SESSION 10

1. Polynomial Regression - Introduction
2. Need for Polynomial Regression
3. Python implementation of polynomial regression
4. Construction of polynomial regression model
5. Displaying the polynomial regression result
6. Polynomial Smooth Regression Using CSV with Python
7. Polynomial Regression with Various Polynomial degree ranges

1. Polynomial Regression - Introduction

- Polynomial regression is a type of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial. In other words, polynomial regression tries to fit a curve to the data points by using a polynomial equation.

- Polynomial regression can be useful when the relationship between the independent variable and the dependent variable is not linear, but rather curved or has some other non-linear pattern. By using a polynomial equation to model the data, we can better capture the non-linear relationship between the variables.

- The degree of the polynomial used in the regression equation can be adjusted to fit the data well. A lower degree polynomial may not capture all the nuances in the relationship between the variables, while a higher degree polynomial may overfit the data and perform poorly on new data.

- Polynomial regression can be performed using various statistical software packages or programming languages such as Python or R. It is a powerful technique for analyzing and modeling complex relationships between variables in various fields such as finance, economics, and engineering.

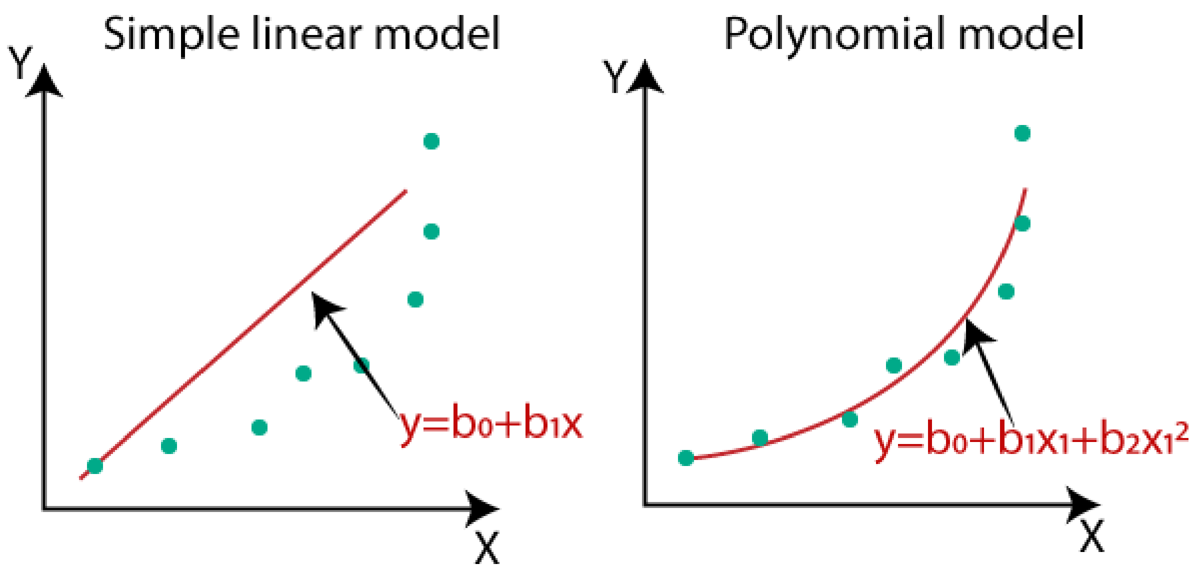
The regression procedure known as "Polynomial Regression" describes the relationship between a dependent variable (y) and an independent variable (x) as an nth degree polynomial. The following is the equation for polynomial regression:

$$y=b_0+b_1x_1+b_2x_1^2+b_2x_1^3+.....b_nx_1^n$$

1. In machine learning, it is also known as the special case of multiple linear regression. because in order to transform the equation for multiple linear regression into polynomial regression, certain polynomial terms are added.
2. It is a linear model that has been modified to improve accuracy.
3. The training dataset for polynomial regression is non-linear in character.
4. To fit the complex and non-linear functions and datasets, it uses a linear regression model.
5. Therefore, "In polynomial regression, the original data are converted into polynomial features of the needed degree (2,3,..,n) and then the model is linear."

2. Need for Polynomial Regression

- A linear model gives us a good result when applied to a linear dataset, as demonstrated in Basic Linear Regression, but when the same model is used to a non-linear dataset without any modifications, the results are drastically different. The loss function will grow as a result, leading to a high mistake rate and declining accuracy.
- In these situations, where the arrangement of the data points is non-linear, the polynomial regression model is required. With the below comparison graphic of the linear dataset and non-linear dataset, we can better grasp it.



