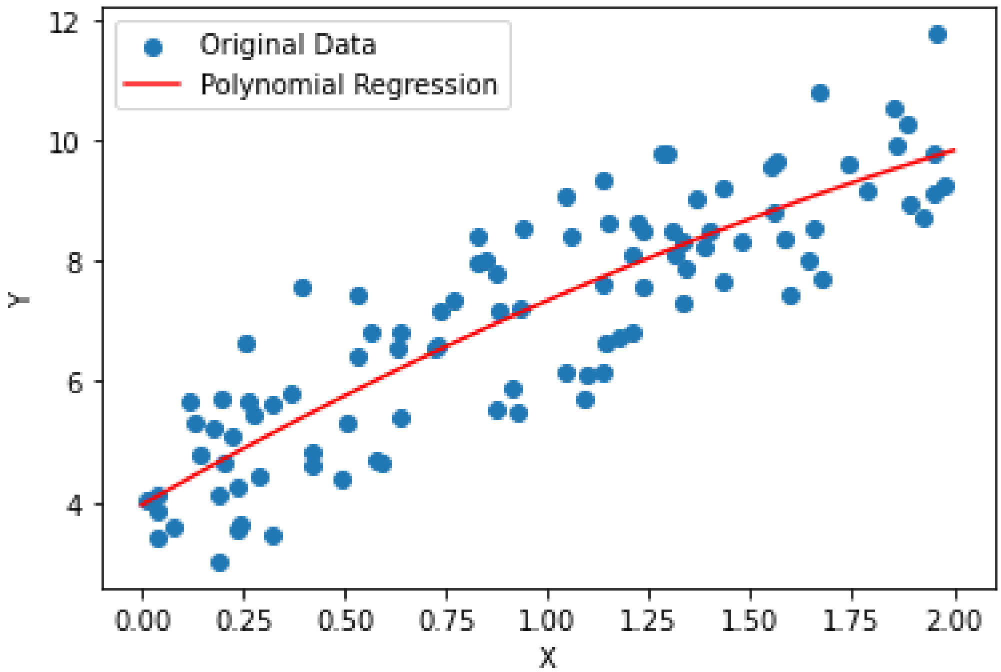
polynomial and logistic regression

August 30, 2023

[31]:

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
# Generate sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
Y = 4 + 3 * X + np.random.randn(100, 1)
# Fit a polynomial regression model
degree = 2 # You can change the degree as needed
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, Y)# Make predictions
X_new = np.linspace(0, 2, 100).reshape(-1, 1)
X_new_poly = poly_features.transform(X_new)
Y_new = model.predict(X_new_poly)
# Plot the original data and the polynomial regression curve
plt.scatter(X, Y, label='Original Data')
plt.plot(X_new, Y_new, 'r-', label='Polynomial Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
# The coefficients of the multivariate polynomial regression model
coefficients = model.coef_
intercept = model.intercept_
print("Coefficients:")
print(coefficients)
print("Intercept:")
print(intercept)
```




```
Coefficients:
[[ 0.          3.84100842 -0.45190593]]
Intercept:
[3.95139826]
```

[32]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2] # We'll use only the first two features for simplicity
y = (iris.target != 0) * 1 # Convert target labels to binary (0 or 1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
# Standardize the feature data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Create a logistic regression model
model = LogisticRegression(solver='liblinear')
```

