


 [learn-co-curriculum](#) / [dsc-polynomial-regression-lab-v2-5](#) Public View license 0 stars  2 forks Star Watch ▾[Code](#) [Issues 1](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) solution ▾

...

This branch is [8 commits ahead](#), [8 commits behind](#) master.**hoffm386** more scaffolding, use StatsModels, and inspect p-values ...on Jul 15  12[View code](#) README.md

Polynomial Regression - Lab

Introduction

In this lab, you'll practice your knowledge on adding polynomial terms to your regression model!

Objectives

You will be able to:

- Determine if polynomial regression would be useful for a specific model or set of data
- Create polynomial terms out of independent variables in linear regression

Dataset

For this lab you'll be using some generated data:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv('sample_data.csv')
df.head()
```

```
<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

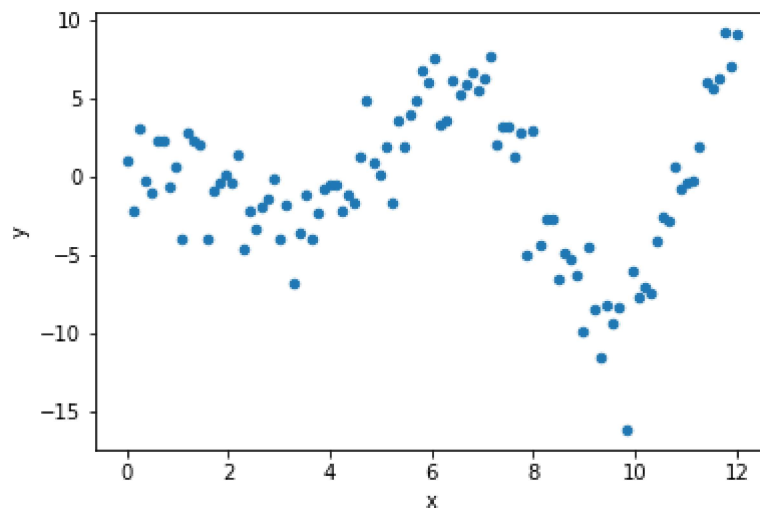
```
.dataframe thead th {
    text-align: right;
}
```

```
</style>
```

	x	y
0	0.000000	0.942870
1	0.121212	-2.261629
2	0.242424	3.100749
3	0.363636	-0.285446
4	0.484848	-1.012210

Let's check out a scatter plot of x vs. y :

```
df.plot.scatter(x="x", y="y");
```



You will notice that the data is clearly of non-linear shape. Begin to think about what degree polynomial you believe will fit it best.

You will fit several different models with different polynomial degrees, then plot them in the same plot at the end.

```
import statsmodels.api as sm
```

```
X = df[["x"]]
```

```
y = df["y"]
```

Build and Evaluate a Quadratic Model

This model should include a constant, x , and x squared. You can use `pandas` or `PolynomialFeatures` to create the squared term.

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly_2 = PolynomialFeatures(degree=2)
```

```
x_2 = pd.DataFrame(poly_2.fit_transform(X), columns=poly_2.get_feature_names(["x"]))  
squared_results = sm.OLS(y, x_2).fit()
```

```
print(f"""  
R-Squared Values  
Quadratic Regression: {squared_results.rsquared_adj}  
""")
```

R-Squared Values

Quadratic Regression: -0.015664188856822303

squared_results.pvalues

```
1      0.938055
x      0.893974
x^2    0.967420
dtype: float64
```

► Answer (click to reveal)

Build and Evaluate a 4th Degree Polynomial Model

In other words, the model should include x^0 (intercept), x^1 , x^2 , x^3 , and x^4 terms.

At this point we recommend importing and using `PolynomialFeatures` if you haven't already!

```
poly_4 = PolynomialFeatures(degree=4)
```

```
x_4 = pd.DataFrame(poly_4.fit_transform(X), columns=poly_4.get_feature_names(["x"]))
poly_4_results = sm.OLS(y, x_4).fit()
```

```
print(f"""
R-Squared Values
Quadratic Regression:          {squared_results.rsquared_adj}
4th Degree Polynomial Regression: {poly_4_results.rsquared_adj}
""")
```

```
R-Squared Values
Quadratic Regression:          -0.015664188856822303
4th Degree Polynomial Regression: 0.5667967820112239
```

poly_4_results.pvalues

```
1      1.995047e-04
x      1.738988e-10
x^2    3.340296e-14
x^3    1.715785e-16
x^4    7.408453e-18
dtype: float64
```

► Answer (click to reveal)

Build and Evaluate an 8th Degree Polynomial Model

This model should include x^0 through x^8 .

```
poly_8 = PolynomialFeatures(degree=8)
```

```
x_8 = pd.DataFrame(poly_8.fit_transform(X), columns=poly_8.get_feature_names(["x"]))
poly_8_results = sm.OLS(y, x_8).fit()
```

```
print(f"""
R-Squared Values
Quadratic Regression:          {squared_results.rsquared_adj}
4th Degree Polynomial Regression: {poly_4_results.rsquared_adj}
8th Degree Polynomial Regression: {poly_8_results.rsquared_adj}
""")
```

```
R-Squared Values
Quadratic Regression:          -0.015664188856822303
4th Degree Polynomial Regression: 0.5667967820112239
8th Degree Polynomial Regression: 0.8188676291759689
```

```
poly_8_results.pvalues
```

```
1      0.683776
x      0.618134
x^2    0.371163
x^3    0.199162
x^4    0.144776
x^5    0.157523
x^6    0.225536
```

```
x^7      0.357352
x^8      0.554141
dtype: float64
```

► Answer (click to reveal)

Plot All Models

Build a single plot that shows the raw data as a scatter plot, as well as all of the models you have developed as line graphs. Make sure that everything is labeled so you can tell the different models apart!

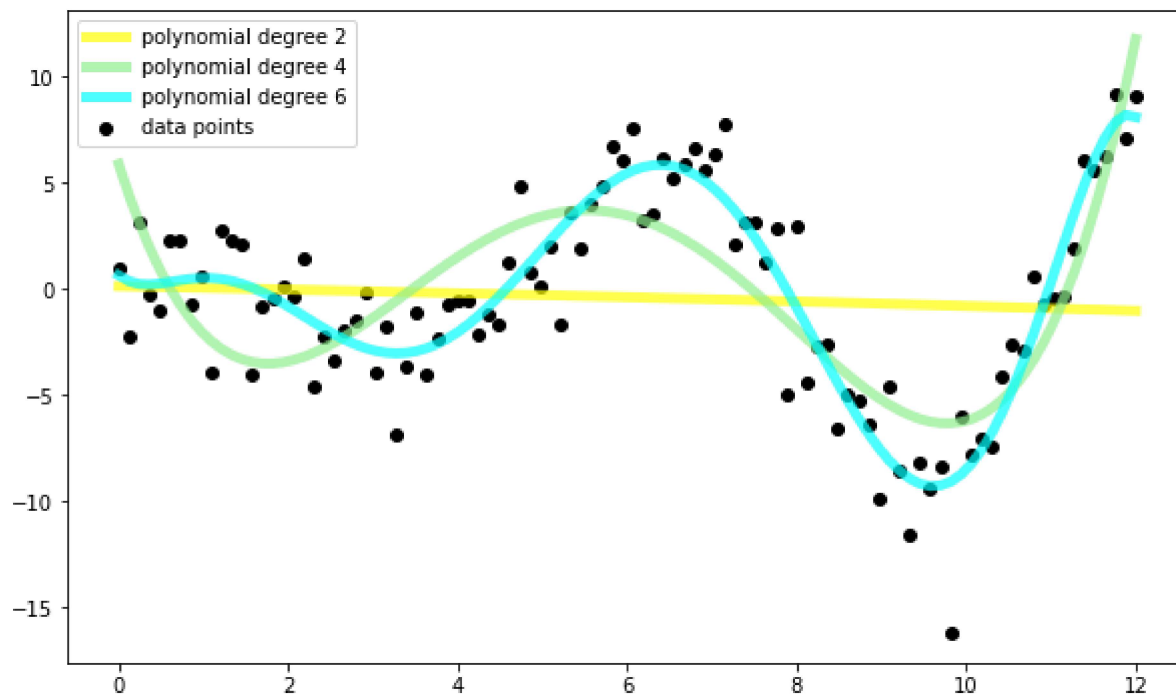
```
fig, ax = plt.subplots(figsize=(10, 6))

models = [squared_results, poly_4_results, poly_8_results]
data = [x_2, x_4, x_8]
colors = ['yellow', 'lightgreen', 'cyan']

ax.scatter(X, y, label="data points", color="black")
for i, model in enumerate(models):
    ax.plot(
        X, # plot same x values for every model
        model.predict(data[i]), # generate predictions using relevant preprocessed d
        label=f"polynomial degree {(i + 1)*2}", # degree happens to be 2 times (i +
        color=colors[i], # select color from list declared earlier
        linewidth=5,
        alpha=0.7
    )

ax.legend();
```





Interpret Findings

Based on the metrics as well as the graphs, which model do you think is the best? Why?

► Answer (click to reveal)

Summary

Great job! You now know how to include polynomials in your linear models as well as the limitations of applying polynomial regression.

Releases

No releases published

Packages

No packages published

Contributors 6



Languages

● Jupyter Notebook 100.0%