Mini Project 1
Linear Regression Algorithm

25 May 2020

by Alex Dance

Purpose and Agenda

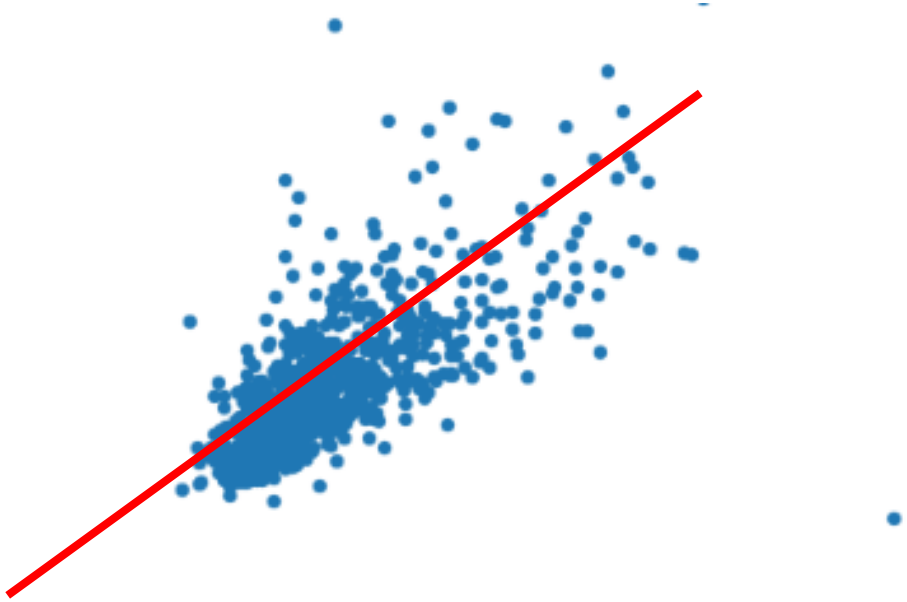
Purpose: Share the linear regression algorithm I built

Agenda

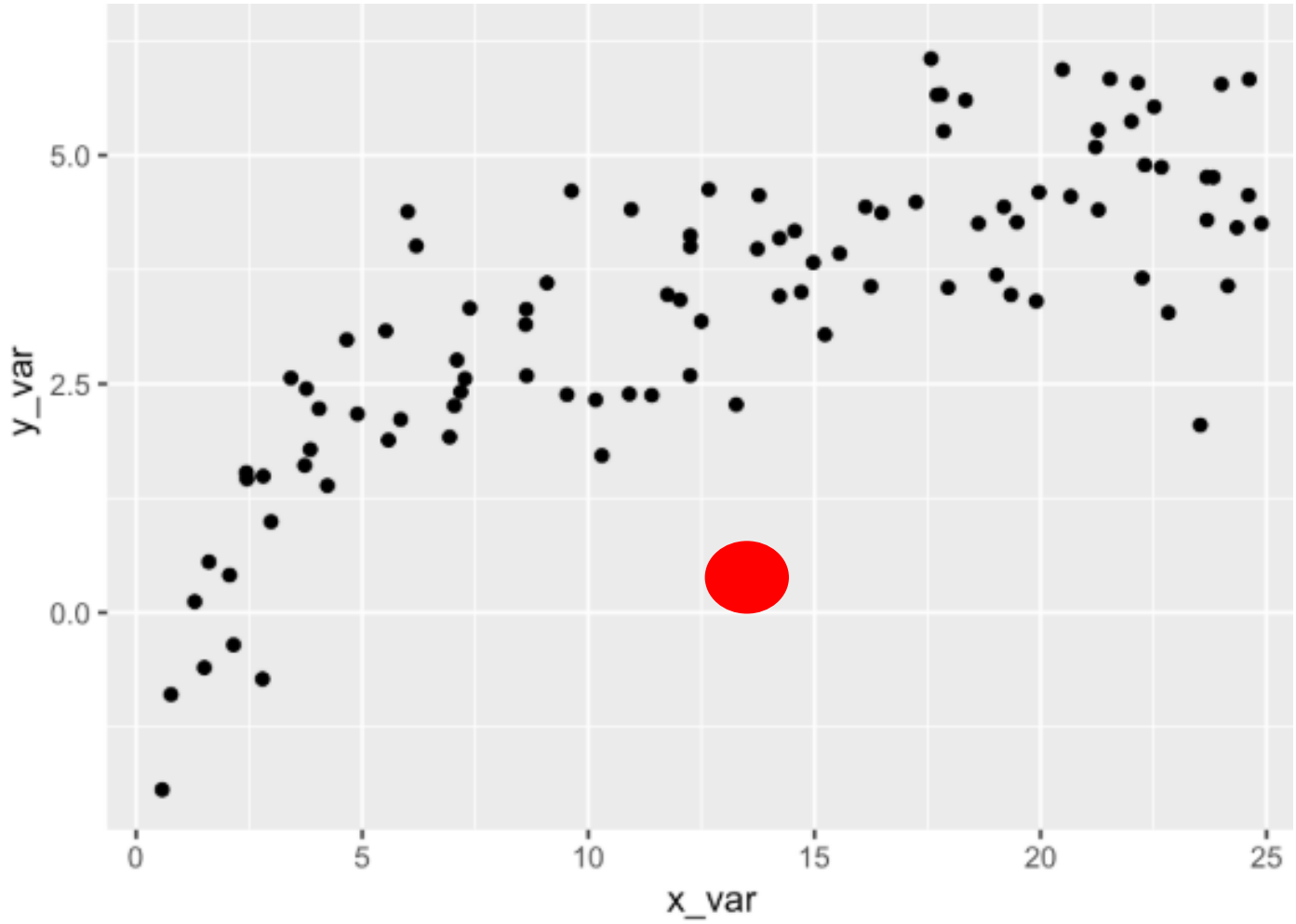
- Where idea came from
- My logic for model
- Explain code
- Show code
- Show examples

Where idea came from

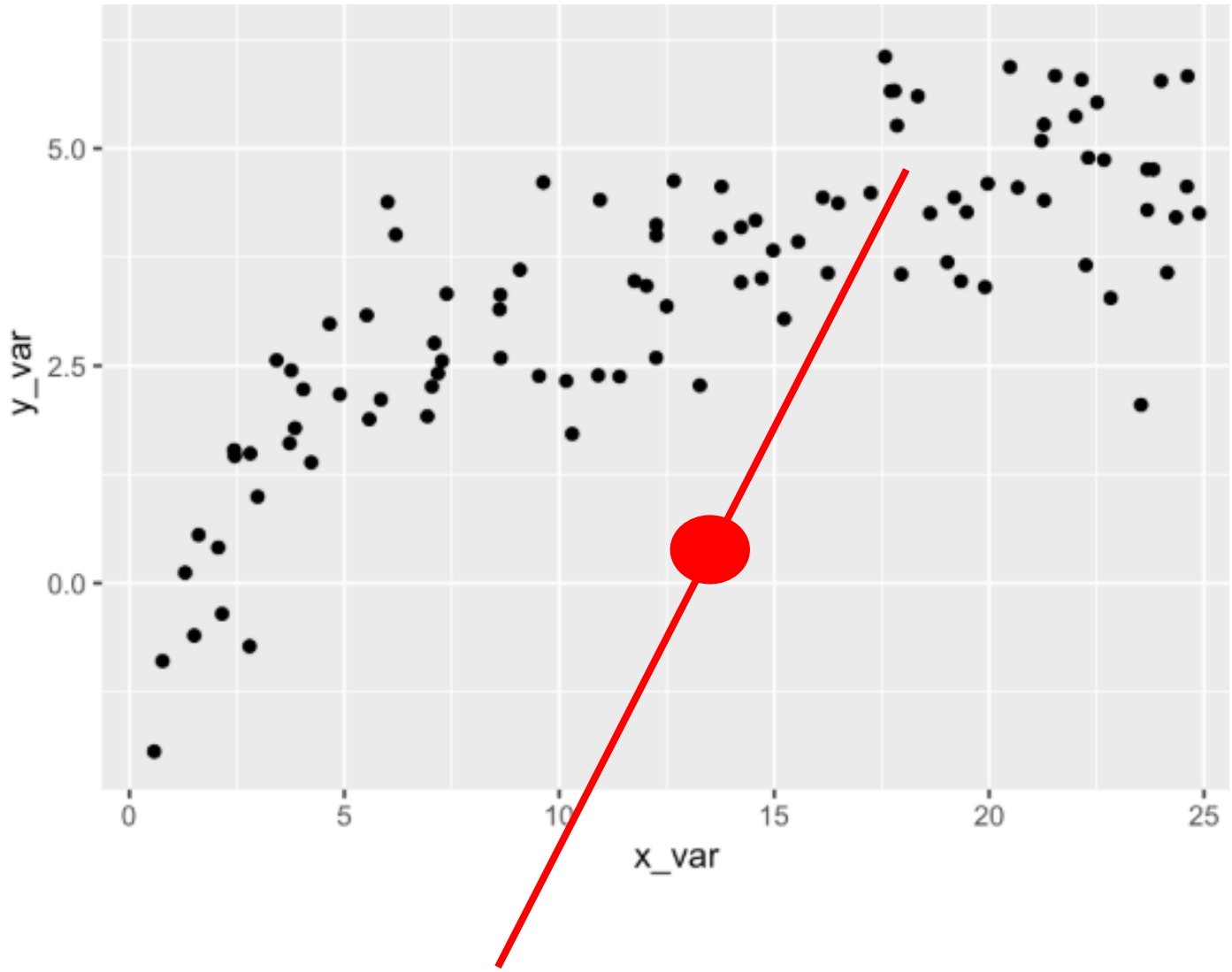
- Calculating Line of Best Fit is easy in Python
- Running all models is relatively easy
- Want to match workings to the actual back end of an algorithm



Step: Choose LOW y value (at x=mean)

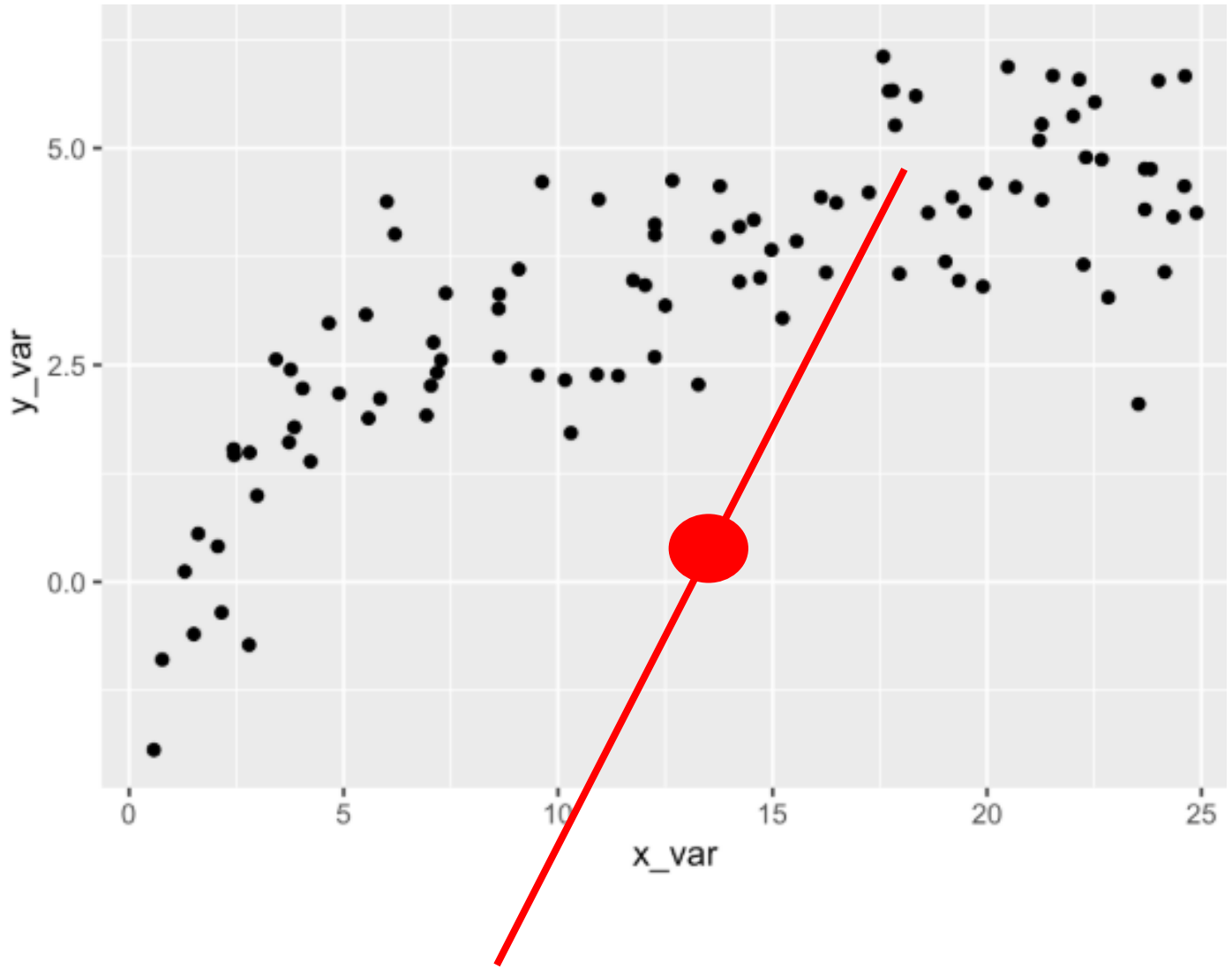


Step: Choose random slope (too steep to start)