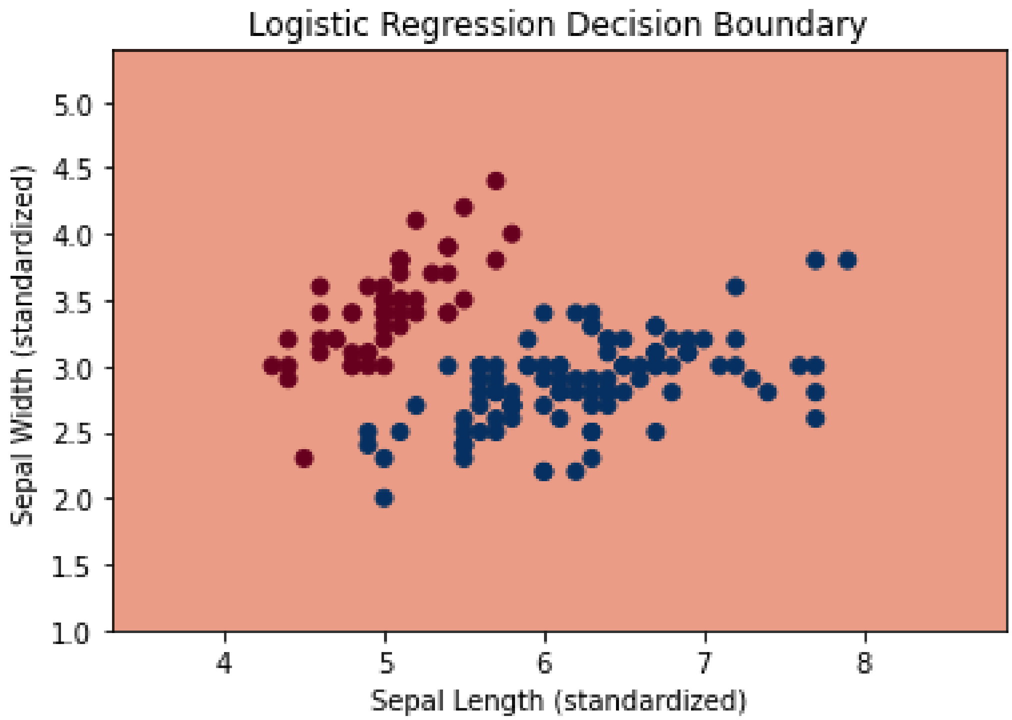
```python
# Train the model
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# Plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.
 ↪01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.RdBu, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdBu)
plt.xlabel('Sepal Length (standardized)')
plt.ylabel('Sepal Width (standardized)')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

```
Confusion Matrix:
[[19  0]
 [ 0 26]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        26

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```



[43]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
df=pd.read_csv('gold.csv')
df.head()
```

[43]:

	Year	Price (24 karat per 10 grams)
0	2022	52950
1	2021	50045
2	2020	48651
3	2019	35220
4	2018	31438

[25]:
```
yr= df['Year'].to_numpy()
gold=df['Price (24 karat per 10 grams)'].to_numpy()
```

[45]:
```
yr=yr.reshape(-1,1)
gold=gold.reshape(-1,1)
```

[55]:
```
degree = 3 # You can change the degree as needed
poly_features = PolynomialFeatures(degree=degree)
```

