Assignment 7: Polynomial Regression

We will be investigating the use of Polynomial Regression on the function y = x3. We want to add some randomness to x3, so we will add a fraction of a random value selected from a normal distribution with a standard deviation of 1 centered around 0:

y = x**3 + 0.5 * np.random.normal(0,1,1)

In [163]:
```
#libraries required
from random import randint
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import math
from math import sqrt

#generating values for y function
def y(x):
    return x**3 + 0.5 * np.random.normal(0,1,1)

print(y(10))
```

```
[1000.80949083]
```

1. Test Points Make a list of 10 random points between -2 and 2 (using a uniform distribution). Pass this list into the function described above to get a set of x and y coordinates. Display the (x,y) coordinates as a data frame.

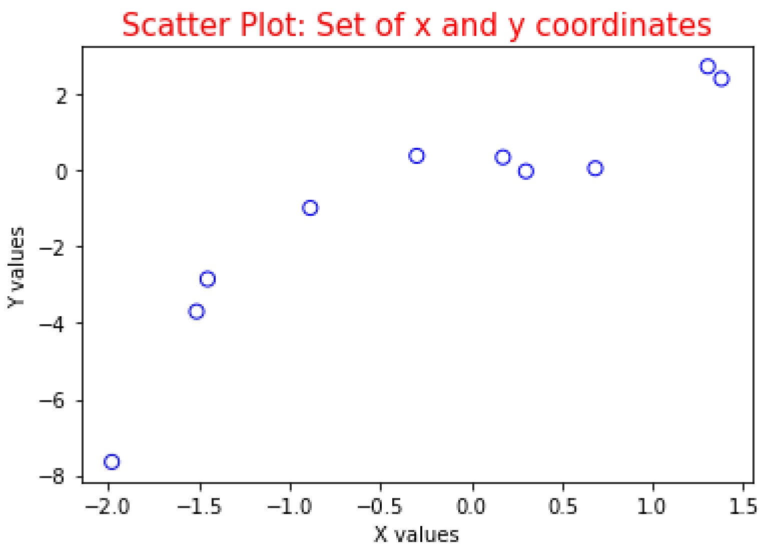
In [164]:
```
fun = np.vectorize(y)

np.random.seed(100)

# list of 10 random points between -2 and 2 using uniform distribution
X = np.sort(np.random.uniform(-2,2,10))

#Pass this list into the function described above to get a set of x and y coordir

Y = fun(X)

#Display the (x,y) coordinates as a data frame.
print("X values")
print(X)
print("\nY values")
print(Y)
```

```
X values
[-1.98112458 -1.51372352 -1.45317364 -0.88652246 -0.30192964  0.17361977
  0.30037332  0.68299634  1.30341102  1.37910453]

Y values
[-7.64812515 -3.69749732 -2.85110484 -0.98853509  0.38089918  0.34159397
 -0.02510465  0.05296668  2.72920563  2.4038915 ]
```

