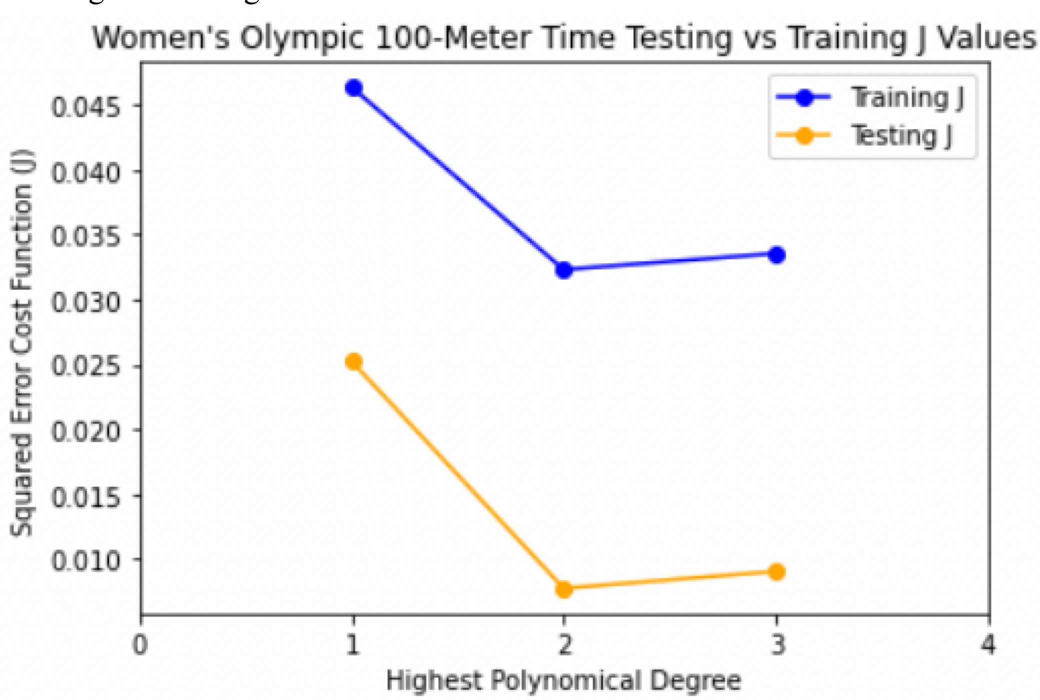
Sydney Trout
Dr. Li
CPSC 4430
9 February 2022

Project 2 Report

J Value Chart:

	Linear	Quadratic	Cubic
1234	0.0438	0.0293	0.0301
5	0.0591	0.0000	0.0000
1235	0.0382	0.0299	0.0310
4	0.0204	0.0000	0.0154
1245	0.0556	0.0300	0.0316
3	0.0073	0.0000	0.0000
1345	0.0473	0.0365	0.0378
2	0.0112	0.0038	0.0060
2345	0.0467	0.0357	0.0373
1	0.0281	0.0349	0.0238
Mean Train	0.0463	0.0323	0.0335
Mean Test	0.0252	0.0078	0.0091

Training and Testing J Value Plot:



The model I chose to predict winning race times was quadratic because the polynomial degree two has the lowest testing J value. This allowed me to calculate the final weights using the complete data set and the quadratic model. The final weights were:

$w_0 = 13.13071952$

$w_1 = -0.04324827$

$w_2 = 0.00020637$

Making the final hypothesis function:

$h_w(x) = 13.13071952 - 0.04324827x + 0.00020637x^2$

```
The final weights are:
w0 =  13.13071952
w1 =  -0.04324827
w2 =  0.00020637
The hw(x) function is:  13.13071952 +  -0.04324827 x + 0.00020637 x^2
```

Using this hypothesis function, the predicted women's Olympic 100-meter race record time for 2022 is 10.926 seconds

```
Enter a year to predict the winning Women's Olympic 100-Meter Race Time for that year: 2022
The predicted winning time for 2022 is 10.926 seconds
```

Python Console:

Console 1/A

```
In [8]: runfile('/Users/sydneytrout/Desktop/spring 2022/cpsc 4430/trout_sydney_P2.py', wdir='/Users/
sydneytrout/Desktop/spring 2022/cpsc 4430')

Enter the name of your file: W100MTimes.txt
             Linear  Quadratic  Cubic
     1234    0.0438   0.0293   0.0301
     5       0.0591   0.0000   0.0000
     1235    0.0382   0.0299   0.0310
     4       0.0204   0.0000   0.0154
     1245    0.0556   0.0300   0.0316
     3       0.0073   0.0000   0.0000
     1345    0.0473   0.0365   0.0378
     2       0.0112   0.0038   0.0060
     2345    0.0467   0.0357   0.0373
     1       0.0281   0.0349   0.0238
Mean Train   0.0463   0.0323   0.0335
Mean Test    0.0252   0.0078   0.0091

The best model to predict the Women's Olympic 100-Meter Times is Quadratic because it has the
smallest testing J value
The final weights are:
w0 =  13.13071952
w1 =  -0.04324827
w2 =  0.00020637
The hw(x) function is:  13.13071952 +  -0.04324827 x + 0.00020637 x^2

Enter a year to predict the winning Women's Olympic 100-Meter Race Time for that year: 2022
The predicted winning time for 2022 is 10.926 seconds

In [9]:
```

IPython console History

Code Copy:

```
#import libraries
import numpy as np
import matplotlib.pyplot as plt
import random

#J value chart that will hold all data
chart = np.zeros((10,3))

#function that takes in the x and y data and caluculates the J value
def calculateJ(xDataLin, xDataCub, xDataQuad, yData, row):
    A = np.linalg.pinv(np.dot(xDataLin.T, xDataLin))
    B = np.dot(xDataLin.T, yData)
    wLinearValue = np.dot(A, B)

    A = np.linalg.pinv(np.dot(xDataQuad.T, xDataQuad))
    B = np.dot(xDataQuad.T, yData)
    wQuadraticValue = np.dot(A, B)

    A = np.linalg.pinv(np.dot(xDataCub.T, xDataCub))
    B = np.dot(xDataCub.T, yData)
    wCubicValue = np.dot(A, B)

    m = len(xDataLin)

    A = np.dot(xDataLin, wLinearValue) - yData
    J = (1/m) * np.dot(A.T, A)
    chart[row,0] = J

    A = np.dot(xDataQuad, wQuadraticValue) - yData
    J = (1/m) * np.dot(A.T, A)
    chart[row,1] = J

    A = np.dot(xDataCub, wCubicValue) - yData
    J = (1/m) * np.dot(A.T, A)
    chart[row,2] = J

#Takes user input for file name and opens file
fileName = input("Enter the name of your file: ")
```

```
dataFile = open(fileName, "r")

#Reads the number of rows and features from the file
readData = dataFile.readline()
fileInfo = readData.split("\t")
rows = int(fileInfo[0])
features = int(fileInfo[1])

xDataOrder = []
yDataOrder = []

#Initializing the bins
xBinsLinear1 = np.zeros((4,features+1))
xBinsLinear2 = np.zeros((4,features+1))
xBinsLinear3 = np.zeros((4,features+1))
xBinsLinear4 = np.zeros((4,features+1))
xBinsLinear5 = np.zeros((3,features+1))
xBinsLinearAll = np.zeros((rows,features+1))

xBinsQuadratic1 = np.zeros((4,features+2))
xBinsQuadratic2 = np.zeros((4,features+2))
xBinsQuadratic3 = np.zeros((4,features+2))
xBinsQuadratic4 = np.zeros((4,features+2))
xBinsQuadratic5 = np.zeros((3,features+2))
xBinsQuadraticAll = np.zeros((rows,features+2))

xBinsCubic1 = np.zeros((4,features+3))
xBinsCubic2 = np.zeros((4,features+3))
xBinsCubic3 = np.zeros((4,features+3))
xBinsCubic4 = np.zeros((4,features+3))
xBinsCubic5 = np.zeros((3,features+3))
xBinsCubicAll = np.zeros((rows,features+3))

yBins1 = np.zeros((4,features))
yBins2 = np.zeros((4,features))
yBins3 = np.zeros((4,features))
yBins4 = np.zeros((4,features))
yBins5 = np.zeros((3,features))
yBinsAll = np.zeros((rows,features))
```

```
#Reading in the x and y data
for i in range(rows):
    readData = dataFile.readline()
    data = readData.split("\t")
    xDataOrder.append(int(data[0]))
    yDataOrder.append(float(data[1]))

#Randomizing the data using a seed so it remains the same with every run
random.seed(10)
randomize = list(zip(xDataOrder,yDataOrder))
random.shuffle(randomize)
xData, yData = zip(*randomize)

#Placing the data into respective folds
for i in range(rows):
    if (i < 4):
        xBinsLinear1[i,0] = 1
        xBinsLinear1[i,1] = xData[i]
        xBinsQuadratic1[i,0] = 1
        xBinsQuadratic1[i,1] = xData[i]
        xBinsQuadratic1[i,2] = xData[i] ** 2
        xBinsCubic1[i,0] = 1
        xBinsCubic1[i,1] = xData[i]
        xBinsCubic1[i,2] = xData[i] ** 2
        xBinsCubic1[i,3] = xData[i] ** 3
        yBins1[i] = yData[i]
    elif (i < 8):
        xBinsLinear2[i-4,0] = 1
        xBinsLinear2[i-4,1] = xData[i]
        xBinsQuadratic2[i-4,0] = 1
        xBinsQuadratic2[i-4,1] = xData[i]
        xBinsQuadratic2[i-4,2] = xData[i] ** 2
        xBinsCubic2[i-4,0] = 1
        xBinsCubic2[i-4,1] = xData[i]
        xBinsCubic2[i-4,2] = xData[i] ** 2
        xBinsCubic2[i-4,3] = xData[i] ** 3
        yBins2[i-4] = yData[i]
    elif (i < 12):
        xBinsLinear3[i-8,0] = 1
        xBinsLinear3[i-8,1] = xData[i]
```

```
        xBinsQuadratic3[i-8,0] = 1