- We have used a dataset that is non-linearly organised in the image above. As a result, when we attempt to cover it with a linear model, it is obvious that it rarely covers any data points. On the other hand, a curve that fits the polynomial model is appropriate to cover the majority of the data points.
- As a result, rather than using the Simple Linear Regression model when the datasets are organised in a non-linear way, we should utilise the Polynomial Regression model.

3. Python implementation of polynomial regression

- Python will be used to implement the polynomial regression in this case.
- By contrasting the Polynomial Regression model with the Basic Linear Regression model, we may better grasp it. -So let's first comprehend the issue for which the model is intended.

Problem Description:

An organisation that provides human resources is planning to hire a new applicant. The applicant disclosed a former salary of 160K per year, and HR must determine whether he is being truthful or lying. However, they only have a dataset from his prior employer, in which the top 10 positions' wages are listed together with their levels, to determine this. We discovered that there is a non-linear relationship between the Position levels and the pay by examining the dataset at hand. Our objective is to create a regression model for a bluffing detector so that HR can hire a trustworthy applicant. The steps to create such a model are listed below.

Position	Level	Salary
Business An	1	45000
Junior Cons	2	50000
Senior Con:	3	60000
Manager	4	80000
Country Ma	5	110000
Region Mar	6	150000
Partner	7	200000
Senior Part	8	300000
C-level	9	500000
CEO	10	1000000

Steps for Polynomial Regression: Listed below are the primary steps in Polynomial Regression:

```
- Data Preparation
- Create a linear regression model, then fit the data to it.
- Create a polynomial regression model and fit the data to it.
- Provide a result visualisation for the linear and polynomial regression models.
  estimating the result.
```

Data Preparation Step:

- With a few modifications, the data pre-processing step will remain the same as in earlier regression models. We won't employ feature scaling or divide our dataset into a training and test set for the polynomial regression model. Due to two factors:
- Because there is not enough information in the dataset to separate it into a test and training set, our model will not be able to identify relationships between wages and levels.
- This model should contain enough data because we want to make extremely accurate pay forecasts.

In [ ]:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
#importing datasets
data_set= pd.read_csv('Images\Position_Salaries.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

Explanation:

- To import and work on the dataset, we imported the necessary Python libraries in the lines of code above.
- The dataset "Position Salaries.csv" has been imported after that. It has three columns (Position, Levels, and Salary), but we will only focus on two of them (Salary and Levels).
- We then took the dependent (Y) and independent (X) variables out of the dataset.
- We chose the parameters [:,1:2] for the x-variable because we needed 1 index (levels), and we included:2 to make it a matrix.

In [ ]:

```
print(x)
print(y)
```

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
[  45000   50000   60000   80000  110000  150000  200000  300000  500000
 1000000]
```

- Three columns are present in the output shown above, as can be seen (Positions, Levels, and Salaries).
- Yet, since Positions are comparable to levels or might be thought of as Positions in an encoded form, we are just taking into account two columns.
- As the candidate has 4+ years of experience as a regional manager, he must be between levels 7 and 6, hence in this case we will anticipate the output for level 6.5.

4. Construction of polynomial regression model

Construction of the linear regression model:

We will now construct and fit the dataset with the linear regression model. We will use the linear regression model as a benchmark while creating the polynomial regression model and compare the outcomes. The key is provided below:

In [ ]:
```
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

Using the lin regs object of the LinearRegression class, we generated the Simple Linear model and fitted it to the dataset variables in the code above (x and y).

In [ ]:
```
print(lin_regs)
```

```
LinearRegression()
```

Regression polynomial model construction:

The Polynomial Regression model will now be constructed; however, it will differ slightly from the Simple Linear model. Because the PolynomialFeatures class of the preprocessing library will be used here. We are utilising this class to enhance our dataset with additional features.

In [ ]:
```
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

Because we are first turning our feature matrix into a polynomial feature matrix and then fitting it to the polynomial regression model, we used poly regs.fit transform(x) in the lines of code above. We can choose the parameter value (degree=2). We can select it based on the characteristics of our polynomial.

We will obtain another matrix, x poly, as a result of running the code, which is visible under the variable explorer option:

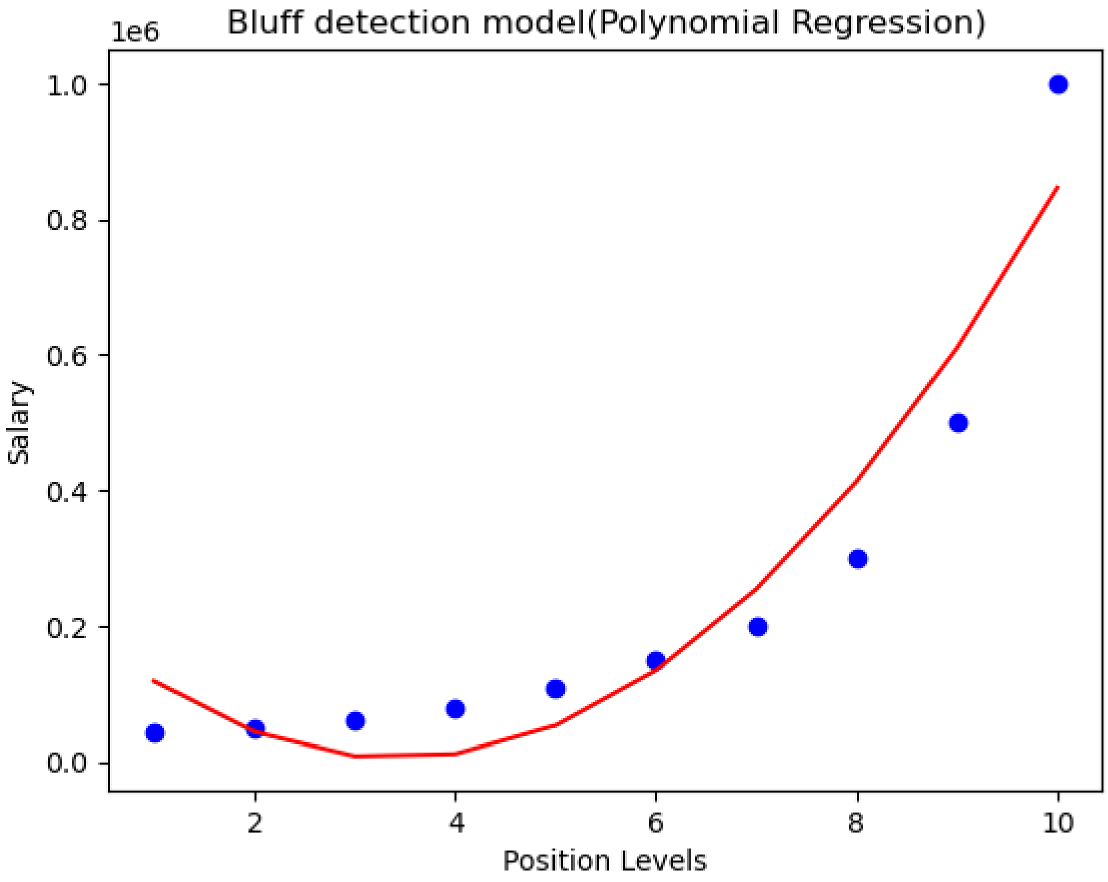
5. Displaying the polynomial regression result

The output of the polynomial regression model, whose code is slightly different from the model mentioned previously, will be shown here.

The following is the code for this:

In [ ]:

```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
#In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x),
#instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.
```



The output graphic above shows that the forecasts are rather close to the actual numbers. As we alter the degree, the plot above will change.

For degree= 3:

If we change the degree=3, then we will give a more accurate plot, as shown in the below image.

In [ ]:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Images\Position_Salaries.csv')
```

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
print(x)
print(y)
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
print(lin_regs)
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 3)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)

#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
#In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x),
#instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.
```

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
[  45000   50000   60000   80000  110000  150000  200000  300000  500000
 1000000]
LinearRegression()
```

So, as we can see in the output image up top, the level 6.5 forecasted income is close to 170K to 190K, which suggests that the future employee is telling the truth about his pay.

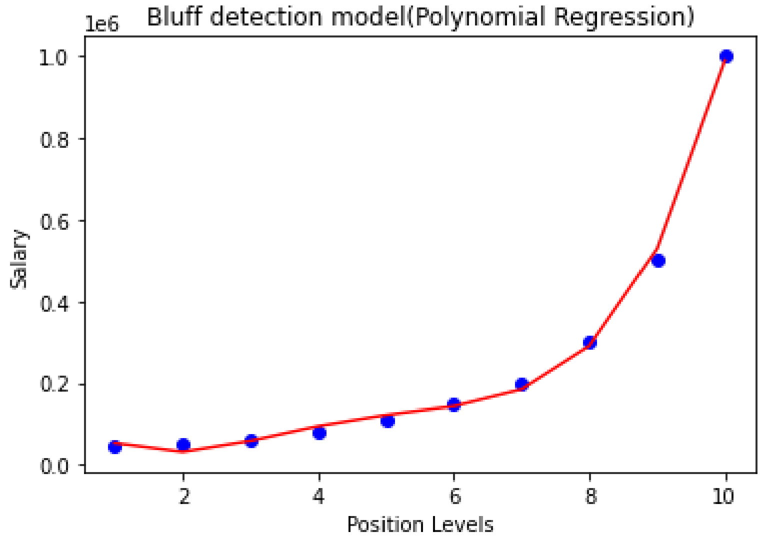
So, as we can see in the output image up top, the level 6.5 forecasted income is close to 170K to 190K, which suggests that the future employee is telling the truth about his pay.

In [ ]:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Images\Position_Salaries.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
print(x)
print(y)
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
print(lin_regs)
```

```
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 4)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)

#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
#In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x),
#instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.
```

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
[  45000   50000   60000   80000  110000  150000  200000  300000  500000
 1000000]
LinearRegression()
```



Using a linear regression model to forecast the outcome:

To determine whether an employee is telling the truth or bluffing, we will now forecast the ultimate result using the linear regression model. Thus, we'll utilise the predict() method and pass the number 6.5 for this. The code is as follows:

In [ ]:

```
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

```
[330378.78787879]
```

Using the polynomial regression model, one may forecast the outcome:

To compare with the linear model, we will now predict the outcome using the polynomial regression model. The code is as follows:

In [ ]:

```
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

```
[158862.45265153]
```

We may conclude that the prospective employee is telling the truth because the projected result for the polynomial regression is [158862.45265153], which is significantly closer to the real value.

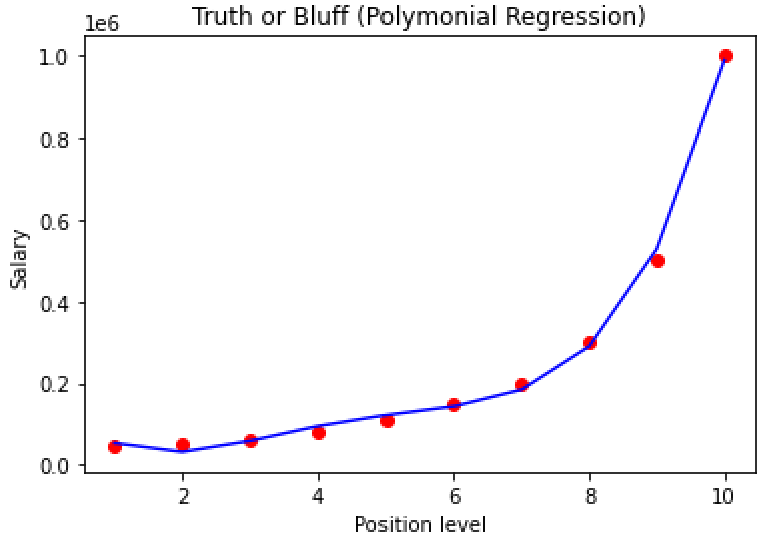
6. Polynomial Smooth Regression Using CSV with Python

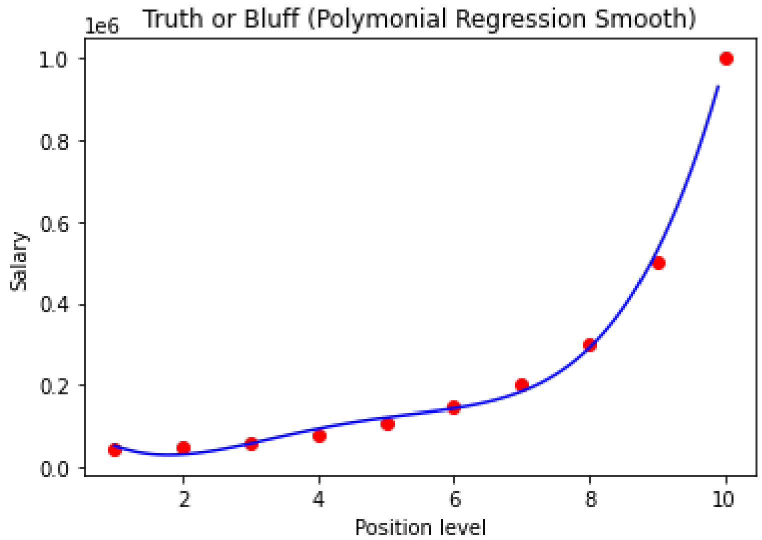
- In this example, we have to use 4 libraries as numpy, pandas, matplotlib and sklearn. Now we have to import libraries and get the data set first
- Code Credit : https://towardsdatascience.com/machine-learning-polynomial-regression-with-python-5328e4e8a386

In [ ]:

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

# Importing the dataset
dataset = pd.read_csv('Images/Position_Salaries.csv')
X = dataset.iloc[:, 1:2].values
y = dataset.iloc[:, 2].values

# Splitting the dataset into the Training set 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, random_state=0)

"""
# Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
"""

# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)

# Visualizing the Linear Regression results
def viz_linear():
    plt.scatter(X, y, color='red')
    plt.plot(X, lin_reg.predict(X), color='blue')
    plt.title('Truth or Bluff (Linear Regression)')
    plt.xlabel('Position level')
    plt.ylabel('Salary')
    plt.show()
    return
##viz_linear()

# Fitting Polynomial Regression to the dataset
```

```python
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(X)
pol_reg = LinearRegression()
pol_reg.fit(X_poly, y)

# Visualizing the Polymonial Regression results
def viz_polymonial():
    plt.scatter(X, y, color='red')
    plt.plot(X, pol_reg.predict(poly_reg.fit_transform(X)), color='blue')
    plt.title('Truth or Bluff (Polymonial Regression)')
    plt.xlabel('Position level')
    plt.ylabel('Salary')
    plt.show()
    return
viz_polymonial()

# Additional feature
# Making the plot line (Blue one) more smooth
def viz_polymonial_smooth():
    X_grid = np.arange(min(X), max(X), 0.1)
    X_grid = X_grid.reshape(len(X_grid), 1)
    # Visualizing the Polymonial Regression results
    plt.scatter(X, y, color='red')
    plt.plot(X_grid, pol_reg.predict(poly_reg.fit_transform(X_grid)), color='blue')
    plt.title('Truth or Bluff (Polymonial Regression Smooth)')
    plt.xlabel('Position level')
    plt.ylabel('Salary')
    plt.show()
    return
viz_polymonial_smooth()

# Predicting a new result with Linear Regression
lin_reg.predict([[5.5]])
#output should be 249500

# Predicting a new result with Polymonial Regression
pol_reg.predict(poly_reg.fit_transform([[5.5]]))
#output should be 132148.43750002
```





Out[ ]:
```
array([132148.43750002])
```

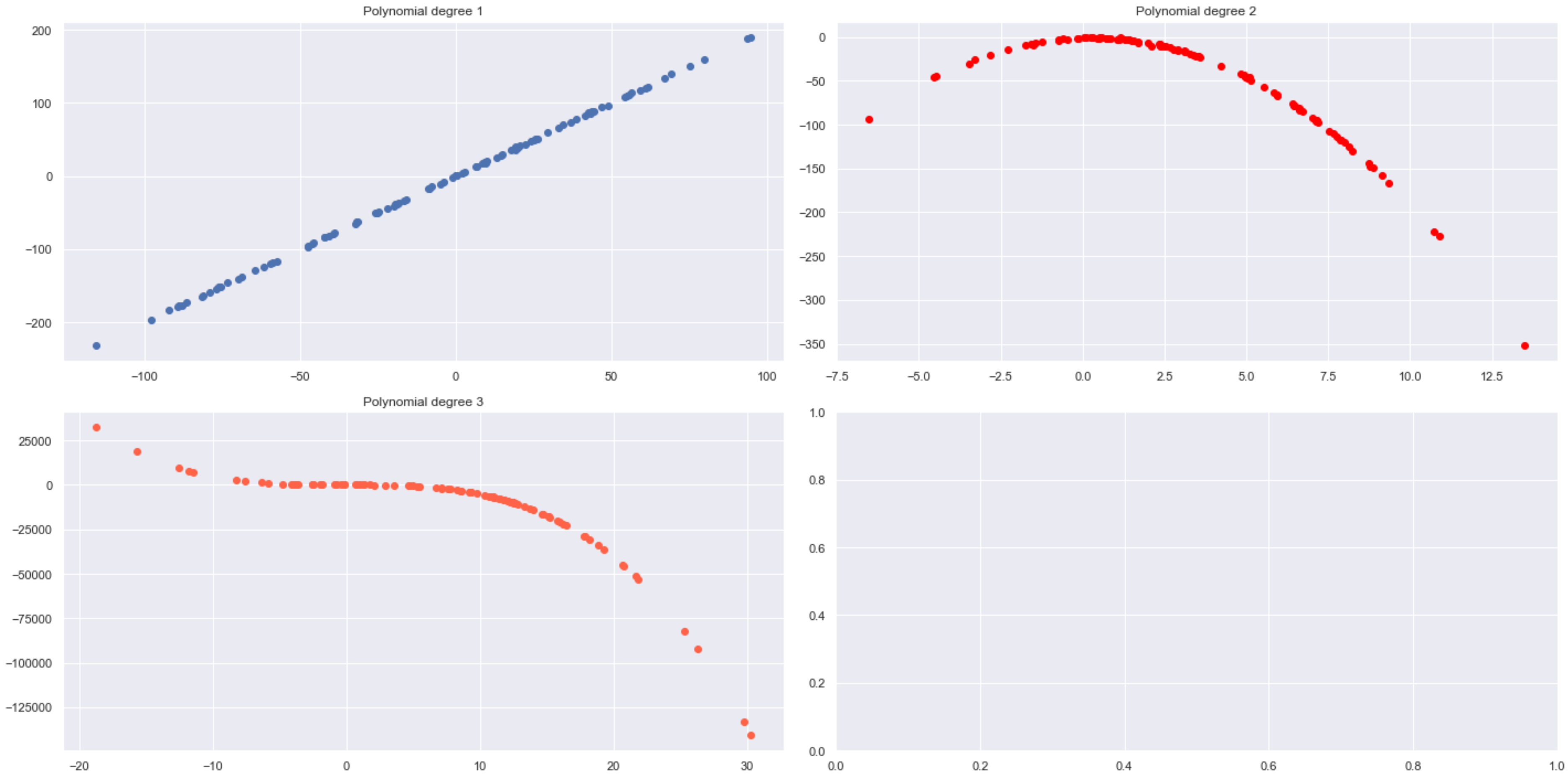
7. Polynomial Regression with Various Polynomial degree ranges

- Use more complex regressions to not so linear data.
- In this example, we have to use 4 libraries as numpy, pandas, seaborn, matplotlib and sklearn. Now we have to import libraries and get the data set first

- Code Credit: https://towardsdatascience.com/polynomial-regression-in-python-dd655a7d9f2b

In [ ]:
```
# Imports
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt

# Poly
from sklearn.preprocessing import PolynomialFeatures

# Metrics
from sklearn.metrics import mean_squared_error



# Datasets

# 1
x1 = 10 * np.random.normal(0,5,100) + 0.3
y1 = 1 + 2*x1 + np.random.normal(-1,1,100)
#x1 = x1[:, np.newaxis]
#y1 = y1[:, np.newaxis]

# 2
x2 = 2 * np.random.normal(1,2,100) + 1.6
y2 = x2 - 2 * (x2 ** 2) + np.random.normal(-1,1,100)
# transforming the data to include another axis
x2 = x2[:, np.newaxis]
y2 = y2[:, np.newaxis]

# 3
x3 = 3 * np.random.normal(1,3,100) + 3
y3 = x3 - 2 * (x3 ** 2) - 5 * (x3 ** 3) + np.random.normal(-2,2,100)
# transforming the data to include another axis
x3 = x3[:, np.newaxis]
y3 = y3[:, np.newaxis]


# Setup figure
fig, [(g1, g2), (g3, g4)] = plt.subplots(2, 2, figsize=(20,10))
```

```
g1.scatter(x1, y1)
g1.set_title('Polynomial degree 1')
g2.scatter(x2, y2, color='red')
g2.set_title('Polynomial degree 2')
g3.scatter(x3, y3, color='tomato')
g3.set_title('Polynomial degree 3')

plt.tight_layout();
```



Homework Problems

1. Code in Python using the scikit-learn library to capture the evaluation metrics of Polynomial Regression Model

```
# Use below sample data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array([4, 5, 6, 9, 10, 11, 12, 13, 14, 15])
```

2. Polynomial Regression with Various Polynomial degree ranges from 1 to 5

- Use more complex regressions to not so linear data.
- In this example, we have to use 4 libraries as numpy, pandas, seaborn, matplotlib and sklearn. Now we have to import libraries and get the data set first

---

For solutions of Homework questions, please refer to the `HomeworkSolution.ipynb` file