```
yr_poly = poly_features.fit_transform(yr)
model = LinearRegression()
model.fit(yr_poly, gold)
```

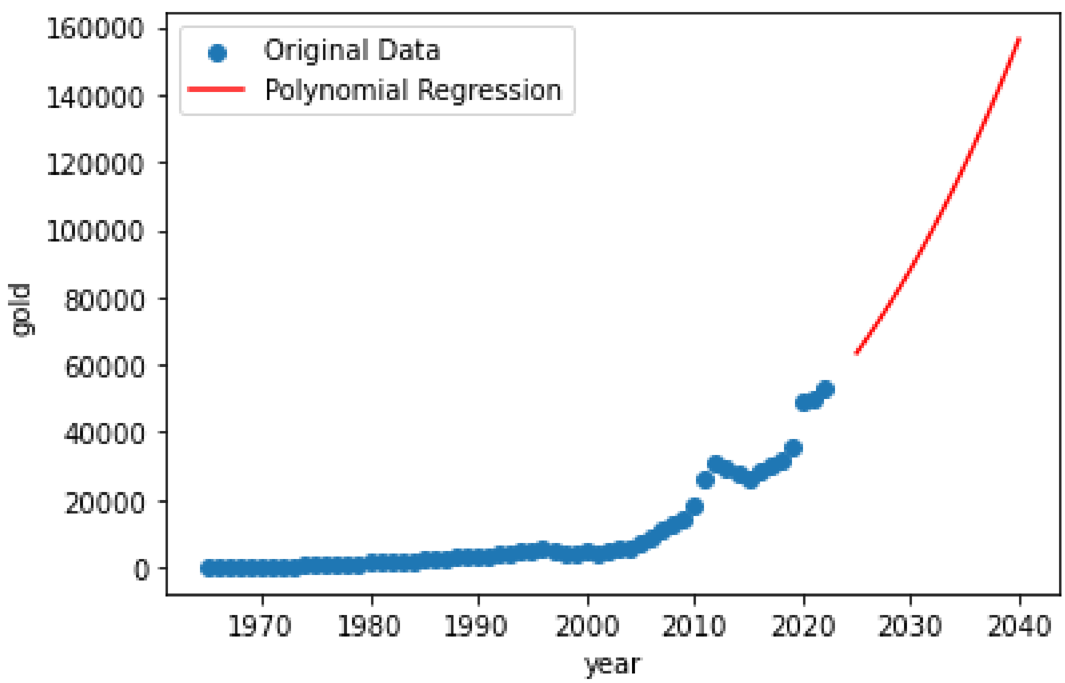
[55]: LinearRegression()

[56]:
```
model.score(yr_poly, gold)
#it gives us a good score
```

[56]: 0.9634754782596965

[57]:
```
yr_new = np.linspace(2025,2040,num=16,dtype=int).reshape(-1, 1)
yr_new_poly = poly_features.transform(yr_new)
gold_new = model.predict(yr_new_poly)
```

[58]:
```
plt.scatter(yr,gold, label='Original Data')
plt.plot(yr_new, gold_new, 'r', label='Polynomial Regression')
plt.xlabel('year')
plt.ylabel('gold')
plt.legend()
plt.show()
```



[72]:
```
df.insert(loc=2,
          column='bin',
          value=1)
```

[71]:

```
df
```

[71]:

	Year	Price (24 karat per 10 grams)	bin	new_column_name
0	2022	52950	1	0
1	2021	50045	1	0
2	2020	48651	1	0
3	2019	35220	1	0
4	2018	31438	1	0
5	2017	29667	1	0
6	2016	28623	1	0
7	2015	26343	1	0
8	2014	28006	1	0
9	2013	29600	1	0
10	2012	31050	1	0
11	2011	26400	1	0
12	2010	18500	1	0
13	2009	14500	1	0
14	2008	12500	1	0
15	2007	10800	1	0
16	2006	8400	1	0
17	2005	7000	1	0
18	2004	5850	1	0
19	2003	5600	1	0
20	2002	4990	1	0
21	2001	4300	1	0
22	2000	4400	1	0
23	1999	4234	1	0
24	1998	4045	1	0
25	1997	4725	1	0
26	1996	5160	1	0
27	1995	4680	1	0
28	1994	4598	1	0
29	1993	4140	1	0
30	1992	4334	1	0
31	1991	3466	1	0
32	1990	3200	1	0
33	1989	3140	1	0
34	1988	3130	1	0
35	1987	2570	1	0
36	1986	2140	1	0
37	1985	2130	1	0
38	1984	1970	1	0
39	1983	1800	1	0
40	1982	1645	1	0
41	1981	1800	1	0
42	1980	1330	1	0
43	1979	937	1	0

44	1978	685	1	0
45	1977	486	1	0
46	1976	432	1	0
47	1975	540	1	0
48	1974	506	1	0
49	1973	279	1	0
50	1972	202	1	0
51	1971	193	1	0
52	1970	184	1	0
53	1969	176	1	0
54	1968	162	1	0
55	1967	103	1	0
56	1966	84	1	0
57	1965	72	1	0

[81]:
```
df.loc[df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 1
df.loc[~df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 0
```

```
/tmp/ipykernel_4688/2541421223.py:1: FutureWarning: Boolean inputs to the
`inclusive` argument are deprecated infavour of `both` or `neither`.
  df.loc[df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 1
/tmp/ipykernel_4688/2541421223.py:2: FutureWarning: Boolean inputs to the
`inclusive` argument are deprecated infavour of `both` or `neither`.
  df.loc[~df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 0
```

[82]:
```
df
```

[82]:

	Year	Price (24 karat per 10 grams)		bin	new_column_name
0	2022	52950	1	0	0
1	2021	50045	1	0	0
2	2020	48651	1	0	0
3	2019	35220	1	0	0
4	2018	31438	1	0	0
5	2017	29667	1	0	0
6	2016	28623	1	0	0
7	2015	26343	1	0	0
8	2014	28006	1	0	0
9	2013	29600	1	0	0
10	2012	31050	1	0	0
11	2011	26400	1	0	0
12	2010	18500	1	0	0
13	2009	14500	1	0	0
14	2008	12500	1	0	0
15	2007	10800	1	0	0
16	2006	8400	1	0	0
17	2005	7000	1	0	0
18	2004	5850	1	0	0

19	2003	5600	1	0	0
20	2002	4990	1	0	0
21	2001	4300	1	0	0
22	2000	4400	1	0	0
23	1999	4234	1	0	0
24	1998	4045	1	0	0
25	1997	4725	1	0	0
26	1996	5160	1	0	0
27	1995	4680	1	1	0
28	1994	4598	1	1	0
29	1993	4140	1	1	0
30	1992	4334	1	1	0
31	1991	3466	1	1	0
32	1990	3200	1	1	0
33	1989	3140	1	1	0
34	1988	3130	1	1	0
35	1987	2570	1	1	0
36	1986	2140	1	1	0
37	1985	2130	1	1	0
38	1984	1970	1	1	0
39	1983	1800	1	1	0
40	1982	1645	1	1	0
41	1981	1800	1	1	0
42	1980	1330	1	1	0
43	1979	937	1	1	0
44	1978	685	1	1	0
45	1977	486	1	1	0
46	1976	432	1	1	0
47	1975	540	1	1	0
48	1974	506	1	1	0
49	1973	279	1	1	0
50	1972	202	1	1	0
51	1971	193	1	1	0
52	1970	184	1	1	0
53	1969	176	1	1	0
54	1968	162	1	1	0
55	1967	103	1	1	0
56	1966	84	1	1	0
57	1965	72	1	1	0

[84]:
```
df = df.drop(columns=['new_column_name'])
```

[85]:
```
df
```

[85]:

	Year	Price (24 karat per 10 grams)	bin
0	2022	52950	0
1	2021	50045	0

2	2020	48651	0
3	2019	35220	0
4	2018	31438	0
5	2017	29667	0
6	2016	28623	0
7	2015	26343	0
8	2014	28006	0
9	2013	29600	0
10	2012	31050	0
11	2011	26400	0
12	2010	18500	0
13	2009	14500	0
14	2008	12500	0
15	2007	10800	0
16	2006	8400	0
17	2005	7000	0
18	2004	5850	0
19	2003	5600	0
20	2002	4990	0
21	2001	4300	0
22	2000	4400	0
23	1999	4234	0
24	1998	4045	0
25	1997	4725	0
26	1996	5160	0
27	1995	4680	1
28	1994	4598	1
29	1993	4140	1
30	1992	4334	1
31	1991	3466	1
32	1990	3200	1
33	1989	3140	1
34	1988	3130	1
35	1987	2570	1
36	1986	2140	1
37	1985	2130	1
38	1984	1970	1
39	1983	1800	1
40	1982	1645	1
41	1981	1800	1
42	1980	1330	1
43	1979	937	1
44	1978	685	1
45	1977	486	1
46	1976	432	1
47	1975	540	1
48	1974	506	1

49	1973	279	1
50	1972	202	1
51	1971	193	1
52	1970	184	1
53	1969	176	1
54	1968	162	1
55	1967	103	1
56	1966	84	1
57	1965	72	1

[98]:
```
yr1=df['Year'].to_numpy()
bin1=df['bin'].to_numpy()
yr1=yr1.reshape(-1,1)
bin1=bin1.reshape(-1,1)
```

[99]:
```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
```

[100]:
```
yr_train, yr_test, bin1_train, bin1_test = train_test_split(yr1, bin1,␣
↪test_size=0.3, random_state=42)
```

[101]:
```
model = LogisticRegression(solver='liblinear')
```

[102]:
```
scaler = StandardScaler()
yr_train = scaler.fit_transform(yr_train)
yr_test = scaler.transform(yr_test)
```

[103]:
```
model.fit(yr_train, bin1_train)
```

```
/usr/lib/python3/dist-packages/sklearn/utils/validation.py:72:
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().
  return f(**kwargs)
```

[103]:
```
LogisticRegression(solver='liblinear')
```

[104]:
```
model.score(yr_train, bin1_train)
```

[104]:
```
0.975
```

[105]:
```
bin_pred = model.predict(yr_test)
```

[109]:
```python
print("Confusion Matrix:")
print(confusion_matrix(bin1_test, bin_pred))
print("\nClassification Report:")
print(classification_report(bin1_test, bin_pred))
```

```
Confusion Matrix:
[[10  0]
 [ 0  8]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         8

    accuracy                           1.00        18
   macro avg       1.00      1.00      1.00        18
weighted avg       1.00      1.00      1.00        18
```

[111]:
```python
data=pd.read_csv('golsil.csv')
data
```

[111]:

	Year	gold	silver
0	2022	52950	2715
1	2021	50045	2720
2	2020	48651	3105
3	2019	35220	3570
4	2018	31438	3955
5	2017	29667	4015
6	2016	28623	4794
7	2015	26343	6066
8	2014	28006	6755
9	2013	29600	6463
10	2012	31050	6646
11	2011	26400	8040
12	2010	18500	5489
13	2009	14500	7124
14	2008	12500	6335
15	2007	10800	7346
16	2006	8400	7345
17	2005	7000	8560
18	2004	5850	7615
19	2003	5600	7900
20	2002	4990	7215
21	2001	4300	7875
22	2000	4400	7695

```
23  1999  4234  11770
24  1998  4045  10675
25  1997  4725  17405
26  1996  5160  19520
27  1995  4680  23625
28  1994  4598  22165
29  1993  4140  27255
30  1992  4334  56900
31  1991  3466  56290
32  1990  3200  54030
33  1989  3140  43070
34  1988  3130  37825
35  1987  2570  36990
36  1986  2140  37825
37  1985  2130  41400
38  1984  1970  40600
39  1983  1800  63435
40  1982  1645  62572
41  1981  1800  55100
```

[135]:
```
sil=data['silver'][::-1].to_numpy().reshape(-1,1)
yr2=data['Year'][::-1].to_numpy().reshape(-1,1)
golyr=data[['Year','gold']].to_numpy().reshape(-1,2)
golyr
```

[135]:
```
array([[ 2022, 52950],
       [ 2021, 50045],
       [ 2020, 48651],
       [ 2019, 35220],
       [ 2018, 31438],
       [ 2017, 29667],
       [ 2016, 28623],
       [ 2015, 26343],
       [ 2014, 28006],
       [ 2013, 29600],
       [ 2012, 31050],
       [ 2011, 26400],
       [ 2010, 18500],
       [ 2009, 14500],
       [ 2008, 12500],
       [ 2007, 10800],
       [ 2006,  8400],
       [ 2005,  7000],
       [ 2004,  5850],
       [ 2003,  5600],
       [ 2002,  4990],
       [ 2001,  4300],
```