2. Create Graphs Now, create scatter plots of the (x,y) coordinates and:

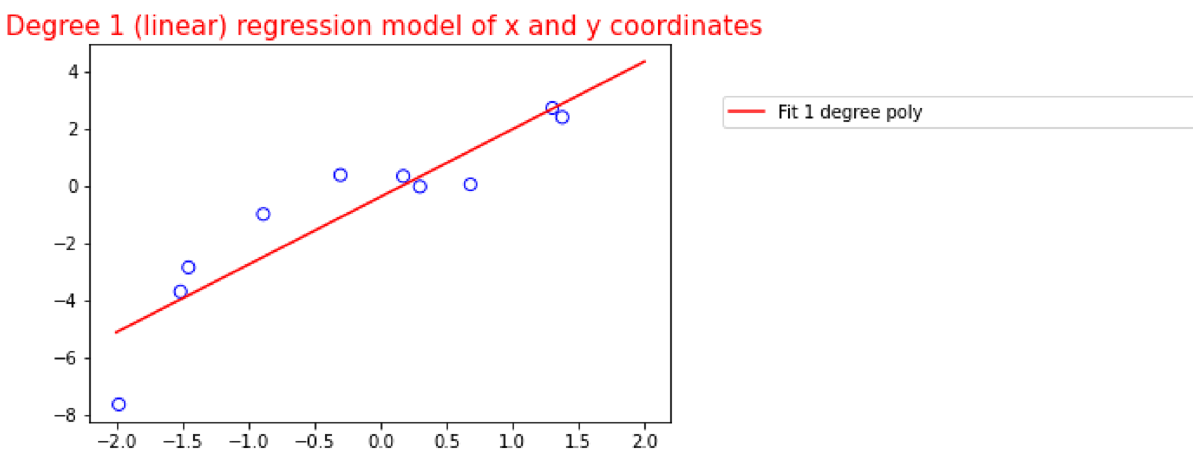
a degree 1 (linear) regression model, degree 2 (quadratic) regression model, degree 3 (cubic) regression model, degree 5, and degree 9. There should be 5 separate graphs for this step each clearly labeled and annotated.

In [165]:
```
#Now, create scatter plots of the (x,y) coordinates
%matplotlib inline

#Plot the (x,y) coordinates

plt.scatter(X,Y, s =50, facecolor = 'none' , edgecolor = 'blue', alpha = 1)
plt.xlabel('X values')
plt.ylabel('Y values')
plt.title('Scatter Plot: Set of x and y coordinates', color = 'Red', fontsize = 1
plt.show()
```



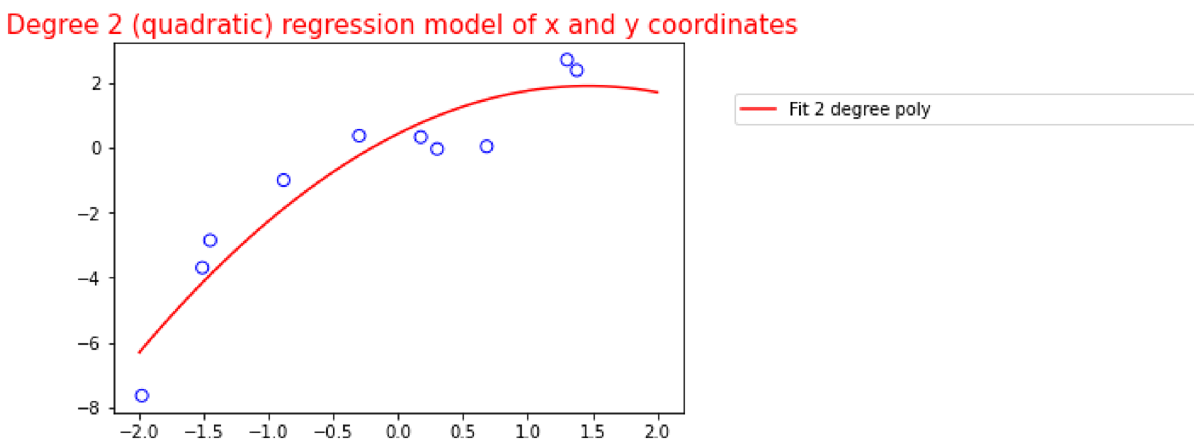
degree 1 (linear) regression model

In [166]:
```
# degree 1 (linear) regression model

X1 = np.linspace(-2,2)

m1 = Pipeline([('poly', PolynomialFeatures(degree=1)),('linear', linear_model.Lir
m1 =m1.fit(X[:,np.newaxis], Y[:,np.newaxis])

Y1 = m1.predict(X1[:, np.newaxis])

plt.scatter(X,Y, s =50, facecolor = 'none' , edgecolor = 'blue', alpha = 1)

plt.plot(X1, Y1, 'r', label="Fit "+str(1)+ " degree poly")

plt.title('\n Degree 1 (linear) regression model of x and y coordinates', color =

plt.legend(bbox_to_anchor=(1., 1., 1., 0.), mode="expand", borderaxespad= 3.)

plt.show()
```



