 score for test data (degree 7):  -6996.5992974478495
```




R2 scores of training and testing data on degree 1:

- R2 score for training data (degree 1): 0.5791322161721189
- R2 score for test data (degree 1): -1.0691716885659526

R2 scores of training and testing data on degree 4:

- Alternative R2 score for training data (degree 4): 0.9262859619872615
- R2 score for test data (degree 4): -56.59525926583845

R2 scores of training and testing data on degree 7:

- Alternative R2 score for training data (degree 7): 0.9765628970633865
- R2 score for test data (degree 7): -6996.5992974478495

22) a)The degree-7 polynomial achieves the highest R2 score for training data but poorly generalizes to the test set. Conversely, the degree-2 polynomial shows the best generalization with a high R2 score for test data.The plots visually confirm that higher-degree polynomials overfit the training data, emphasizing the importance of selecting a degree that strikes a balance between complexity and generalization.

b)In data modeling and prediction, guarding against overfitting and underfitting is crucial. Overfitting, where models closely follow training data, underscores the need for balanced complexity to prevent poor generalization. Conversely, underfitting emphasizes the importance of selecting models that adequately capture every data. Ethical considerations are also important, demanding vigilance against biases, transparent communication of ethical implications, and a commitment to fairness and transparency throughout the analysis process.And also is Data accuracy.One of the lobster catch example that we saw in class which had inaccurate data before a certain year is a good exmaple for this.

c)Overfitting or underfitting on the training data does not guarantee the same behavior on the test data.The impact on test data depends on the model's ability to generalize.Overfitting, capturing training data too closely, may or may not lead to similar issues in the test set.Similarly, underfitting, indicating a lack of complexity, does not necessarily translate to underfitting in the test data.

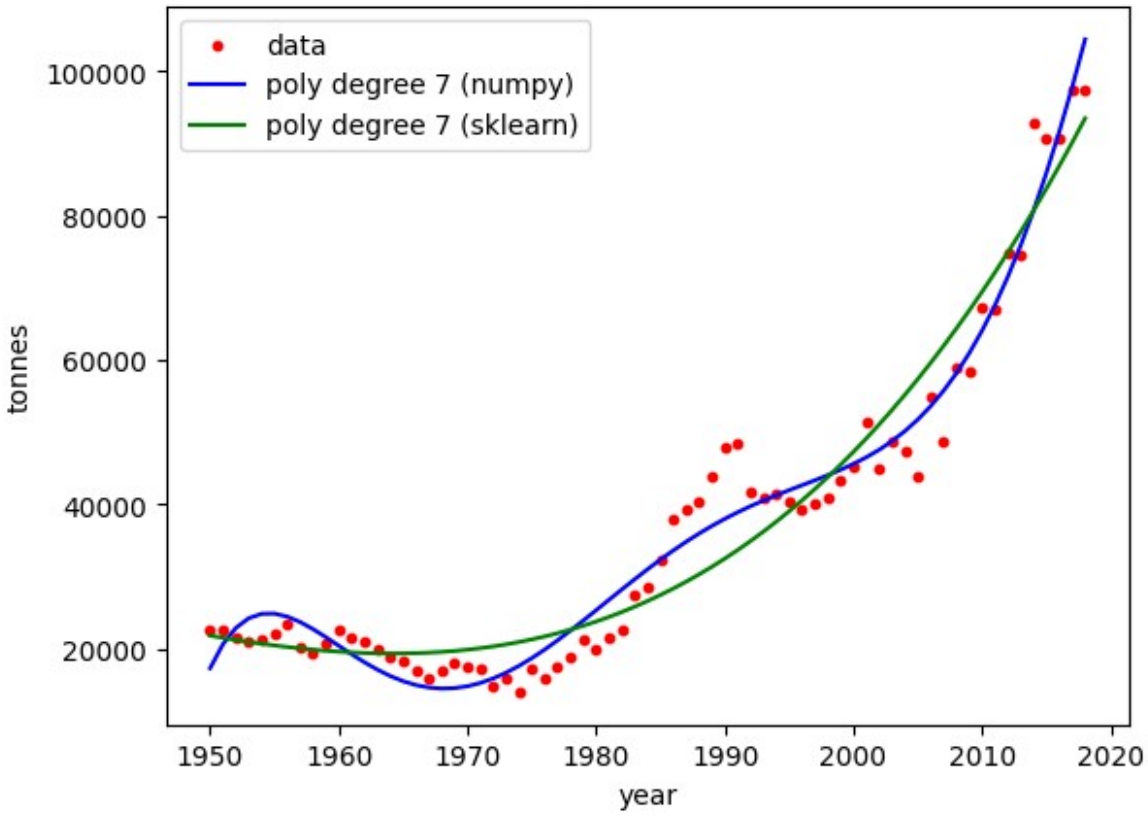
d)Overfitting or underfitting past data may result in poor predictions for future occurrences. Overfit models, by capturing noise and specific past patterns, may struggle to generalize to new data. Similarly, underfit models, being too simplistic, can also lead to inadequate predictions for future occurrences.

e)Optimal predictions for future occurrences require avoiding both overfitting and underfitting. Overfitting, by fitting training data too closely, leads to poor generalization, while underfitting, being too simplistic, results in inadequate predictions. Striking a balance with moderate model complexity, achieved through techniques like cross-validation, ensures accurate predictions for new data while capturing essential patterns from the past.

```
# Use 'years' instead of 'years after 2000'
x = lob['year']
y = lob['tonnes']

# Polynomial regression using numpy
deg = 7
coefficients_np = np.polyfit(x, y, deg)
poly_np = np.poly1d(coefficients_np)

# Polynomial regression using sklearn
X_poly = PolynomialFeatures(deg).fit_transform(x.values.reshape(-1,
```

```
1))
model_sklearn = LinearRegression()
model_sklearn.fit(X_poly, y.values.reshape(-1, 1))

# Print R2 scores
print("R2 score (numpy):", metrics.r2_score(y, poly_np(x)))
print("R2 score (sklearn):", model_sklearn.score(X_poly, y))

# Plot results
plt.scatter(x, y, 10, color='r', marker='o', label='data')
plt.plot(x, poly_np(x), color='b', label='poly degree '+str(deg)+'
(numpy)')
plt.plot(x, model_sklearn.predict(X_poly), color='g', label='poly
degree '+str(deg)+' (sklearn)')
plt.legend(loc='upper left')
plt.xlabel('year')
plt.ylabel('tonnes')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/
interactiveshell.py:3553: RankWarning: Polyfit may be poorly
conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)

R2 score (numpy): 0.9684264707755983
R2 score (sklearn): 0.9333775238375897
```




From what i read from the articles is that this could occur due to the higher-degree polynomials in the fitting process.