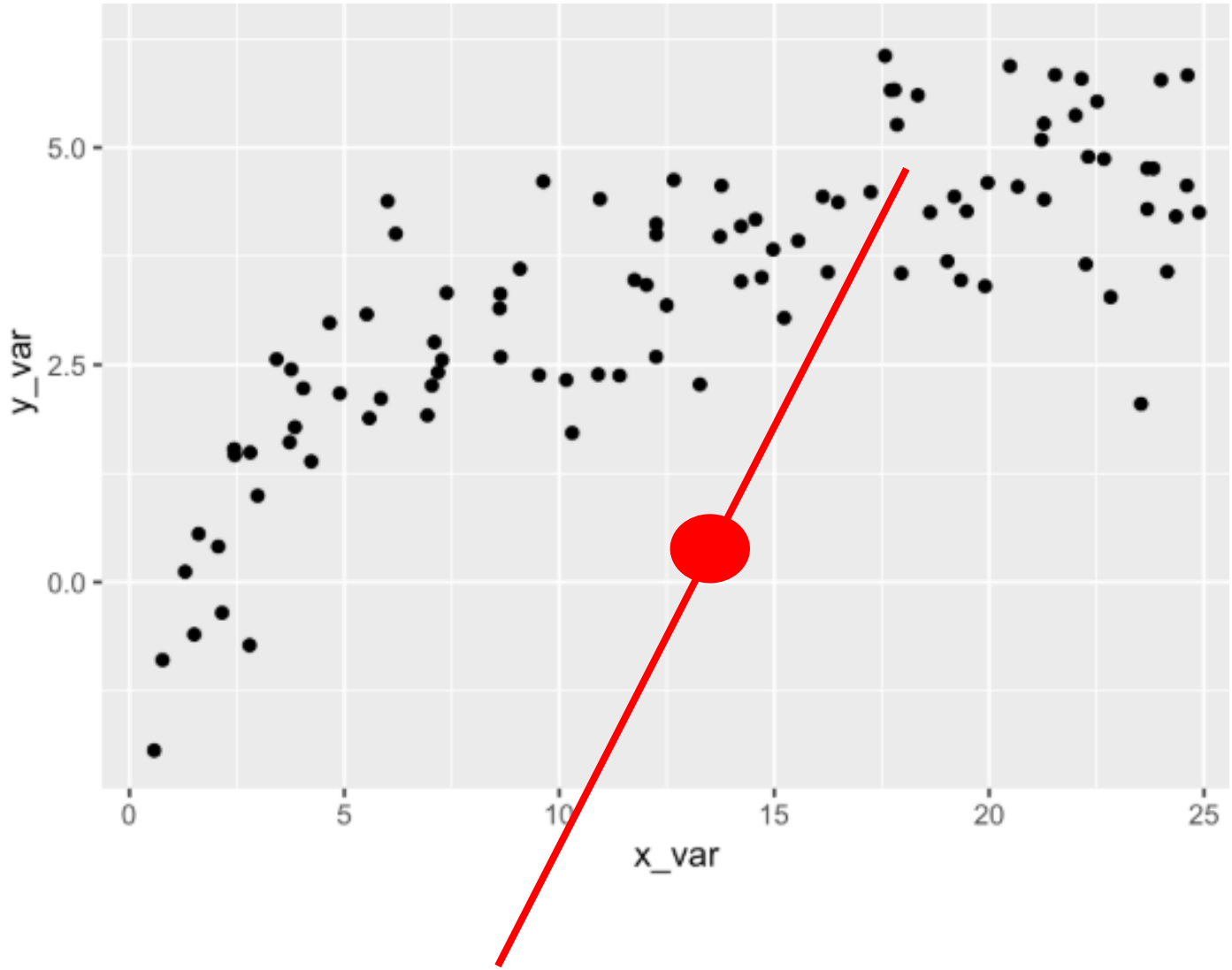


Step: Find where it hits the y AXIS(x = 0)

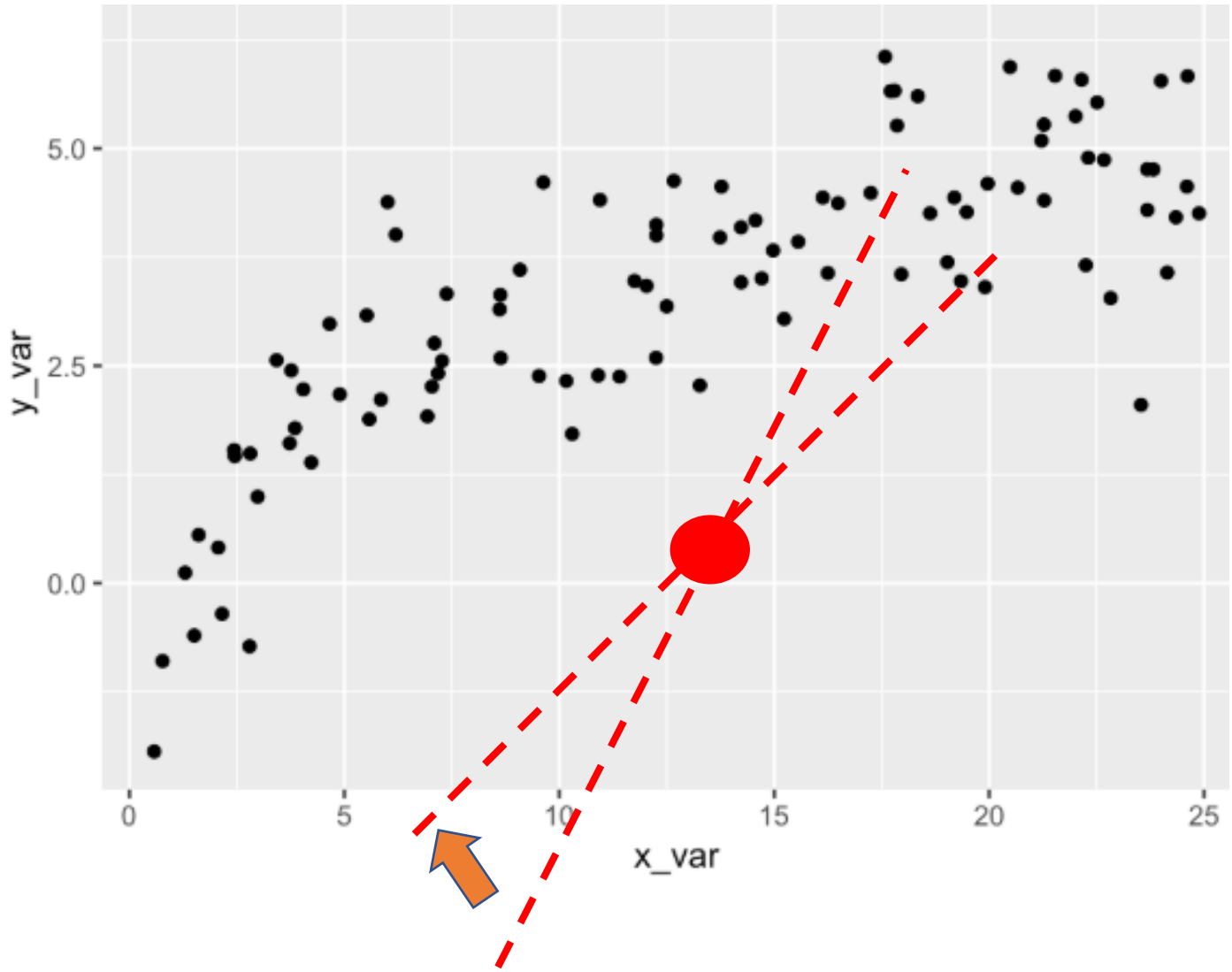
(to get b in y=mx +b)



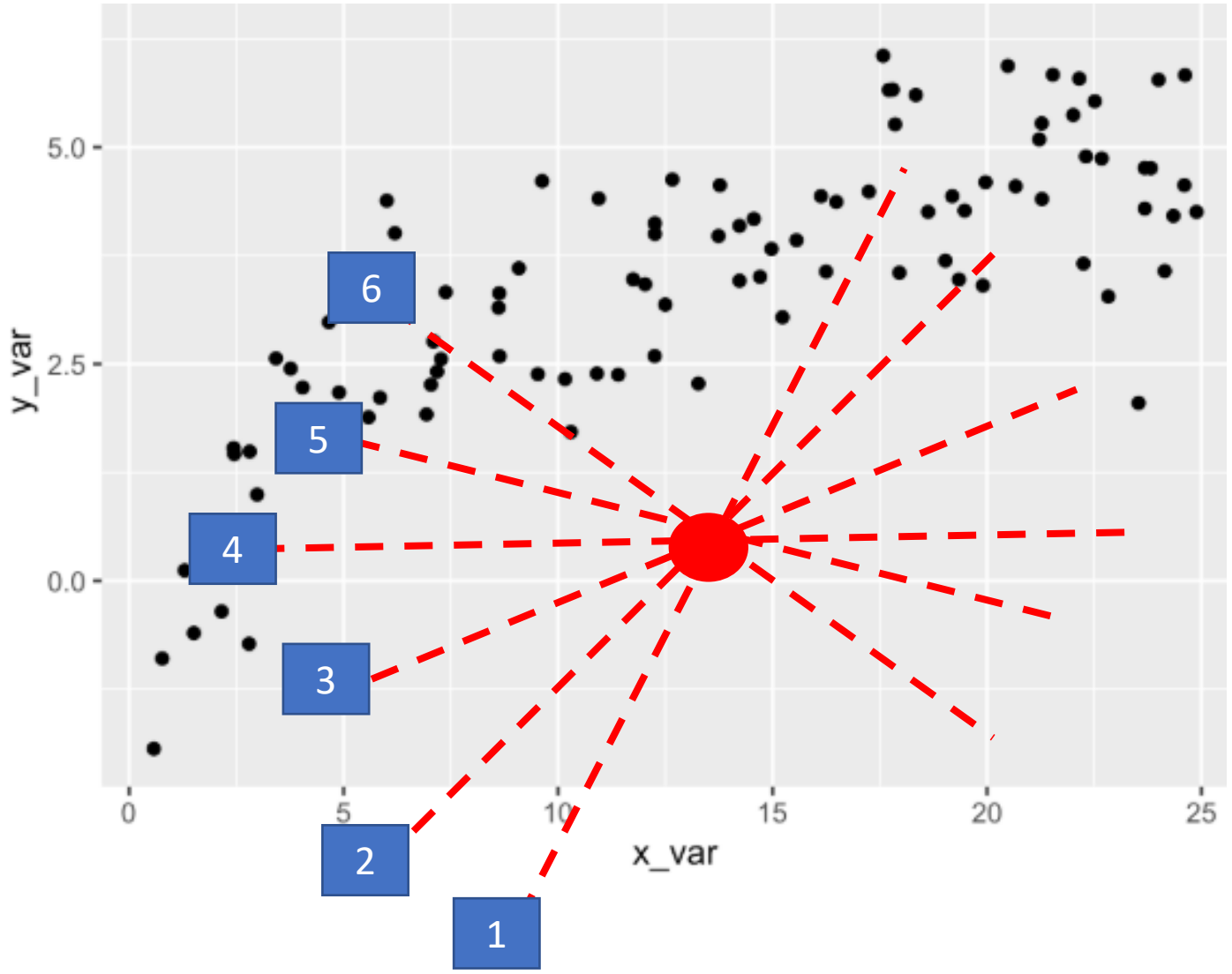
Step: Calculate R squared (as now have m and b in y=mx+b)



Step: Repeat with a slightly different slope



Step: Repeat until R is NOT better for this Y value

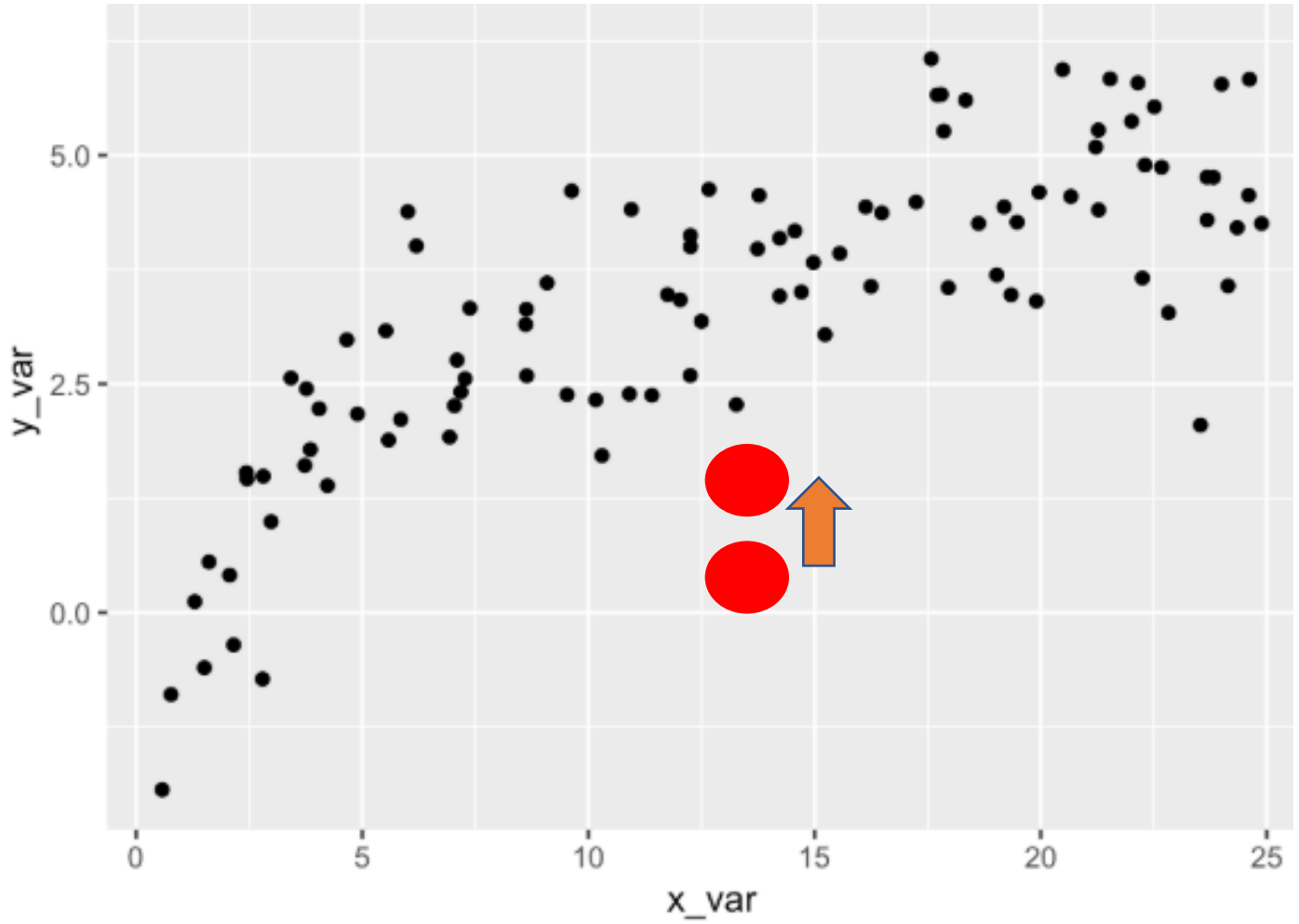


R at 2 is better than R at 1
R at 3 is better than R at 2

keep going until R is not better

This saves a lot of processing steps

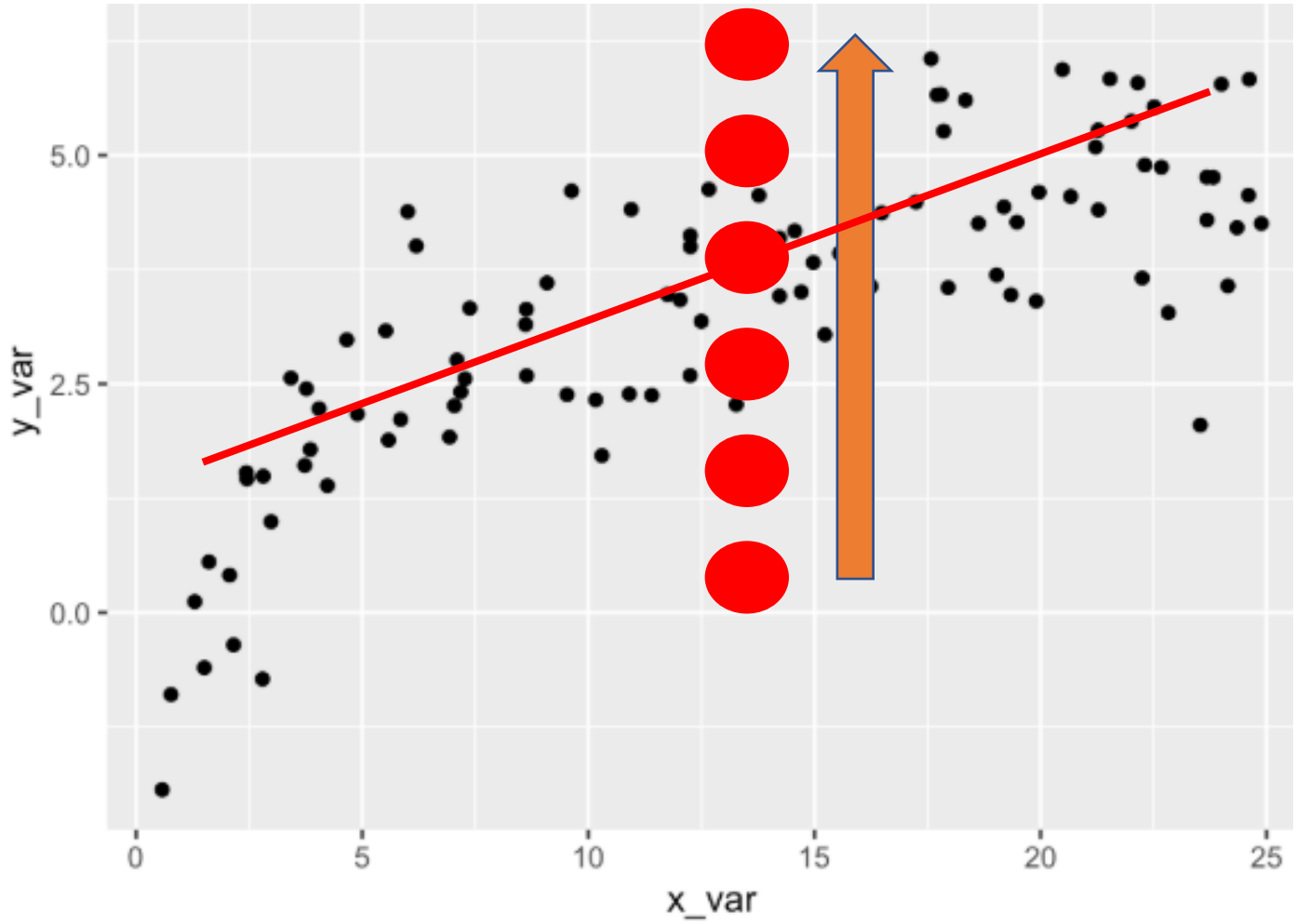
Step: Choose higher Y value



Repeat Steps

- Find where meets Axis
- Start with initial slope again
- Calculate R

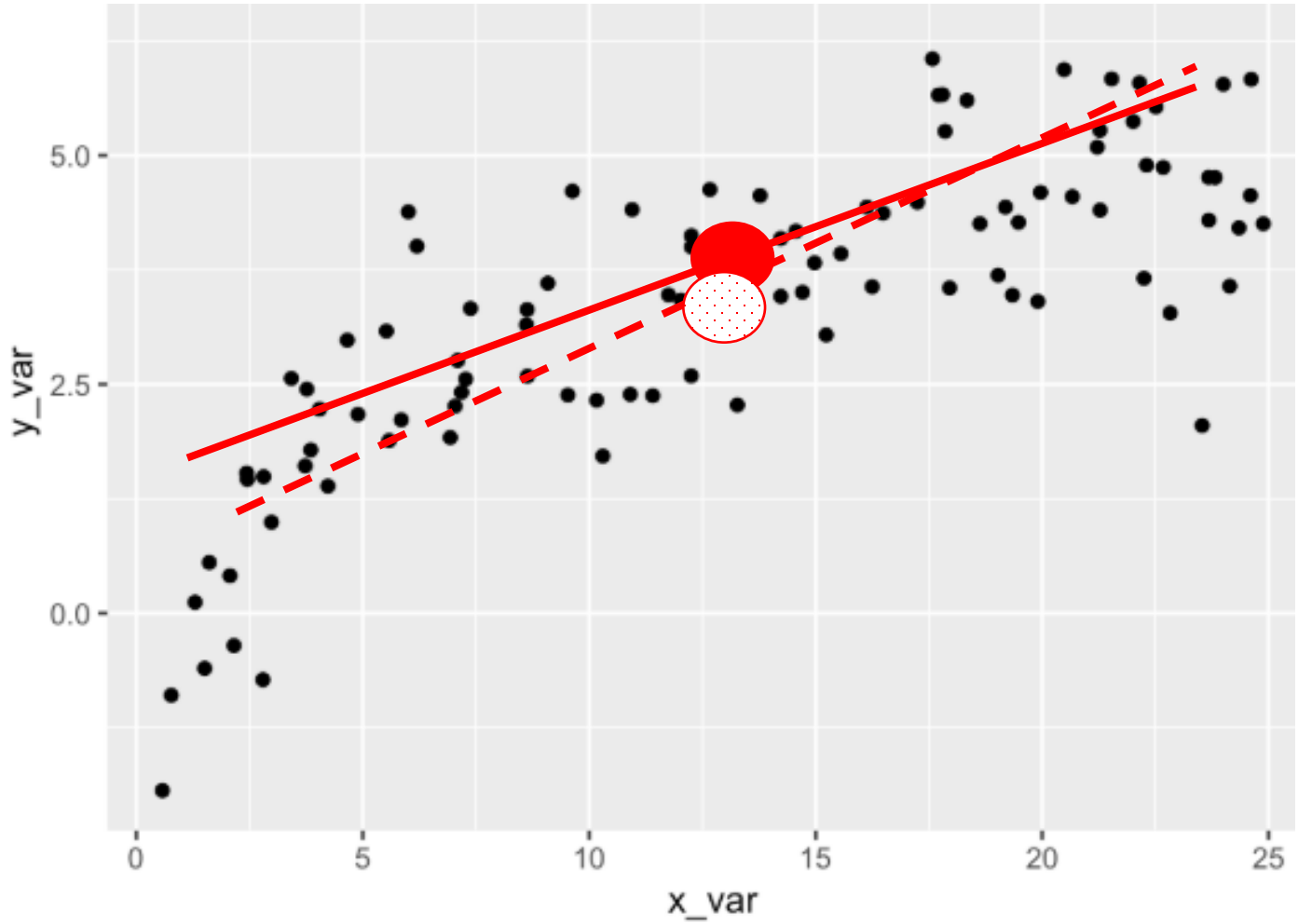
Keep moving up until end up with best R squared



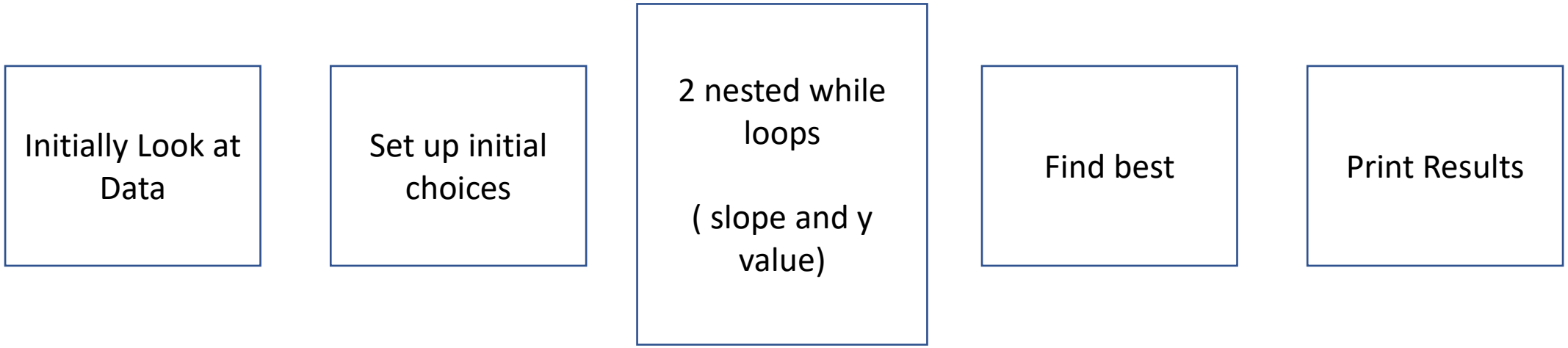
At this point we have the

- Best slope
- Best meeting of Y Axis

Can add more or less iterations to get better or worse results



How it works (theory)



The Set up

```
bikes = pd.read_csv(r'C:/Users/alexd/Alex Folder 1/Module 3/bikeshare.csv')
bikes.head()
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

1

Have a quick look at the values

2

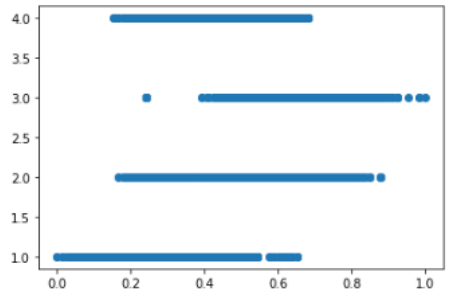
Choose X and Y

```
In [89]:
#xmean = bikes['temp'].mean()
#ymean = bikes['atemp'].mean()

xmean = X.mean()
ymean = Y.mean()
print("INIIIAL RESULTS")
print("Mean of X : ", xmean)
print("Mean of Y : ", ymean)
print(" ")
plt.scatter(X, Y) # extra stuff from Alex
```

```
INIIIAL RESULTS
Mean of X :  0.4757751021347581
Mean of Y :  2.5016399102364923
```

```
Out[89]: <matplotlib.collections.PathCollection at 0x22049de2308>
```



3

CHOOSE INITIAL STARTING POINTS

```
:
minitial = -0.5 #choosing aslope that is too low a slope (eg if think slope should be 1 then choose 0.5)
maddfactor = 0.05
mtoofar=100 # Choosing aslope that is too HIGH a slope (eg if think slope should be 1 then choose 2)

ystart = 0.1
yaddfactor = 0.5
ytoofar = 20 # matching x mean
```

To avoid bad results on extreme data (eg price Vs bedrooms slope of 64K)

Had 3 functions

```
def calculateRsquaredB (df, Xc,Yc,Bzc, mc):
    ymean = np.mean(Yc)
    xmean = np.mean(Xc)
    yhat = Bzc + (mc * Xc)
    SSresidual= pow((Yc - yhat),2)
    SSresidualTotal=sum(SSresidual)
    SStot = sum(pow((Yc - ymean), 2))
    Rsquared = 1 - (SSresidualTotal /SStot )
    return(Rsquared)
```

```
def CalculateBNil (mc, yc, xmeanc):
    Beet = yc - (xmeanc * mc)
    return(Beet)
```

```
def StopBecauseR (OneAgo, ThisGo):
    if (OneAgo < ThisGo):
        Stop_as_R = False
    else:
         Stop_as_R = True
    return(Stop_as_R)
    #return(False)  # used to check impact of this function
```

Could have done Rsquared with a function but did it manually instead

Main Code

```
### Run from here
BestR = -10000000
Bestm = 0
BestB = 0
Besti = 0
Bestj = 0
Previous_RMAX = -10000000

yuse = ystart
RList =[]
MLidt =[]
NewList =[]
ind =[]
totrunthrough = 0
j = 0
NotChangeY = True
Old_R = Previous_RMAX
```

Initialising

```
while NotChangeY: #j <20: #ChangeY == False: # moving the y start up and down
    j += 1
    Old_R = -1000 ######
    if yuse > ytoofar:
        NotChangeY = False
    yuse = yuse + yaddfactor
    mchange = minitial
    Bee = CalculateBNil (mchange,yuse, xmean)
    i=1
    NotChangeM = True
    StopDuetoR = False
    while  NotChangeM: #i<20: #ChangeM == False: # moving the m around
        if mchange > mtoofar:
                NotChangeM = False
        i += 1
        totrunthrough += 1
        Tresult = calculateRsquaredB (datatouse,X, Y,Bee, mchange)
        Result1 = float(Tresult)
        mchange = mchange + maddfactor

        # beginning of R calculaitons
        StopDuetoR = StopBecauseR (Old_R, Result1) # beginning of R calculaitons
        Old_R = Result1
        if StopDuetoR:
            NotChangeM = False
            Old_R = Previous_RMAX
        # end of R Calculaitons
        print(" ")
        print("Result1 ",Result1, "yuse ",yuse,  "mchange ",mchange,"i(m) ",i, "j(y) ",j, "BestR ",BestR )
        if Result1 > BestR:
            print("------Bestttttttttt")
            BestR = Result1
            Bestm = mchange
            BestB = Bee
            Besty= yuse
            Besti = i
            Bestj = j

print("The END")
```

Going through Y change and initialise

For this Y and M check B then check R

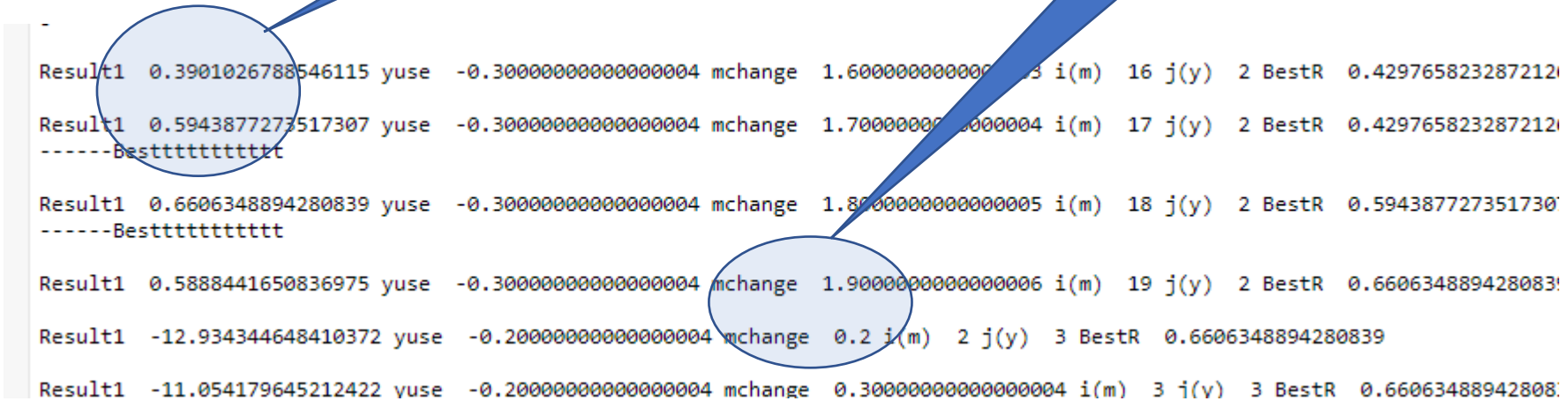
Check if this R is better than the last R for this Y and M

If Best keep the results

Looking at the log results

As scroll through R changes

When go to a new Y THEN go to initial m and R is bad again

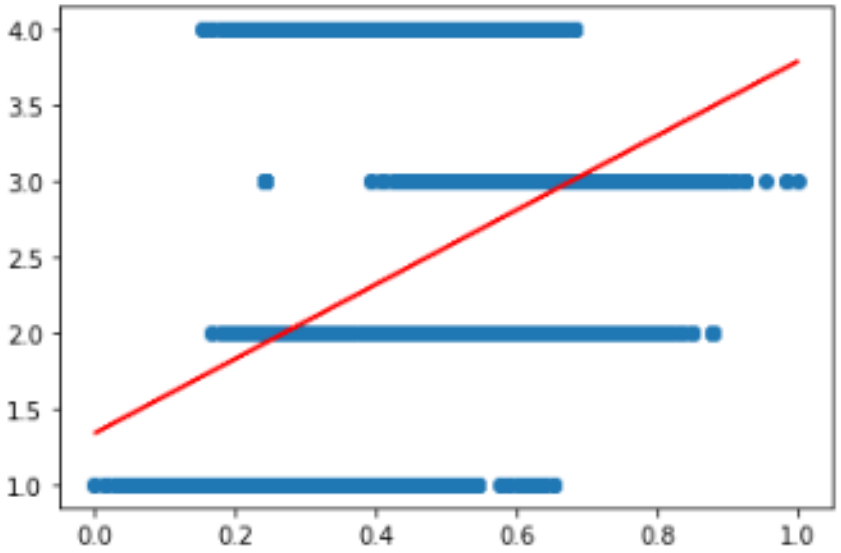


Look at the final results

```
1  print("ALEX REGRESSION - RESULTS")
2  print(" ")
3  print("Iterations ran through ", totrunthrough)
4  print("BestR ",BestR)
5  print("Bestm ",Bestm)
6  print("Cross Y axis ",BestB)
7  print("  ")
8
9  printline=[]
10 for a in range(len(bikes)):
11     printline = (Bestm * X) + BestB
12
13 plt.plot(X,printline, color='r')
14 plt.scatter(X, Y)
15 plt.show()
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  279
BestR  0.09877963060646155
Bestm  2.4499999999999993
Cross Y axis  1.337887551067379
```



Choose different data

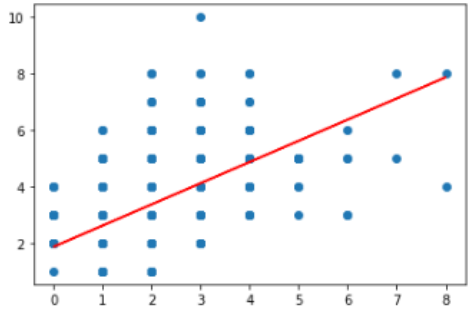
CHOOSE X AND Y

```
#X = bikes['atemp']
#Y = bikes['season']
#datatouse = bikes


X = melb['Bathroom']
Y = melb['Rooms']
datatouse = melb
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  180
BestR  0.3441097231074395
Bestm  0.7500000000000001
Cross Y axis  1.867120765832106
```



```
X = bikes['atemp']
Y = bikes['temp']
datatouse = bikes


#X = melb['Bathroom']
#Y = melb['Rooms']
#datatouse = melb
```

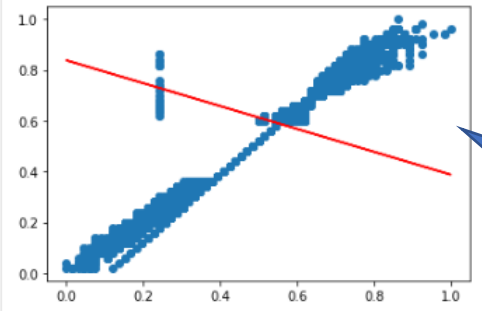
```
minitial = -0.5 #choosing aslope th
maddfactor = 0.05
mtoofar=100 # Choosing aslo that

ystart = 0.1
yaddfactor = 0.5
ytoofar = 20 # matching x mean
```

Seeded Wrong

```
ALEX REGRESSION - RESULTS

Iterations ran through  53
BestR  -1.366806243223306
Bestm  -0.45
Cross Y axis  0.837887551067379
```



Got Wrong Data

```
X = bikes['atemp']
Y = bikes['temp']
datatouse = bikes


#X = melb['Bathroom']
#Y = melb['Rooms']
#datatouse = melb
```

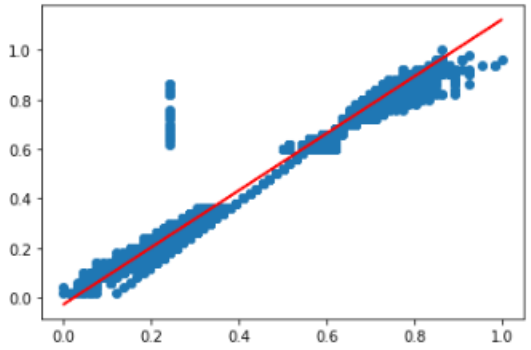
Better Seeding

