In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Task 1: Implementing MSE Loss Function

What is a loss/cost function?

Explanations:

1. It's a function that determines how well a Machine Learning model performs for a given set of data
2. Cost function helps us reach the optimal solution
3. A mechanism that returns the error between predicted outcomes and the actual outcomes

In this code block, we will implement one type of loss function:

- Mean Squared Error $\frac{1}{m} * \sum_{i=1}^{m}(y_{pred}^{i} - y^{i})^2$

There are many more cost functions that Machine Learning programmers use. You can check out this link.

In [7]:
```python
#The mean squared error function computes the mean squared error between point guesses a
def MSE(y_pred, y):
  """
  INPUT:
  y_pred : a numpy array of values that your model predict
  y : a numpy array of ground truth labels/values

  OUTPUT:
  Mean Squared Error Loss
  """
  ########################
  ## YOUR CODE STARTS HERE
  ########################
  m = len(y)
  sum_error_squared = np.sum(np.square(y_pred - y))
  return sum_error_squared / m


  ########################
  ## YOUR CODE ENDS HERE
  ########################
```

Task 2: Fitting a line by brute force

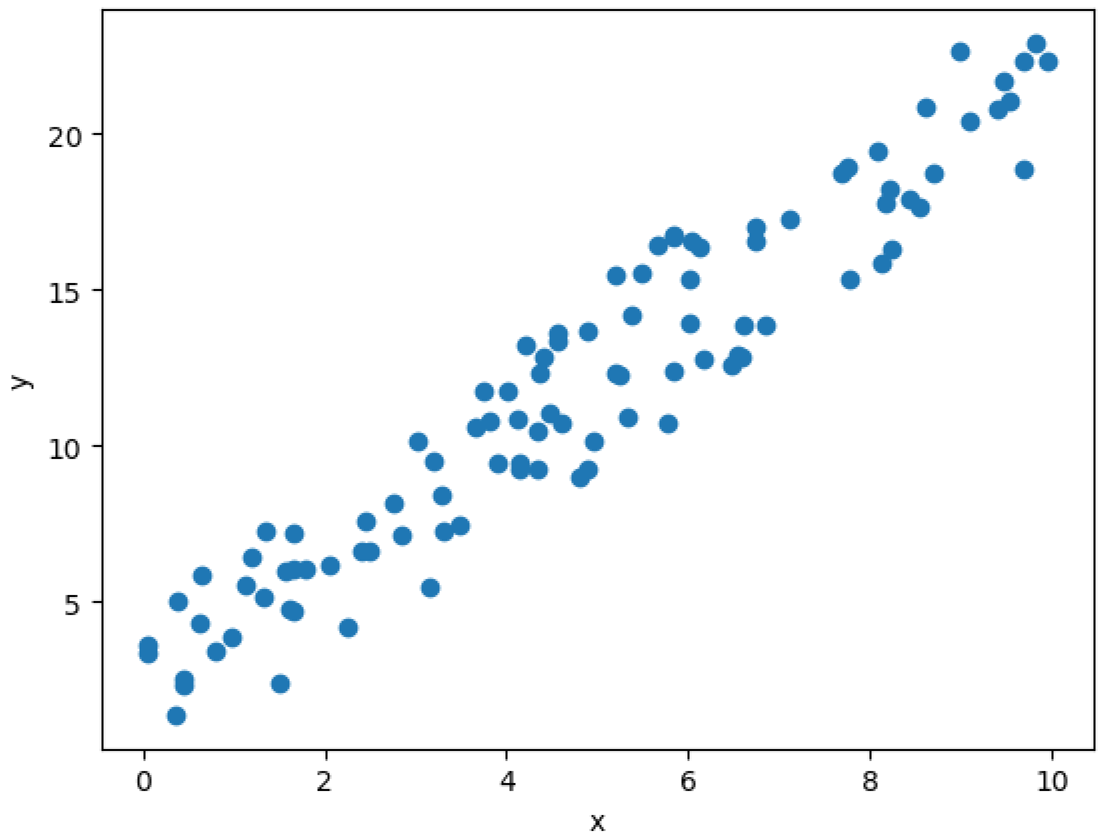
1. Download and import the line_fitting.csv data from Bruinlearn
2. Graph the points using plt.scatter() - note that the first column of the csv is the x coordinates and the second is the y coordinates
3. Find values of beta1 and beta0 that create a line that fits your data well
4. Use the mean squared error function provided mse() to calculate the error of your line (note: the ypred and y of MSE must be the same size, when you are calculating your y_guess be sure to calculate the guesses for the points in x_guess)

How low can you get the error of your line? Can you get the error below 3?

1. Use a brute force approach to find the lowest mean squared error you can get, try 500 values for beta0 and beta1

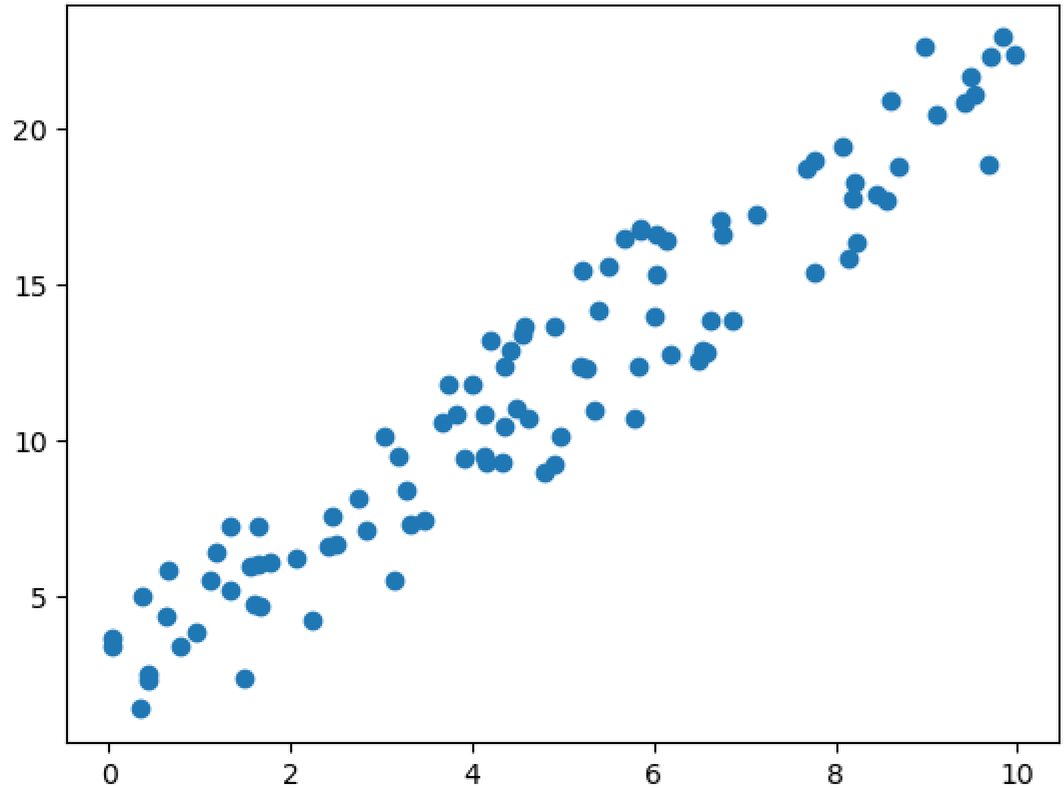
In [8]:
```
line_fitting = pd.read_csv('line_fitting.csv')

x_points = np.array(line_fitting.x)
y_points = np.array(line_fitting.y)

plt.scatter(x_points, y_points)
plt.xlabel('x')
plt.ylabel('y')
```

Out[8]:
```
Text(0, 0.5, 'y')
```



In [9]:
```
########################
## YOUR CODE STARTS HERE
########################

# guess two values for beta0 and beta1
beta0 = 0.5   # example guess
beta1 = 0.5   # example guess

# calculate your model's guesses for each x in x_points, and store it in y_guess
y_guess = beta0 + beta1 * x_points

# plot your y_guess on the same plot as the scattered points
plt.scatter(x_points, y_points)


# compute and print your mean squared error with your y_guess
error = np.mean(np.square(y_guess - y_points))
print(error)

########################
## YOUR CODE ENDS HERE
########################
```

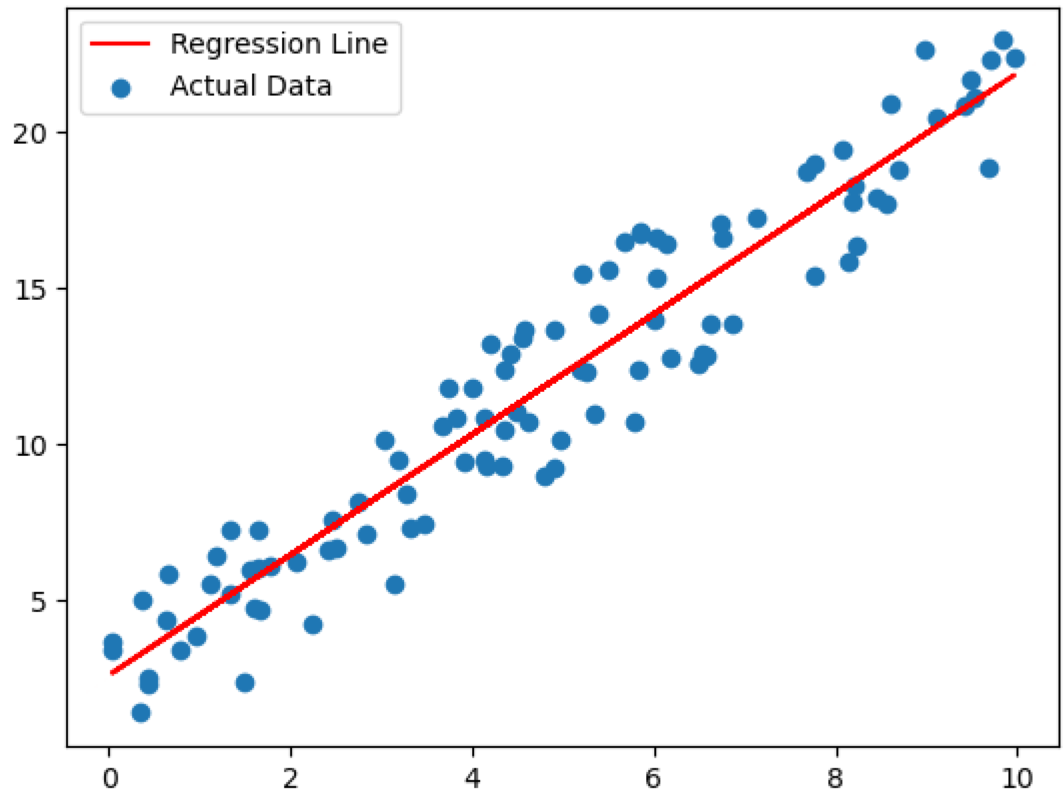
```
97.59432896772762
```




```
In [11]: # update your beta0 and beta1 to be the best value from 250,000 combinations (brute forc
best_error = float('inf')
best_params = (None, None)
for beta1_guess in np.arange(0, 5, 0.01):
  for beta0_guess in np.arange(0, 5, 0.01):
    ########################
    ## YOUR CODE STARTS HERE
    ########################
    # Calculate model's predictions
    y_guess = beta0_guess + beta1_guess * x_points
    # Compute the MSE
    error = np.mean(np.square(y_guess - y_points))
    # Update best parameters if this error is lower
    if error < best_error:
        best_error = error
        best_params = (beta0_guess, beta1_guess)
    ########################
    ## YOUR CODE ENDS HERE
    ########################

########################
## YOUR CODE STARTS HERE
########################

# print your best model parameters
beta0, beta1 = best_params
print(beta1, beta0)

# calculate your model's guesses for each x in x_points, and store it in y_guess
y_guess = beta0 + beta1 * x_points

# plot your y_guess on the same plot as the scattered points
plt.plot(x_points, y_guess, color="red", label="Regression Line")
plt.scatter(x_points, y_points, label="Actual Data")
plt.legend()
plt.show()
# compute and print your mean squared error with your y_guess
```

```
print(error)

########################
## YOUR CODE ENDS HERE
########################
```

```
1.93 2.58
```




```
363.12108822830794
```

Task 3: Fitting a line using polyfit 🚦

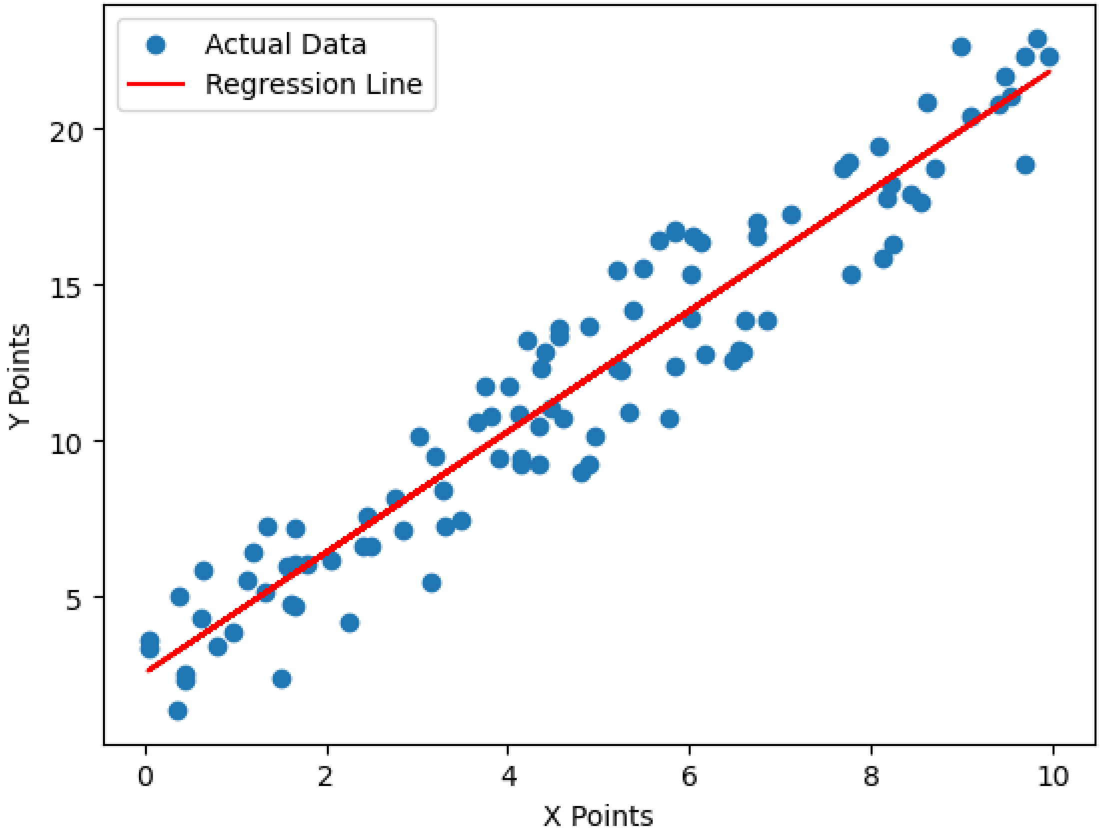
1. Use the np.polyfit() function to find the solution to the best fit line for the data
2. Calculate the mean squared error, how similar is it to the one you found by brute force?

In [12]:

```
########################
## YOUR CODE STARTS HERE
########################

# find and print values for beta0 and beta1 using np.polyfit
beta1, beta0 = np.polyfit(x_points, y_points, 1)
print(beta1, beta0)

# calculate your model's guesses for each x in x_points, and store it in y_guess
y_guess = beta0 + beta1 * x_points

# plot your y_guess on the same plot as the scattered points
plt.scatter(x_points, y_points, label="Actual Data")
plt.plot(x_points, y_guess, color="red", label="Regression Line")
plt.xlabel("X Points")
plt.ylabel("Y Points")
plt.legend()
plt.show()


# compute and print your mean squared error with your y_guess
error = np.mean(np.square(y_guess - y_points))
print(error)
```

```
########################
## YOUR CODE ENDS HERE
########################
```

```
1.9349777917522766 2.555030766526259
```




```
2.6549986410378392
```

Write your report here

Run the next two cells in Google Colab to export your notebook to a pdf for submission

In [ ]:

```
'''
    This is a linear regression project where I
    took data from a set of data and modelled a line
    of best fit by leveraging different methods.

    No difficulties were incurred.
'''
```