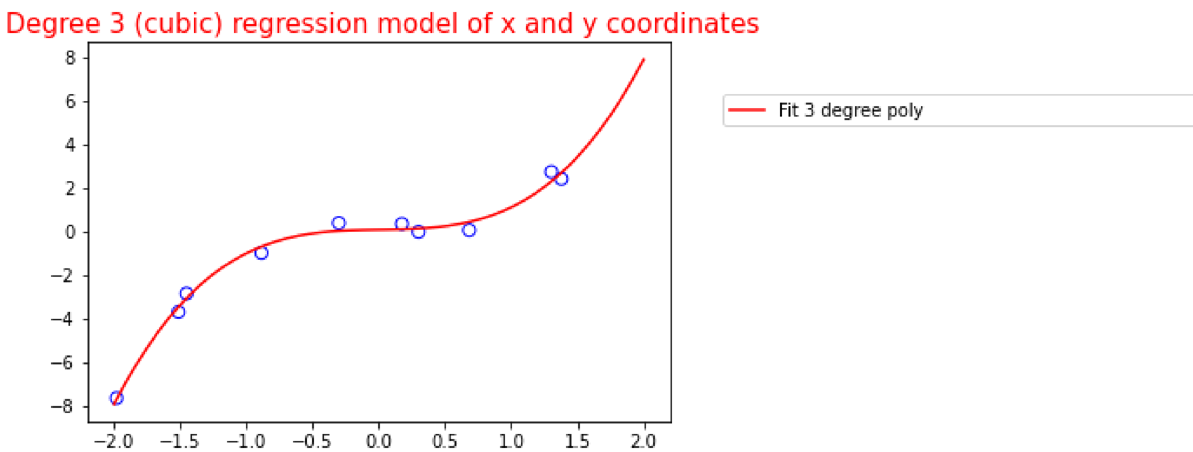
Degree 2 (quadratic) regression model

In [167]:
```
# degree 2 (quadratic) regression model

m2 = Pipeline([('poly', PolynomialFeatures(degree=2)),('quadratic', linear_model.
m2 =m2.fit(X[:,np.newaxis], Y[:,np.newaxis])

Y2 = m2.predict(X1[:, np.newaxis])

plt.scatter(X,Y, s =50, facecolor = 'none' , edgecolor = 'blue', alpha = 1)
plt.plot(X1, Y2, 'r', label="Fit "+str(2)+ " degree poly")
plt.title('\n Degree 2 (quadratic) regression model of x and y coordinates', colo
plt.legend(bbox_to_anchor=(1., 1., 1., 0.), mode="expand", borderaxespad= 3.)


plt.show()
```



