Lecture 18: Correlation and Regression

Just some code related to these topics for lecture examples

In [32]:

```
# General useful imports


import numpy as np
from numpy import arange,linspace,mean, var, std
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from numpy.random import seed,random, randint, uniform, choice, binomial, geomet
ric, poisson, exponential, normal
from numpy.linalg import inv
import math
from collections import Counter
import pandas as pd
%matplotlib inline


# Numpy basic stats functions

# https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.statistics.html


X = [1,2,3]

# mean of a list
mean(X)

# population variance
var(X)

# sample variance
var(X,ddof=1)

# population standard deviation
std(X)

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


# Scipy statistical functions

from scipy.stats import norm,expon

# https://docs.scipy.org/doc/scipy/reference/stats.html

# given random variable X (e.g., housing prices) normally distributed with  mu =
60, sigma = 40

#a. Find P(X<50)
norm.cdf(x=50,loc=60,scale=40) # 0.4012936743170763

#b. Find P(X>50)
norm.sf(x=50,loc=60,scale=40) # 0.5987063256829237

#c. Find P(60<X<80)
norm.cdf(x=80,loc=60,scale=40) - norm.cdf(x=60,loc=60,scale=40)

#d. how much top most 5% expensive house cost at least? or find x where P(X>x) =
```

```
0.05
norm.isf(q=0.05,loc=60,scale=40)

#e. how much top most 5% cheapest house cost at least? or find x where P(X<x) =
 0.05
norm.ppf(q=0.05,loc=60,scale=40)

#f give the endpoints of the range for the central alpha percent of the distribu
tion
norm.interval(alpha=0.3, loc=60, scale=140)

# Same for exponential distribution
beta = 5    # mean of distribution

expon.pdf(x=3,loc=0,scale=beta)       # don't change loc = starting point

# functions from numpy.random library

# https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.random.html

#g generate random variates from a normal distribution
normal(loc=60,scale=40,size=10)

#h generate random variates from a poisson distribution with rate parameter lam
(bda)
poisson(lam=2,size=10)

#i generate random variates from an exponential distribution
# scale = beta = mean of distribution; beta = 1 / lambda    (lambda = rate param
eter)
exponential(scale=5,size=100)

# Utility functions

# 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]
```

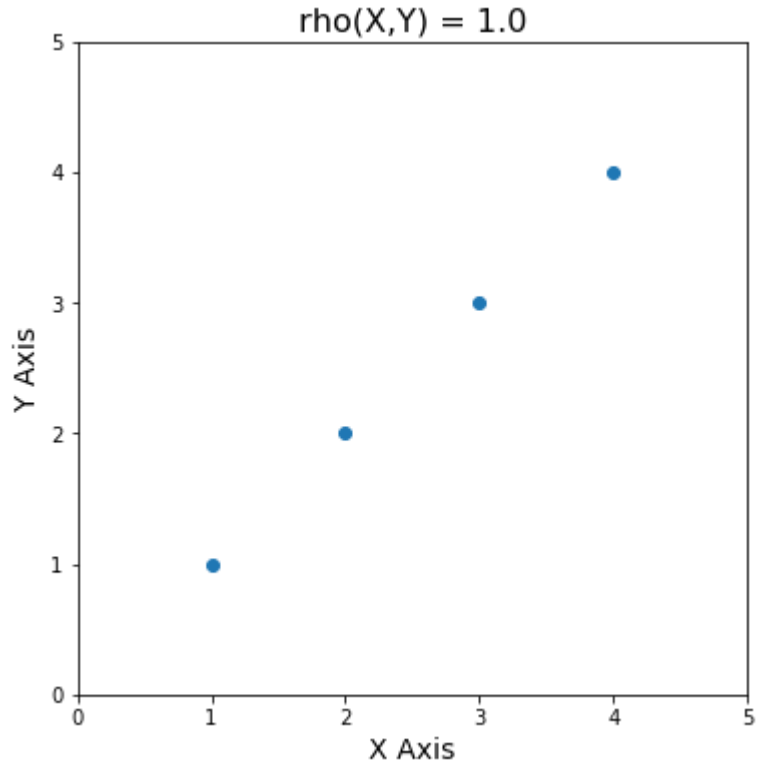
Correlation

In [33]:

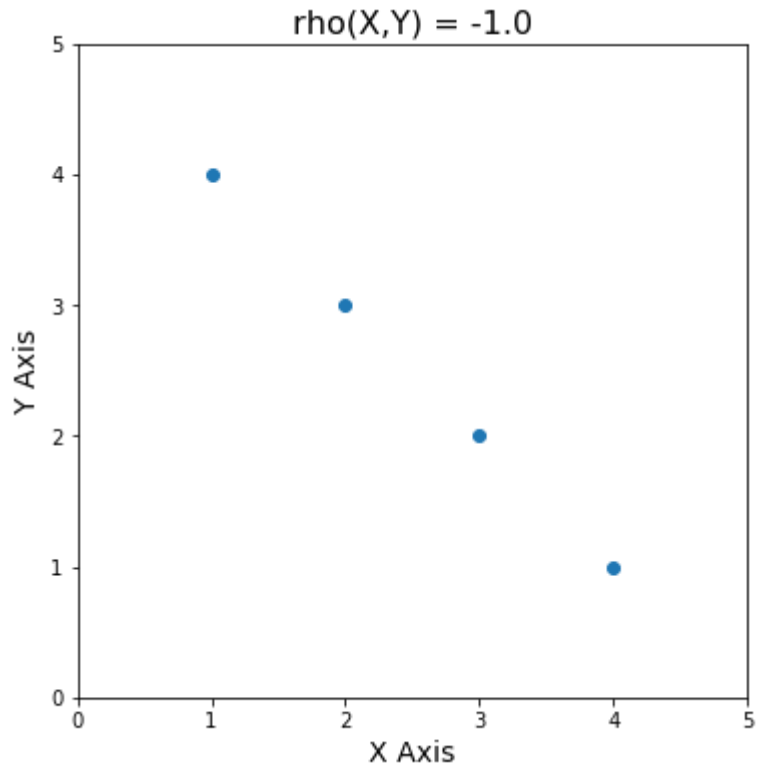
```
def rho(x,y):
    mux = mean(x)
    muy = mean(y)
    sdx = std(x)
    sdy = std(y)

    sum = 0
    for i in range(len(x)):
        sum += x[i]*y[i]
    expXY = sum/len(x)

    return (expXY - mux*muy)/(sdx*sdy)

def displayXY(X,Y):
    print()
    if (len(X) + len(Y) < 50):
        print("X = " + str(X))
        print("Y = " + str(Y))
    fig = plt.figure(figsize=(6,6))
    xlo = min(X)-1
    xhi = max(X)+1
    ylo = min(Y)-1
    yhi = max(Y)+1
    plt.axis([xlo,xhi,ylo,yhi])
    plt.title("rho(X,Y) = " + str(round4(rho(X,Y))), fontsize=16)
    plt.xlabel("X Axis", fontsize=14)
    plt.ylabel("Y Axis", fontsize=14)
    if(len(X) > 20):
        m = "+"
    else:
        m = "o"
    plt.scatter(X,Y, marker=m)
    plt.show()


def ex1():
    X = [1,2,3,4]
    Y = [1,2,3,4]
    displayXY(X,Y)

def ex2():
    X = [1,2,3,4]
    Y = [4,3,2,1]
    displayXY(X,Y)

def ex3():
    X = [1,2,3,4]
    Y = [2,4,6,8]
    displayXY(X,Y)

def ex4():
    X = [1,2,3,4]
    Y = [1,1.1,1.2,1.3]
    displayXY(X,Y)

def ex5():
    X = [1,2,3,4]
    Y = [2,2,2,2]
    displayXY(X,Y)
```

```
def ex5b():
    X = [2,2,2,2]
    Y = [1,2,3,4]
    displayXY(X,Y)

def ex6():
    X = [0,1,2,3]
    Y = [0,2,4,3]
    displayXY(X,Y)

def ex7():
    X = [0,1,2,3,4,5,6,7,8,9,10]
    Y = [0,1,4,9,16,25,36,49,64,81,100]
    displayXY(X,Y)

def ex8():
    X = [-1,-1,1,1]
    Y = [-1,1,-1,1]
    displayXY(X,Y)

def ex9():
    X = [-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8]
    Y = [y**2 for y in X]
    displayXY(X,Y)


ex1()

ex2()

ex3()

ex4()

ex5()

ex5b()

ex6()

ex7()

ex8()

ex9()


print()
```

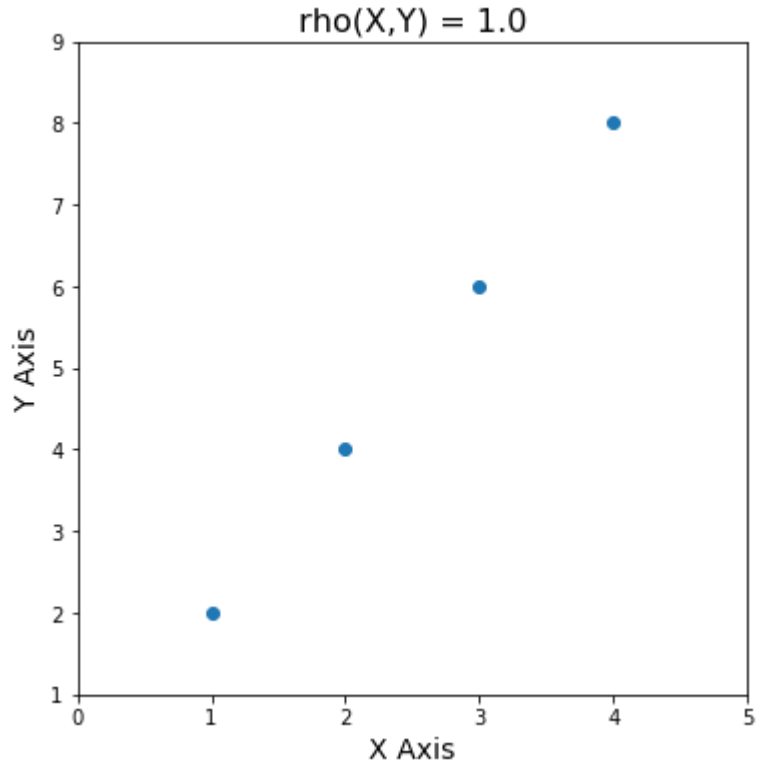
```
X = [1, 2, 3, 4]
Y = [1, 2, 3, 4]
```



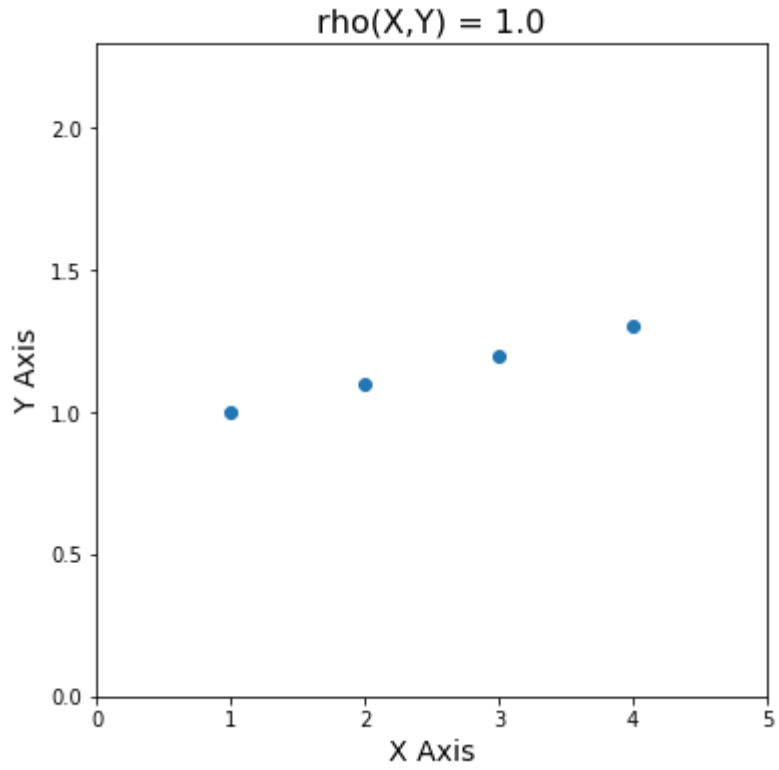

```
X = [1, 2, 3, 4]
Y = [4, 3, 2, 1]
```




```
X = [1, 2, 3, 4]
Y = [2, 4, 6, 8]
```

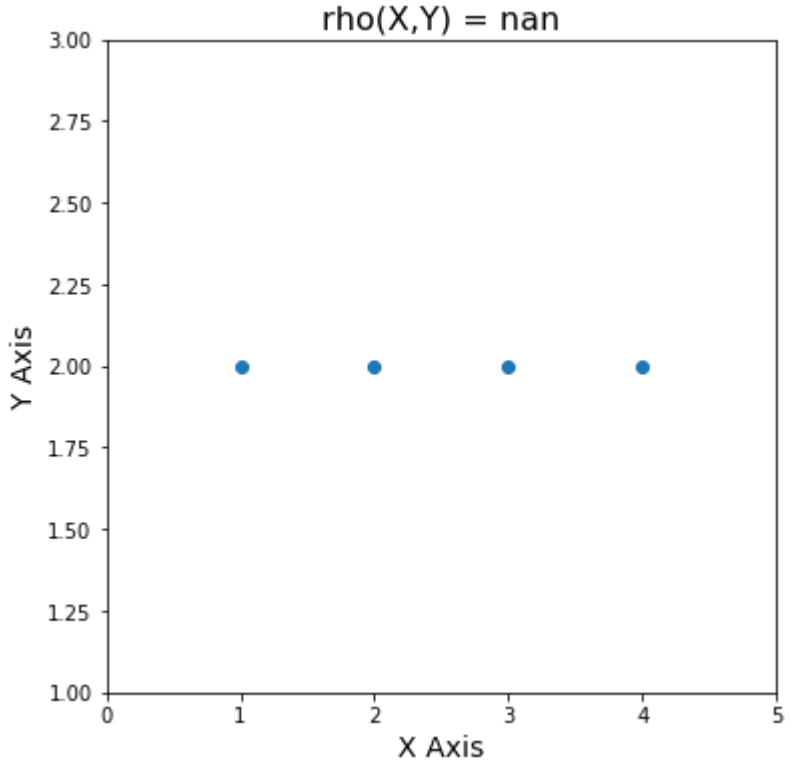



```
X = [1, 2, 3, 4]
Y = [1, 1.1, 1.2, 1.3]
```

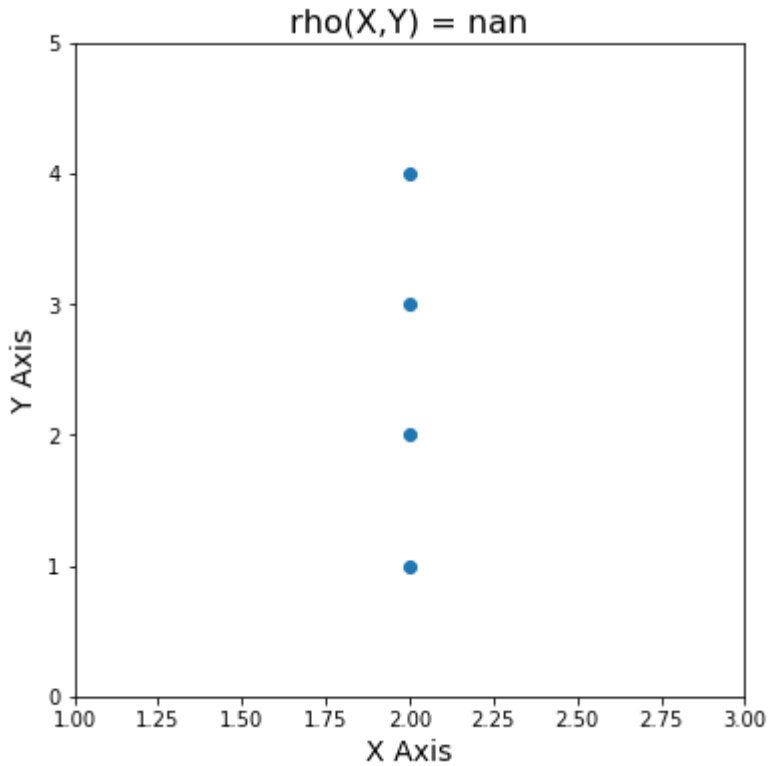



```
X = [1, 2, 3, 4]
Y = [2, 2, 2, 2]
```

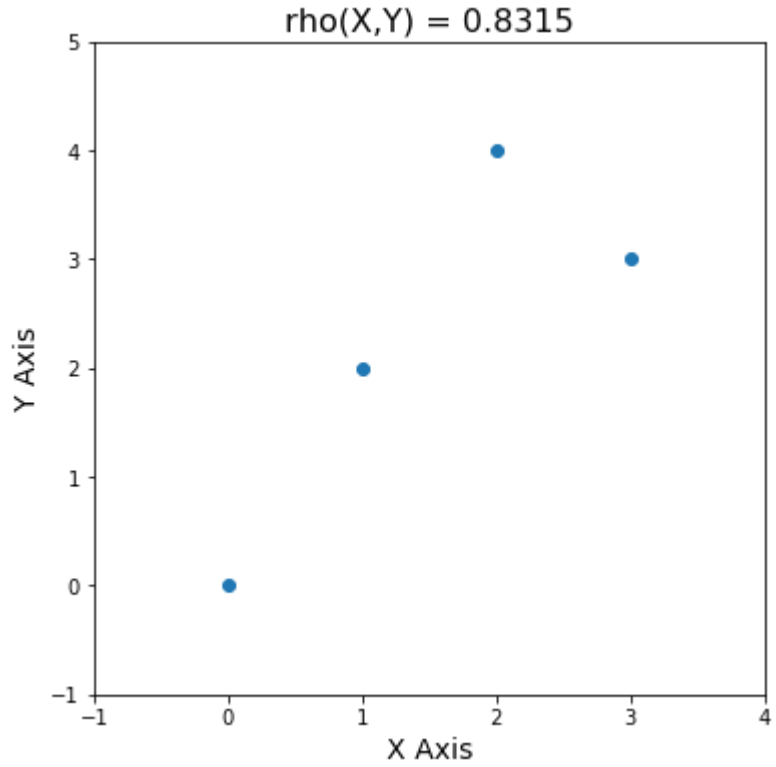
```
//anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:12: Ru
ntimeWarning: invalid value encountered in double_scalars
  if sys.path[0] == '':
```



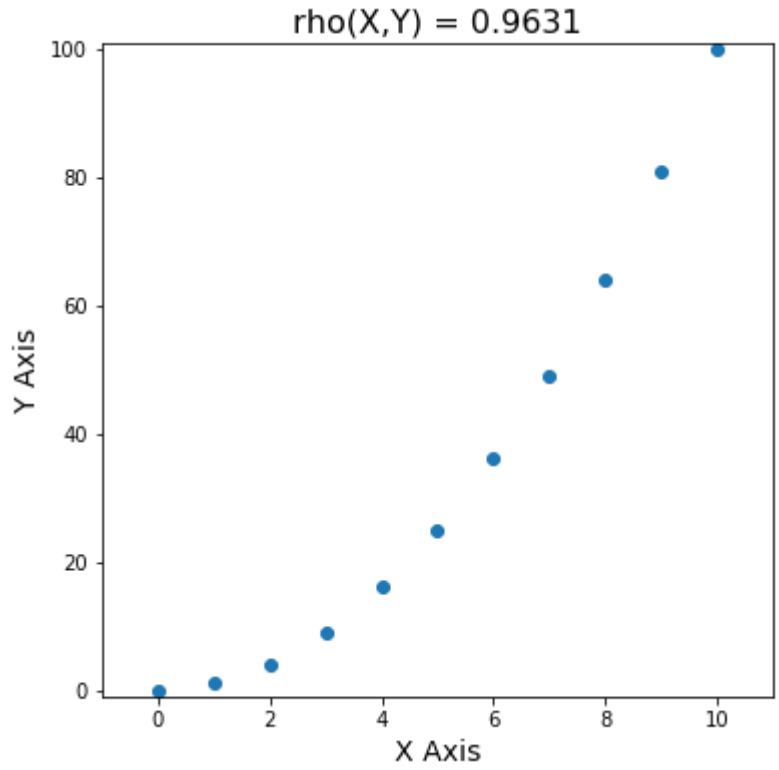

```
X = [2, 2, 2, 2]
Y = [1, 2, 3, 4]
```



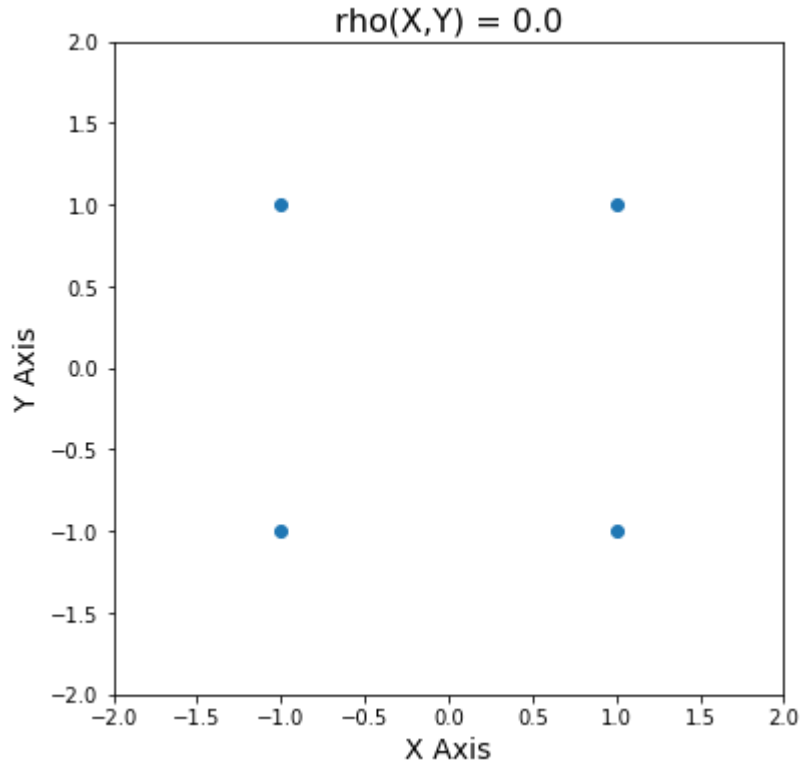

```
X = [0, 1, 2, 3]
Y = [0, 2, 4, 3]
```



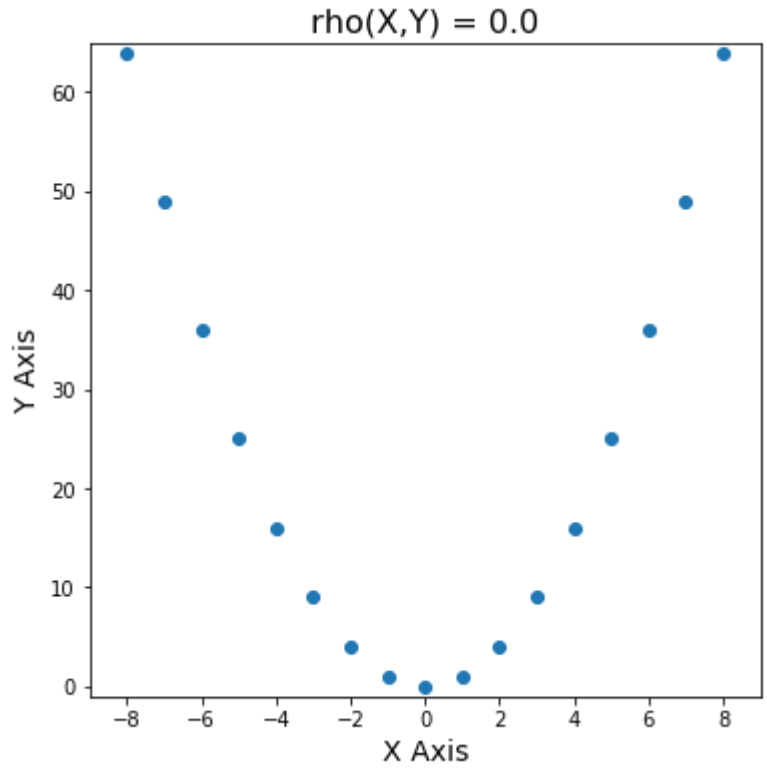

```
X = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Y = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```




```
X = [-1, -1, 1, 1]
Y = [-1, 1, -1, 1]
```

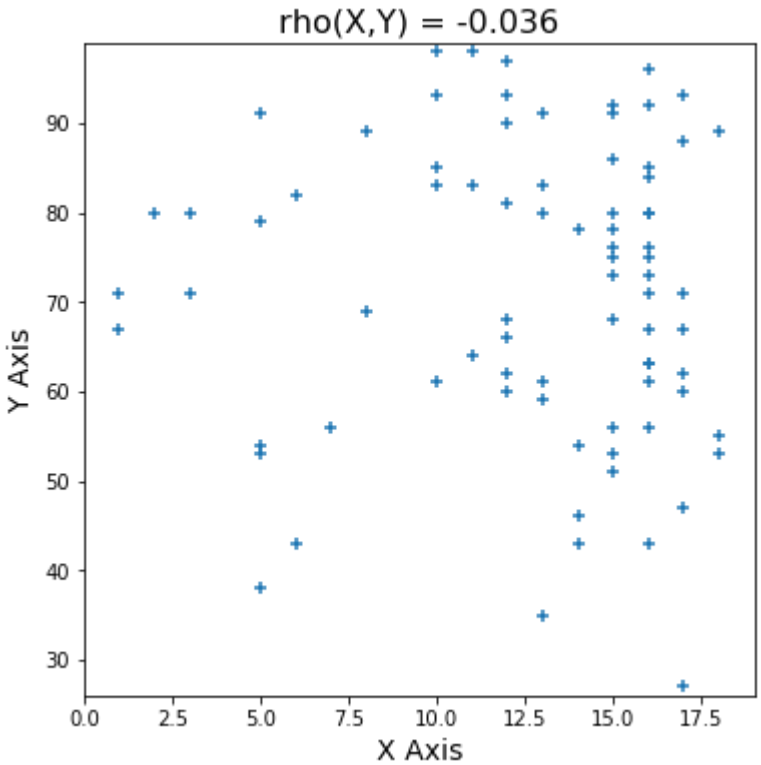



```
X = [-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8]
Y = [64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64]
```



In [34]:

```
# Midterm Data   X = how many minutes after noon turned in   Y = score

midtermY = [98, 98, 97, 96, 93, 93, 93, 92, 92, 91, 91, 91, 90, 89, 89, 88, 86,
     85, 85, 84, 83, 83, 83, 82, 81, 80, 80, 80, 80, 80, 80, 79, 78, 78,
     76, 76, 75, 75, 73, 73, 71, 71, 71, 71, 69, 68, 68, 67, 67, 67, 66,
     64, 63, 63, 62, 62, 61, 61, 61, 60, 60, 59, 56, 56, 56, 55, 54, 54,
     53, 53, 53, 51, 47, 46, 43, 43, 43, 38, 35, 27]

midtermX = [10, 11, 12, 16, 17, 10, 12, 16, 15, 5,  15, 13, 12, 8,  18, 17, 15,
     10, 16, 16, 13, 11, 10, 6,  12, 15, 3,  16, 16, 13, 2,  5,  15, 14, 15,
     16, 15, 16, 15, 16, 17, 1,  16, 3,  8,  15, 12, 1,  17, 16, 12, 11, 16,
     16, 12, 17, 10, 13, 16, 17, 12, 13, 15, 16, 7,  18, 14, 5,  15, 18, 5,
     15, 17, 14, 16, 14, 6, 5, 13, 17]

displayXY(midtermX,midtermY)
```

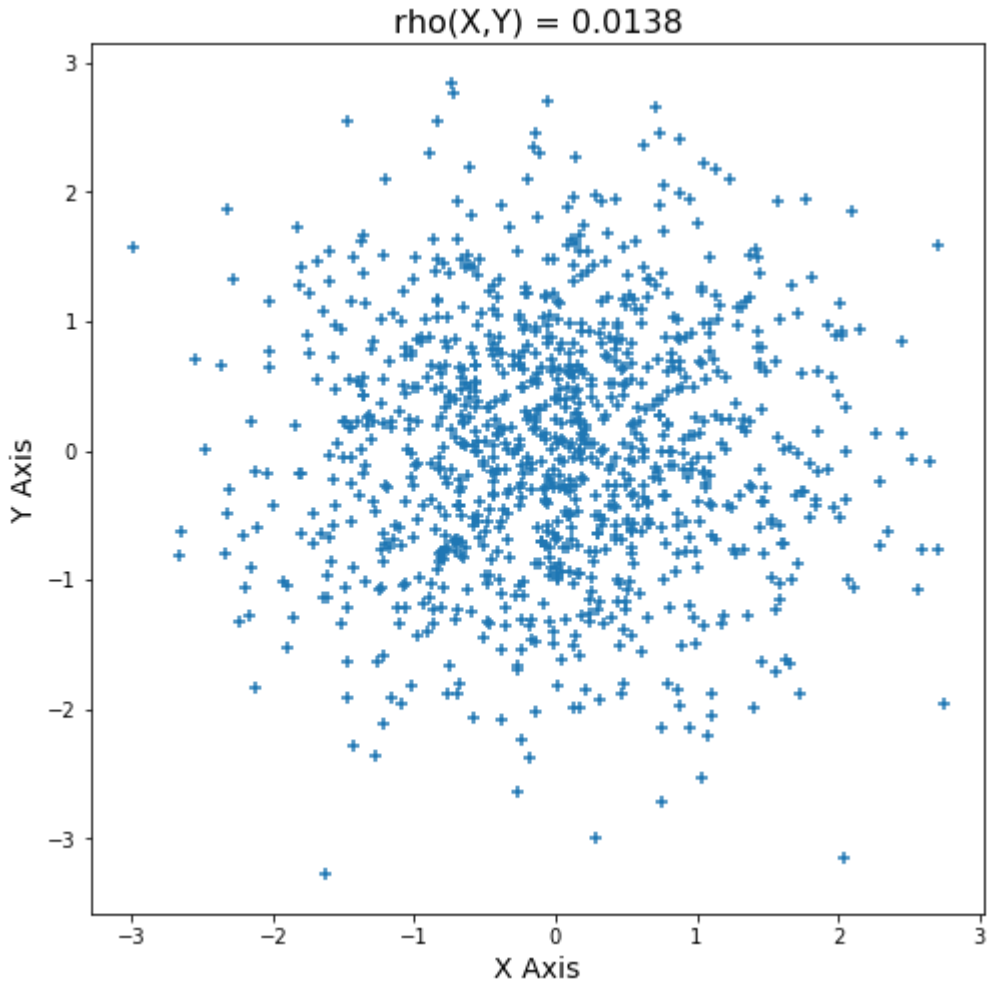


In [35]:

```
# This generates a joint random variable consisting of two independent
# standard normal distributions

def jointNormal(num_trials):
    X = normal(size=num_trials)
#    X = uniform(size=num_trials)
#    X = exponential(3,size=num_trials)
    Y = normal(size=num_trials)
#    Y = uniform(size=num_trials)
#    Y = exponential(3,size=num_trials)
    return (X,Y)


num_trials = 1000
(X,Y) = jointNormal(num_trials)

print()
fig = plt.figure(figsize=(8,8))

plt.title("rho(X,Y) = " + str(round4(rho(X,Y))), fontsize=16)
plt.xlabel("X Axis", fontsize=14)
plt.ylabel("Y Axis", fontsize=14)

plt.scatter(X,Y, marker="+")
```

Out[35]:

```
<matplotlib.collections.PathCollection at 0x1a21634e10>
```



In [36]:

```
# Draw scatterplot for bivariate data and draw linear regression line
# with midpoint (mux,muy)

def ScatterTrendline(X,Y,titl="Scatterplot with Trendline", xlab="X",ylab="Y",sh
owStats=True,showResiduals=False):

    n = len(X)

    # basic
    mux = mean(X)
    muy = mean(Y)
 #   sdx = std(X,ddof=1)
 #   sdy = std(Y,ddof=1)
    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])

    # regression sum of squares -- deviation of line from mean of y
    regss = 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)] )

    # alternate way to compute r^2 statistic

    #r2 = regss / tss

    if(n < 20):
        fmt = 'bo'
        fmt2 = 'o'
        bord = 3
    elif(n < 100):
        fmt = 'b+'
        fmt2 = '+'
        bord = 4
    else:
        fmt = 'b,'
        fmt2 = '.'
        bord = 5
```

```
    # set axes so border around data

    minx = min(X)
    miny = min(Y)
    maxx = max(X)
    maxy = max(Y)
    maxe = max(E)
    mine = min(E)

    xran = maxx - minx
    yran = maxy - miny
    eran = maxe - mine
    xlo = minx - xran/bord
    xhi = maxx + xran/bord
    ylo = miny - yran/bord
    yhi = maxy + yran/bord
    elo = mine - eran/bord
    ehi = maxe + eran/bord

    plt.figure(figsize=(8,6))
    plt.axis([xlo,xhi,ylo,yhi])

    linex = [(minx-xran/10),(maxx+xran/10)]
    liney = [(m*x+b) for x in linex]

    plt.plot(X,Y, fmt,linex,liney,'r--',[mux],[muy],'rD')
    plt.title(titl,fontsize=16)
    plt.legend(["Data","Trendline","Midpoint"],loc='best')
    plt.xlabel(xlab,fontsize=14)
    plt.ylabel(ylab,fontsize=14)
    plt.show()

    if showResiduals:
        plt.figure(figsize=(10,5))
        plt.axis([xlo,xhi,elo,ehi])
        plt.scatter(X,E,marker=fmt2)
        plt.plot([xlo,xhi],[0,0],'--', color='k')
        plt.title("Residual Plot",fontsize=16)
        plt.xlabel("X",fontsize=14)
        plt.ylabel("Residuals",fontsize=14)
        plt.show()

    if showStats:
        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)) + "\tRegression SS: " + str
(round4(regss)) + "\tTotal SS:   " + str(round4(tss)))

        if(b >= 0):
            print("\nRegression Line: y = " + str(round4(m)) + " * x + " + str(r
ound4(b)))
        else:
            print("\nRegression Line: y = " + str(round4(m)) + " * x - " + str(r
ound4(-b)))
```

Simple Linear Regression

Linear Regression is the process of constructing a model for a bivariate random variable (X,Y) which shows a linear relationship between X (the independent variable) and Y (the dependent variable). It is assumed that the values taken on by Y are mostly explained by a linear relationship between X and Y with some variation explained as errors (random deviations from the linear model).

For example, suppose we have two thermometers which measure the daily temperature, one in Farenheit and one in Celsius. We take 10 measurements on 10 different days, obtaining the following pairs of values (X is the Farenheit measurements and Y is the Celsius), which for convenience we have sorted along the X axis:

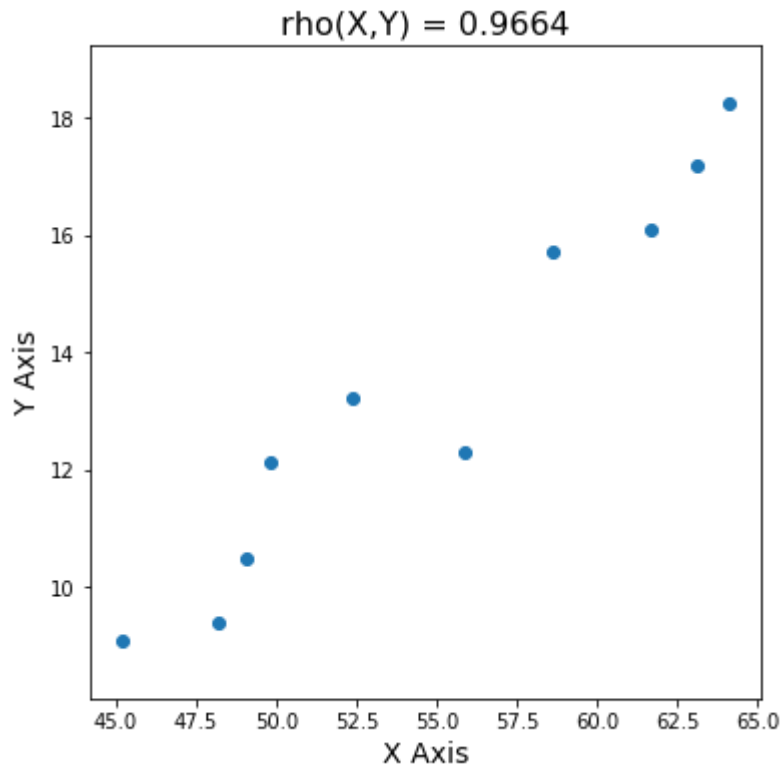
```
[(45.2, 9.0994), (48.2, 9.4023), (49.1, 10.4809), (49.8, 12.132), (52.4, 1
3.2032), (55.9, 12.303), (58.6, 15.7304), (61.7, 16.1), (63.1, 17.1773),
(64.1, 18.2468)]
```

and which correspond to the following lists of measurements:

In [37]:

```
X = [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Y = [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.1, 17.1773, 1
8.2468]

displayXY(X,Y)
print()
#ScatterTrendline(X,Y)
```

```
X = [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Y = [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.
1, 17.1773, 18.2468]
```



The correlation coefficient is $\rho = 0.9664$, so the experimental data show a strong linear correlation. It seems that we should be able to find a line that summarizes this linear correlation, kind of a 2D way of measuring the "central tendency" which we first saw with the mean of a 1D set of data.

Therefore we will assume we have a linear model of the relationship with unknown parameters θ_0 and θ_1. So we are trying to find a model

$$\theta_0 + \theta_1 X$$

which ***best*** fits the data, by determining θ_0 and θ_1. (This is just the equation for a line, e.g., $y = mx + b$, but using notation which is more common in the literature.)

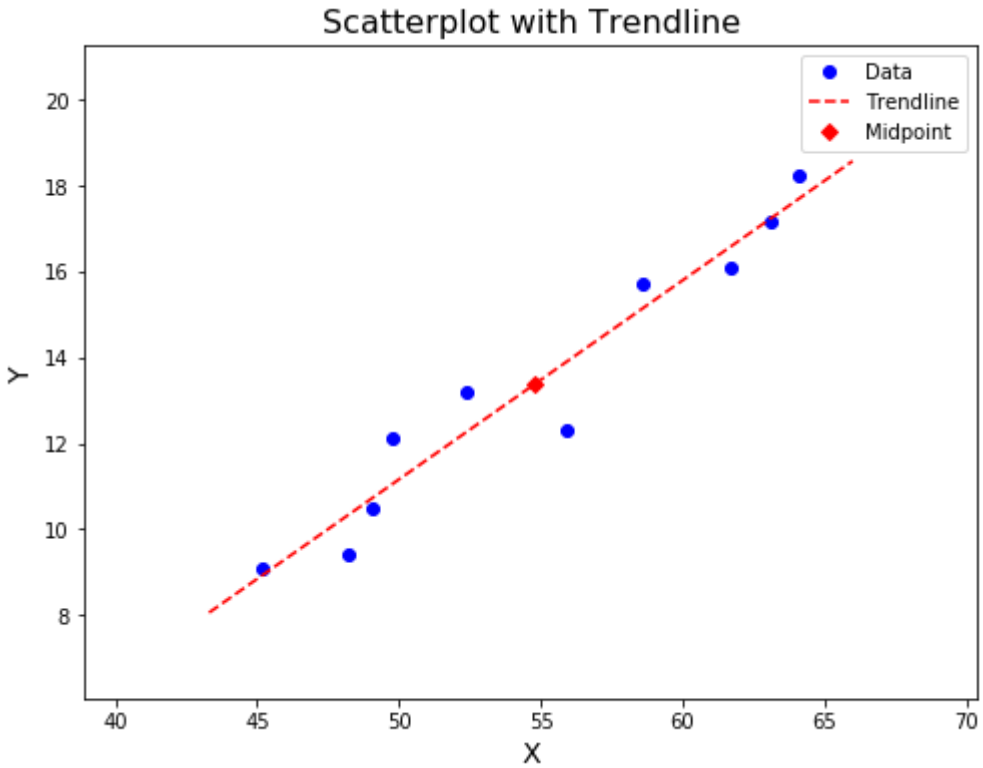
The problem is, of course, what does ***best fits the data*** mean? Since the points do not in fact fall in a line, there is a general linear trend with deviations (or "errors" or "residuals") from that trend. The predicted values due to the model will be denoted as

$$\hat{y}_i = \theta_0 + \theta_1 x_i$$

(where we denote the predicted nature of the variable by a hat). The trendline gives us a model of the data, which attempts to explain why Y varies when X changes.

In [38]:

```
X = [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Y = [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.1, 17.1773, 1
8.2468]
ScatterTrendline(X,Y)
```

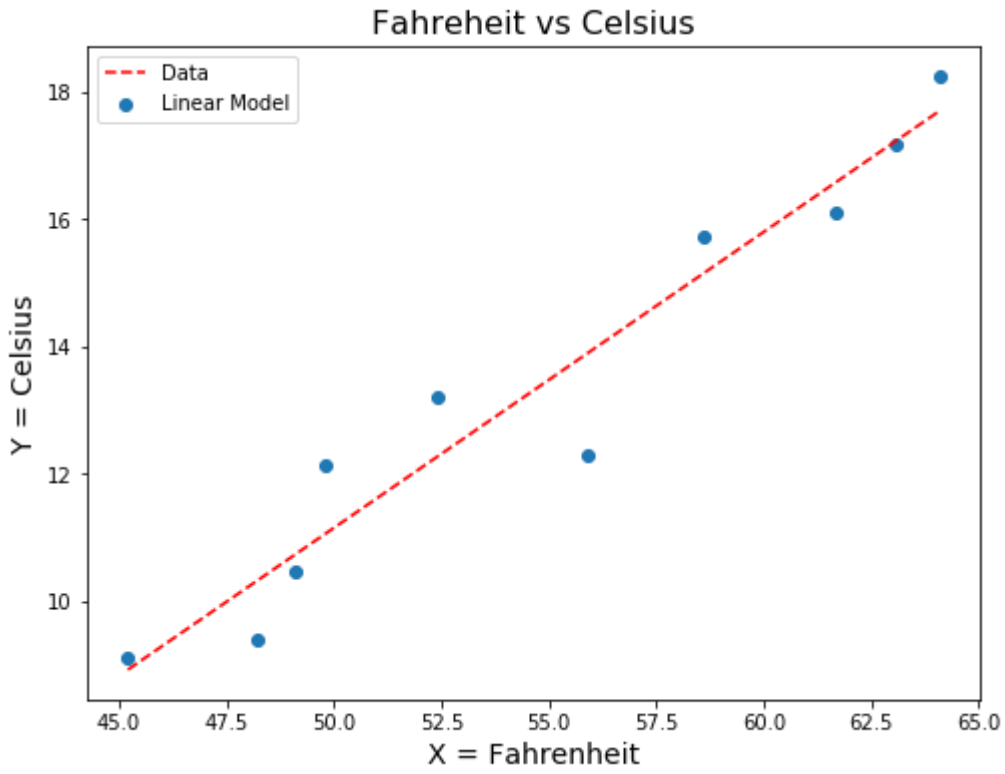



```
mean(x):        54.81   std(x): 6.4623
mean(y):        13.3875 std(y): 3.1037

rho:   0.9664   r^2:   0.9339

Residual SS:   6.3631   Regression SS: 89.9635  Total SS:   96.3265

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

In [39]:

```
X = [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Y = [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.1, 17.1773, 1
8.2468]
Yhat = [8.9254, 10.3177, 10.7354, 11.0603, 12.2669, 13.8913, 15.1444, 16.5831, 1
7.2328, 17.6969]
E = [ Y[i] - Yhat[i] for i in range(len(Y)) ]
print("X:",X)
print("Y:",Y)
print("YHat:",round4List(Yhat))
print("E:",round4List(E))
print("sum(E):",round4(sum(E)))

#def line(x):
#    return 0.4641 * x - 12.0519

#YHat = [line(x) for x in X]


plt.figure(figsize=(8,6))
plt.scatter(X,Y)
plt.plot(X,Yhat,'r--')
plt.title("Fahreheit vs Celsius",fontsize=16)
plt.legend(["Data","Linear Model"],loc='best')
plt.xlabel("X = Fahrenheit",fontsize=14)
plt.ylabel("Y = Celsius",fontsize=14)
plt.show()
```

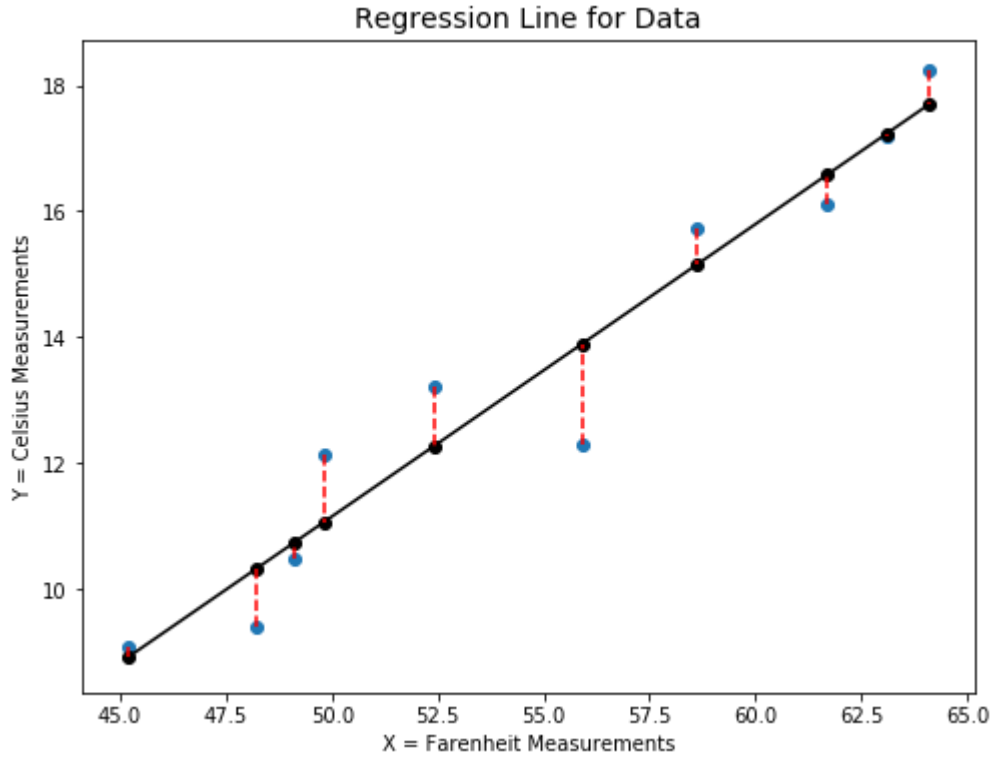
```
X: [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Y: [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.1,
17.1773, 18.2468]
YHat: [8.9254, 10.3177, 10.7354, 11.0603, 12.2669, 13.8913, 15.1444,
16.5831, 17.2328, 17.6969]
E: [0.174, -0.9154, -0.2545, 1.0717, 0.9363, -1.5883, 0.586, -0.483
1, -0.0555, 0.5499]
sum(E): 0.0211
```

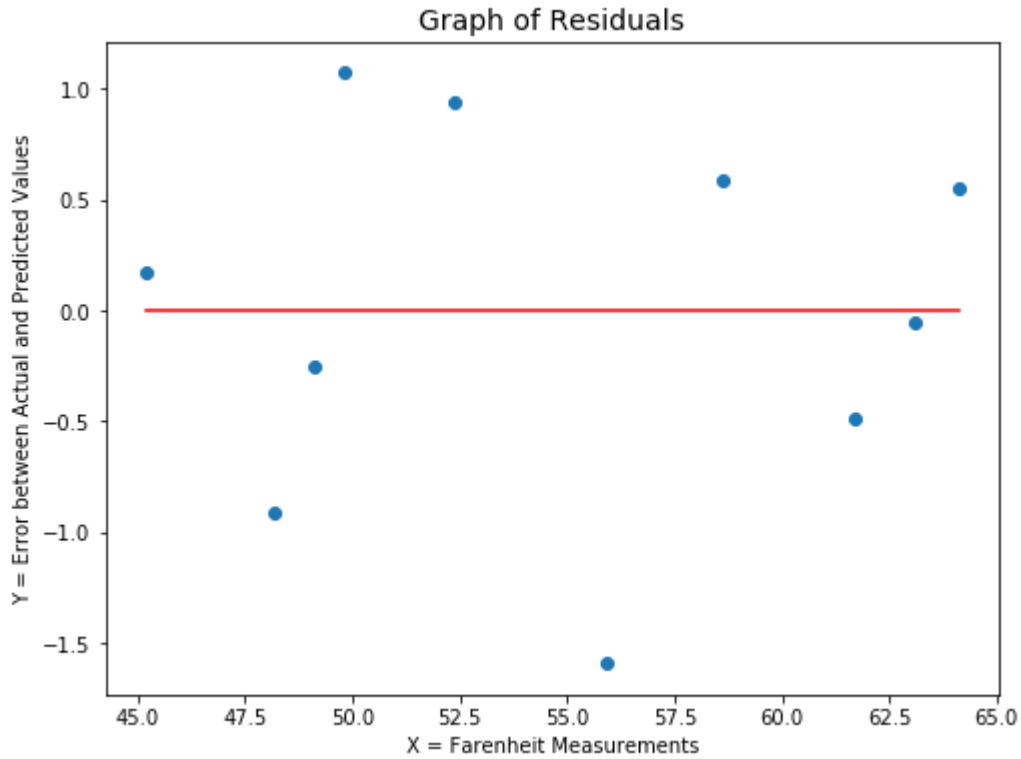


The deviations are the ***errors*** in the model, or simply factors in the random process that are not accounted for by the model. Factoring out the line to just look at the distribution of the errors can tell us quite a lot about our model.

In [40]:

```
X = [45.2, 48.2, 49.1, 49.8, 52.4, 55.9, 58.6, 61.7, 63.1, 64.1]
Y = [9.0994, 9.4023, 10.4809, 12.132, 13.2032, 12.303, 15.7304, 16.1, 17.1773, 1
8.2468]
Yhat = [8.9254, 10.3177, 10.7354, 11.0603, 12.2669, 13.8913, 15.1444, 16.5831, 1
7.2328, 17.6969]


plt.figure(figsize=(8,6))
plt.title("Regression Line for Data",fontsize=14)
plt.xlabel("X = Farenheit Measurements")
plt.ylabel("Y = Celsius Measurements")
plt.scatter(X,Y)
plt.plot(X,Yhat,color='black')
plt.scatter(X,Yhat,marker='o',color="black")
for k in range(len(X)):
    plt.plot([X[k],X[k]],[Y[k],Yhat[k]], '--', color='red')
plt.show()

plt.figure(figsize=(8,6))
plt.title("Graph of Residuals",fontsize=14)
plt.xlabel("X = Farenheit Measurements")
plt.ylabel("Y = Error between Actual and Predicted Values")
z = [0 for x in X ]
#plt.xlim(-3,4)
plt.scatter(X,E)
plt.plot(X,z,color='red')
plt.show()
print("Sum of errors: ",sum(E))
```






```
Sum of errors:  0.021100000000000563
```

Summary

$$\hat{y}_i = \theta_0 + \theta_1 x_i$$
$$y_i = \theta_0 + \theta_1 x_i + e_i.$$

The goal of Linear Regression is to determine the parameters θ_1 and θ_0 which minimize the errors; as with calculating the variance, we can not simply add the errors together (since those of opposite sign would cancel out) and using absolute values is a pain, so we square the errors, obtaining the Residual Sum of Squares (RSS):

$$RSS = \sum_{i=1}^{n} e_i^2.$$

which is also sometimes expressed as the Mean Square Error (MSE):

$$MSE = \frac{\sum_{i=1}^{n} e_i^2}{n}.$$

(Note that minimizing one is the same as minimizing the other, since n is a constant.)

There is a theoretical way to determine θ_1 and θ_0 and an experimental way; we consider the theoretical way now, and the experimental in the next lecture.

Calculating the Regression Line

Simply put, for any set of points for which ρ is defined there is a formula for the ***regression line*** which minimizes the RSS.

$$\rho(X,Y) = \frac{E(X*Y) - \mu_X * \mu_Y}{\sigma_X * \sigma_Y}$$
$$\theta_1 = \frac{Cov(X,Y)}{Var(X)} = \rho(X,Y)\frac{\sigma_Y}{\sigma_X}$$

Then, using the fact that the midterm is on the regression line, we can easily calculate the y-intercept:

$$\mu_Y = \theta_0 + \theta_1 \mu_X$$

and so

$$\theta_0 = \mu_Y - \theta_1 \mu_X$$

Thus, in our running example, we find that the linear regression line for this data would be

$$\hat{Y} = -12.0519 + 0.4641 * X$$

The symbol y indicates that the value for y has been estimated, hence, we have a set of estimated values for y:

$$\hat{Y} = [(-12.0519 + 0.4641 * x) \text{ for } x \text{ in } X]$$

that is,

$$\hat{Y} = [8.9254, 10.3177, 10.7354, 11.0603, 12.2669, 13.8913, 15.1444, 16.5831, 17.2328, 17.6969]$$

In other words, our estimates for the actual (unknown) parameters are

$$\theta_0 = -12.0519$$

and

$$\theta_1 = 0.4641$$

The residuals are thus:

$$e_i = (y_i - y_i) \quad \text{for } n = 1, \ldots, n,$$

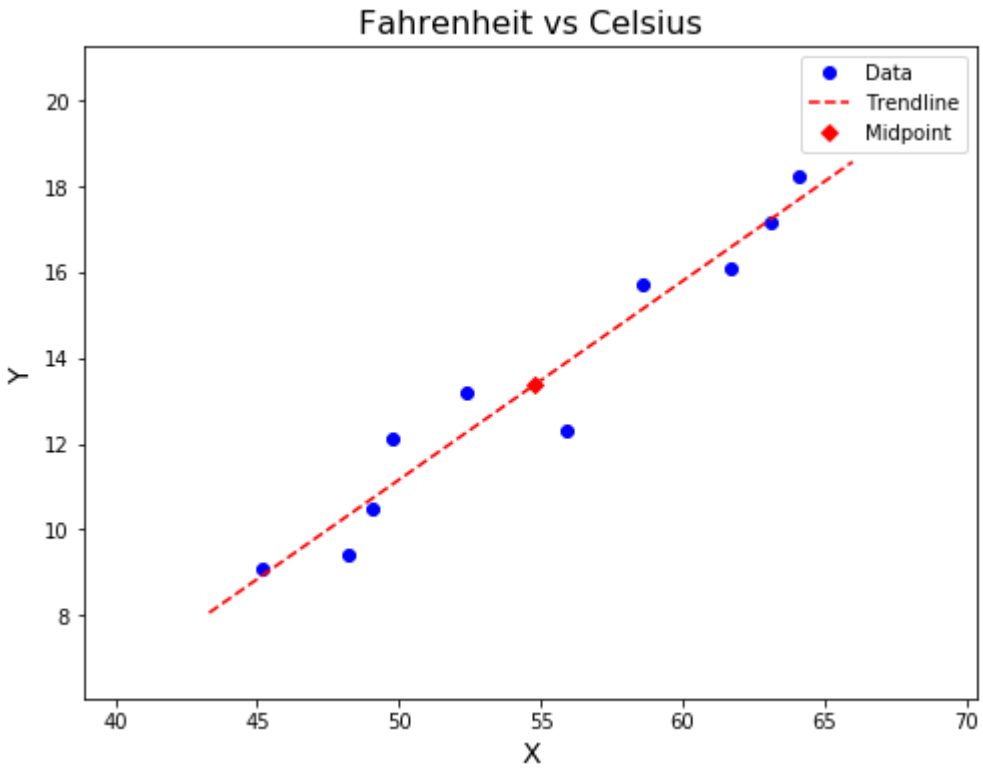
or

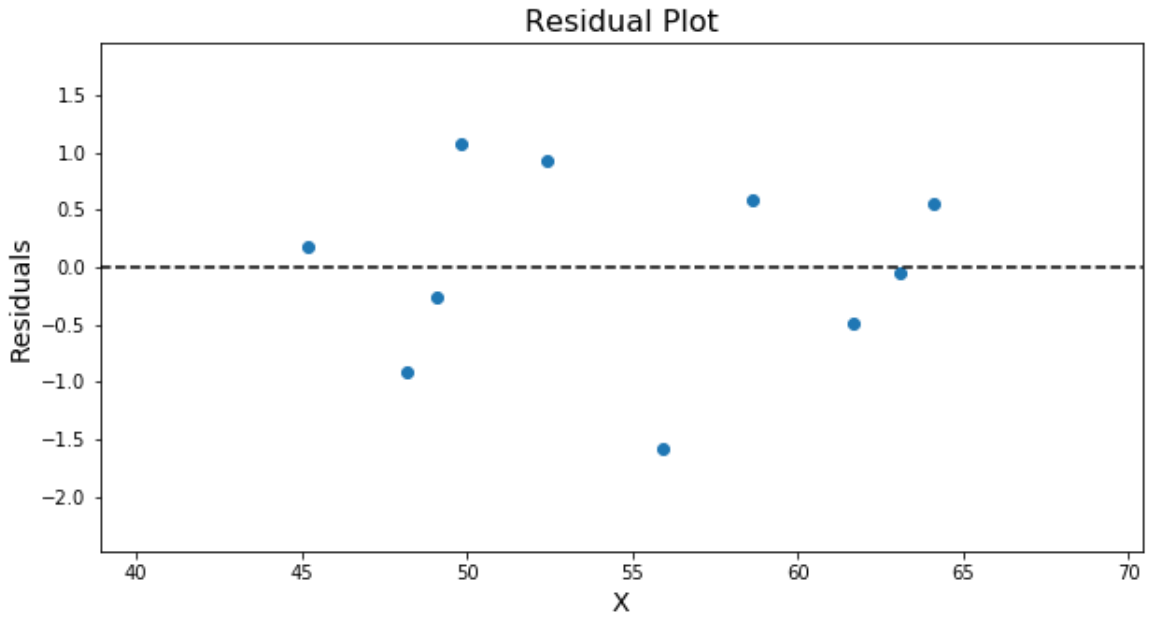
$$[0.5399, -0.274, 0.5877, -0.5188, -1.4347, 0.2941, 1.5323, 0.2958, 0.1317, 0.6001]$$

and the RSS = 6.3631.

In [41]:

```
ScatterTrendline(X,Y,"Fahrenheit vs Celsius",showResiduals=True)
```






```
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   Regression SS: 89.9635  Total SS:   96.3265

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

In [42]:

```
# RSS

def sumSqDiff(X,Y):
    s = 0
    for k in range(len(X)):
        s += (X[k] - Y[k])**2
    return s

print("RSS: ", round4(sumSqDiff(Y,Yhat)))
print("MSE: ", round4(sumSqDiff(Y,Yhat)/len(X)))
```

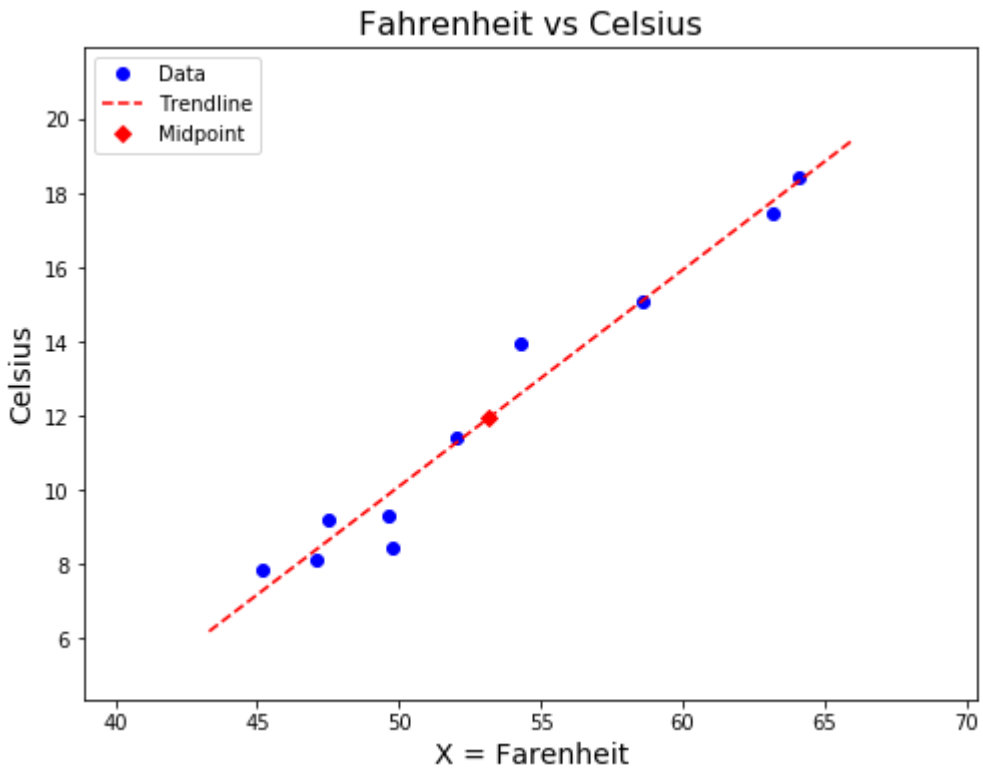
```
RSS:  6.3632
MSE:  0.6363
```

Linear Regression Lines: Examples

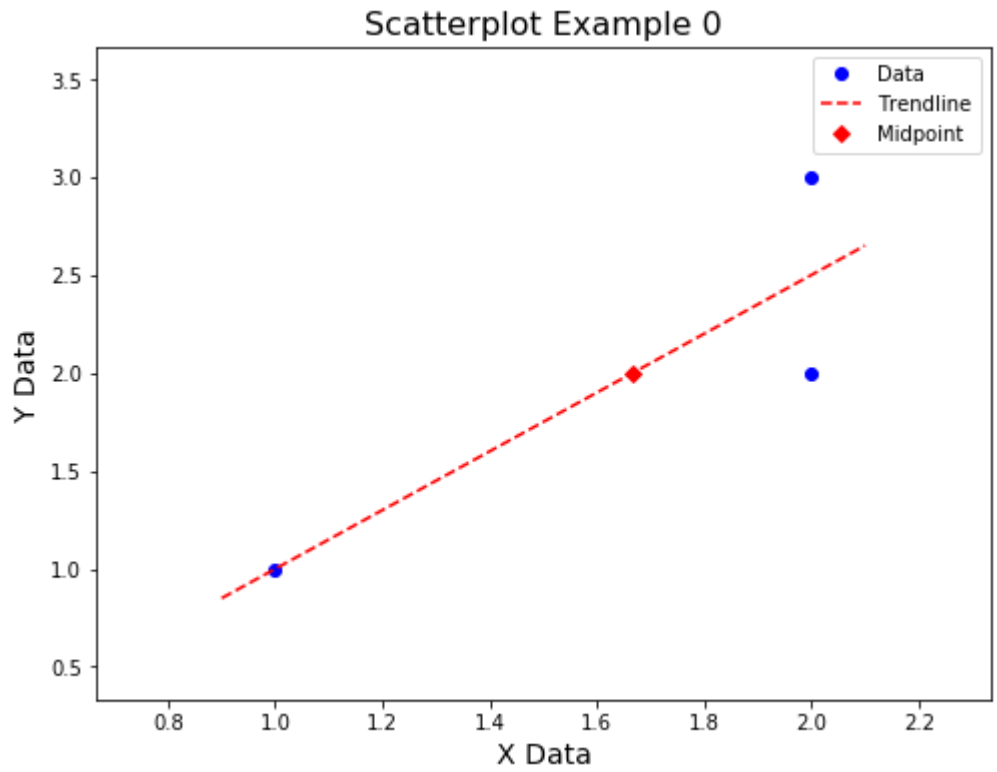
In [43]:

```
def Ex0():
    x = [1,2,2]
    y = [1,2,3]
    ScatterTrendline(x,y,titl="Scatterplot Example 0",xlab="X Data", ylab="Y Dat
a")



def Ex1():
    x = [5,6,6,7,7,8]
    y = [6,6,7,8,9,10]
    ScatterTrendline(x,y,titl="Scatterplot Example 1",xlab="X Data", ylab="Y Dat
a")


def Ex2():
#    x,y = getBioMetricData()
    x = [50,45,40,38,32,40,55]
    y = [2.5,5.0,6.2,7.4,8.3,4.7,1.8]
    ScatterTrendline(x,y,"Scatterplot Example 2","X Data", "Y Data")


def Ex3():
    studs = pd.read_csv("biometricdata.csv")
    x = [0]*len(studs)
    y = [0]*len(studs)
    for i in range(len(studs)):
        x[i] = studs['Height'][i]
        y[i] = studs['Weight'][i]
    ScatterTrendline(x,y,"Height vs Weight for 25K Individuals","Height", "Weigh
t")

def Ex4():
    studs = pd.read_csv("StudentData3.csv")
    x = [0]*len(studs)
    y = [0]*len(studs)
    for i in range(len(studs)):
        x[i] = studs['SAT_TOTAL'][i]
        y[i] = studs['BU_GPA'][i]
    ScatterTrendline(x,y,titl="SAT vs GPA", xlab="X = Sat Total",ylab="Y = BU GP
A")

def Ex5():
    studs = pd.read_csv("StudentData3.csv")
    x = [0]*len(studs)
    y = [0]*len(studs)
    for i in range(len(studs)):
        x[i] = studs['HS_GPA'][i]
        y[i] = studs['BU_GPA'][i]
    ScatterTrendline(x,y,titl="HS GPA vs BU GPA", xlab="X = HS GPA",ylab="Y = BU
GPA")

def midterm():
    ScatterTrendline(midtermX,midtermY,titl="CS 237 Midterm Data", xlab="X = MT
 Score",ylab="Y = Time Turned In")

X = [45.2, 47.1, 47.5, 49.6, 49.8, 52.0, 54.3, 58.6, 63.2, 64.1]
Y = [7.8752, 8.117, 9.2009, 9.3167, 8.4564, 11.4075, 13.9236, 15.0762, 17.4678,
```

```
18.4362]

ScatterTrendline(X,Y,"Fahrenheit vs Celsius",xlab="X = Farenheit",ylab="Celsius"
)

print("\n\nExample 0")
Ex0()
print("\n\nExample 1")
Ex1()
print("\n\nExample 2")
Ex2()
print("\n\nExample 3")
Ex3()
print("\n\nExample 4")
Ex4()
print("\n\nExample 5")
Ex5()
print("\n\nMidterm Data")
midterm()
```

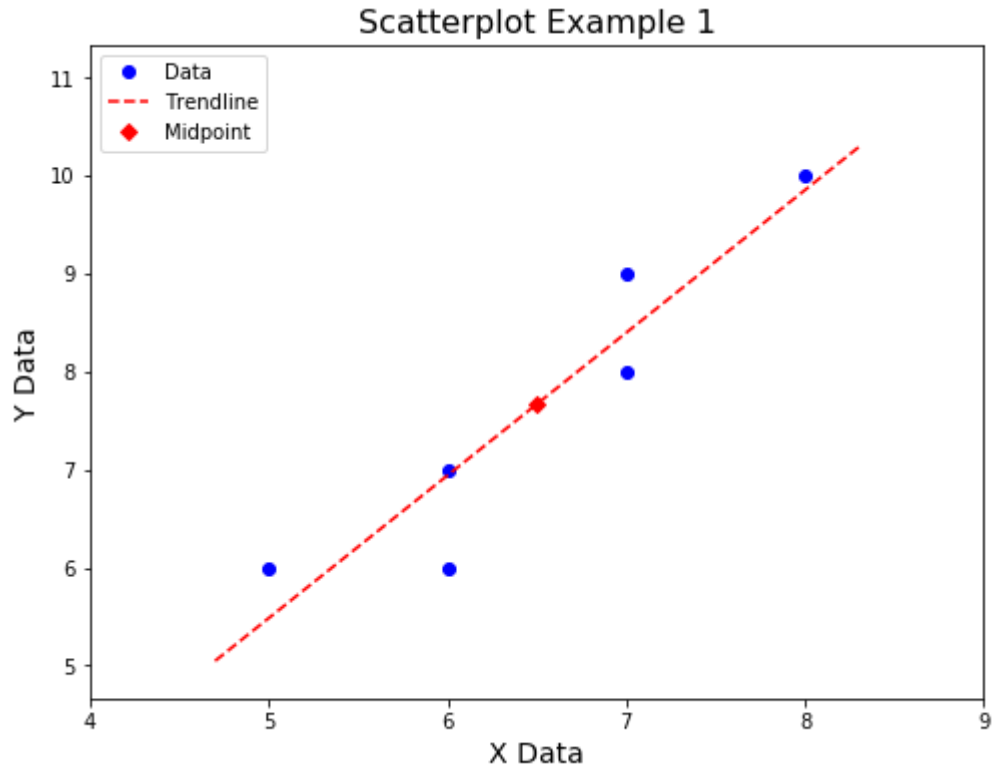



```
mean(x):        53.14   std(x): 6.3938
mean(y):        11.9278 std(y): 3.8009

rho:   0.9817   r^2:   0.9638

Residual SS:   5.2364   Regression SS: 139.231  Total SS:   144.4674

Regression Line: y = 0.5836 * x - 19.0844


Example 0
```



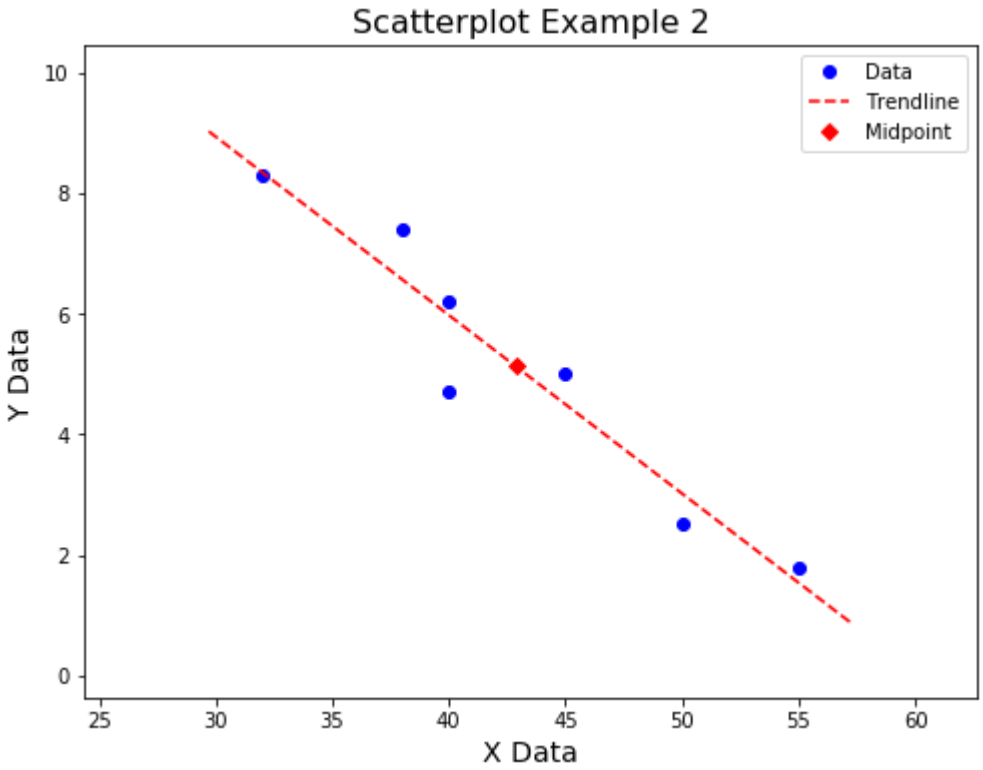

```
mean(x):         1.6667  std(x): 0.4714
mean(y):         2.0     std(y): 0.8165

rho:   0.866    r^2:   0.75

Residual SS:   0.5      Regression SS: 1.5      Total SS:   2.0

Regression Line: y = 1.5 * x - 0.5
```

```
Example 1
```



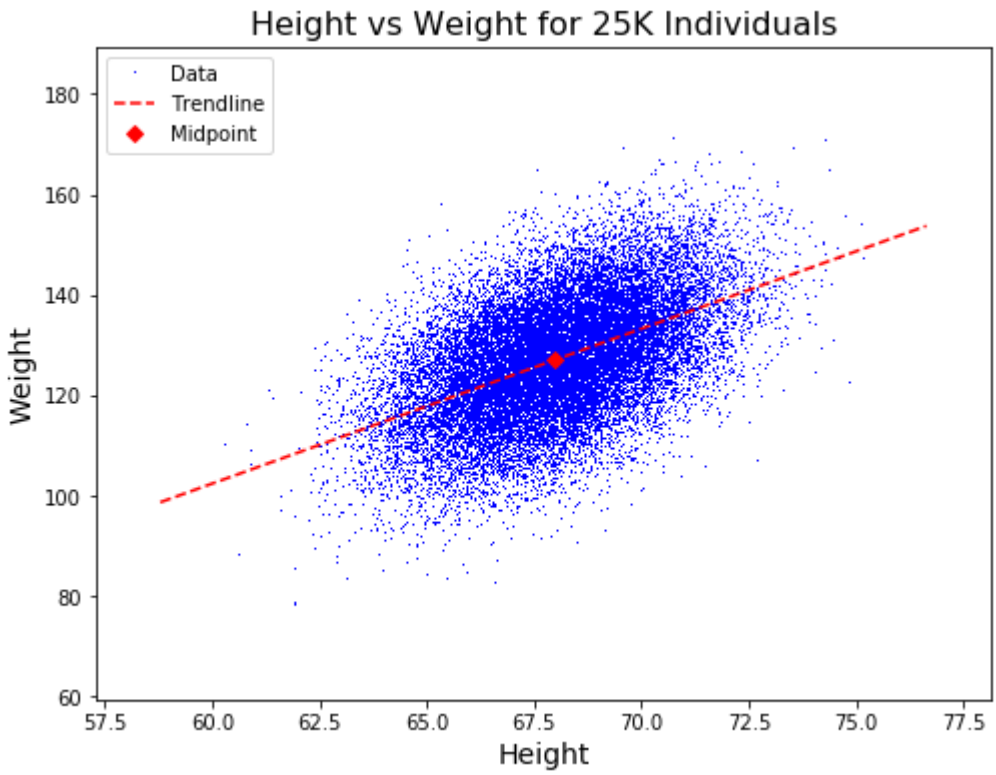
```
mean(x):        6.5     std(x): 0.9574
mean(y):        7.6667  std(y): 1.4907

rho:   0.9342   r^2:   0.8727

Residual SS:   1.697    Regression SS: 11.6364  Total SS:   13.3333

Regression Line: y = 1.4545 * x - 1.7879
```

Example 2

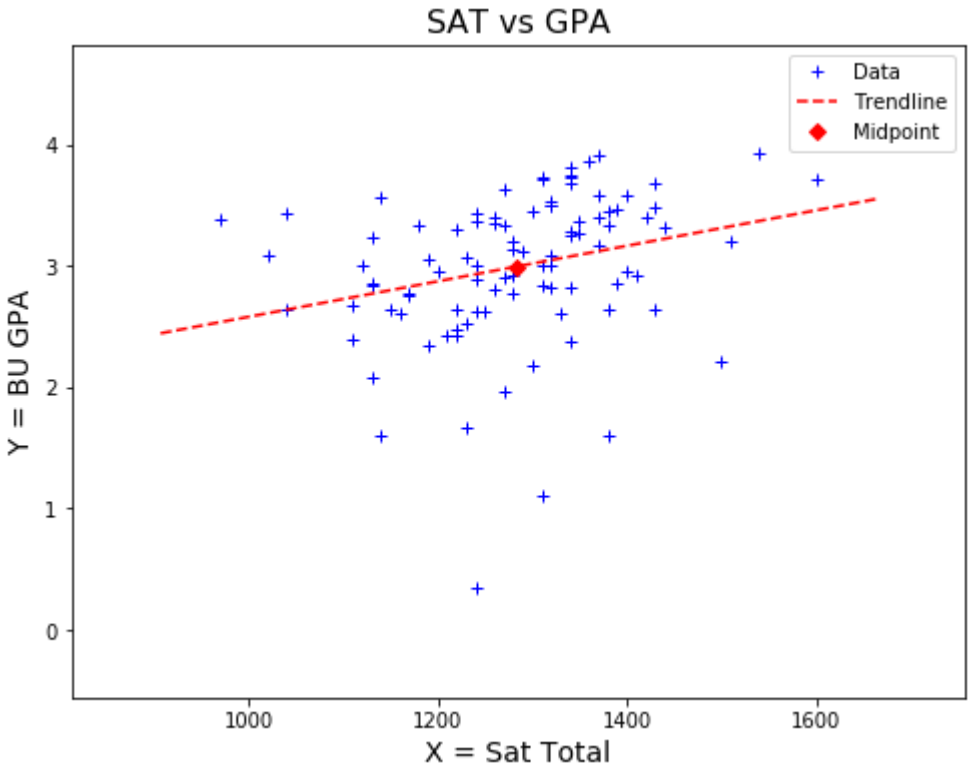



```
mean(x):        42.8571 std(x): 7.1799
mean(y):        5.1286  std(y): 2.2218

rho:   -0.9562  r^2:   0.9143

Residual SS:   2.9625   Regression SS: 31.5918  Total SS:   34.5543

Regression Line: y = -0.2959 * x + 17.8093
```

Example 3

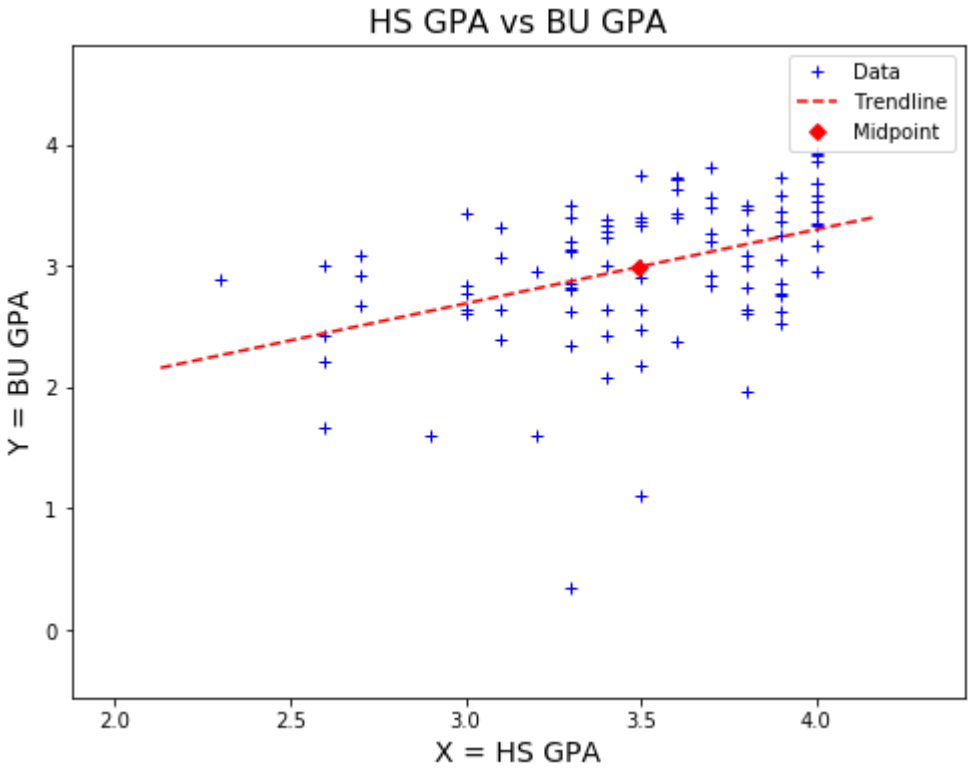



```
mean(x):         67.9931 std(x): 1.9016
mean(y):         127.0794        std(y): 11.6607

rho:   0.5029   r^2:   0.2529

Residual SS:   2539713.3122     Regression SS: 859564.011       Tota
l SS:   3399277.3232

Regression Line: y = 3.0835 * x - 82.5757


Example 4
```

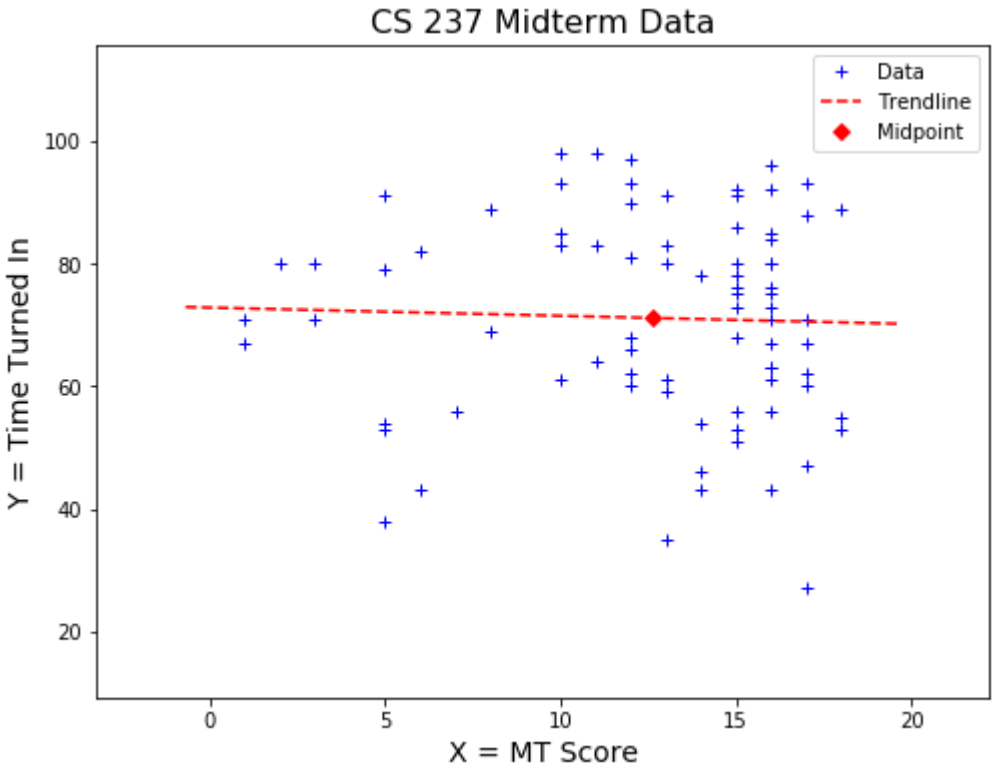



```
mean(x):        1282.4211       std(x): 112.7056
mean(y):        2.9943  std(y): 0.6129

rho:   0.269    r^2:   0.0724

Residual SS:   33.1077  Regression SS: 2.583    Total SS:   35.6907

Regression Line: y = 0.0015 * x + 1.1181
```

Example 5




```
mean(x):        3.4968  std(x): 0.4035
mean(y):        2.9943  std(y): 0.6129

rho:   0.4015   r^2:   0.1612

Residual SS:   29.9369  Regression SS: 5.7538   Total SS:   35.6907

Regression Line: y = 0.6099 * x + 0.8617


Midterm Data
```




```
mean(x):        12.6125 std(x): 4.4229
mean(y):        71.1375 std(y): 16.4474

rho:   -0.036   r^2:   0.0013

Residual SS:   21613.3788       Regression SS: 28.1087  Total SS:
21641.4875

Regression Line: y = -0.134 * x + 72.8278
```