```
minitial = 0.1 #choosing aslop
maddfactor = 0.05
mtoofar=100 # Choosing aslope th

ystart = -0.5
yaddfactor = 0.01
ytoofar = 20 # matching x mean
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  4826
BestR  0.9754211425815303
Bestm  1.1500000000000004
Cross Y axis  -0.0275775102134755
```



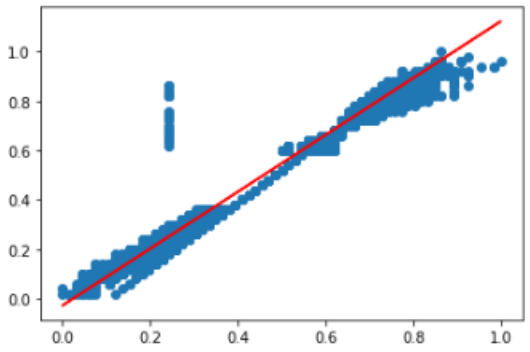
Less increments gave a worse result

```
minitial = 0.1 #choosing aslope
maddfactor = 0.05
mtoofar=100 # Choosing aslope th

ystart = -0.5
yaddfactor = 0.01
ytoofar = 20 # matching x mean
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  4826
BestR  0.9754211425815303
Bestm  1.1500000000000004
Cross Y axis  -0.0275775102134755
```

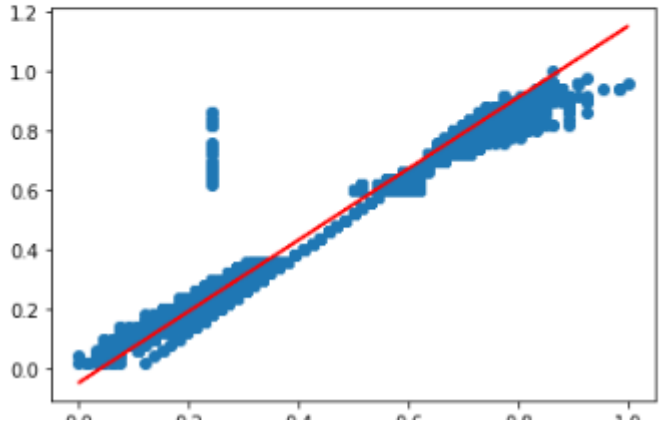



```
minitial = 0.1 #choosing aslope that
maddfactor = 0.1
mtoofar=100 # Choosing aslope that i

ystart = -0.5
yaddfactor = 0.1
ytoofar = 20 # matching x mean
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  371
BestR  0.9633247470231415
Bestm  1.2
Cross Y axis  -0.04757751021347584
```

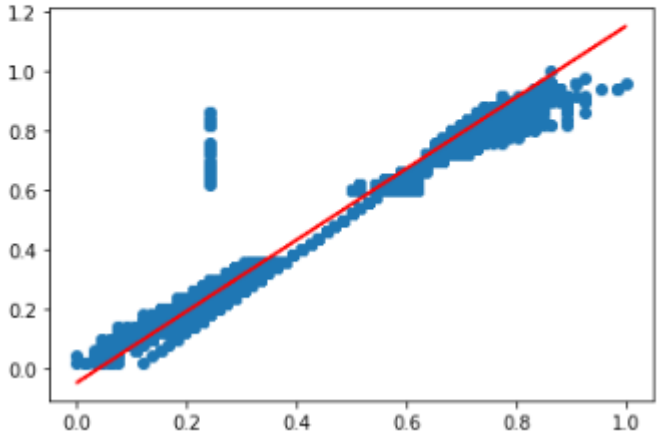


Not checking for a change in R improved speed and gave the same R (from 754 to 147 iterations)

```
def StopBecauseR (OneAgo, ThisGo):
    if (OneAgo < ThisGo):
        Stop_as_R = False
    else:
         Stop_as_R = True
    #return(Stop_as_R)
    return(False)  # used to check impact of this functio
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  147
BestR  0.9633247470231415
Bestm  1.2
Cross Y axis  -0.04757751021347584
```

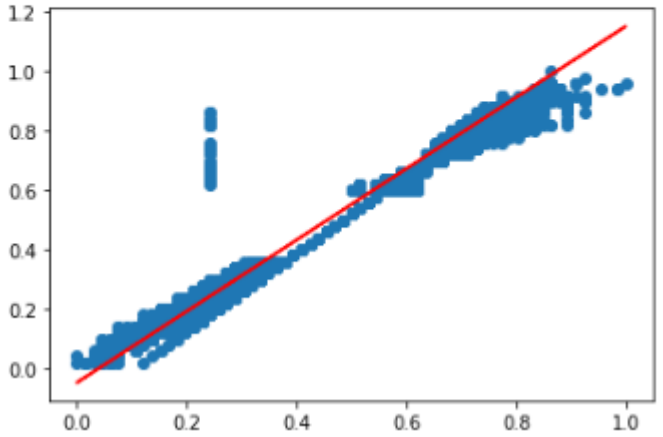



```
minitial = 0.1 #choosing aslope t
maddfactor = 0.1
mtoofar=3 # Choosing aslope that

ystart = -0.5
yaddfactor = 0.1
ytoofar = 2 # matching x mean
```

```
ALEX REGRESSION - RESULTS

Iterations ran through  754
BestR  0.9633247470231415
Bestm  1.2
Cross Y axis  -0.04757751021347584
```



Making it better OPTIONS

Not moved in simple increments

BUT

Looked at slope of R change

AND

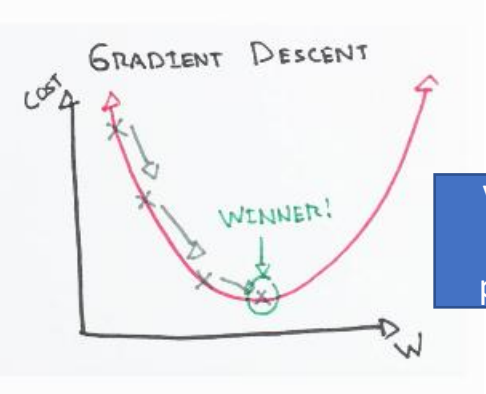
Moved through Faster WHEN A LONG WAY AWAY

Moved through Slower WHEN CLOSE

OR

Bounced back and forth when close to best

Added and Compared Gradient Descent code to my model as an extra option (UWR is Updated Weights Result from gradient



UWR seeding	m		B		# of iterations	
	Original	UWR	Original	UWR	Original	UWR
M= 0.5 * Alex best B = 0.5 * Alex Best Learning Rate = 1	1.2	0.854	-0.047	0.447	147	17,379
M= Alex best B = Alex Best Learning Rate = 1		1.169		-0.100		
M + B no change Learning Rate = 2		1.139		-0.153		
M + B no change Learning Rate = 0.001		1.199969		-0.0476		

Very BAD for UVR – particularly B

Almost identical

Very Close

```
def update_weights(m, b, X, Y, learning_rate):
    m_deriv = 0
    b_deriv = 0
    N = len(X)
    for i in range(N):
        # Calculate partial derivatives
        # -2x(y - (mx + b))
        m_deriv += -2*X[i] * (Y[i] - (m*X[i] + b))

        # -2(y - (mx + b))
        b_deriv += -2*(Y[i] - (m*X[i] + b))

    # We subtract because the derivatives point in direction of steepest ascent
    m -= (m_deriv / float(N)) * learning_rate
    b -= (b_deriv / float(N)) * learning_rate

    return m, b
```

Comment

- Alex's is better above a certain learning rate
- Changing the learning rate does not change the number of iterations
- Seeding the model with slightly bad data gives bad results

Source: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Thanks