CS 237 Fall 2018, Homework 11

Due date: Friday December 6th at 11:59 pm (10% off if up to 24 hours late) via Gradescope

General Instructions

Please complete this notebook by filling in solutions where indicated. Be sure to "Run All" from the Cell menu before submitting.

You may use ordinary ASCII text to write your solutions, or (preferably) Latex. A nice introduction to Latex in Jupyter notebooks may be found here: http://data-blog.udacity.com/posts/2016/10/latex-primer/ (http://data-blog.udacity.com/posts/2016/10/latex-primer/)

As with previous homeworks, just upload a PDF file of this notebook. Instructions for converting to PDF may be found on the class web page right under the link for homework 1.

In [2]:
```
# General useful imports
import numpy as np
from numpy import arange,linspace,mean, var, std, corrcoef, transpose, ones,log
from numpy.linalg import inv
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.mlab as mlab
from numpy.random import random, randint, uniform
import math
from collections import Counter
import pandas as pd
%matplotlib inline

# Basic Numpy statistical functions

X = [1,2,3]

# mean of a list
mean(X)             # might need to use np.mean, np.var, and np.std

# population variance
var(X)

# sample variance    ddof = delta degrees of freedom, df = len(X) - ddof
var(X,ddof=1)

# population standard deviation
std(X)

# sample standard deviation
std(X,ddof=1)


# Scipy statistical functions

# Scipy Stats Library Functions, see:
# https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.statistics.html


# Calculate the correlation coefficient rho(X,Y)


def rho(X,Y):
    return corrcoef(X,Y)[0][1]

def R2(X,Y):
    return corrcoef(X,Y)[0][1] ** 2

# Utility functions


# Round to 2 decimal places
def round2(x):
    return round(float(x)+0.00000000001,2)

# Round to 4 decimal places
def round4(x):
    return round(float(x)+0.00000000001,4)

def round4List(X):
    return [round(float(x)+0.00000000001,4) for x in X]

def probToPercent(p):
    pc = p*100
    if round(pc) == pc:
        return str(round(pc)) + "%"
    else:
        return str(round(pc,2))+ "%"
```

Homework 11 General Instructions

This homework contains programming problems with some commentary on interpreting the results. Its goal is simply to exercise your understanding of regression and Monte Carlo algorithms, as covered in lecture before Thanksgiving break.

Problem One: Linear Regression

For this problem, in Part A you will fill in the template for a function that draws a scatterplot, the linear regression line, the midpoint, and the relevant statistics, as shown in lecture. In lab 9, problem 1, you drew a diagram just to practice using the matplotlib library, and you may want to refer to your solution to get started.

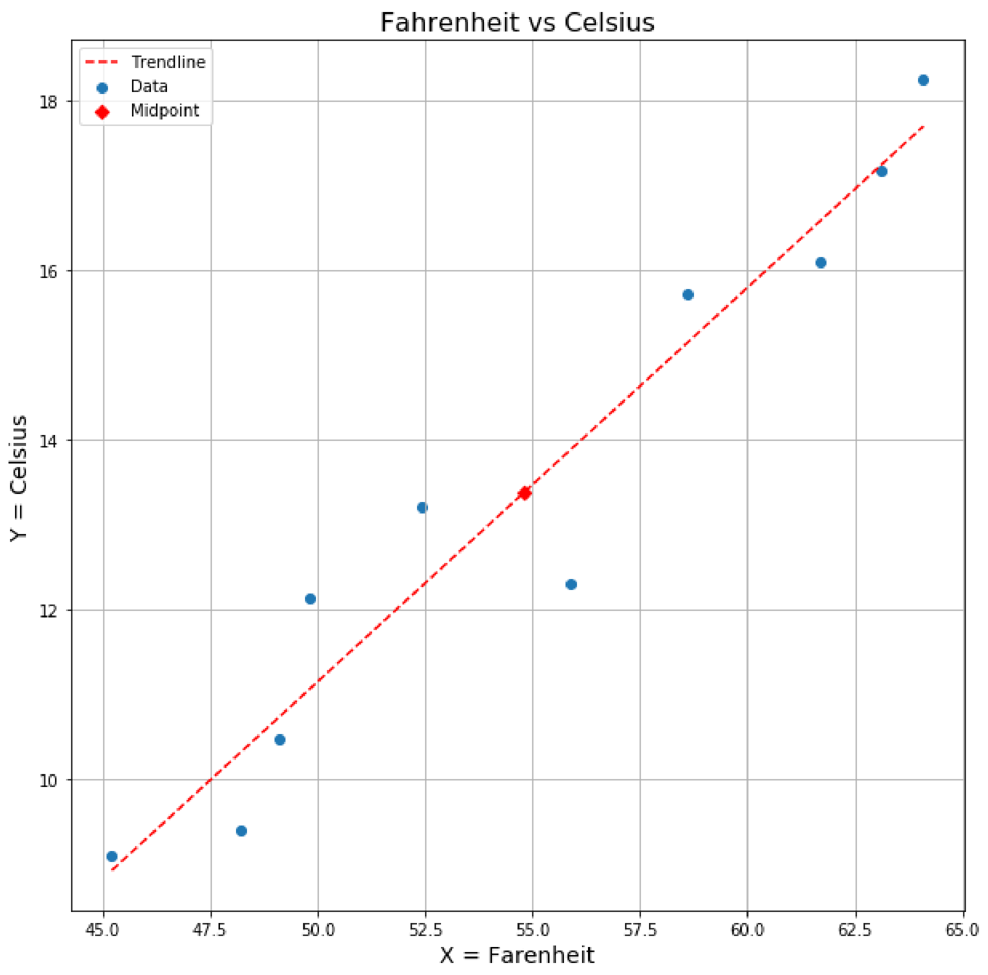
For Part B you will simply draw the regression line and residual plot for some data related to our class (anonymous of course!). For Part C you will think about what you are seeing and answer some questions about the data and whether a linear model is appropriate for this data.

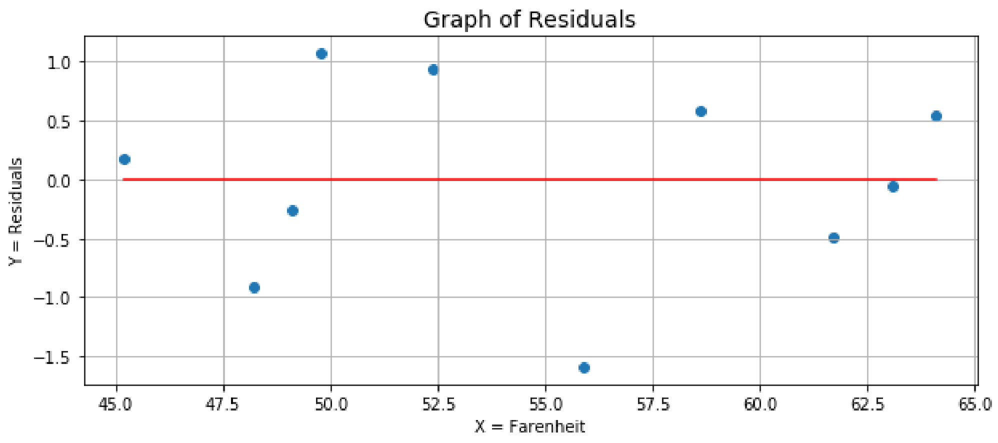
Part A (Farentheit vs Celsius)

Complete the following template to draw the example Farenheit/Celsius data from lecture

In [3]:
```
# Draw scatterplot for bivariate data and draw linear regression line
# with midpoint (mux,muy)

def LinearRegression(X,Y,titl="Linear Regression", xlab="X",ylab="Y"):

    n = len(X)

    # basic
    mux = mean(X)
    muy = mean(Y)
    sdx = std(X)
    sdy = std(Y)

    expXY = sum( [X[i]*Y[i] for i in range(n)] ) / n
    r = (expXY - mux*muy)/(sdx*sdy)

    r2 = r**2

    m = r * sdy / sdx
    b = muy - m*mux

    # Predicted values from regression line

    Yhat = [(m*X[i]+b) for i in range(n)]

    # Residuals

    E = [(Y[i] - Yhat[i]) for i in range(n)]

    # residual sum of squares -- deviations of data from line
    rss = sum( [ e**2 for e in E])

    # explained sum of squares -- deviation of line from mean of y
    egss = sum( [ (Yhat[i] - muy)**2 for i in range(n)])

    # total sum of squares -- deviation of data from mean of y
    tss = sum( [ (Y[i] - muy)**2 for i in range(n)] )

    plt.figure(figsize=(10,10))
    plt.grid()

    linex = [min(X),max(X)]
    liney = [(m*x+b) for x in linex]

    plt.scatter(X,Y,label="Data")
    plt.plot(linex,liney,'r--',label="Trendline")
    plt.scatter([mux],[muy],marker='D',color='red',label="Midpoint")           # add diamond sh
ape?  'rD'
    plt.title(titl,fontsize=16)
    plt.legend()
    plt.xlabel(xlab,fontsize=14)
    plt.ylabel(ylab,fontsize=14)
    plt.show()

    plt.figure(figsize=(10,4))
    plt.title("Graph of Residuals",fontsize=14)
    plt.grid()
    plt.xlabel(xlab)
    plt.ylabel("Y = Residuals")
    Yhat = [0 for x in X ]
    plt.scatter(X,E)
    plt.plot(X,Yhat,color='red')
    plt.show()



    print("\nmean(x):\t" + str(round4(mux)) + "\tstd(x):\t" + str(round4(sdx)))
    print("mean(y):\t" + str(round4(muy)) + "\tstd(y):\t" + str(round4(sdy)))
    print("\nrho:    " + str(round4(r)) + "\tR^2:    " + str(round4(r2)))
    print("\nResidual SS:    " + str(round4(rss)) + "\tExplained SS: " + str(round4(egss)) + "\t
Total SS:    " + str(round4(tss)))
```

```
    if(b >= 0):
        print("\nRegression Line: y = " + str(round4(m)) + " * x + " + str(round4(b)))
    else:
        print("\nRegression Line: y = " + str(round4(m)) + " * x - " + str(round4(-b)))

Xfarenheit = [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Ycelsius = [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.1, 17.1773, 18.2468]


LinearRegression(Xfarenheit,Ycelsius,"Fahrenheit vs Celsius",xlab="X = Farenheit",ylab="Y = Cel
sius")
```





```
mean(x):        54.81   std(x): 6.4623
mean(y):        13.3875 std(y): 3.1037

rho:   0.9664   R^2:   0.9339

Residual SS:   6.3631   Explained SS: 89.9635   Total SS:   96.3265

Regression Line: y = 0.4641 * x - 12.0519
```

In [4]:
```
studs = pd.read_csv('http://www.cs.bu.edu/fac/snyder/cs237/Homeworks,%20Labs,%20and%20Code/clas
sdata.csv')
print(studs[:10])
```

```
    GPA  Midterm  HWAvg
0  3.51     86.0  86.29
1  3.34     91.0  51.18
2  3.94     92.0  94.06
3  3.20     78.0  38.82
4  3.91     97.0  89.18
5  3.59     92.0  88.76
6  3.17     97.0  68.65
7  3.45     97.0  55.06
8  3.45     84.0  92.47
9  3.50     94.0  89.76
```

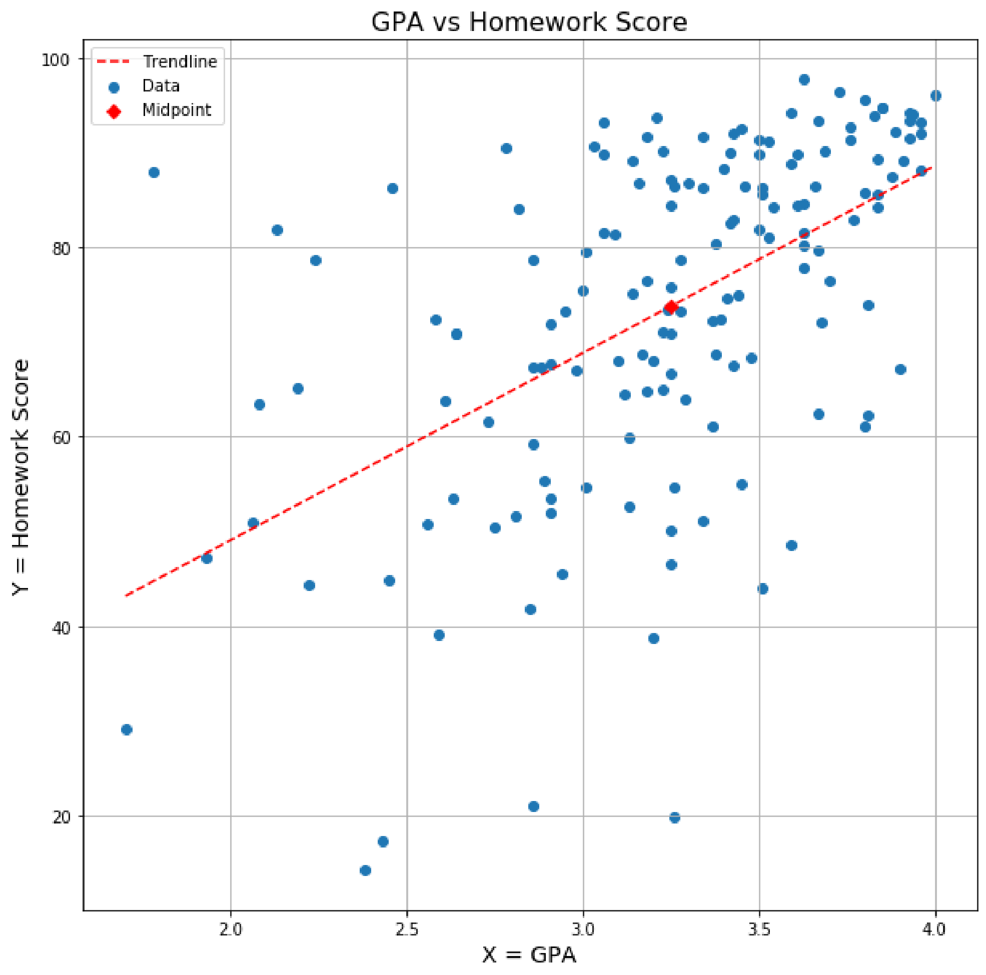
In [5]:
```
GPA = studs['GPA']
MID = studs['Midterm']
HWS = studs['HWAvg']
```

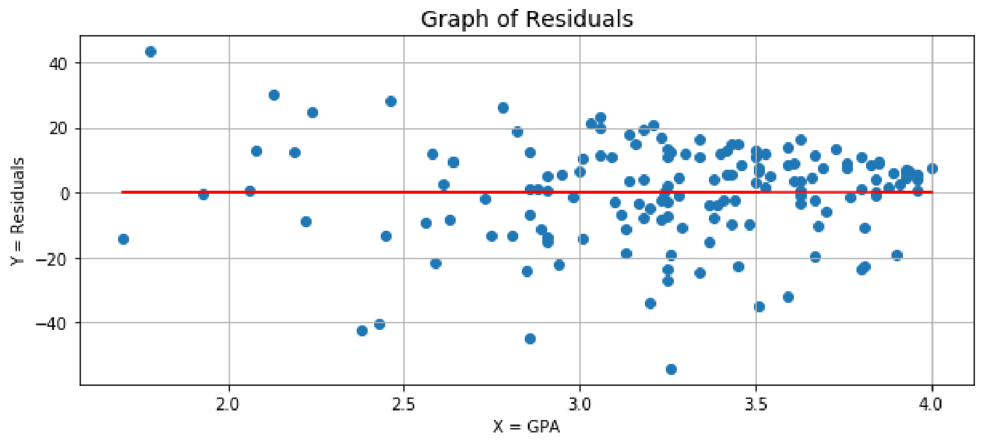
Part B (GPA vs HWS)

Apply the function from Part A to the following data:

```
X = GPA
Y = HWS
```

```
In [6]: LinearRegression(GPA,HWS,"GPA vs Homework Score",xlab="X = GPA",ylab="Y = Homework Score")
```





```
mean(x):        3.249   std(x): 0.4903
mean(y):        73.7222 std(y): 18.0138

rho:    0.5371    R^2:    0.2885

Residual SS:    35093.5105       Explained SS: 14229.7885        Total SS:    49323.299

Regression Line: y = 19.7359 * x + 9.6
```

Part C

Looking at the data and the statistics for Part B, answer the following questions to the best of your ability:

> (i) Just looking at the graphical display of the data, would you think there is a linear trend to this data, i.e., is there a correlation between GPA and homework scores in CS 237? What do you see?

> (ii) Now, what does the R^2 value tell you about fitting a linear model to this data?

Linear regression isn't always appropriate, and not only because of the R^2 score. Please read through the following page outlining the principal conditions necessary for linear regression:

https://www.statisticshowto.datasciencecentral.com/assumptions-conditions-for-regression/ (https://www.statisticshowto.datasciencecentral.com/assumptions-conditions-for-regression/)

> (iii) Do you see any other problems, related to the conditions you read about above? (Hint: look at the residual plot).) Can you think of any reason why this might be the case for this data set? (Hint: Just answer these by "eyeballing" the data, don't worry about doing a precise analysis.)

Answers to Part C Questions

(i) Offhand, there appears to be a general positive correlation between GPA and Homework Score; students who generally get good grades will get good grades on this particular aspect of grading.

(ii) The R^2 is very low, indicating that there is not much of a linear trend to this data.

(iii) The data appears to violate Homoscedasticity, because the variance of the residuals seems to decrease as the scores increase; this is probably because better students tend to be more regular in their habits and vary less in their attention to their schoolwork. It is less clear, but I am also not so sure that the residuals are normally distributed either.

Problem Two: Linear Regression

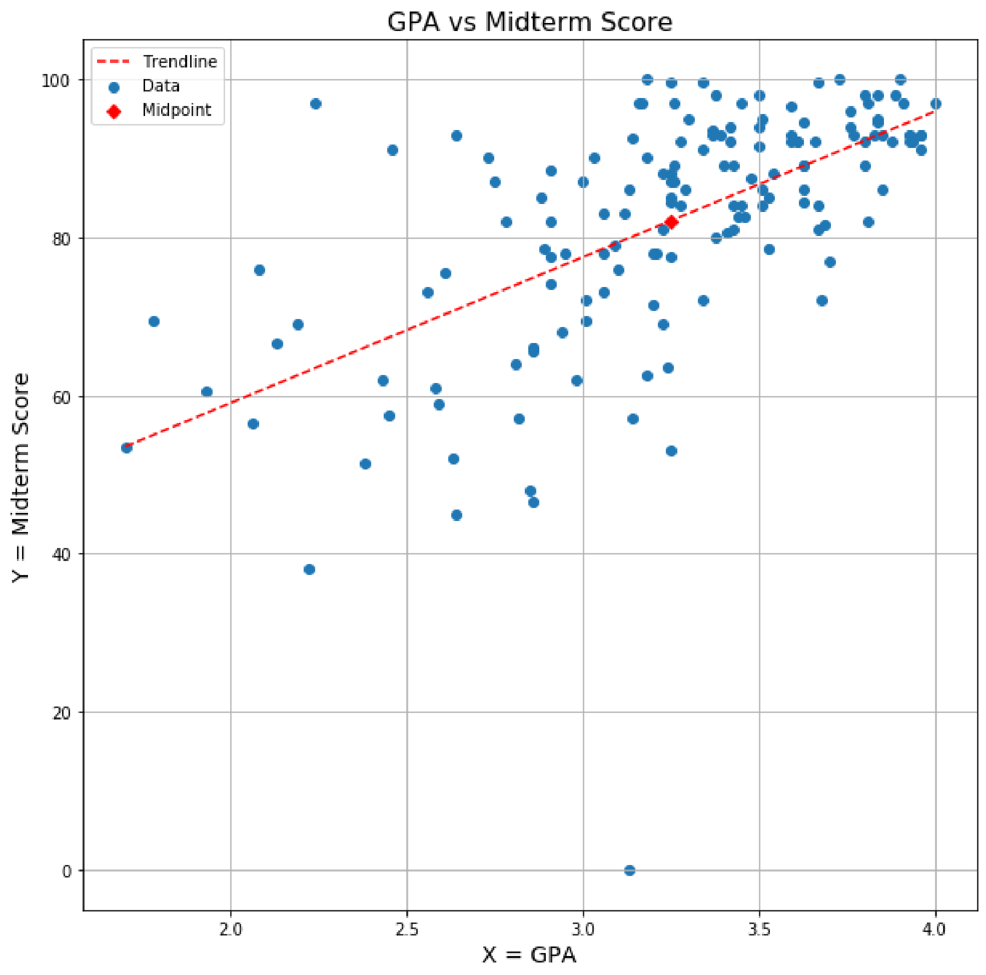
Now we will look at pairing the Midterm Score data with GPA. Again, you will apply your function from Problem One (A) to display the data and statistics, and then think about what you are seeing.

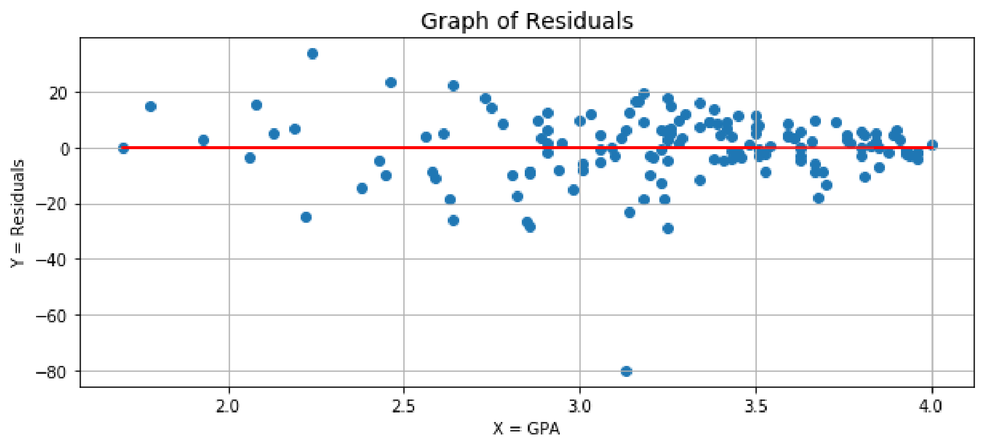
Part A (GPA vs MID)

Apply the function from Part A to the following data:

```
X = GPA
Y = MID
```

```
In [7]: LinearRegression(GPA,MID,"GPA vs Midterm Score",xlab="X = GPA",ylab="Y = Midterm Score")
```





```
mean(x):         3.249   std(x): 0.4903
mean(y):         82.0296 std(y): 15.1626

rho:   0.5952   R^2:   0.3542

Residual SS:   22567.1516       Explained SS: 12378.4652        Total SS:   34945.6168

Regression Line: y = 18.4073 * x + 22.2239
```

Part B (Dealing with Outliers: GPA2 vs MID2)

In Part A there is clearly a violation of one of the conditions for linear regression, in that there is an outlier in the Midterm data, because of a student in the data that didn't take the midterm and then at some point dropped the class. It is always a serious question whether an inconvenient data point is really an outlier, but in this case it is clear that if a student dropped the class, then the data on homeworks AND midterm are not really relevant to what we want to know, which is the relationship between GPA and performance in two parts of the class marks for those who complete the class.

The outlier occurs at index 128:

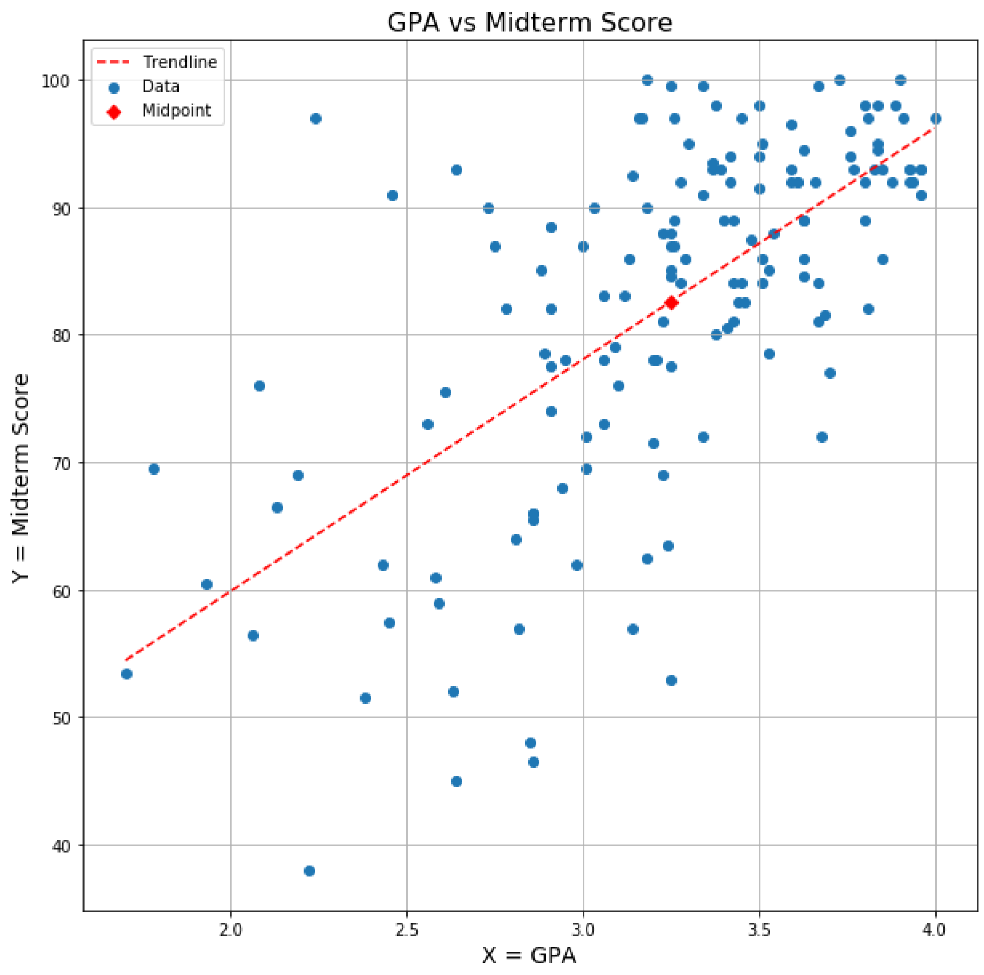
In [8]:
```
print(studs[125:132])
```

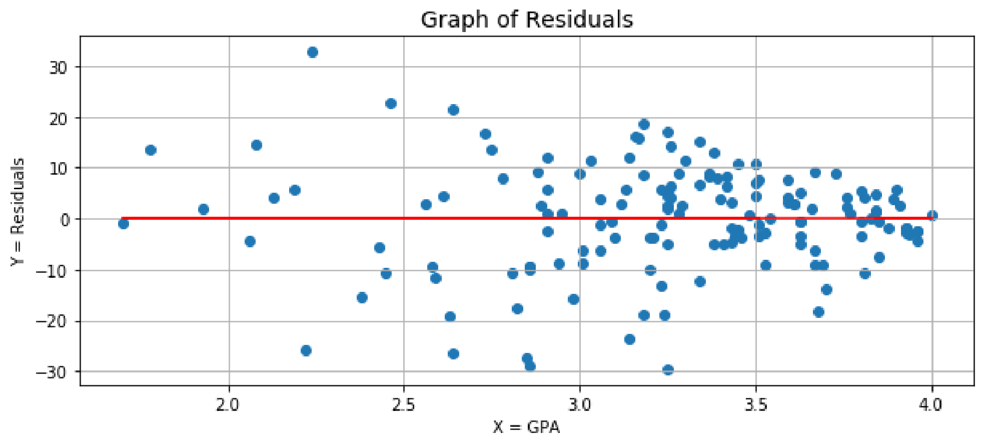
```
      GPA  Midterm  HWAvg
125  3.81     82.0  62.29
126  3.84     95.0  84.24
127  2.95     78.0  73.24
128  3.13      0.0  52.59
129  3.30     95.0  86.82
130  3.59     96.5  48.53
131  2.85     48.0  41.82
```

Eliminate this data point from ALL three lists and call the new sets GPA2, HWS2, and MID2. [Hint: check out the `del` function in Python.]

Now display the data for GPA2 vs MID2 (as in Part A, but with the outlier removed).

In [9]:
```
GPA2 = list(GPA)
del GPA2 [128]

HWS2 = list(HWS)
del HWS2 [128]

MID2 = list(MID)
del MID2 [128]

LinearRegression(GPA2,MID2,"GPA vs Midterm Score",xlab="X = GPA",ylab="Y = Midterm Score")
```



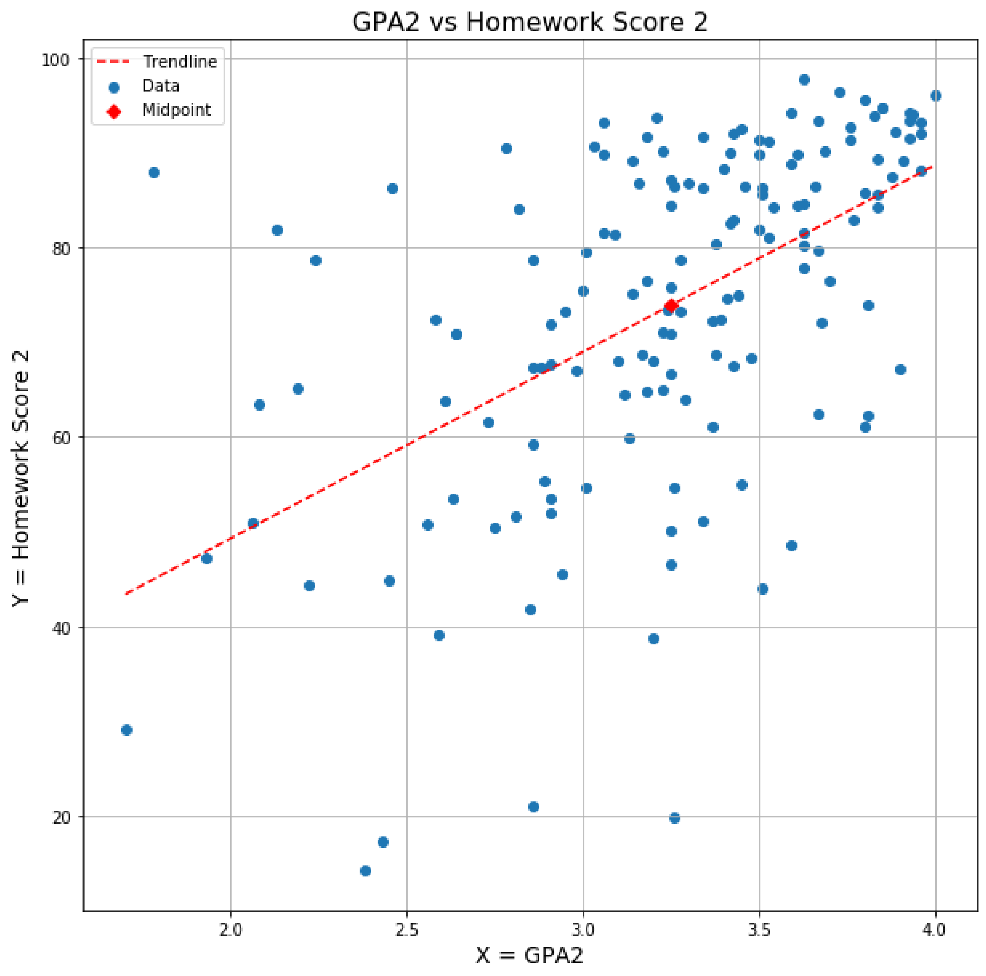


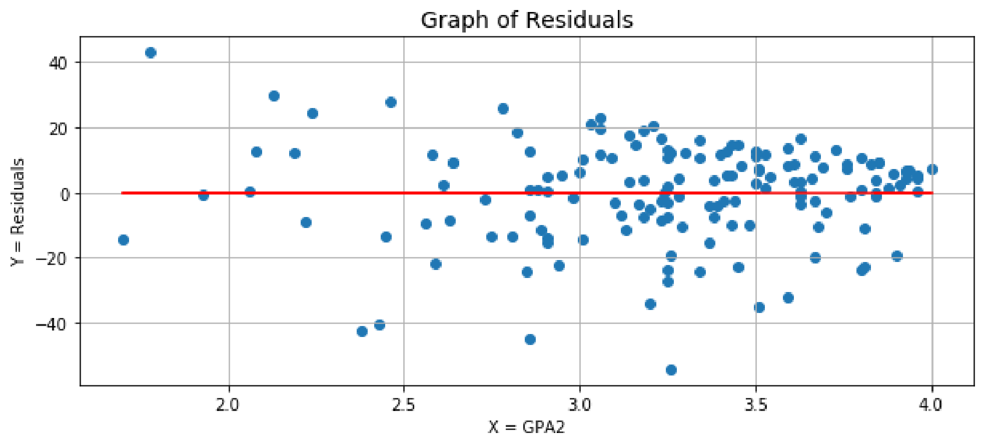
```
mean(x):         3.2498  std(x): 0.4918
mean(y):         82.5728 std(y): 13.6591

rho:   0.6533   R^2:   0.4268

Residual SS:   16148.1846      Explained SS: 12024.0141       Total SS:   28172.1987

Regression Line: y = 18.1454 * x + 23.6038
```

Part C (GPA2 vs HWS2)

Well, now we could wonder whether this new data (with one student eliminated who dropped the class) would change our results in Problem One. Redo the data display from Problem One (B) with this new data.

```
In [10]: LinearRegression(GPA2,HWS2,"GPA2 vs Homework Score 2",xlab="X = GPA2",ylab="Y = Homework Score
          2")
```





```
mean(x):        3.2498  std(x): 0.4918
mean(y):        73.8621 std(y): 17.9908

rho:    0.5378    R^2:    0.2892

Residual SS:    34738.2215       Explained SS: 14135.5514        Total SS:    48873.7729

Regression Line: y = 19.6743 * x + 9.9246
```

Part D

Now answer the following questions:

> (i) How much did removing the outlier affect our results with GPA vs MID (Problem 2 A compared with 2 B)? In particular, look at $var(y)$ and the R^2 value.

> (ii) What does this say (with this one example as evidence) about the sensitivity of R^2 to outliers in the data set?

> (iii) How what you saw in 2 C differ with what you saw with Problem One (B)?

Answers to Part D

(i) It made a significant difference in two respects:

```
Before removing sample 128:

mean(x):    3.249   std(x): 0.4903
mean(y):    82.0296 std(y): 15.1626

rho:   0.5952   R^2:   0.3542

Residual SS:   22567.1516   Explained SS: 12378.4652    Total SS:   34945.6168

Regression Line: y = 18.4073 * x + 22.2239

After removing the sample:

mean(x):    3.2498  std(x): 0.4918
mean(y):    82.5728 std(y): 13.6591

rho:   0.6533   R^2:   0.4268

Residual SS:   16148.1846   Explained SS: 12024.0141    Total SS:   28172.1987

Regression Line: y = 18.1454 * x + 23.6038
```

It significantly decreased the std(y) value, and increased the R^2 value by a factor of 1.4. The values for the slope and bias for the regression line were not affected as much.

(ii) Clearly the variance and the R^2 are affected by outliers, due to the fact that they are based on squaring the deviations, which exaggerates the outliers.

(iii) There was not as much of an effect:

```
Before (Problem 1 B):

mean(x):    3.249   std(x): 0.4903
mean(y):    73.7222 std(y): 18.0138

rho:   0.5371   R^2:   0.2885

Residual SS:   35093.5105   Explained SS: 14229.7885    Total SS:   49323.299

Regression Line: y = 19.7359 * x + 9.6

After (Problem 2 C):

mean(x):    3.2498  std(x): 0.4918
mean(y):    73.8621 std(y): 17.9908

rho:   0.5378   R^2:   0.2892

Residual SS:   34738.2215   Explained SS: 14135.5514    Total SS:   48873.7729

Regression Line: y = 19.6743 * x + 9.9246
```

Some small changes throughout, you can see `std(y)` went down slightly, and R^2 went up slightly.

Problem Three: Linear Regression

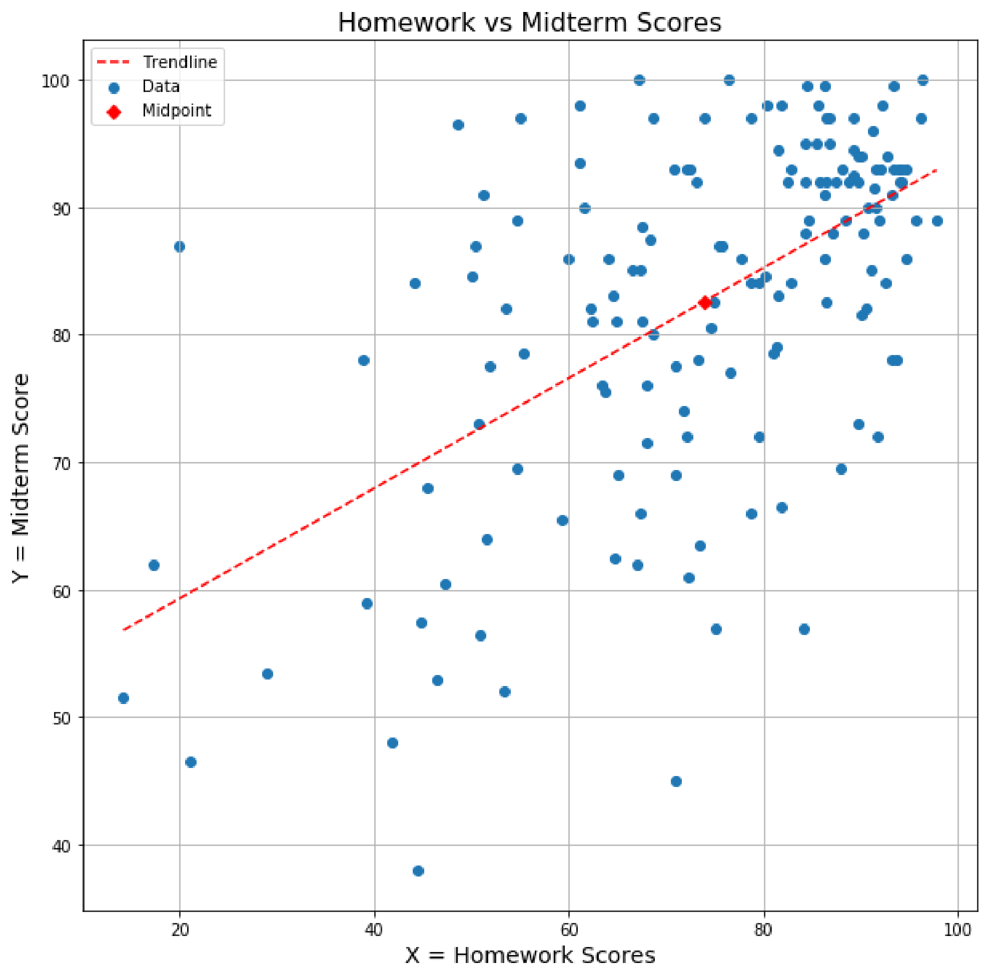
In this last problem on linear regression you will finish your analysis by comparing HWS2 and MID2 data, using the regression line to do some prediction, and thinking about what you have seen in all the problems so far.

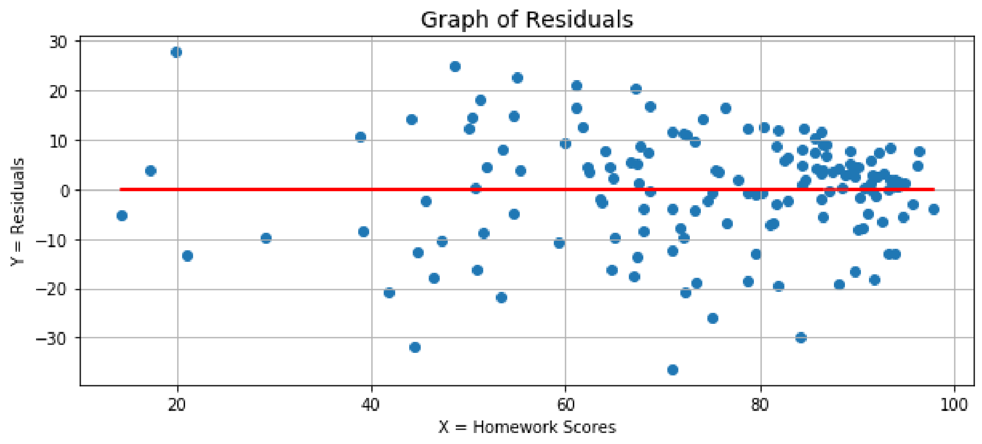
Part A

Apply the function from Problem One A to the following (modified) data:

```
X = HWS2
Y = MID2
```

```
In [11]: LinearRegression(HWS2,MID2,"Homework vs Midterm Scores",xlab="X = Homework Scores",ylab="Y = Mi
         dterm Score")
```





```
mean(x):        73.8621 std(x): 17.9908
mean(y):        82.5728 std(y): 13.6591

rho:    0.5688   R^2:    0.3235

Residual SS:    19058.132        Explained SS: 9114.0667 Total SS:    28172.1987

Regression Line: y = 0.4318 * x + 50.6766
```

Part B

We eliminated a student from the class who dropped the course before the midterm. Hm, I wonder how our model would have predicted his/her score for homeworks (for which we have a score) and midterm (for which we don't)? We will use the corrected values, after we removed the outlier (HWS2, MID2, GPA2).

The outlier student had the following values:

```
GPA:            3.13
Midterm:        0.0
HWAvg:          52.59
```

> (i) Calculate, based on the appropriate regression equation, what the midterm score would be predicted to be for this student based on the GPA.

> (ii) Using the regression line for HWS2 vs MID2, what would be the prediction for the midterm score be for this student based on his/her homework average? How well does this compare with the value from (i)?

(iii) Do you have any explanation for the results? (Hint: remember that the student dropped the class.)

Solution Part B

(i) The regression equation is y = 18.1454 *x + 23.6038, so our student would be predicted to have a midterm score of y = 18.1454* 3.13 + 23.6038 = 80.4.

(ii) Using y = 0.4318 *x + 50.6766, we have y = 0.4318* 52.59 + 50.6766 = 73.4.

(iii) Perhaps the student was doing poorly and was not handing in homeworks before dropping, so did not do as well as expected on homeworks.

Part C

Answer the following questions in a short sentence or two:

> (i) Did you see any other problems with the necessary conditions for linear regression in the other pairings of data from this set?

> (ii) Let us consider these three pairs of data, after eliminating the outlier but not worrying about any of the other conditions for linear regression. Using the R^2 value, in what order do you put them from weakest to strongest linear relationship? What do you think might be the causes for them being ordered in this way? Is there anything surprising about this? How certain are you of your conclusions, based on the R^2 values?

Answers to Part C

(i) I think that GPA2 vs HWS2 also suffered from homoscedasticity, for the same reasons given above: good students tend to be more regular in their attention to their school work. It is also not clear that the residuals are distributed normally.

(ii) Ranking the pairs of data in order of R^2 from highest to lowest:

```
GPA2 vs MID2:  0.4268
HWS2 vs MID2:  0.32351
GPA2 vs HWS2:  0.2892
```

None of these values shows much statistical significance, so we don't really know much about this data! That being said, it seems that GPA is a better predictor of midterm performance (knowledge) than homework performance (how much time you spend on them and how well you use your resources). It is surprising that homeworks are not the most important predictor of midterm success! This however could be disrupted by the "drop the lowest two" policy, since we are only half way through the grading process....

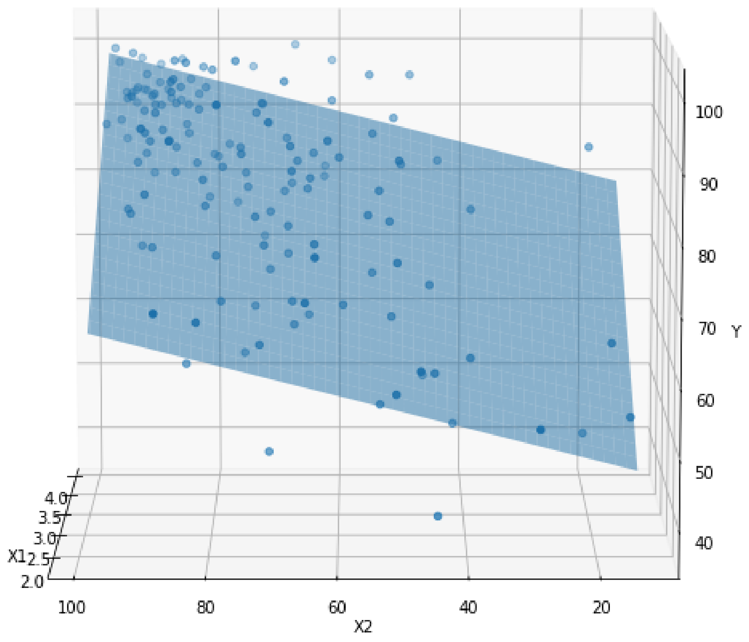
Problem Four Multiple Linear Regression

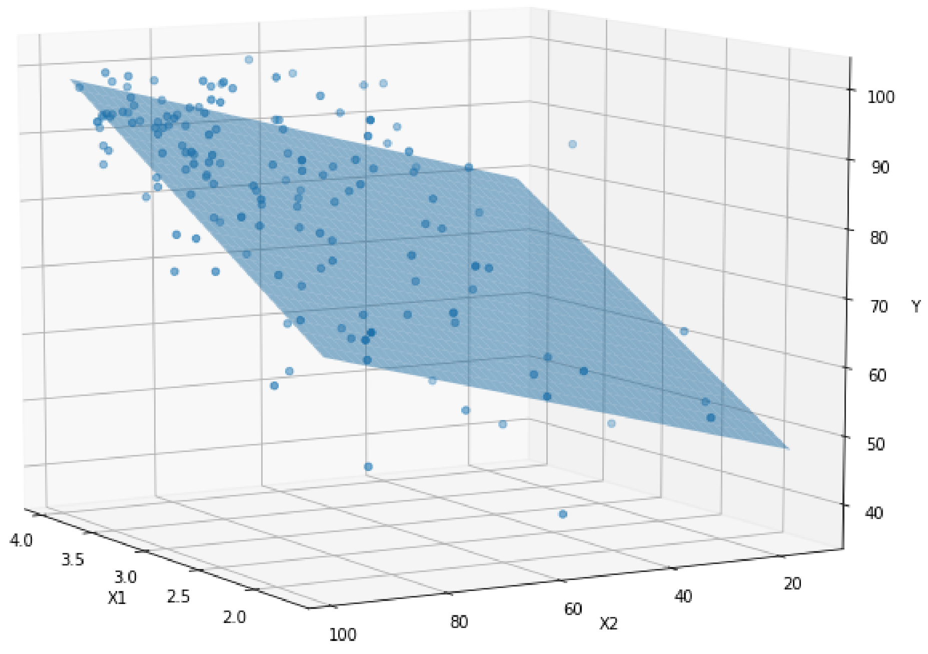
In this problem we will apply the technique of multiple linear regression to some data from a class and predict some values from the regression plane. The next cell contains the code for displaying the regression plane.

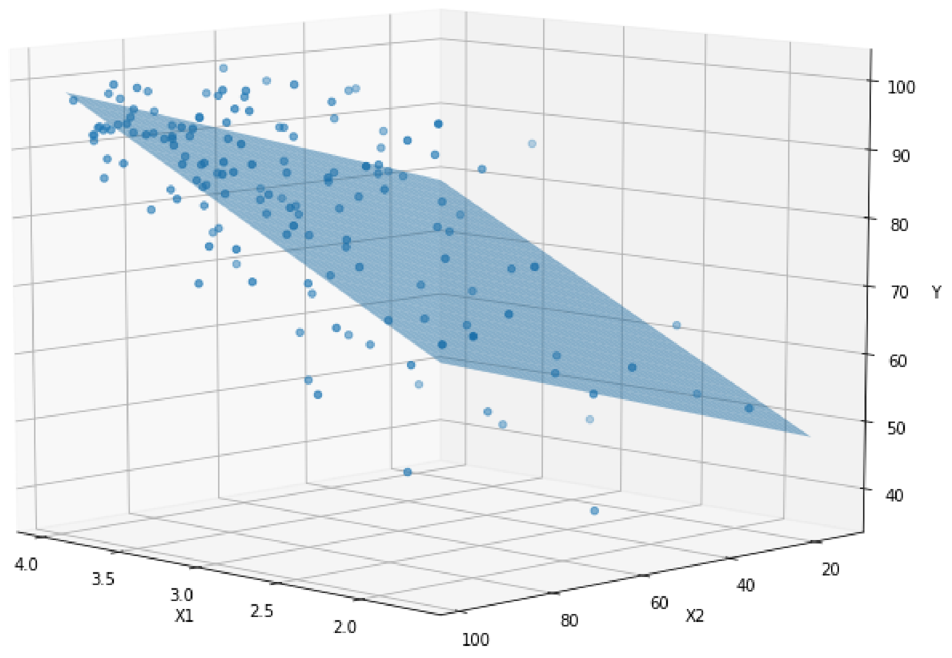
Run this cell and observe how it creates a series of 3D views of the linear regression plane.

In [12]:
```
# Given data vectors X and Y (must be same length), calculate the vector Theta
# which contains the parameters produced by regression.

def getTheta(X,Y):
    return inv(X.T @ X) @ X.T @ Y

def multiple_regression(X1,X2,Y,verbose=False,el=10,az=180):

    # make sure they are numpy arrays
    X1 = np.array(X1)
    X2 = np.array(X2)
    Y = np.array(Y)

    # Collect the dependent variables and add the bias term 1 to front
    X = transpose([ones(len(X1)),X1,X2])

    Theta = getTheta(X,Y)

    if verbose:
        print("\nRegression Plane:  y = " + str(round4(Theta[0])) + " + "
                                          + str(round4(Theta[1])) + " * x1 + "
                                          + str(round4(Theta[2])) + " * x2")

    # Plot the surface.
    Xplot = np.arange(min(X1), max(X1)+0.1, 0.1)
    Yplot = np.arange(min(X2), max(X2)+0.1, 0.1)
    Xplot, Yplot = np.meshgrid(Xplot, Yplot)
    Zplot = np.zeros_like(Xplot)
    for r in range(len(Zplot)):
        for c in range(len(Zplot[0])):
            Zplot[r][c] = Theta[0] + Theta[1]*Xplot[r][c] + Theta[2]*Yplot[r][c]


    fig = plt.figure(figsize=(12,10))
    ax = fig.gca(projection='3d')
    ax.view_init(elev=el, azim=az)

 #   ax.set_xlim(min(X),)
 #   ax.set_ylim(lo,hi)
 #   ax.set_zlim(lo,hi)
    ax.set_xlabel("X1")
    ax.set_ylabel("X2")
    ax.set_zlabel("Y")
    ax.scatter(X1,X2,Y)
    ax.plot_surface(Xplot,Yplot,Zplot,alpha=0.5)

    plt.show()

X1 = GPA2
X2 = HWS2
Y = MID2

multiple_regression(X1,X2,Y,True,el=10,az=180)
multiple_regression(X1,X2,Y,el=10,az=150)
multiple_regression(X1,X2,Y,el=10,az=135)
multiple_regression(X1,X2,Y,el=10,az=120)
multiple_regression(X1,X2,Y,el=10,az=90)
```

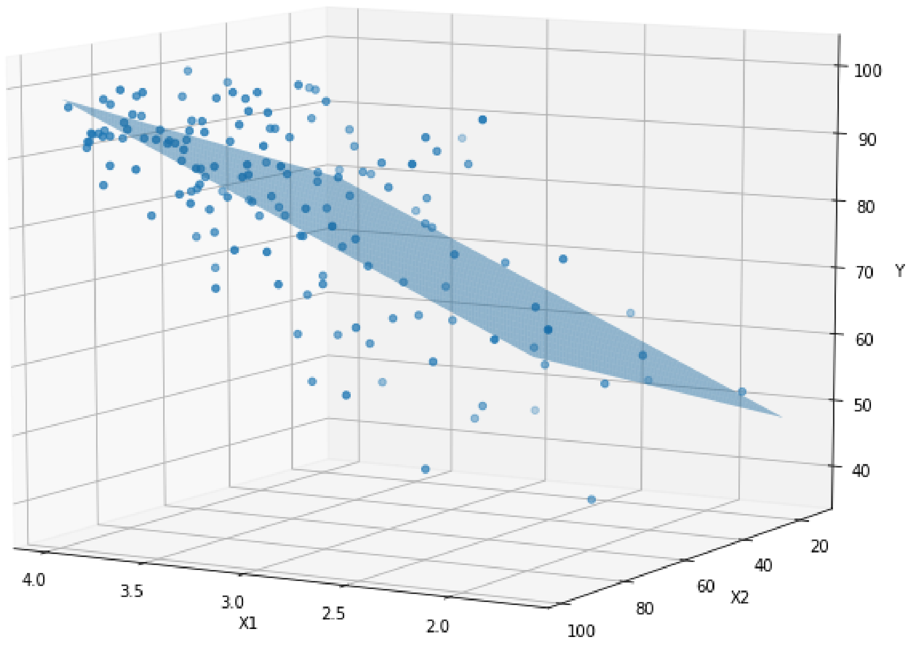
```
Regression Plane:  y = 21.2987 + 13.5759 * x1 + 0.2323 * x2
```



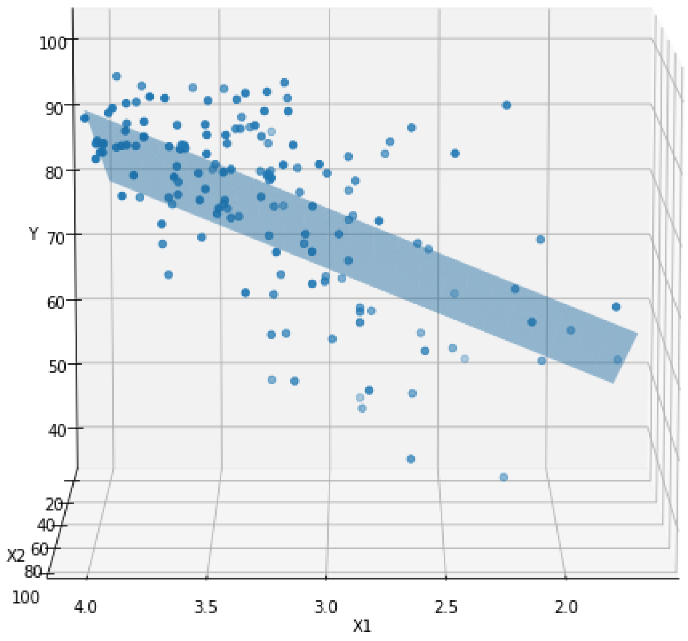




X1
X2
Y



X1
X2
Y



Part A

Now we will determine which pair of data best predict the other two. Using the formula for SSR, SST, and R^2:

$$SSR = \sum_{i=0}^{N} (\hat{y}_i - \mu_Y)^2$$
$$SST = \sum_{i=0}^{N} (y_i - \mu_Y)^2$$
$$R^2 = \frac{SSR}{SST}$$

and the regression equation determined for this data set:

$$\hat{Y} = 21.2987 + 13.5759 \cdot X_1 + 0.2323 \cdot X_2$$

calculate the R^2 value for

```
X1 = GPA2
X2 = HWS2
Y = MID2
```

In [13]:
```
# Solution

def h(x1,x2):
    return 21.2987 + 13.5759 * x1 + 0.2323 * x2

def SSR(X1,X2,Y):
    muY = mean(Y)
    N = len(Y)
    res = 0
    for k in range(N):
        res += (h(X1[k],X2[k]) - muY)**2
    print(res)
    return res

def SST(Y):
    muY = mean(Y)
    N = len(Y)
    res = 0
    for k in range(N):
        res += (Y[k] - muY)**2
    print(res)
    return res

def R2(X1,X2,Y):
    return SSR(X1,X2,Y)/SST(Y)


print("The R^2 for (GPA2, HWS2) -> MID2 is",round4(R2(GPA2,HWS2,MID2)))
```

```
13899.678167632714
28172.19867549669
The R^2 for (GPA2, HWS2) -> MID2 is 0.4934
```

Part B

Now consider the other 2 possibilities for two of these data predicting the other one (note that you don't need to consider reordering X1 and X2), print out your results, and then answer the following question: Which had the highest correlation and was it relevant?

Hint: You will need to recalculate the regression line, look at how it is done in the function `multiple_regression(...)` .

In [ ]:

Part C

Now answer the following questions:

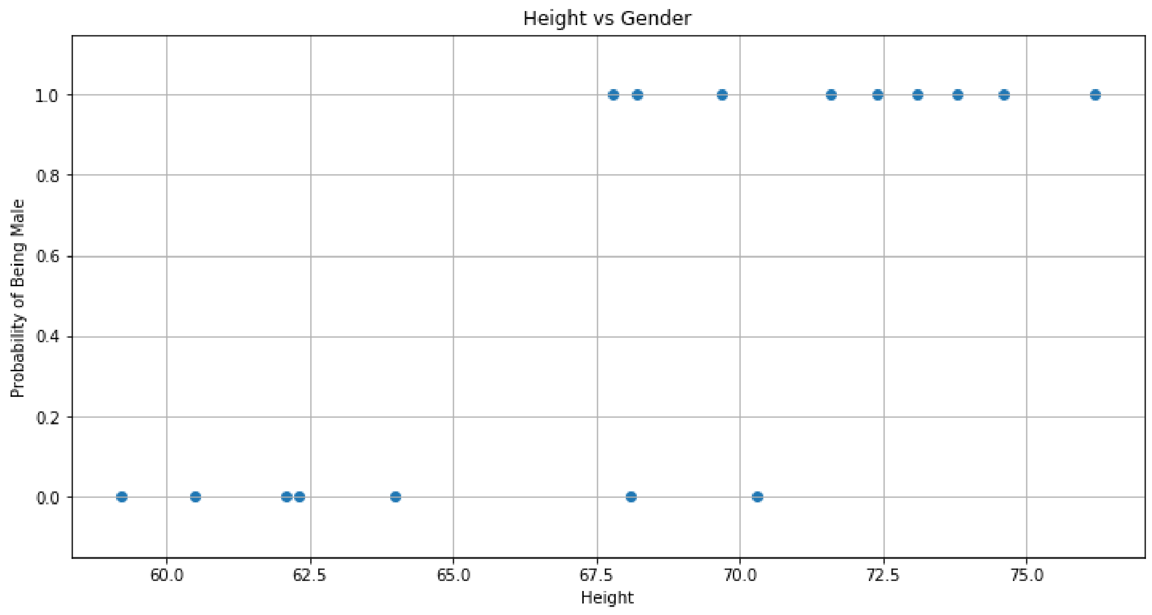
> (i) Which produced a better linear model?

> (ii) Were the R^2 values high enough to reasonably use this as a predictor of student performance?

Problem Five: Logistic Regression

This problem will review the process of logistic regression, using an example similar to what was shown in class.

```
In [14]: Height = [59.2, 60.5, 62.1, 62.3, 73.8, 64.0, 71.6, 67.8,68.1,68.2,69.7,70.3, 72.4,73.1,74.6,7
         6.2]

         Gender = [0,0,0,0,1,0,1,1,0,1,1,0,1,1,1,1]

         plt.figure(figsize=(12,6))
         plt.grid()
         plt.title("Height vs Gender")
         plt.ylim([-0.15,1.15])
         plt.xlabel("Height")
         plt.ylabel("Probability of Being Male")
         plt.scatter(Height,Gender)
         plt.show()
```



In [15]:
```
def s(z):
    return 1/(1+np.exp(-z))

def Cost(y,yHat):
    if y == 1:
        return -log(yHat)
    else:
        return -log(1 - yHat)

def h(b,m,xi):
    return b + m*xi

def J(b,m,X,Y):
    N = len(X)
    res = 0
    for k in range(len(X)):
        res += Cost(Y[k],s(h(b,m,X[k])))

    return res/N

def update_weights(b,m, X, Y, blam,mlam,verbose=False):
    m_deriv = 0
    b_deriv = 0
    N = len(X)
    for i in range(N):
        # Calculate partial derivatives
        m_deriv += X[i] * (s(h(b,m,X[i])) - Y[i])
        b_deriv += (s(h(b,m,X[i])) - Y[i])
    m_deriv /= N
    b_deriv /= N
    # We subtract because the derivatives point in direction of steepest ascent
    m -= m_deriv * mlam
    b -= b_deriv * blam

    return b,m,b_deriv,m_deriv

def Bderiv(b,m,X,Y):
    (b,m,bd,md) = update_weights(b,m,X,Y,0.001,0.001)
    return bd

def Mderiv(b,m,X,Y):
    (b,m,bd,md) = update_weights(b,m,X,Y,0.001,0.001)
    return md

def gradient_descent(b,m,X,Y,blam,mlam,limit,verbose=False):
    blast = b
    mlast = m
    inc = int(limit/10)
    for k in range(limit):
        (b,m,bd,md) = update_weights(blast,mlast,X,Y,blam,mlam,verbose)
        if verbose and (k % inc == 0):
            #print("  b_deriv = ",bd," \tm_deriv = ",md)
            #print("  b_delta = ",(b-blast),"\tm_delta = ",(m-mlast))
            print("b = ",b,"\tm = ",m, "J = ", J(b,m,X,Y))
        blast = b
        mlast = m
    return (b,m)

# Given an x-axis value for height, what is the prediction that is male?
def predict(b,m,xi):
    return s(h(b,m,xi))
```

Part A

Now run the gradient descent algorithm to find the best values for m and b. Experiment with different values for limit, mlam, and blam. The code will print out the results ten times during the calculation.

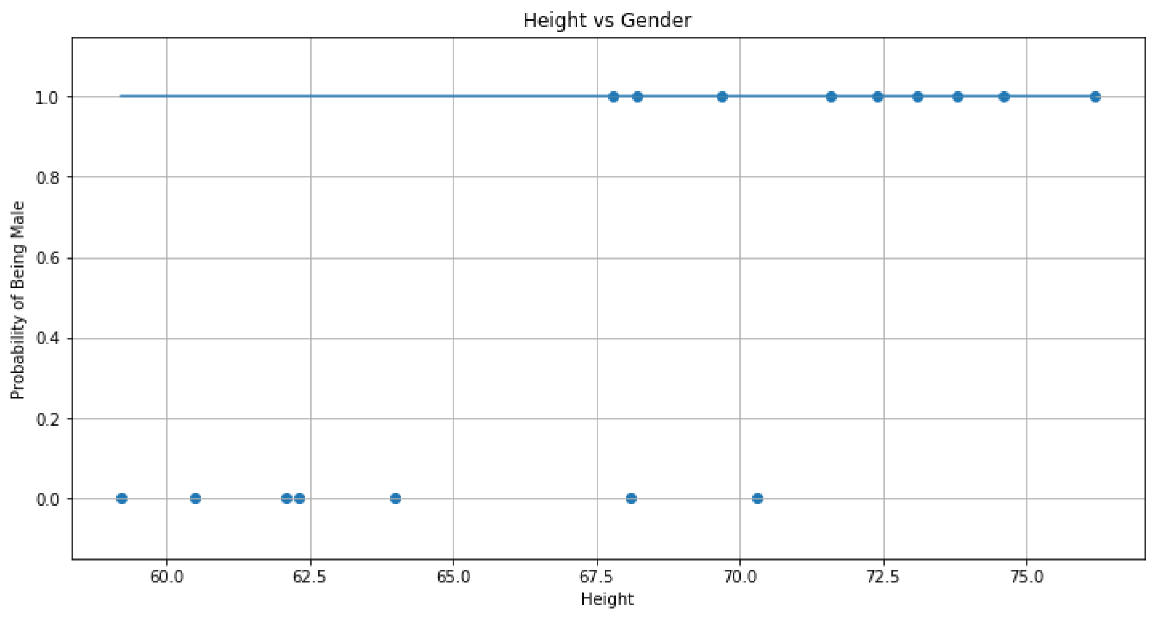
In [16]:
```
limit = 10**5           # Try as large as you can on your machine!

mlam = 0.01
blam = 0.1

b = 0
m = 1

b =  -39.1207393411      # after 10^6 iterations
m =  0.576274026396

b =  -43.1572903319   # after 2 * 10 ^ 6
m =  0.634947972275

b =  -44.3476779714
m =  0.652248017712

b =  -44.7443494217
m =  0.658012728303

b =  -44.8812980915
m =  0.660002949265

b =  -44.9291364732
m =  0.660698164062

b =  -44.9550000219    # after 10**7
m =  0.66107402767

b = 0
m = 1

# There is no improvement in the cost after this.

verbose = True
print("\nb = ",b,"\t               m = ",m)
(b1,m1) = gradient_descent(b,m,Height,Gender,blam,mlam,limit,verbose)
print("\nb = ",b1,"\nm = ",m1)
```

```
b =  0                m =  1
b =  -0.043750000000000004      m =  0.7209375 J =  inf

//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: RuntimeWarning: divide by ze
ro encountered in log


b =  -26.468314478613625        m =  0.6704379543455644 J =  7.129521772793141
b =  -46.207901903950216        m =  0.5443461084529149 J =  3.9675467409047394
b =  -59.422916945989414        m =  1.1455464673343523 J =  5.970398255195786
b =  -73.92611600487228         m =  1.2415633983480914 J =  2.3841666215095865
b =  -80.1161967936033  m =  1.2401330374837847 J =  0.7464386588468445
b =  -80.54904346768463         m =  1.245347305590861 J =  0.7350645135692216
b =  -80.89065251018282         m =  1.24953804032928 J =  0.7269881929001657
b =  -81.16605979556941         m =  1.2529577452904463 J =  0.7209639403427859
b =  -81.39139036333361         m =  1.255779790717121 J =  0.7163197113563061

b =  -81.59192675621709
m =  1.1608615740790762
```

Part B

Now complete this template by adding code to show the prediction curve calculated by the algorithm. (Hint: this is just the function `predict(...)` .)

In [17]:
```
plt.figure(figsize=(12,6))
plt.grid()
plt.title("Height vs Gender")
plt.ylim([-0.15,1.15])
plt.scatter(Height,Gender)
plt.xlabel("Height")
plt.ylabel("Probability of Being Male")
X = np.linspace(min(Height),max(Height),100)     # now plot the probability curve against the h
eight
Y = [predict(b,m,x) for x in X]
plt.plot(X,Y)
plt.show()
```



Part C

Now answer the following two questions:

```
 (i)  Your professor is 70 inches tall. What is the probability that he is male, according to this
algorithm?

 (ii) The decision rule associated with this algorithm is that an input height is classified as ma
le if the probability is 0.5 or higher.  Does this algorithm classify your gender correctly?
```

In [18]:
```
predict(b,m,70)
```

Out[18]:
```
1.0
```

Solution

(i) predict(b,m,70) => 78.4%

(ii) Correct!

Problem Six (Bloom Filters)

This is another famous example of a Monte Carlo algorithm, but in this case it is a data structure. From Wikipedia:

> A *Bloom filter* is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not – in other words, a query returns either "possibly in set" or "definitely not in set". Elements can be added to the set, but not removed (though this can be addressed with a "counting" filter); the more elements that are added to the set, the larger the probability of false positives.

The basic algorithm for a Bloom Filter is actually extremely simple:

- Create a list or array A of bits of length m, initialized to all 0's;
- Assume you need to store n keys (which are integers or strings or really anything that can be hashed);
- Assume you have k independent hash functions $h_1 \dots h_k$ which map the keys into the range [0.. m-1];
- To insert a key x into the Bloom Filter, for each i, $1 \le i \le k$, set the bit $A[h_i(x)]$ to 1;
- To test for the membership of an integer y in the Bloom Filter, check if all the bits $A[h_i(x)]$ are set to 1; if so, then return "Member!"; otherwise return "Not a member!"

Two things should be clear about this algorithm: first, if m and k are fixed ahead of time, then this is an $O(1)$ algorithm (like a hash table); second, when a number x has been inserted, then a subsequent membership test will always return true ("true positive"), and if the membership test returns False, then the value was not inserted ("true negative"); the problem is that there may be "false positives," i.e., a number z which was not inserted appears to be in the set because the combination of insertions (not including z) has accidentally set the bits corresponding to z to 1.

As always with Monte Carlo algorithms, the question is how likely false results will be returned. For Bloom Filters, it can be shown that the probability of an erroneous result (a false positive) is

$$(1 - e^{\frac{kn}{m}})^k,$$

which means that you can "tune" the error rate with the parameters k and m. For example, suppose we are storing 32-bit integers, and we assume that $m = 32n$, i.e., the Bloom Filter uses exactly as many bits as would be necessary to store the original list of n numbers, then for $k = 3$ we have a false positive rate of

$$(1 - e^{3})^3 = 0.0007$$

and for $k = 10$ we have

$$(1 - e^{10})^{10} = 0.000002$$

This is obviously quite accurate! We will choose these parameters as follows for this problem:

In [19]:
```
k = 10           # number of hash functions
n = 100         # number of keys to insert
m = 1000        # size of bit list


print("The false positive rate is " + str((1 - np.e **(-k*n/m))**k))
```

```
The false positive rate is 0.010185894032016 96
```

Your Turn!

Complete the following program stub to implement a Bloom Filter.

In [20]:
```
import hashlib

def hashVector(obj,size,k):
    h1 = hashlib.md5(obj.encode())
    h2 = hashlib.md5(h1.encode())

    return [( (h1 + (i*h2)) % size) for i in range(k)]
```

In [21]:
```
k = 3           # number of hash functions
n = 3          # number of keys to insert
m = 17        # size of bit list (prime number)


A = [0]*m

def insert(x):
    hv = hashVector(x,m,k)
    for i in range(len(hv)):
        A[hv[i]] = 1

def member(x):
    hv = hashVector(x,m,k)
    for i in range(k):
        if A[hv[i]] != 1:
            return False
    return True


B = [str(randint(100)) for k in range(n)]
print(A)
print(B)
for i in range(len(B)):
    print()
    print(B[i])
    insert(B[i])
    print(A)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
['74', '45', '27']

74

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-ce1150621c51> in <module>
     25     print()
     26     print(B[i])
---> 27     insert(B[i])
     28     print(A)

<ipython-input-21-ce1150621c51> in insert(x)
      7 
      8 def insert(x):
----> 9     hv = hashVector(x,m,k)
     10     for i in range(len(hv)):
     11         A[hv[i]] = 1

<ipython-input-20-0caa94b8a372> in hashVector(obj, size, k)
      3 def hashVector(obj,size,k):
      4     h1 = hashlib.md5(obj.encode())
----> 5     h2 = hashlib.md5(h1.encode())
      6 
      7     return [( (h1 + (i*h2)) % size) for i in range(k)]

AttributeError: '_hashlib.HASH' object has no attribute 'encode'
```

In [ ]:

In [ ]:
```
# Now we test the algorithm to find the actual false positive rate
A = [0]*m

seed(0)

# generate n numbers to insert, in the range [0 .. 10^8 - 1]
N = [ randint(10**5) for k in range(n)]

for i in range(len(N)):
    insert(N[i])

num_tests = 10**4
# generate another num_tests numbers, to test the algorithm
Test = [randint(10**5) for k in range(num_tests)]

errors = 0
for i in range(len(Test)):
    if member(Test[i]) and not (Test[i] in N):
        errors += 1

print("False positive rate is " + str(errors/num_tests))
```

Problem 4

In this problem we will study another practical Monte Carlo algorithm, for comparing two files quickly. Here is the scenario: Wayne and Roman are working, separately and remotely, on a project that involves a very large file. They both have a copy of the file, but how do they know the files are the same? What would be good is to have a fast check to confirm, with high probability, that the files are indeed the same; if this test gives false positives (says the files are the same when they are in fact different), then this causes complications in backing up the updates to the file to find the common version, but it does not cause a permanent error.

The absolutely correct and deterministic algorithm is to send the file over the network and compare bit by bit, however the file is large and it seems that the Monte Carlo approach will help avoid this expensive operation.

Here is a simple Monte Carlo algorithm using hashing to check if two files A and B (thought of as strings of characters) are the same:

Processing at the sender who owns version A:

- let na = hash(A)
- Send na to receiver

Processing at the receiver (who owns B):

- let na = number received from sender
- let nb = hash(B)
- if na == nb:

```
print("Files Same with High Probability!")
```

  else:

```
print("Files are Different!")
```

This is MUCH more efficient (in terms of the bandwidth necessary to send the file) if we use a cryptographic hash function (https://simple.wikipedia.org/wiki/Cryptographic_hash_function#:~:targetText=A%20cryptographic%20hash%20function%20is,digest'%20or% such as SHA256, since that returns a 256-bit number. Sending such a number over the network is obviously very efficient!

Finally, let us observe the two essential features of a Monte Carlo algorithm:

- There is no possibility of a false negative, i.e., reporting that the files are different when they are in fact the same, since the hash function is assumed to be correct!
- A false positive occurs when two different strings collide at the same hash value; since there are 2^{256} possible outputs of the SHA256 algorithm, and such a strong cryptographic hash function distributes its inputs equiprobably over the range, this probability is
$$\frac{1}{2^{256}} = 8.6 * 10^{-78}$$
This is the probability of flipping a coin 256 times and getting all heads, or, put another way, choosing your favorite atom in our galaxy, and having a friend do the same, and find that you have randomly chosen the same atom! Hence, it is a very acceptable false positive rate!

In [ ]:
```python
import hashlib
hash_object = hashlib.sha256(b'Hello World')
hex_dig = hash_object.hexdigest()
print(hex_dig)
```

In [ ]:
```python
def hash(s):
    return hashlib.sha256(s.encode('ascii')).hexdigest()

hash("A simple message")
```

In [ ]:
```
# Just a demonstration of what these do

w = "Arms folded\n to the moon,\namong the cows.\n  -Jack Kerouac"
print("Original file (text string):\n\n" + w)
print()
b  = hash(w)
print("Representation as hexidecimal number:\n\n" + str(b))
print()
```

In [ ]:
```
## Ok, your turn!
We will use text strings instead of files, but the idea is the same.

Follow the pseudo-code to complete the algorithm.
```

In [ ]:
```
A = '''SECTION 3

The Senate of the United States shall be composed of two Senators from each State, chosen by th
e Legislature thereof, for six Years; and each Senator shall have one Vote.

Immediately after they shall be assembled in Consequence of the first Election, they shall be d
ivided as equally as may be into three Classes. The Seats of the Senators of the first Class sh
all be vacated at the Expiration of the second Year, of the second Class at the Expiration of t
he fourth Year, and of the third Class at the Expiration of the sixth Year, so that one third m
ay be chosen every second Year; and if Vacancies happen by Resignation, or otherwise, during th
e Recess of the Legislature of any State, the Executive thereof may make temporary Appointments
until the next Meeting of the Legislature, which shall then fill such Vacancies.

No Person shall be a Senator who shall not have attained to the Age of thirty Years, and been n
ine Years a Citizen of the United States, and who shall not, when elected, be an Inhabitant of
 that State for which he shall be chosen.

The Vice President of the United States shall be President of the Senate, but shall have no Vot
e, unless they be equally divided.

The Senate shall chuse their other Officers, and also a President pro tempore, in the Absence o
f the Vice President, or when he shall exercise the Office of President of the United States.

The Senate shall have the sole Power to try all Impeachments. When sitting for that Purpose, th
ey shall be on Oath or Affirmation. When the President of the United States is tried, the Chief
Justice shall preside: And no Person shall be convicted without the Concurrence of two thirds o
f the Members present.

Judgment in Cases of impeachment shall not extend further than to removal from Office, and disq
ualification to hold and enjoy any Office of honor, Trust or Profit under the United States: bu
t the Party convicted shall nevertheless be liable and subject to Indictment, Trial, Judgment a
nd Punishment, according to Law."
'''
```

In [ ]:
```
B = A[:1724] + "I" + A[1725:]

print(A[1710:1800])
print(B[1710:1800])
```

In [ ]:
```
A == A
```

In [ ]:
```
hash(A) == hash(A)
```

In [ ]:
```
A == B
```

In [ ]:
```
hash(A) == hash(B)
```