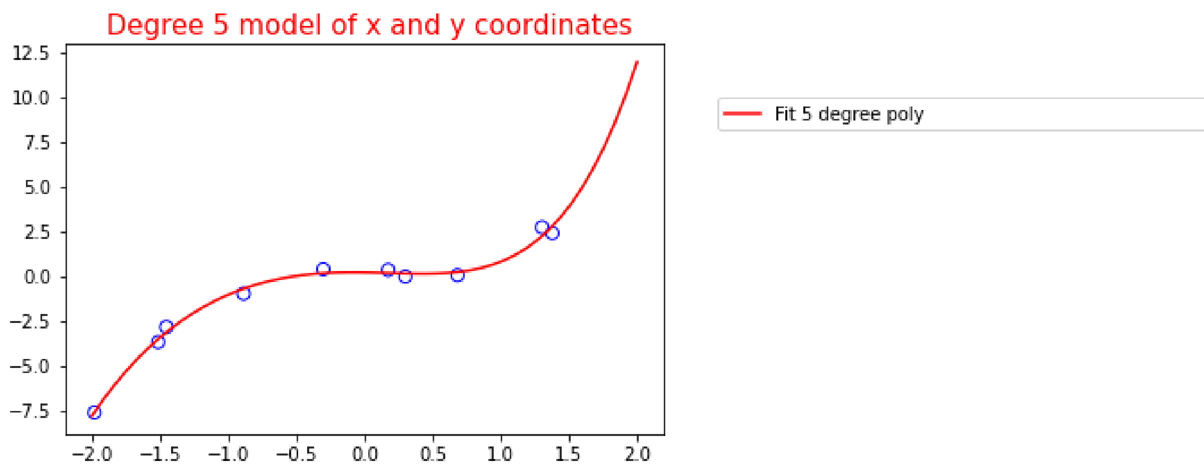
Degree 3 (cubic) regression model

In [168]:
```
# degree 3 (cubic) regression model

m3 = Pipeline([('poly', PolynomialFeatures(degree=3)),('degree 3', linear_model.L
m3 = m3.fit(X[:,np.newaxis], Y[:,np.newaxis])

Y3 = m3.predict(X1[:, np.newaxis])

plt.scatter(X,Y, s =50, facecolor = 'none' , edgecolor = 'blue', alpha = 1)
plt.plot(X1, Y3, 'r', label="Fit "+str(3)+ " degree poly")
plt.title('\n Degree 3 (cubic) regression model of x and y coordinates', color =
plt.legend(bbox_to_anchor=(1., 1., 1., 0.), mode="expand", borderaxespad= 3.)

plt.show()
```



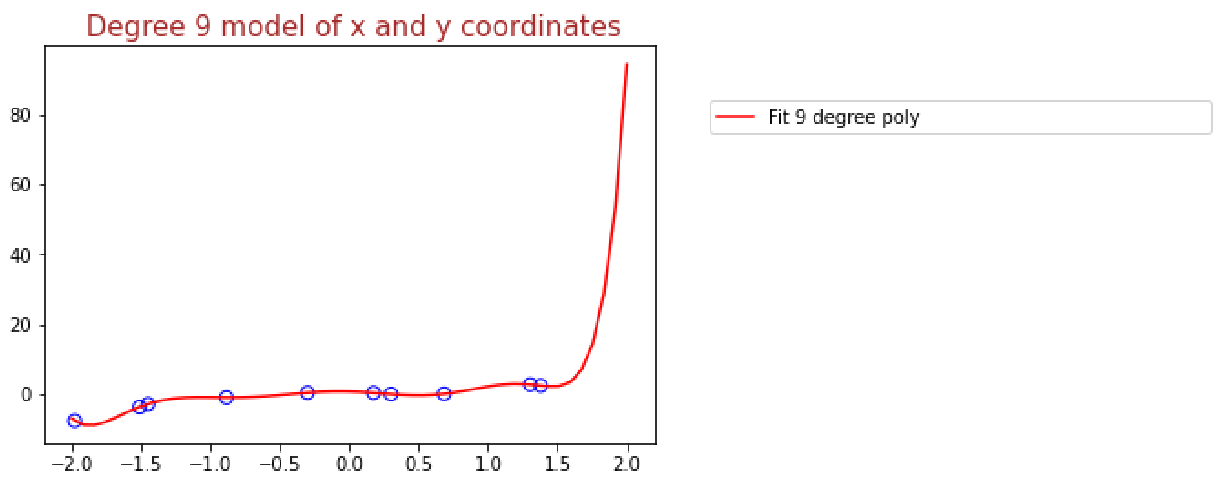
Degree 5

In [169]:
```
# degree 5

m5 = Pipeline([('poly', PolynomialFeatures(degree=5)),('degree 5', linear_model.L
m5 =m5.fit(X[:,np.newaxis], Y[:,np.newaxis])

Y5 = m5.predict(X1[:, np.newaxis])

plt.scatter(X,Y, s =50, facecolor = 'none' , edgecolor = 'blue', alpha = 1)
plt.plot(X1, Y5, 'r', label="Fit "+str(5)+ " degree poly")
plt.title('\n Degree 5 model of x and y coordinates', color = 'red', fontsize = '
plt.legend(bbox_to_anchor=(1., 1., 1., 0.), mode="expand", borderaxespad= 3.)

plt.show()
```



