```
#### Importing the libraries
##"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')

#@"""



#### Importing the dataset
##"""

dataset = pd.read_csv('./Datasets/Position_Salaries.csv')
# print(dataset)
'''
            Position  Level   Salary
0   Business Analyst      1    45000
1  Junior Consultant      2    50000
2  Senior Consultant      3    60000
3            Manager      4    80000
4    Country Manager      5   110000
5     Region Manager      6   150000
6            Partner      7   200000
7     Senior Partner      8   300000
8            C-level      9   500000
9                CEO     10  1000000

'''

input_feature = dataset.iloc[ : , 1:-1].values
dv = dataset.iloc[ : , -1].values # // dependent variable

# print(input_feature)
# print(dv)


### Description:
## Line 34: Input feature (matrix of features)
'''
# // ** Here we can take only 1 col since both cols have the same
  meaning.
# / so, we will take the 2nd col., since it's numerical as well.

'''

#@"""



#### Training The Linear Regression Model on the Whole Dataset
##"""

from sklearn.linear_model import LinearRegression
```

```
57
58 ## Building the model
59 linear_R = LinearRegression()
60 ## Training the model
61 linear_R.fit(input_feature, dv)
62
63
64 ### Description:
65
66 #@"""
67
68
69
70 #### Training The Polynomial Regression Model on the Whole Dataset
71 ##"""
72
73 from sklearn.preprocessing import PolynomialFeatures
74
75 ## Building the model
76 polynomial_R = PolynomialFeatures(degree=6)
77 ## Training the model
78 poly_input_features = polynomial_R.fit_transform(input_feature)
79
80 ## Building and training another Linear Regression model on the new
   input features resulted from PolyFeatures
81 poly_linear_R = LinearRegression()
82 poly_linear_R.fit(poly_input_features, dv)
83
84 ### Description:
85 ## Line 73: Importing the Polynomial Features
86 '''
87 # // Here, we're converting (x1) feature to (x1 ^ 2) and (x1 ^ n), so
   it's kind of a data preprocessing tool,
88 # / That's why it's in the preprocessing section.
89 '''
90
91 ## Line 76: Building the Polynomial Regressor model
92 '''
93 ** "degree" parameter:
94 # // This is basically the (n) in the Polynomial Reg. eq., how many
   powers do we give.
95 # // note: you get a new coefficient (b) with every (n) you give.
96 # // you can try different values to see which is better.
97 # // we will start with (n=2), this will turn the single feature (x1)
   into (x1, x1^2)
98
99 # // I then tried with degree = 3, degree = 4 and degree = 6
100
101 # // degree in the code is n in the eq.
102 '''
103
104 ## Line 78: Training the Poly_R model
105 '''
106 # // If you noticed, here we are using ".fit_transform" because we are
107 # / transforming the input feature (x1) into (x1, x1^2).
108 '''
109
```

```
## Line 81, 82: This is the Polynomial Linear Regression Model
'''
# // Basically preprocessing the data to transform it into a
  polynomial data
# / then fitting it into/ feeding it to the Linear Regressor model.
'''

#@"""



#### Visualizing the LR results
"""

## real points
plt.scatter(input_feature, dv, color="orange")
## LR prediction (plotting the line of the LR predictions)
plt.plot(input_feature, linear_R.predict(input_feature), color="blue")

plt.title("Linear Regression model predictions with the real values")
plt.xlabel("Position Label.")
plt.ylabel("Salary.")

plt.show()


### Description:
## Line 120: plotting the scattering points of the real values
'''
** 1st arg: x-axis values -> input_feature

** 2nd arg: y-axis values -> dv

** 3rd arg: color of the scattered points
'''

## Line 122: Plotting the line of LR predictions
'''
** 2nd arg: y-axis values -> predictions of the model

# // It's a line because the model returns a value for each value in
  the input features and values for the
# / range between them.
'''

"""



#### Visualizing the PLR results
##"""

## real points
plt.scatter(input_feature, dv, color="orange")
## LR prediction (plotting the line of the LR predictions)
plt.plot(input_feature, poly_linear_R.predict(poly_input_features),
  color="blue")
```

```
164
165 plt.title("Polynomial Regression predictions with the real values")
166 plt.xlabel("Position Label.")
167 plt.ylabel("Salary.")
168
169 plt.show()
170
171
172 ### Description:
173
174 #@"""
175
176
177
178 #### Predicting a new result with Linear Regression
179 ##"""
180
181 print("LR prediction: ", linear_R.predict([[6.5]]) )
182 # -> LR prediction: [330378.78787879]
183
184 ### Description:
185 ## Line 181: predicting a level between 6 and 7
186 '''
187 # // reminder: you have to create a vector with 2 square brackets,
188 # / if you create 1 square brackets it will be seen as a list.
189
190 # // The prediction of the model is that the salary of the given level
     is $330k, which is wrong
191 # / because it's too high.
192 '''
193
194 #@"""
195
196
197
198 #### Predicting a new result with Polynomial Regression
199 ##"""
200
201 ## for n=6 Polynomial Regression model
202
203 print("PLR prediction: ", poly_linear_R.predict([[0, 6.5, 6.5 ** 2, 6.
    5 ** 3, 6.5 ** 4, 6.5 ** 5, 6.5 ** 6]]) )
204 # -> PLR prediction:  [174192.81930603]
205
206 print("PLR prediction: ", poly_linear_R.predict(polynomial_R.
    fit_transform([[6.5]]) ) )
207 # -> PLR prediction:  [174192.81930603]
208
209
210 ### Description:
211 ## Line 201:
212 '''
213 # // Here we can't give a single value, we would need to give an array
    (vector) corresponding to
214 # / the value of 6.5.
215 # // ** Don't forget the "b0" value, we will give the y-intercept (b0
    ) = 0
```

```
# // [[0, 6.5, 6.5 ^ 2, 6.5 ^ 3, 6.5 ^ 4, 6.5 ^ 5, 6.5 ^ 6]]

# // ** or use the Polynomial Features preprocessing tool

'''

#@"""
```