        xBinsQuadratic3[i-8,1] = xData[i]
        xBinsQuadratic3[i-8,2] = xData[i] ** 2
        xBinsCubic3[i-8,0] = 1
        xBinsCubic3[i-8,1] = xData[i]
        xBinsCubic3[i-8,2] = xData[i] ** 2
        xBinsCubic3[i-8,3] = xData[i] ** 3
        yBins3[i-8] = yData[i]
    elif (i < 16):
        xBinsLinear4[i-12,0] = 1
        xBinsLinear4[i-12,1] = xData[i]
        xBinsQuadratic4[i-12,0] = 1
        xBinsQuadratic4[i-12,1] = xData[i]
        xBinsQuadratic4[i-12,2] = xData[i] ** 2
        xBinsCubic4[i-12,0] = 1
        xBinsCubic4[i-12,1] = xData[i]
        xBinsCubic4[i-12,2] = xData[i] ** 2
        xBinsCubic4[i-12,3] = xData[i] ** 3
        yBins4[i-12] = yData[i]
    else:
        xBinsLinear5[i-16,0] = 1
        xBinsLinear5[i-16,1] = xData[i]
        xBinsQuadratic5[i-16,0] = 1
        xBinsQuadratic5[i-16,1] = xData[i]
        xBinsQuadratic5[i-16,2] = xData[i] ** 2
        xBinsCubic5[i-16,0] = 1
        xBinsCubic5[i-16,1] = xData[i]
        xBinsCubic5[i-16,2] = xData[i] ** 2
        xBinsCubic5[i-16,3] = xData[i] ** 3
        yBins5[i-16] = yData[i]
    xBinsLinearAll[i,0] = 1
    xBinsLinearAll[i,1] = xData[i]
    xBinsQuadraticAll[i,0] = 1
    xBinsQuadraticAll[i,1] = xData[i]
    xBinsQuadraticAll[i,2] = xData[i] ** 2
    xBinsCubicAll[i,0] = 1
    xBinsCubicAll[i,1] = xData[i]
    xBinsCubicAll[i,2] = xData[i] ** 2
    xBinsCubicAll[i,3] = xData[i] ** 3
    yBinsAll[i] = yData[i]
```

```
#Calculating the J values
k1234Lin = np.append(xBinsLinear1,xBinsLinear2,axis = 0)
k1234Lin = np.append(k1234Lin,xBinsLinear3,axis = 0)
k1234Lin = np.append(k1234Lin,xBinsLinear4,axis = 0)
k1234Quad = np.append(xBinsQuadratic1,xBinsQuadratic2,axis = 0)
k1234Quad = np.append(k1234Quad,xBinsQuadratic3,axis = 0)
k1234Quad = np.append(k1234Quad,xBinsQuadratic4,axis = 0)
k1234Cub = np.append(xBinsCubic1,xBinsCubic2,axis = 0)
k1234Cub = np.append(k1234Cub,xBinsCubic3,axis = 0)
k1234Cub = np.append(k1234Cub,xBinsCubic4,axis = 0)
k1234y = np.append(yBins1,yBins2, axis = 0)
k1234y = np.append(k1234y,yBins3, axis = 0)
k1234y = np.append(k1234y,yBins4, axis = 0)
calculateJ(k1234Lin, k1234Quad, k1234Cub, k1234y, 0)
calculateJ(xBinsLinear5, xBinsQuadratic5, xBinsCubic5, yBins5, 1)

k1235Lin = np.append(xBinsLinear1,xBinsLinear2,axis = 0)
k1235Lin = np.append(k1235Lin,xBinsLinear3,axis = 0)
k1235Lin = np.append(k1235Lin,xBinsLinear5,axis = 0)
k1235Quad = np.append(xBinsQuadratic1,xBinsQuadratic2,axis = 0)
k1235Quad = np.append(k1235Quad,xBinsQuadratic3,axis = 0)
k1235Quad = np.append(k1235Quad,xBinsQuadratic5,axis = 0)
k1235Cub = np.append(xBinsCubic1,xBinsCubic2,axis = 0)
k1235Cub = np.append(k1235Cub,xBinsCubic3,axis = 0)
k1235Cub = np.append(k1235Cub,xBinsCubic5,axis = 0)
k1235y = np.append(yBins1,yBins2, axis = 0)
k1235y = np.append(k1235y,yBins3, axis = 0)
k1235y = np.append(k1235y,yBins5, axis = 0)
calculateJ(k1235Lin, k1235Quad, k1235Cub, k1235y, 2)
calculateJ(xBinsLinear4, xBinsQuadratic4, xBinsCubic4, yBins4, 3)

k1245Lin = np.append(xBinsLinear1,xBinsLinear2,axis = 0)
k1245Lin = np.append(k1245Lin,xBinsLinear4,axis = 0)
k1245Lin = np.append(k1245Lin,xBinsLinear5,axis = 0)
k1245Quad = np.append(xBinsQuadratic1,xBinsQuadratic2,axis = 0)
k1245Quad = np.append(k1245Quad,xBinsQuadratic4,axis = 0)
k1245Quad = np.append(k1245Quad,xBinsQuadratic5,axis = 0)
k1245Cub = np.append(xBinsCubic1,xBinsCubic2,axis = 0)
k1245Cub = np.append(k1245Cub,xBinsCubic4,axis = 0)
```

```
k1245Cub = np.append(k1245Cub,xBinsCubic5,axis = 0)
k1245y = np.append(yBins1,yBins2, axis = 0)
k1245y = np.append(k1245y,yBins4, axis = 0)
k1245y = np.append(k1245y,yBins5, axis = 0)
calculateJ(k1245Lin, k1245Quad, k1245Cub, k1245y, 4)
calculateJ(xBinsLinear3, xBinsQuadratic3, xBinsCubic3, yBins3, 5)

k1345Lin = np.append(xBinsLinear1,xBinsLinear3,axis = 0)
k1345Lin = np.append(k1345Lin,xBinsLinear4,axis = 0)
k1345Lin = np.append(k1345Lin,xBinsLinear5,axis = 0)
k1345Quad = np.append(xBinsQuadratic1,xBinsQuadratic3,axis = 0)
k1345Quad = np.append(k1345Quad,xBinsQuadratic4,axis = 0)
k1345Quad = np.append(k1345Quad,xBinsQuadratic5,axis = 0)
k1345Cub = np.append(xBinsCubic1,xBinsCubic3,axis = 0)
k1345Cub = np.append(k1345Cub,xBinsCubic4,axis = 0)
k1345Cub = np.append(k1345Cub,xBinsCubic5,axis = 0)
k1345y = np.append(yBins1,yBins3, axis = 0)
k1345y = np.append(k1345y,yBins4, axis = 0)
k1345y = np.append(k1345y,yBins5, axis = 0)
calculateJ(k1345Lin, k1345Quad, k1345Cub, k1345y, 6)
calculateJ(xBinsLinear2, xBinsQuadratic2, xBinsCubic2, yBins2, 7)

k2345Lin = np.append(xBinsLinear2,xBinsLinear3,axis = 0)
k2345Lin = np.append(k2345Lin,xBinsLinear4,axis = 0)
k2345Lin = np.append(k2345Lin,xBinsLinear5,axis = 0)
k2345Quad = np.append(xBinsQuadratic2,xBinsQuadratic3,axis = 0)
k2345Quad = np.append(k2345Quad,xBinsQuadratic4,axis = 0)
k2345Quad = np.append(k2345Quad,xBinsQuadratic5,axis = 0)
k2345Cub = np.append(xBinsCubic2,xBinsCubic3,axis = 0)
k2345Cub = np.append(k2345Cub,xBinsCubic4,axis = 0)
k2345Cub = np.append(k2345Cub,xBinsCubic5,axis = 0)
k2345y = np.append(yBins2,yBins3, axis = 0)
k2345y = np.append(k2345y,yBins4, axis = 0)
k2345y = np.append(k2345y,yBins5, axis = 0)
calculateJ(k2345Lin, k2345Quad, k2345Cub, k2345y, 8)
calculateJ(xBinsLinear1, xBinsQuadratic1, xBinsCubic1, yBins1, 9)

#Calculating the mean values
meanTrainLin = (chart[0,0] + chart[2,0] + chart[4,0] + chart[6,0] + chart[8,0]) / 5
meanTestLin = (chart[1,0] + chart[3,0] + chart[5,0] + chart[7,0] + chart[9,0]) / 5
```

```
meanTrainQuad = (chart[0,1] + chart[2,1] + chart[4,1] + chart[6,1] + chart[8,1]) / 5
meanTestQuad = (chart[1,1] + chart[3,1] + chart[5,1] + chart[7,1] + chart[9,1]) / 5
meanTrainCub = (chart[0,2] + chart[2,2] + chart[4,2] + chart[6,2] + chart[8,2]) / 5
meanTestCub = (chart[1,2] + chart[3,2] + chart[5,2] + chart[7,2] + chart[9,2]) / 5

#Printing the J value chart
print("\t\t   \t Linear  Quadratic  Cubic")
print("\t 1234 \t","{0:.4f}".format(chart[0,0])," ","{0:.4f}".format(chart[0,1]),"
","{0:.4f}".format(chart[0,2]))
print("\t 5    \t","{0:.4f}".format(chart[1,0])," ","{0:.4f}".format(chart[1,1]),"
","{0:.4f}".format(chart[1,2]))
print("\t 1235 \t","{0:.4f}".format(chart[2,0])," ","{0:.4f}".format(chart[2,1]),"
","{0:.4f}".format(chart[2,2]))
print("\t 4    \t","{0:.4f}".format(chart[3,0])," ","{0:.4f}".format(chart[3,1]),"
","{0:.4f}".format(chart[3,2]))
print("\t 1245 \t","{0:.4f}".format(chart[4,0])," ","{0:.4f}".format(chart[4,1]),"
","{0:.4f}".format(chart[4,2]))
print("\t 3    \t","{0:.4f}".format(chart[5,0])," ","{0:.4f}".format(chart[5,1]),"
","{0:.4f}".format(chart[5,2]))
print("\t 1345 \t","{0:.4f}".format(chart[6,0])," ","{0:.4f}".format(chart[6,1]),"
","{0:.4f}".format(chart[6,2]))
print("\t 2    \t","{0:.4f}".format(chart[7,0])," ","{0:.4f}".format(chart[7,1]),"
","{0:.4f}".format(chart[7,2]))
print("\t 2345 \t","{0:.4f}".format(chart[8,0])," ","{0:.4f}".format(chart[8,1]),"
","{0:.4f}".format(chart[8,2]))
print("\t 1    \t","{0:.4f}".format(chart[9,0])," ","{0:.4f}".format(chart[9,1]),"
","{0:.4f}".format(chart[9,2]))
print("Mean Train\t","{0:.4f}".format(meanTrainLin)," ","{0:.4f}".format(meanTrainQuad),"
","{0:.4f}".format(meanTrainCub))
print("Mean Test \t","{0:.4f}".format(meanTestLin)," ","{0:.4f}".format(meanTestQuad),"
","{0:.4f}".format(meanTestCub))

#Plotting the testing and training J values
plt.plot([1,2,3], [meanTrainLin,meanTrainQuad,meanTrainCub], color = "blue", marker = "o",
label = "Training J")
plt.plot([1,2,3], [meanTestLin,meanTestQuad,meanTestCub], color = "orange", marker = "o",
label = "Testing J")
x = [0, 1, 2, 3, 4]
plt.xticks(x)
plt.xlabel("Highest Polynomical Degree")
```

```
plt.ylabel("Squared Error Cost Function (J)")
plt.title("Women's Olympic 100-Meter Time Testing vs Training J Values")
plt.legend(loc='upper right')
plt.show()

#Determines which model is best and takes user input to predict a winning time
#Due to the random seed, it will always be Quadratic, but may change if seed is removed
if(((meanTestLin) < (meanTestQuad)) and ((meanTestLin) < (meanTestCub))):
    A = np.linalg.pinv(np.dot(xBinsLinearAll.T, xBinsLinearAll))
    B = np.dot(xBinsLinearAll.T, yBinsAll)
    wComplete = np.dot(A, B)
    print("\nThe best model to predict the Women's Olympic 100-Meter Times is Linear because it
has the smallest testing J value")
    print("The final weights are:")
    print("w0 = ", "{0:.8f}".format(wComplete[0,0]))
    print("w1 = ", "{0:.8f}".format(wComplete[1,0]))
    print("The hw(x) function is: ", "{0:.8f}".format(wComplete[0,0]), "+ ",
"{0:.8f}".format(wComplete[1,0]), "x")
    year = input("Enter a year to predict the winning Women's Olympic 100-Meter Race Time for
that year: ")
    year = int(year)
    yearAbv = year - 1900
    time = (wComplete[0,0]) + (wComplete[1,0]*yearAbv)
    print("The predicted winning time for", year, "is", "{0:.3f}".format(time), "seconds")
elif(((meanTestQuad) < (meanTestLin)) and ((meanTestQuad) < (meanTestCub))):
    A = np.linalg.pinv(np.dot(xBinsQuadraticAll.T, xBinsQuadraticAll))
    B = np.dot(xBinsQuadraticAll.T, yBinsAll)
    wComplete = np.dot(A, B)
    print("\nThe best model to predict the Women's Olympic 100-Meter Times is Quadratic
because it has the smallest testing J value")
    print("The final weights are:")
    print("w0 = ", "{0:.8f}".format(wComplete[0,0]))
    print("w1 = ", "{0:.8f}".format(wComplete[1,0]))
    print("w2 = ", "{0:.8f}".format(wComplete[2,0]))
    print("The hw(x) function is: ", "{0:.8f}".format(wComplete[0,0]), "+ ",
"{0:.8f}".format(wComplete[1,0]), "x +", "{0:.8f}".format(wComplete[2,0]), "x^2")
    year = input("Enter a year to predict the winning Women's Olympic 100-Meter Race Time for
that year: ")
    year = int(year)
    yearAbv = year - 1900
```

```
    time = (wComplete[0,0]) + (wComplete[1,0]*yearAbv) + 
(wComplete[2,0]*(yearAbv*yearAbv))
    print("The predicted winning time for", year, "is", "{0:.3f}".format(time), "seconds")
else:
    A = np.linalg.pinv(np.dot(xBinsCubicAll.T, xBinsCubicAll))
    B = np.dot(xBinsCubicAll.T, yBinsAll)
    wComplete = np.dot(A, B)
    print("\nThe best model to predict the Women's Olympic 100-Meter Times is Cubic because it 
has the smallest testing J value")
    print("The final weights are:")
    print("w0 = ", "{0:.8f}".format(wComplete[0,0]))
    print("w1 = ", "{0:.8f}".format(wComplete[1,0]))
    print("w2 = ", "{0:.8f}".format(wComplete[2,0]))
    print("w3 = ", "{0:.8f}".format(wComplete[3,0]))
    print("The hw(x) function is: ", "{0:.8f}".format(wComplete[0,0]), "+ ", 
"{0:.8f}".format(wComplete[1,0]), "x +", "{0:.8f}".format(wComplete[2,0]), "x^2 +", 
"{0:.8f}".format(wComplete[3,0]), "x^3")
    year = input("Enter a year to predict the winning Women's Olympic 100-Meter Race Time for 
that year: ")
    year = int(year)
    yearAbv = year - 1900
    time = (wComplete[0,0]) + (wComplete[1,0]*yearAbv) + 
(wComplete[2,0]*(yearAbv*yearAbv) + (wComplete[3,0]*(yearAbv*yearAbv*yearAbv)))
    print("The predicted winning time for", year, "is", "{0:.3f}".format(time), "seconds")
```