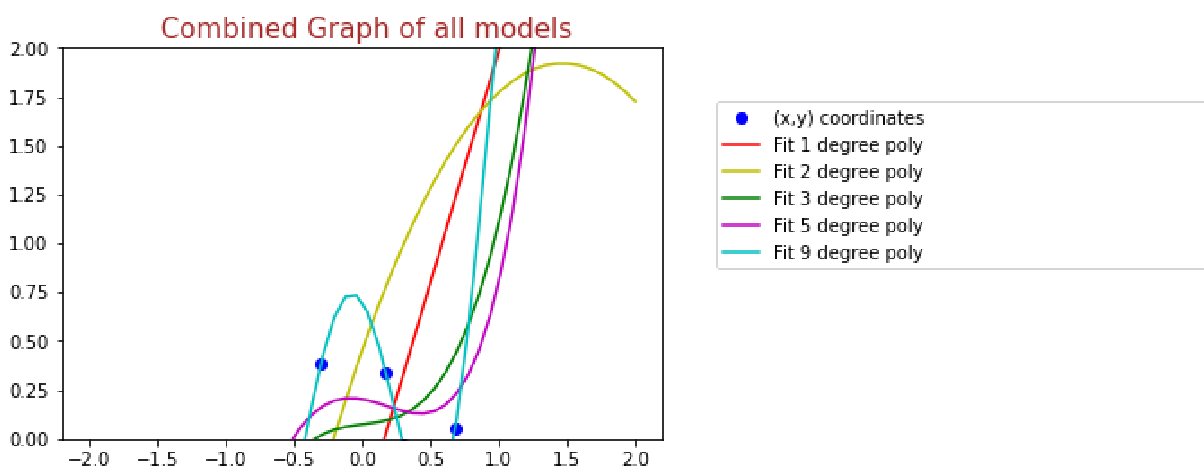
Degree 9

In [170]:
```
# degree 9

m9 = Pipeline([('poly', PolynomialFeatures(degree=9)),('degree 9', linear_model.L
m9 =m9.fit(X[:,np.newaxis], Y[:,np.newaxis])

Y9 = m9.predict(X1[:, np.newaxis])

plt.scatter(X,Y, s =50, facecolor = 'none' , edgecolor = 'blue', alpha = 1)
plt.plot(X1, Y9, 'r', label="Fit "+str(9)+ " degree poly")
plt.title('\n Degree 9 model of x and y coordinates', color = 'brown', fontsize =
plt.legend(bbox_to_anchor=(1., 1., 1., 0.), mode="expand", borderaxespad= 3.)

plt.show()
```



3. Combine Graphs Now, create single graph that contains all 5 regression models from the last step as well as the (x,y) coordinates. Make sure there is a useful legend as well.

In [177]:
```
#Combined Graph

colour =['r', 'y', 'g','m','c']
plt.plot(X, Y, 'bo', label="(x,y) coordinates")

d = [1,2,3,5,9]

i = 0
for degree in d:
    dval = 'degree ' + str(degree)
    m = Pipeline([('poly', PolynomialFeatures(degree=degree)),(dval, linear_model
    m=m.fit(X[:,np.newaxis], Y[:,np.newaxis])

    newY = m.predict(X1[:,np.newaxis])

    plt.plot(X1, newY , colour[i], label="Fit "+str(degree)+ " degree poly")
    plt.legend(bbox_to_anchor=(1., 1., 1., 0.), mode="expand", borderaxespad= 3.)

    plt.ylim(0,2)

    i = i+1

plt.title('\n Combined Graph of all models', color = 'brown', fontsize = '15')

plt.show()
```



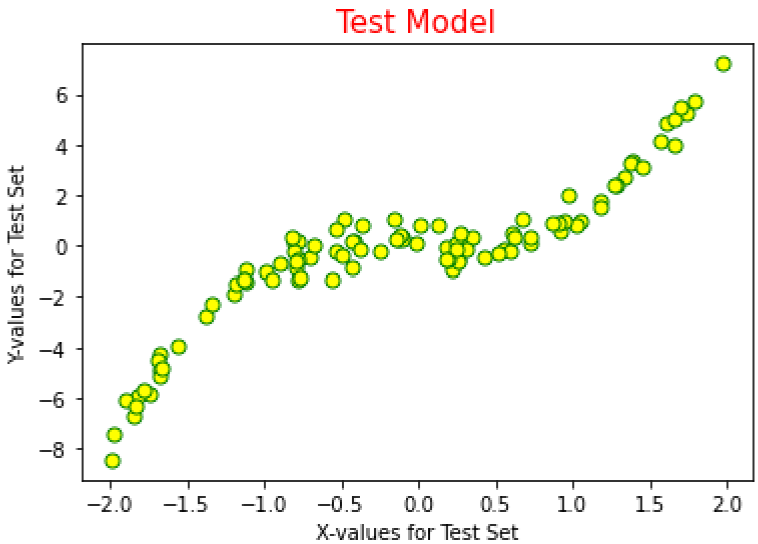
4. Test the Models We want to see which model was the best predictor for our function. Which one do you think would be the best? To figure this out, create a set of 100 x coordinates randomly generated from a uniform distribution between -2 and 2. Then generate the corresponding y coordinates by passing in the x's to the x3 + (degree of randomness) function from earlier. Display the first few (x,y) coordinates to make sure they are what you expect.

Test Model

In [185]:
```
#create a set of 100 x coordinates randomly generated from a uniform distribution

Xtest = np.random.uniform(-2,2,100)

#generate the corresponding y coordinates by passing in the x's to the x3 + (degr

Ytest = fun(Xtest)

# Displaying the coordinates
print("X values for test model")
print(Xtest)

print("\nY values for test model")
print(Ytest)

%matplotlib inline
plt.scatter(Xtest,Ytest, s =50, facecolor = 'yellow' , edgecolor = 'green', alpha

plt.xlabel('X-values for Test Set')
plt.ylabel('Y-values for Test Set')
plt.title('Test Model', color = 'red', fontsize = '15')
plt.show()
```

```
X values for test model
[ 1.56671811  0.55630684 -1.83833108  0.66805244 -0.70334043 -1.67085667
  0.1807794  -1.11473476 -0.53991589 -0.80704032  1.33366626 -1.68469309
  0.32787918  0.24945729 -0.99266763 -1.37777118  0.35317722  0.42737389
  0.91947922 -0.77550078  1.04431201  1.74279622 -0.41114834  1.38579465
  0.59825289  0.25205856  1.18071821 -0.79348962 -0.89630671  0.22056566
 -1.74372295 -0.01395001 -0.49132973  1.03001052 -0.80349939 -1.81582947
 -0.55988497 -0.11013804  0.25955221  1.60980051 -0.43491058  1.18291251
  1.37915488  0.97830079 -0.66986924 -0.94815144  0.94666376  0.90821201
 -0.43364748  0.18752882 -0.12840752 -0.47894364  0.01722541  0.27523142
  0.72487711 -0.24640747 -1.19040439  0.1341047  -0.77972765 -0.15197812
  0.72248378  0.30616393  0.6062591   1.66592097 -1.17643551  0.61701693
  1.28615655 -1.90219224 -0.81633124  1.66260656 -0.37252684  1.45160555
 -1.83656128 -1.67497274 -1.12176742 -1.13124765 -0.10227185  0.24044259
 -1.67980698  1.69976015 -0.76725197  1.26874735 -1.34343109 -0.7835702
  1.7943309  -1.55818121 -0.535411    0.86781153  0.18162866 -1.98503305
 -0.79687519 -1.65912288 -1.9797152  -1.77585362 -0.11279535  0.52396715
 -0.14416692 -0.37058992  1.96565008 -1.12440462]

Y values for test model
[ 4.12192431 -0.17417647 -6.74257261  1.03172562 -0.43189049 -4.28473618
 -0.09840278 -1.40931767 -0.24907773  0.11587867  2.71452802 -4.52756682
  0.24332156  0.08700194 -1.00897756 -2.77276223  0.37458198 -0.46937507
  0.60368926  0.21279329  0.98988425  5.2534209   0.17783892  3.34798041
 -0.25026427 -0.51212614  1.74716145 -0.83872697 -0.73106832 -0.9394545
 -5.8138794   0.10753131 -0.38195851  0.83054301 -0.24522186 -5.92785354
 -1.32816901  0.35865955 -0.62476694  4.85021925  0.16008101  1.52717049
  3.30055979  1.97428001  0.03285316 -1.3592396   0.94667135  0.85457441
 -0.85774182 -0.5487225   0.22310556  1.03029219  0.84630043  0.46293533
  0.1116444  -0.20329821 -1.90307074  0.79274166 -1.3164083   1.0644095
  0.37110149 -0.11684461  0.50405872  4.98920002 -1.5241405   0.30834667
  2.38233313 -6.11249155  0.29911969  4.00741532 -0.14646177  3.07590754
 -6.36159782 -4.89025508 -0.92519104 -1.29334404  0.22730868 -0.16595117
```

```
-5.16382527  5.51674012 -1.2242299   2.38387661 -2.2465421  -0.54033285
 5.71376945 -3.96031298  0.65783367  0.89745352 -0.55500288 -8.45764318
-0.62762492 -4.85036953 -7.44367139 -5.66501819  0.41399123 -0.3330181
 0.22710717  0.77952203  7.21281982 -1.35771285]
```



5. The Results To find the best model, we compare the root mean square error for each polynomial regression model. The model with the lowest error is the best!

In [186]:
```
#getting the root mean square error for each polynomial regression model

prediction_Y1 = m1.predict(Xtest[:, np.newaxis])
Square1 = sqrt(mean_squared_error(Ytest,prediction_Y1))
print('Root mean square error for Degree 1')
print(Square1)

prediction_Y2 = m2.predict(Xtest[:, np.newaxis])
Square2 = sqrt(mean_squared_error(Ytest,prediction_Y2))
print('\nRoot mean square error for Degree 2')
print(Square2)

prediction_Y3 = m3.predict(Xtest[:, np.newaxis])
Square3 = sqrt(mean_squared_error(Ytest,prediction_Y3))
print('\nRoot mean square error for Degree 3')
print(Square3)

prediction_Y5 = m5.predict(Xtest[:, np.newaxis])
Square5 = sqrt(mean_squared_error(Ytest,prediction_Y5))
print('\nRoot mean square error for Degree 5')
print(Square5)


prediction_Y9 = m9.predict(Xtest[:, np.newaxis])
Square9 = sqrt(mean_squared_error(Ytest,prediction_Y9))
print('\nRoot mean square error for Degree 9')
print(Square9)

print()

#we compare the root mean square error for each polynomial regression model
#The model with the lowest error is the best!

if min(Square1,Square2,Square3,Square5,Square9)==Square1:
    print(f'Degree 1 polynomial regression model has lowest error {Square1}. Henc
elif min(Square1,Square2,Square3,Square5,Square9)==Square2:
    print(f'Degree 2 polynomial regression model has lowest error {Square2}. Henc
elif min(Square1,Square2,Square3,Square5,Square9)==Square3:
    print(f'Degree 3 polynomial regression model has lowest error {Square3}. Henc
elif min(Square1,Square2,Square3,Square5,Square9)==Square5:
    print(f'Degree 5 polynomial regression model has lowest error {Square5}. Henc
elif min(Square1,Square2,Square3,Square5,Square9)==Square9:
    print(f'Degree 9 polynomial regression model has lowest error {Square9}. Henc
```

```
Root mean square error for Degree 1
1.33256156196513

Root mean square error for Degree 2
1.3919437628868163

Root mean square error for Degree 3
0.469413421025787

Root mean square error for Degree 5
0.7016126121079768
```

```
Root mean square error for Degree 9
7.070311324496551

Degree 3 polynomial regression model has lowest error 0.469413421025787. Henc
e, Degree 3 is the best model!
```

In [ ]: