In [181]:
```python
import numpy as np
import pandas as pd
import  matplotlib.pyplot as plt
import seaborn as sns
#set seed for random elements
np.random.seed(100)
```

Statistical Models

This is a space where I develop my analytical skills through developing a deep understanding of Statistics

Linear Regression

$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_i$

With loss function of $\frac{\sum (y_i - \hat{y}_i)}{N}$

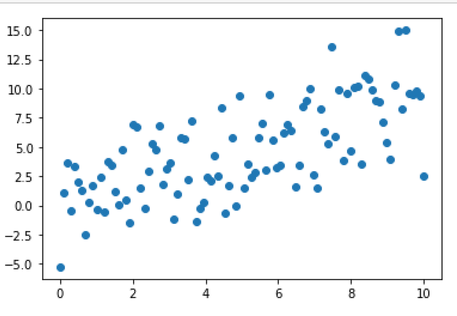
Equivalent to $\frac{\sum (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))}{N}$

Where after differentiating $\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{(x_i - \bar{x})^2}$

And $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$

Create Random data from a sequence by adding a random factor

In [182]:
```python
x = np.linspace(0,10,num =100)
random = np.random.normal(0,3,100)
y = x + random
plt.scatter(x,y)
plt.show()
```

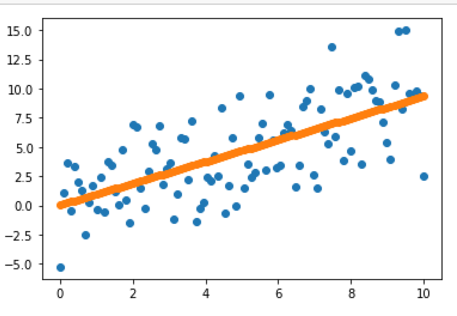


Create a regression model

In [183]:
```python
#executing formulas with python code
ssx = (x - np.mean(x))
ssy = (y - np.mean(y))
b_1 =  np.sum(  ssx * ssy)/np.sum(ssx**2)
b_0 = np.mean(y) - b_1*np.mean(x)
y_hat = b_0 + b_1*x
```

Seems like the code worked! Lets take it a step further

In [184]:
```python
plt.scatter(x,y)
plt.scatter( x,y_hat)
plt.show()
```



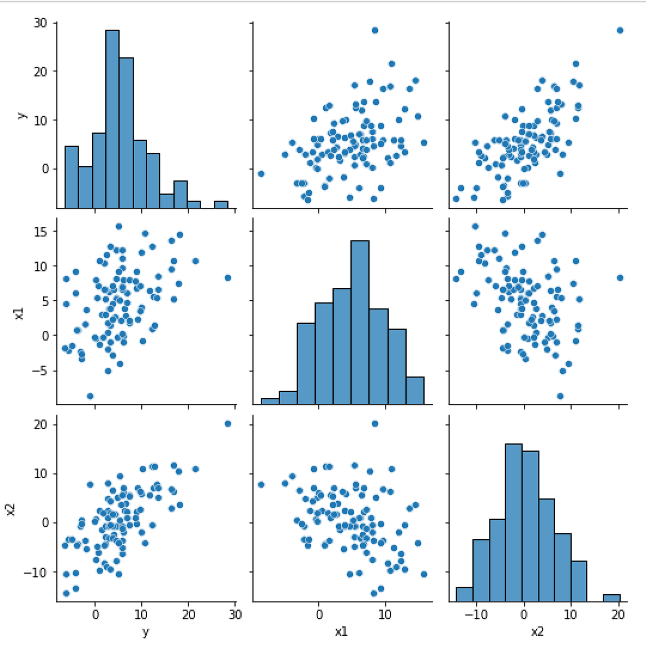
Multivariate Regression

$\hat{Y} = \sum \hat{\beta}_j x_{ij}$

Now this is the same just with more variables so we'll reproduce the sample but now in 4 dimensions

In [185]:
```python
random1 = np.random.normal(0,3,100)
random2 = np.random.normal(15,5,100)
x1 = np.linspace(0,10,num =100)+random1
x2 = -np.linspace(10,20,100) +random2

Y = (x1 + x2 )
data = pd.DataFrame({"y":Y,"x1":x1,"x2":x2})
sns.pairplot(data)
plt.show()
```



Above we can see the distributions of the generated data, seems like the data is as expected

Now again we will take the loss function and derive it

Loss function of $\frac{\sum (Y_i - \hat{Y}_i)}{N}$

Loss function of $\frac{\sum (Y_i - (\sum \hat{\beta}_j x_{ij}))}{N}$**>**

$\hat{\beta} = (X^T X)^{-1} X^T Y$

In [186]:
```python
#create x matrix
x0= np.ones(100)
X=np.array([x0,x1,x2]).T
B=np.linalg.inv(X.T@X)@X.T@Y
```

In [187]:
```python
X0=B[0]*X[0]
X1=B[1]*X[1]
X2=B[2]*X[2]
Y_hat= x0+x1+x2
```

Now due to us not being able to clearly see the results, we will use a more mathematical approach

R2 which is simply a way to "score" how well we did

$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$

$SS_{\text{res}} = \sum_i (y_i - \hat{y}_i)^2$

$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$

Lets see

In [188]:
```python
ssres=np.sum( (Y  - Y_hat)**2)
sst = np.sum ((Y - np.mean(Y))**2)
R2 = 1 - ssres/sst
print("R squared is ", np.round(R2,3))
```

```
R squared is  0.975
```

We did pretty well!