```
       [ 2000,  4400],
       [ 1999,  4234],
       [ 1998,  4045],
       [ 1997,  4725],
       [ 1996,  5160],
       [ 1995,  4680],
       [ 1994,  4598],
       [ 1993,  4140],
       [ 1992,  4334],
       [ 1991,  3466],
       [ 1990,  3200],
       [ 1989,  3140],
       [ 1988,  3130],
       [ 1987,  2570],
       [ 1986,  2140],
       [ 1985,  2130],
       [ 1984,  1970],
       [ 1983,  1800],
       [ 1982,  1645],
       [ 1981,  1800]])
```

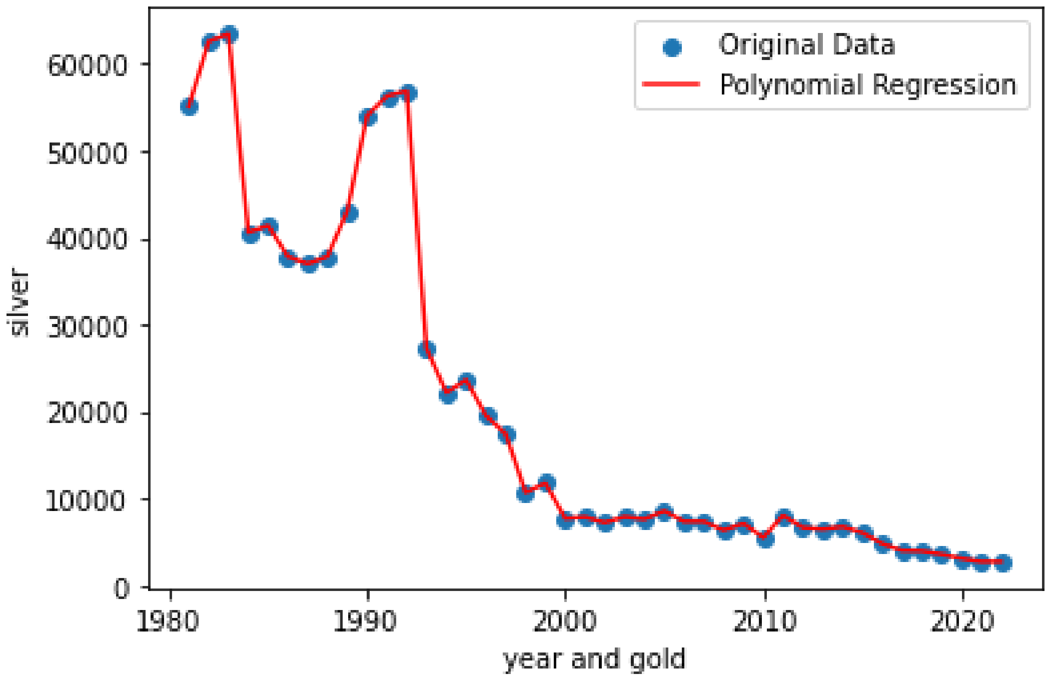
[129]:
```
degree = 3 # You can change the degree as needed
poly_features = PolynomialFeatures(degree=degree)
golyr_poly = poly_features.fit_transform(golyr)
model = LinearRegression()
model.fit(golyr_poly, sil)
```

[129]:
```
LinearRegression()
```

[131]:
```
model.score(golyr_poly, sil)
#it gives us a good score
```

[131]:
```
0.9889592081235473
```

[136]:
```
plt.scatter(yr2,sil, label='Original Data')
plt.plot(yr2, sil, 'r', label='Polynomial Regression')
plt.xlabel('year and gold')
plt.ylabel('silver')
plt.legend()
plt.show()
```



[148]:
```
silyr=data[['Year','silver']].to_numpy().reshape(-1,2)
data.insert(loc=2,
          column='nan',
          value=0)
bin1=data['bin']
```

[149]:
```
data.loc[df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 1
data.loc[~df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 0
```

```
/tmp/ipykernel_4688/2379100525.py:1: FutureWarning: Boolean inputs to the
`inclusive` argument are deprecated infavour of `both` or `neither`.
  data.loc[df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 1
/tmp/ipykernel_4688/2379100525.py:2: FutureWarning: Boolean inputs to the
`inclusive` argument are deprecated infavour of `both` or `neither`.
  data.loc[~df['Year'].between(1964, 1995, inclusive=True), 'bin'] = 0
```

[150]:
```
data
```

[150]:

	Year	gold	nan	na	bin	silver
0	2022	52950	0	0	0	2715
1	2021	50045	0	0	0	2720
2	2020	48651	0	0	0	3105
3	2019	35220	0	0	0	3570
4	2018	31438	0	0	0	3955
5	2017	29667	0	0	0	4015

6	2016	28623	0	0	0	4794
7	2015	26343	0	0	0	6066
8	2014	28006	0	0	0	6755
9	2013	29600	0	0	0	6463
10	2012	31050	0	0	0	6646
11	2011	26400	0	0	0	8040
12	2010	18500	0	0	0	5489
13	2009	14500	0	0	0	7124
14	2008	12500	0	0	0	6335
15	2007	10800	0	0	0	7346
16	2006	8400	0	0	0	7345
17	2005	7000	0	0	0	8560
18	2004	5850	0	0	0	7615
19	2003	5600	0	0	0	7900
20	2002	4990	0	0	0	7215
21	2001	4300	0	0	0	7875
22	2000	4400	0	0	0	7695
23	1999	4234	0	0	0	11770
24	1998	4045	0	0	0	10675
25	1997	4725	0	0	0	17405
26	1996	5160	0	0	0	19520
27	1995	4680	0	0	1	23625
28	1994	4598	0	0	1	22165
29	1993	4140	0	0	1	27255
30	1992	4334	0	0	1	56900
31	1991	3466	0	0	1	56290
32	1990	3200	0	0	1	54030
33	1989	3140	0	0	1	43070
34	1988	3130	0	0	1	37825
35	1987	2570	0	0	1	36990
36	1986	2140	0	0	1	37825
37	1985	2130	0	0	1	41400
38	1984	1970	0	0	1	40600
39	1983	1800	0	0	1	63435
40	1982	1645	0	0	1	62572
41	1981	1800	0	0	1	55100

[153]:
```
silyr_train, silyr_test, bin1_train, bin1_test = train_test_split(silyr, bin1,
    test_size=0.3, random_state=42)
```

[154]:
```
model = LogisticRegression(solver='liblinear')
```

[155]:
```
scaler = StandardScaler()
silyr_train = scaler.fit_transform(silyr_train)
silyr_test = scaler.transform(silyr_test)
```

[156]:
```
model.fit(silyr_train, bin1_train)
```

[156]: LogisticRegression(solver='liblinear')

[157]:
```
model.score(silyr_train, bin1_train)
```

[157]: 1.0

[158]:
```
bin_pred = model.predict(yr_test)
```

[159]:
```
print("Confusion Matrix:")
print(confusion_matrix(bin1_test, bin_pred))
print("\nClassification Report:")
print(classification_report(bin1_test, bin_pred))
```

```
Confusion Matrix:
[[0 9]
 [0 4]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         9
           1       0.31      1.00      0.47         4

    accuracy                           0.31        13
   macro avg       0.15      0.50      0.24        13
weighted avg       0.09      0.31      0.14        13


/usr/lib/python3/dist-packages/sklearn/metrics/_classification.py:1221:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```

```