Simple Linear Regression - Lab

Introduction

Regression analysis forms the basis of machine learning experiments. Understanding regression will help you to get the foundations of most machine learing algorithms. Ever wondered what's at the heart of an artificial neural network processing unstructured data like music and graphics? It can be linear regression!

Objectives

You will be able to:

- Calculate the slope of a line using standard slope formula
- Calculate the y-intercept using the slope value
- Draw a regression line based on calculated slope and intercept
- Predict the label of a previously unseen data element

Let's get started

A first step towards understanding regression is getting a clear idea about "linear" regression and basic linear algebra.

The calculation for the best-fit line's slope, `m` is calculated as :

$$m = \frac{\overline{x} \cdot \overline{y} - \overline{xy}}{(\overline{x})^2 - \overline{x^2}}$$

As in our previous lesson, let's break down the formula into its parts. First we shall import the required libraries and define some data points to work with. We shall first create some toy data as numpy arrays. Let's do this for you to give you a head start.

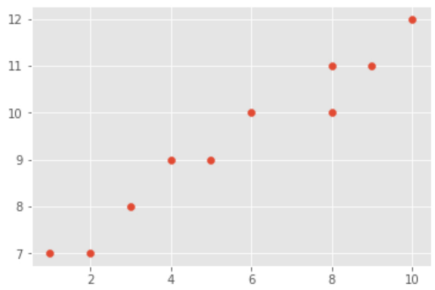
In [2]:
```
# import necessary libraries

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')

# Initialize vectors X and Y with given values and create a scatter plot
X = np.array([1,2,3,4,5,6,8,8,9,10], dtype=np.float64)
Y = np.array([7,7,8,9,9,10,10,11,11,12], dtype=np.float64)
```

Show a scatter plot between X and Y and comment on the output

In [3]:
```
# Scatter plot
plt.scatter(X,Y)
plt.show()
```



In [4]:
```
# Your observations about relationship in X and Y

# The relationship is very linear but not perfectly linear
# THe best fit line should be able to explain this relationship with very low error
```

In a data analysis context, we can think of these points as two vectors:

- **vector X**: the features of our model

- **vector Y**: the labels for given features

Write a function calc_slope()

Write a function `calc_slope()` that takes in x and y vectors and calculates the slope using the formula shown above.

In [5]:
```
# Write the function to calculate slope as:
# (mean(x) * mean(y) – mean(x*y)) / ( mean (x)^2 – mean( x^2))
def calc_slope(xs,ys):

    # Use the slope formula above and calculate the slope
    m = (((np.mean(xs)*np.mean(ys)) - np.mean(xs*ys)) /
         ((np.mean(xs)**2) - np.mean(xs*xs)))

    return m

calc_slope(X,Y)

# 0.5393518518518512
```

Out[5]:
```
0.5393518518518512
```

Great, so we have our slope. Next we calculate the intercept.

As a reminder, the calculation for the best-fit line's y-intercept is:

$$b = \bar{y} - m\bar{x}$$

Write a function best_fit()

Write a function `best_fit()` that takes in X and Y, calculates the slope using above above and intercept using the formula. The function should return slope and intercept values.

In [8]:
```
def best_fit(xs,ys):

    # use the slope function with intercept formula to return calculate slop and intercept from data points
    m = calc_slope(xs,ys)
    b = np.mean(ys) - m*np.mean(xs)

    return m, b

m, b = best_fit(X,Y)
m,b
# (0.5393518518518512, 6.379629629629633)
```

Out[8]:
```
(0.5393518518518512, 6.379629629629633)
```

We now have a working model with `m` and `b` as model parameters. We can create a line for the data points using the calculated slope and intercept:

- Recall that `y=mx+b`. We can now use slope and intercept values along with X data points (features) to calculate the Y data points (labels) of the regression line.

Write a function reg_line()

Write a function `reg_line()` that takes in slope, intercept and X vector and calculates the regression line using Y= mX+b for each point in X

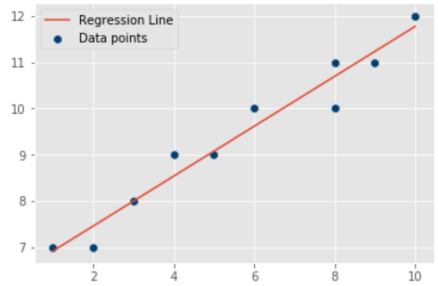
In [9]:
```
def reg_line (m, b, xs):

    return [(m*x)+b for x in xs]

regression_line = reg_line(m,b,X)
```

Plot the (x,y) data points and draw the calculated regression line for visual inspection

In [12]:
```
plt.scatter(X,Y,color='#003F72', label="Data points")
plt.plot(X, regression_line, label= "Regression Line")
plt.legend()
plt.show()
```



So there we have it, our least squares regression line. This is the best fit line and does describe the data pretty well (still not perfect though).

Describe your Model Mathematically and in words

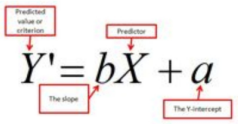
In [ ]:
```
# y = 6.37 + 0.53x

# The line crosses y-axis at 6.37 (shown in the graph) - intercept
# The slope of line is 0.53 - a slope 0 would a horizontal line , and slope = 1 would be a vertical one
# Our slope creates an angle roughly around 45 degree between x and y .
```

Predicting label for new data

So, how might you go about actually making a prediction based on this model you just made?

Now that we have a working model with m and b as model parameters. We can fill in a value of x with these parameters to identify a corresponding value of y according to our model. Recall the formula



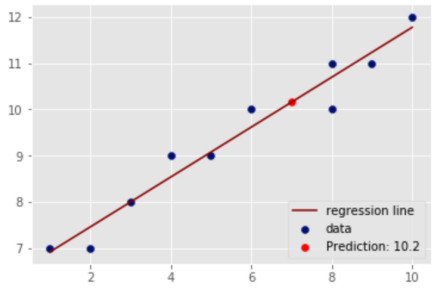
Let's try to find a y prediction for a new value of x = 7 and unknown y, and plot the new prediction with existing data

In [18]:
```
x_new = 7
y_predicted = (m*x_new)+b
y_predicted

# 10.155092592592592
```

Out[18]: 10.155092592592592

In [19]:
```
plt.scatter(X,Y,color='#000F72',label='data')
plt.plot(X, regression_line, color='#880000', label='regression line')
plt.scatter(x_new,y_predicted,color='r',label='Prediction: '+ str(np.round(y_predicted,1)))
plt.legend(loc=4)
plt.show()
```



We now know how to create our own models, which is great, but we're stilling missing something integral: how accurate is our model? This is the topic for discussion in the next lab.

Summary

In this lesson, we learnt how we can draw a best fit line for given data labels and features, by first calculating the slope and intercept. The calculated regression line was then used to predict the label (y-value) of a previously unseen feature (x-value). The lesson uses a simple set of data points for demonstration. Students should be able to plug in other datasets and practice with predictions for accuracy.