Logistic Regression

Now lets the define the logistic model

Now assuming a linear relashionship between the predictors and the log odds we get

$\ln \frac{p}{1-p} = \beta_0 + \sum \beta_j x_j$

Re arange to solve for p= $P(Y) = 1$

$p = \frac{1}{1 + e^{-(\beta_0 + \sum \beta_j x_j)}}$

A more genereal representation

$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}}$

Lets checkout the generated data

In [189]:
```python
def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))

def predict_sigmoid(x,beta):
    z= np.dot(x,beta)
    return sigmoid(z)
```

the loss function is derived using Maximum likelihood

Below we have the log likelihood(we can use either one but log is much easier to work with)

$-\frac{1}{n} \sum y_i \ln h_\theta(x_i) + (1 - y_i) \ln(1 - h_\theta(x_i))$

In [190]:
```python
def logistic_loss(y,h):
    y_is_1 = -y * np.log(h)
    y_is_0 = (1-y)*(np.log(1-h))
    cost = y_is_1 - y_is_0
    logistic_loss = np.sum(cost)/len(y)
    return logistic_loss
```

Now basic gradient descent using betas as our sequancially updated parameter

$\beta_{n+1} = \beta_n - \gamma \nabla F(\beta_n)$

Here γ is our learning rate or how fast we jump around the curve and $\nabla F(\beta_n)$ is the gradient of our loss function evaluated at the current value of β this simple algorithm allows us to find a minimum in our loss function

See simple stuff

In [191]:
```python
def update_betas(beta,learning_rate,x,y,logistic_loss):
    gradient = np.dot(x.T,(logistic_loss-y) )
    gradient *= learning_rate
    gradient /= len(x)
    beta -= gradient
    return beta
```

In [192]:
```python
def wallmart_gradient_descent(y,x,beta,iterations,learning_rate,thresh):
    loss = []
    iter_loss = logistic_loss(y,predict_sigmoid(x,beta))
    for i in range(iterations):

        loss.append(iter_loss)
        beta = update_betas(beta,learning_rate,x,y,iter_loss)
        iter_loss = logistic_loss(y,predict_sigmoid(x,beta))
        if( np.absolute(iter_loss) <= thresh):
            break;



    return beta,loss
```

In [193]:
```python
print( np.shape(betas),np.shape(y), np.shape(X))
```

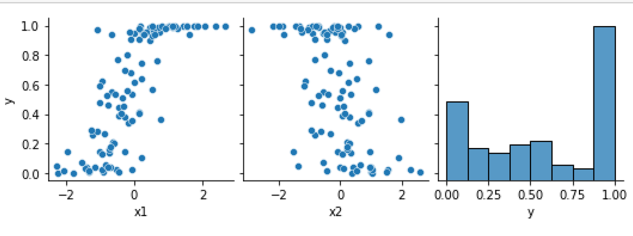
```
(3, 1) (100,) (100, 3)
```

Ok we have a basis, lets generate some data and experiment

In [201]:
```python
x1 = np.random.randn(100)
x2 = np.random.randn(100)
z = 1 + 3*x1 -2*x2
p = 1 / ( 1 + np.exp(-z))
y = np.random.binomial(1,p,100)
data = pd.DataFrame({'y':p,'x1':x1,'x2':x2})
betas = np.ones((3,1))
y=np.array([y]).T
intercept = np.ones( (100))
X = np.array([intercept,x1,x2]).T
```

Lets checkout the generated data

In [202]:
```python
sns.pairplot(data,x_vars = ['x1','x2','y'] , y_vars = ['y'])
plt.show()
```



now we know the true regression coefficients $\beta_1 = 3$ and $\beta_2 = -2$

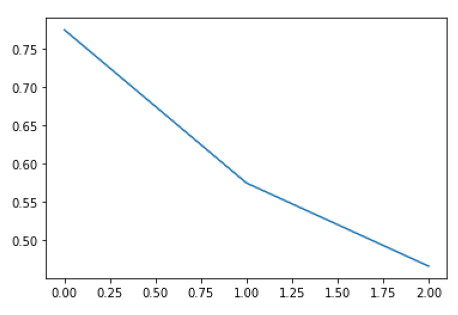
but lets solve for the using the logistic model

In [196]:
```python
final_beta,loss_list = wallmart_gradient_descent(y,X,betas,15,2,.45)
```

In [203]:
```python
print("betas",final_beta)
plt.plot(loss_list)
```

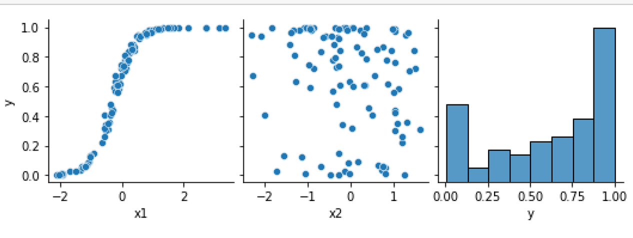
```
betas [[ 0.90798287]
 [ 2.93103401]
 [-0.21905637]]
```

Out[203]:
```
[<matplotlib.lines.Line2D at 0x20989e22648>]
```



In [198]:
```python
y_hat = predict_sigmoid(X,final_beta)
y_hat = np.reshape(y_hat,(100))
data = pd.DataFrame({'y':y_hat,'x1':x1,'x2':x2})
```

In [200]:
```python
sns.pairplot(data,x_vars = ['x1','x2','y'] , y_vars = ['y'])
plt.show()
```



Since we know the true values of beta we cacn do a direct comparison of the model efficiancy

in which we see we did a good job of prediction beta_0 and beta_1 but not sucha good job with beta_2

a more genereal and useful approach would be to use a Likelihood ratio test but for I will save that for when I do model evaluations

We could tweak the learning rate and try and optimizez for this specific problem, but for now this fulfills the objective of the example.

In [ ]: