Olatomiwa Akinlaja ACML Assignment 1 (Programming Exercise)

March 25, 2021

1 INTRODUCTION

The aim of this assignment is to implement a simple linear regression (using gradient descent) model on a dataset

> Two columns are extracted from the bank client data namely: 1. age (numeric) - X 1. balance: client's total balance? (numeric) - Y (Target)

The linear regression model should be able to predict the client's balance based on their age

2 IMPORTING THE DATASET

```
In [1]: import pandas as pd
        import numpy as np
        import warnings
        import matplotlib.pyplot as plt
        plt.style.use('ggplot')
        warnings.filterwarnings('ignore')
```

```
In [2]: bank = pd.read_csv('bank-full.csv')
        bank.head()
```

```
Out[2]:    age           job  marital  education default  balance housing loan  \
        0   58    management  married   tertiary      no     2143     yes   no
        1   44    technician   single  secondary      no       29     yes   no
        2   33  entrepreneur  married  secondary      no        2     yes  yes
        3   47   blue-collar  married    unknown      no     1506     yes   no
        4   33       unknown   single    unknown      no        1      no   no

           contact  day month  duration  campaign  pdays  previous poutcome   y
        0  unknown    5   may       261         1     -1         0  unknown  no
        1  unknown    5   may       151         1     -1         0  unknown  no
        2  unknown    5   may        76         1     -1         0  unknown  no
        3  unknown    5   may        92         1     -1         0  unknown  no
        4  unknown    5   may       198         1     -1         0  unknown  no
```

```
In [3]: len(bank)
```

```
Out[3]: 45211
```

```
In [4]: # the bank dataset consists of over 45000 rows so lets focus on random 100 customers
        bank = bank.iloc[3100:3200, :]
```

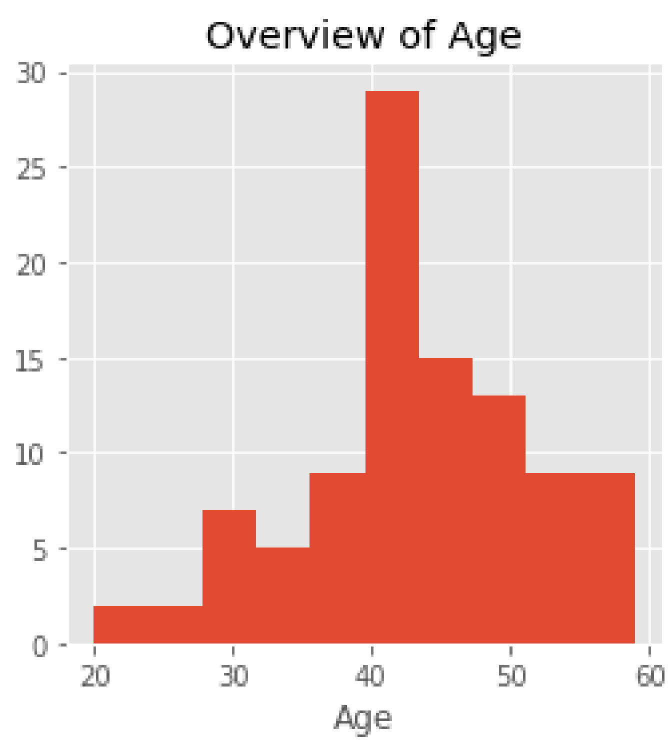
```
In [5]: # Check/Remove null values
        bank.isnull().values.any()
```

```
Out[5]: False
```

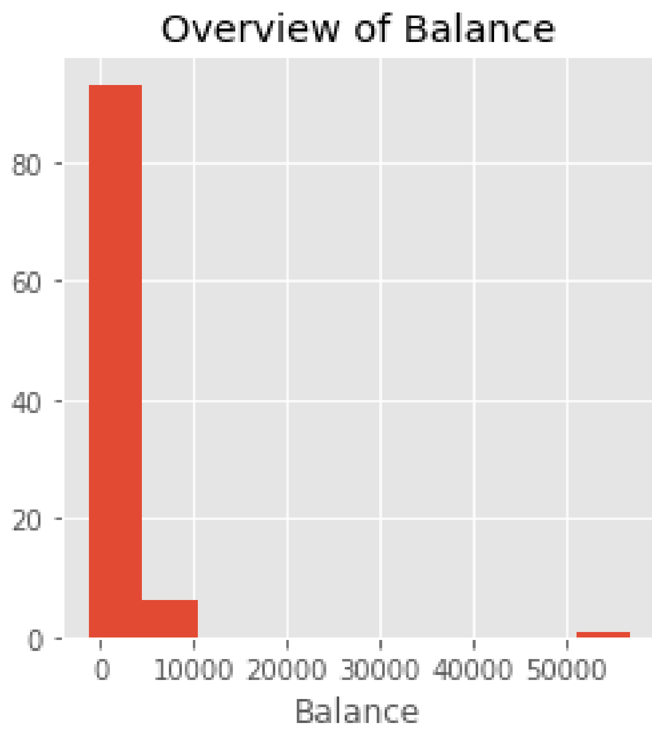
3 Exploring the data & implementing the Linear Regression model with the data as is

```
In [6]: # Predict Balance based on Age
        X = bank['age'] # Feature
        Y = bank['balance'] # Target
```

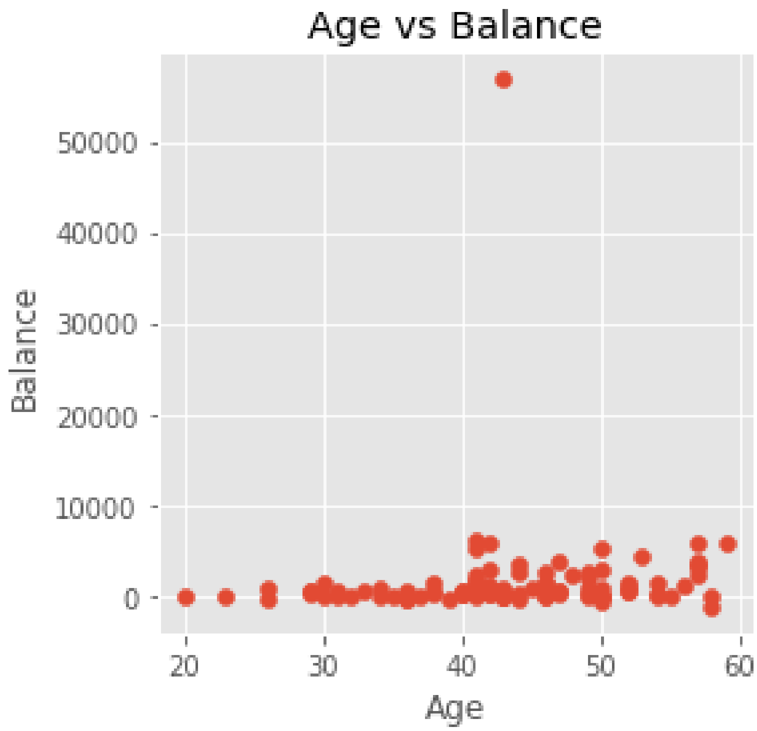
```
In [7]: plt.figure(figsize = (4,4))
        plt.hist(X)
        plt.xlabel('Age')
        plt.title("Overview of Age")
        plt.show()
```



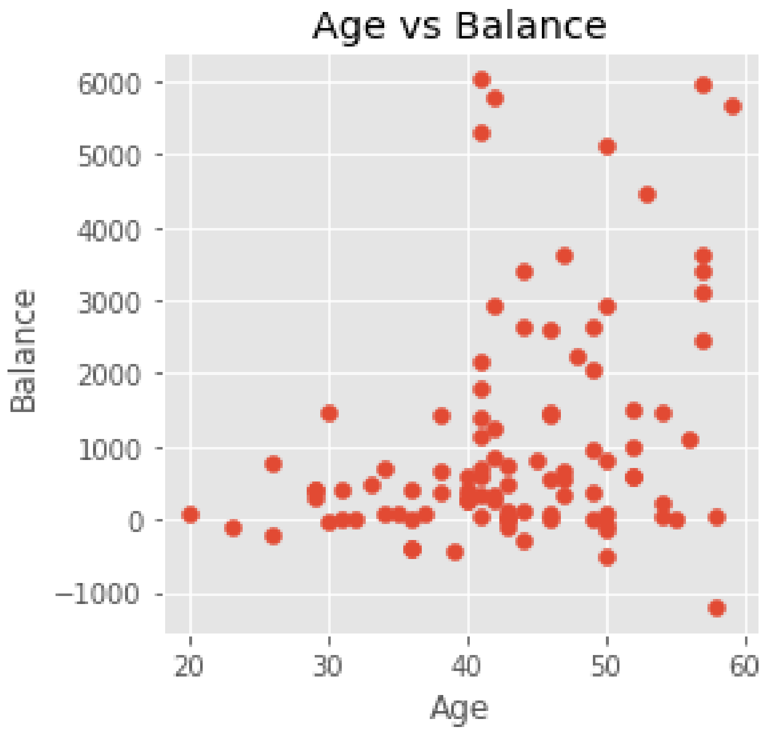
In [8]:
```
plt.figure(figsize = (4,4))
plt.hist(Y)
plt.xlabel('Balance')
plt.title("Overview of Balance")
plt.show()
```



In [9]:
```
plt.figure(figsize = (4,4))
plt.scatter(X,Y)
plt.xlabel('Age')
plt.ylabel('Balance')
plt.title("Age vs Balance")
plt.show()
```




```
In [10]: # removing the outlier
         bank = bank[bank['balance'] != bank['balance'].max()]

In [11]: X = bank['age'] # Feature
         Y = bank['balance'] # Target

In [12]: plt.figure(figsize = (4,4))
         plt.scatter(X,Y)
         plt.xlabel('Age')
         plt.ylabel('Balance')
         plt.title("Age vs Balance")
         plt.show()
```



4 Building the model

```
In [13]: m = 0 # gradient
         c = 0 # Intercept

         alpha = 0.005  # The learning Rate
         epochs = 100  # The number of iterations to perform gradient descent

         n = float(len(X)) # Number of elements in X

         # Performing Gradient Descent
         for i in range(epochs):
             Y_pred = m*X + c  # The current predicted value of Y
             D_m = (-2/n) * sum(X * (Y - Y_pred))  # Partial derivative wrt m
             D_c = (-2/n) * sum(Y - Y_pred)  # Partial derivative wrt c
             m = m - alpha * D_m  # Update m
             c = c - alpha * D_c  # Update c

             print("Iteration: {}".format(i+1), "\nGradient: {}".format(m), "\nIntercept: {}".:

         print("\nProcess Complete... after {} epochs".format(epochs))
         print("Gradient(m): {}".format(m), "Intercept(c): {}".format(c))
```

```
Iteration: 1
Gradient: 521.789898989899
Intercept: 11.029797979797982

Iteration: 2
Gradient: -9134.364625905524
Intercept: -204.2123642995613

Iteration: 3
Gradient: 169565.6002481972
Intercept: 3768.006891642485

Iteration: 4
Gradient: -3137509.779618302
Intercept: -69754.19799192189

Iteration: 5
Gradient: 58064228.211913906
Intercept: 1290858.8651336674

Iteration: 6
Gradient: -1074553655.267811
Intercept: -23889069.325772442

Iteration: 7
Gradient: 19886015641.805317
Intercept: 442098290.4532609

Iteration: 8
Gradient: -368016622715.0586
Intercept: -8181604814.652449

Iteration: 9
Gradient: 6810626987185.738
Intercept: 151411254483.9497

Iteration: 10
Gradient: -126039524010132.92
Intercept: -2802062494616.931

Iteration: 11
Gradient: 2332525572558898.0
Intercept: 51855816466449.39

Iteration: 12
Gradient: -4.316642409886063e+16
Intercept: -959659431784086.4
```

```
Iteration: 13
Gradient: 7.988509071043288e+17
Intercept: 1.7759747850231842e+16

Iteration: 14
Gradient: -1.4783776629722096e+19
Intercept: -3.286672680509649e+17

Iteration: 15
Gradient: 2.735930440759623e+20
Intercept: 6.082415921613136e+18

Iteration: 16
Gradient: -5.063195666543197e+21
Intercept: -1.1256302966487112e+20

Iteration: 17
Gradient: 9.370103119501844e+22
Intercept: 2.083125490039541e+21

Iteration: 18
Gradient: -1.7340596384662574e+24
Intercept: -3.85509506999946e+22

Iteration: 19
Gradient: 3.20910324188363e+25
Intercept: 7.134355596816227e+23

Iteration: 20
Gradient: -5.93886357113802e+26
Intercept: -1.3203054362503735e+25

Iteration: 21
Gradient: 1.0990640642613899e+28
Intercept: 2.443397194513561e+26

Iteration: 22
Gradient: -2.0339612164542376e+29
Intercept: -4.5218247885973213e+27

Iteration: 23
Gradient: 3.764110177526558e+30
Intercept: 8.368225790176473e+28

Iteration: 24
Gradient: -6.965976201482702e+31
Intercept: -1.548649188088007e+30
```

```
Iteration: 25
Gradient: 1.2891446358116332e+33
Intercept: 2.8659770516482044e+31

Iteration: 26
Gradient: -2.3857300742545976e+34
Intercept: -5.303863860036041e+32

Iteration: 27
Gradient: 4.415104270762761e+35
Intercept: 9.815490961317521e+33

Iteration: 28
Gradient: -8.170725570367821e+36
Intercept: -1.8164844602751782e+35

Iteration: 29
Gradient: 1.5120991997484328e+38
Intercept: 3.3616411114073376e+36

Iteration: 30
Gradient: -2.7983365371760024e+39
Intercept: -6.221154768476254e+37

Iteration: 31
Gradient: 5.178686277062358e+40
Intercept: 1.1513057274913005e+39

Iteration: 32
Gradient: -9.583833538227222e+41
Intercept: -2.1306412193293293e+40

Iteration: 33
Gradient: 1.7736132365320905e+43
Intercept: 3.943029116511953e+41

Iteration: 34
Gradient: -3.28230232740845e+44
Intercept: -7.297089004292793e+42

Iteration: 35
Gradient: 6.074328013911389e+45
Intercept: 1.3504213730908003e+44

Iteration: 36
Gradient: -1.1241335239743485e+47
Intercept: -2.499130658578532e+45
```

```
Iteration: 37
Gradient: 2.0803555172340454e+48
Intercept: 4.624966823764291e+46

Iteration: 38
Gradient: -3.8499688745025707e+49
Intercept: -8.559103561670856e+47

Iteration: 39
Gradient: 7.124868904304227e+50
Intercept: 1.5839736061021393e+49

Iteration: 40
Gradient: -1.3185498002261162e+52
Intercept: -2.9313494885887606e+50

Iteration: 41
Gradient: 2.440148161359202e+53
Intercept: 5.4248440700946263e+51

Iteration: 42
Gradient: -4.5158120295218264e+54
Intercept: -1.0039380599073094e+53

Iteration: 43
Gradient: 8.357098396277322e+55
Intercept: 1.8579181541578795e+54

Iteration: 44
Gradient: -1.5465899189000638e+57
Intercept: -3.438319559145036e+55

Iteration: 45
Gradient: 2.8621661057727876e+58
Intercept: 6.36305822425088e+56

Iteration: 46
Gradient: -5.2968112082747315e+59
Intercept: -1.1775668104355765e+58

Iteration: 47
Gradient: 9.802439110545472e+60
Intercept: 2.179240774121109e+59

Iteration: 48
Gradient: -1.8140690452746768e+62
Intercept: -4.0329689232964176e+60
```

```
Iteration: 49
Gradient: 3.357171071313755e+63
Intercept: 7.463534332425615e+61

Iteration: 50
Gradient: -6.212882377009751e+64
Intercept: -1.3812242492006354e+63

Iteration: 51
Gradient: 1.1497748136931585e+66
Intercept: 2.556135393243161e+64

Iteration: 52
Gradient: -2.1278080639270125e+67
Intercept: -4.7304615107733125e+65

Iteration: 53
Gradient: 3.9377859933893956e+68
Intercept: 8.754335221858506e+66

Iteration: 54
Gradient: -7.287385921978343e+69
Intercept: -1.620103767933288e+68

Iteration: 55
Gradient: 1.3486256913148768e+71
Intercept: 2.998213059419967e+69

Iteration: 56
Gradient: -2.495807515544303e+72
Intercept: -5.548583817654936e+70

Iteration: 57
Gradient: 4.618816914702445e+73
Intercept: 1.0268377120436608e+72

Iteration: 58
Gradient: -8.547722353856626e+74
Intercept: -1.9002969433751722e+73

Iteration: 59
Gradient: 1.5818673653429972e+76
Intercept: 3.516747028908764e+74

Iteration: 60
Gradient: -2.927451615702262e+77
Intercept: -6.5081984731146015e+75
```

```
Iteration: 61
Gradient: 5.417630548576083e+78
Intercept: 1.2044269041038828e+77

Iteration: 62
Gradient: -1.0026031037860172e+80
Intercept: -2.2289488762856323e+78

Iteration: 63
Gradient: 1.8554476439622787e+81
Intercept: 4.124960241394994e+79

Iteration: 64
Gradient: -3.433747558216152e+82
Intercept: -7.63377624947779e+80

Iteration: 65
Gradient: 6.3545971409771e+83
Intercept: 1.4127297335448678e+82

Iteration: 66
Gradient: -1.1760009767605734e+85
Intercept: -2.614440395968224e+83

Iteration: 67
Gradient: 2.176342994937949e+86
Intercept: 4.838362513202805e+84

Iteration: 68
Gradient: -4.027606205449435e+87
Intercept: -8.954020082181548e+85

Iteration: 69
Gradient: 7.453609924495054e+88
Intercept: 1.657058052457466e+87

Iteration: 70
Gradient: -1.3793876082364342e+90
Intercept: -3.0666017766459305e+88

Iteration: 71
Gradient: 2.5527364498956257e+91
Intercept: 5.675146047286336e+89

Iteration: 72
Gradient: -4.7241713233578403e+92
Intercept: -1.0502597012532918e+91
```

```
Iteration: 73
Gradient: 8.742694410678035e+93
Intercept: 1.9436423853868828e+92

Iteration: 74
Gradient: -1.6179494841895935e+95
Intercept: -3.5969634155860375e+93

Iteration: 75
Gradient: 2.9942262767324124e+96
Intercept: 6.6566493457534944e+94

Iteration: 76
Gradient: -5.5412057569680364e+97
Intercept: -1.2318996718264127e+96

Iteration: 77
Gradient: 1.0254723058059561e+99
Intercept: 2.279790811595231e+97

Iteration: 78
Gradient: -1.897770081272674e+100
Intercept: -4.219049865423137e+98

Iteration: 79
Gradient: 3.5120707414356927e+101
Intercept: 7.807901354980727e+99

Iteration: 80
Gradient: -6.499544394006281e+102
Intercept: -1.4449538524948384e+101

Iteration: 81
Gradient: 1.202825354035711e+104
Intercept: 2.674075325641491e+102

Iteration: 82
Gradient: -2.225984999264450e+105
Intercept: -4.948724718687992e+103

Iteration: 83
Gradient: 4.119475200888759e+106
Intercept: 9.158259719358732e+104

Iteration: 84
Gradient: -7.623625467532367e+107
Intercept: -1.6948552577696285e+106
```

```
Iteration: 85
Gradient: 1.410851199120438e+109
Intercept: 3.1365504285900416e+107

Iteration: 86
Gradient: -2.6109639233154365e+110
Intercept: -5.804595139312819e+108

Iteration: 87
Gradient: 4.831928847708896e+111
Intercept: 1.0742159419537868e+110

Iteration: 88
Gradient: -8.942113746127297e+112
Intercept: -1.9879765293747444e+111

Iteration: 89
Gradient: 1.6548546298771186e+114
Intercept: 3.6790095240598034e+112

Iteration: 90
Gradient: -3.0625240561402624e+115
Intercept: -6.808486356918804e+113

Iteration: 91
Gradient: 5.6675996943213334e+116
Intercept: 1.2599990885915405e+115

Iteration: 92
Gradient: -1.0488631503373286e+118
Intercept: -2.3317924425862914e+116

Iteration: 93
Gradient: 1.9410578860003225e+119
Intercept: 4.315285657373333e+117

Iteration: 94
Gradient: -3.5921804628109e+120
Intercept: -7.985998223786084e+118

Iteration: 95
Gradient: 6.647797868609357e+121
Intercept: 1.477912997980636e+120

Iteration: 96
Gradient: -1.2302615906803773e+123
Intercept: -2.735070517664843e+121
```

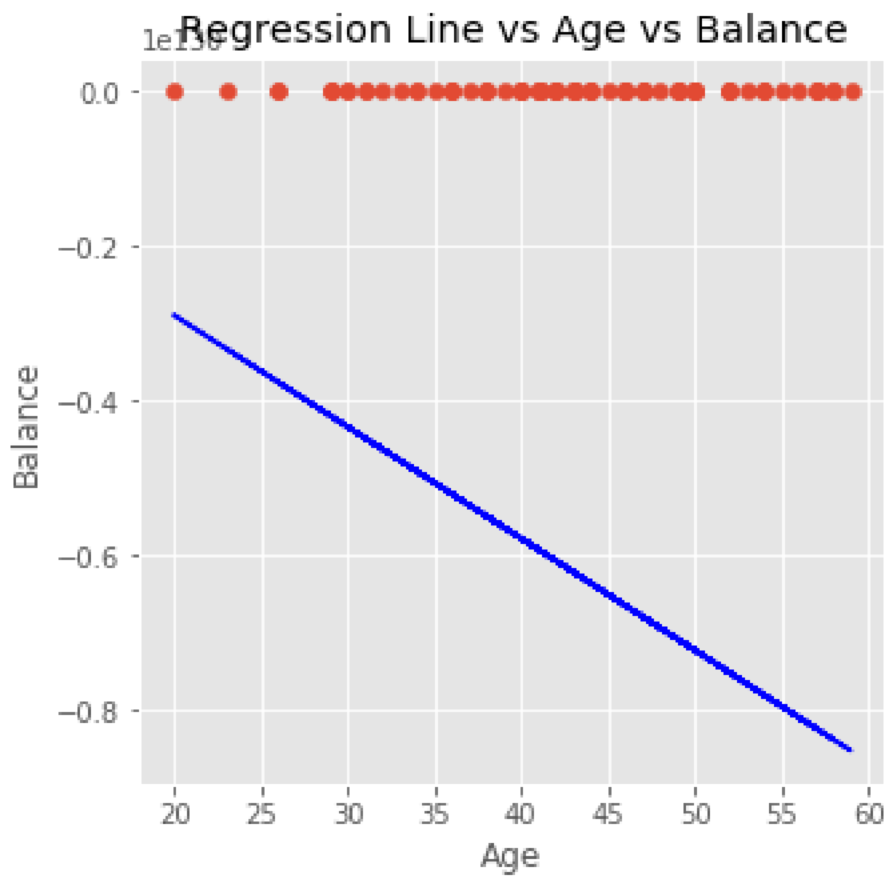
```
Iteration: 97
Gradient: 2.2767593290558745e+124
Intercept: 5.061604266841583e+122

Iteration: 98
Gradient: -4.2134397120991405e+125
Intercept: -9.36715802705618e+123

Iteration: 99
Gradient: 7.797519035468681e+126
Intercept: 1.7335146107460257e+125

Iteration: 100
Gradient: -1.443032469027667e+128
Intercept: -3.208094597091305e+126


Process Complete... after 100 epochs
Gradient(m): -1.443032469027667e+128 Intercept(c): -3.208094597091305e+126
```

5 Making predictions

- Using the m & c generated after training the model and hopefully reaching global/local minima

In [14]:
```
Y_pred = m*X + c
```

In [15]:
```
plt.figure(figsize=(5,5))
plt.scatter(X, Y)
plt.plot(X, Y_pred, color='blue') # regression line
plt.xlabel('Age')
plt.ylabel('Balance')
plt.title("Regression Line vs Age vs Balance")
plt.show()
```



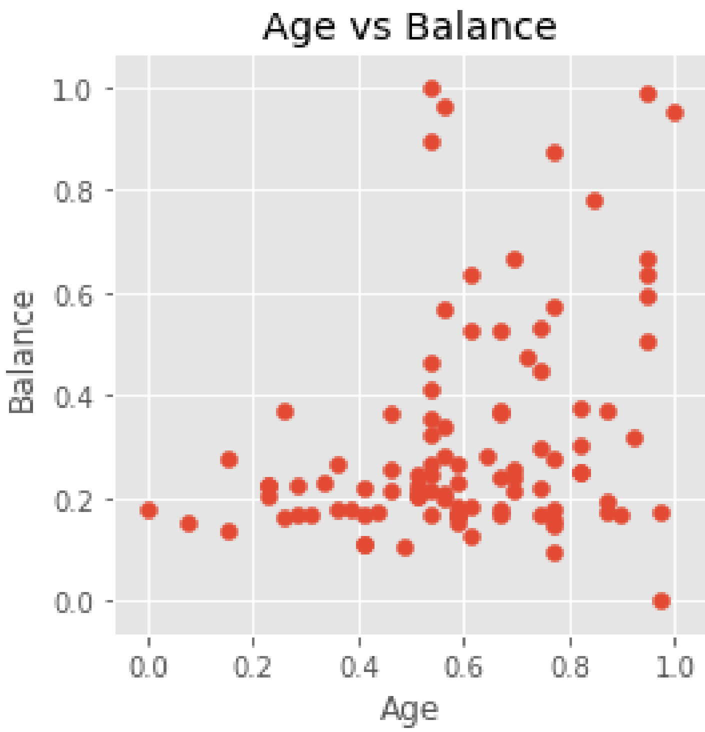
- As seen above the model performs poorly and fails to produce a line of best fit due to each column not being represented on the same scale
- Lets see how the model performs after each column is standardized.

  Making use of a Standardardazion / Normalising technique to normalise columns with high variance: scaling between 0 and 1

6 Performing Feature Standardization before implementing the model

```
In [16]: # for every value within the column value v: apply the formula
         # taking the minimum and maximum value within the balance column and using the standa

         new_X = (X - X.min()) / (X.max() - X.min())
         new_Y = (Y - Y.min()) / (Y.max() - Y.min())

In [17]: plt.figure(figsize = (4,4))
         plt.scatter(new_X,new_Y)
         plt.xlabel('Age')
```

```
plt.ylabel('Balance')
plt.title("Age vs Balance")
plt.show()
```



7 Fitting the model on the Standardized variables

- alpha = 0.0001 & iterations = 100

```
In [18]: new_m = 0 # gradient
         new_c = 0 # Intercept

         new_alpha = 0.0001  # The learning Rate
         new_epochs = 100  # The number of iterations to perform gradient descent

         new_n = float(len(new_X)) # Number of elements in X

         # Performing Gradient Descent
         for i in range(new_epochs):
             new_Y_pred = new_m*new_X + new_c  # The current predicted value of Y
             new_D_m = (-2/new_n) * sum(new_X * (new_Y - new_Y_pred))  # Partial derivative wr
             new_D_c = (-2/new_n) * sum(new_Y - new_Y_pred)  # Partial derivative wrt c
             new_m = new_m - new_alpha * new_D_m  # Update m
             new_c = new_c - new_alpha * new_D_c  # Update c
```

```
            print("Iteration: {}".format(i+1), "\nGradient: {}".format(new_m), "\nIntercept: 

        print("\nProcess Complete... after {} epochs".format(new_epochs))
        print("Gradient(m): {}".format(new_m), "Intercept(c): {}".format(new_c))
```

```
Iteration: 1
Gradient: 4.119467050954942e-05
Intercept: 6.363957918283176e-05

Iteration: 2
Gradient: 8.237838637196234e-05
Intercept: 0.00012726149903912782

Iteration: 3
Gradient: 0.0001235511505884527
Intercept: 0.00019086576441213336

Iteration: 4
Gradient: 0.00016471296615941153
Intercept: 0.0002544523801437657

Iteration: 5
Gradient: 0.0002058638360844073
Intercept: 0.0003180213510746146

Iteration: 6
Gradient: 0.0002470037633621861
Intercept: 0.00038157268204394264

Iteration: 7
Gradient: 0.0002881327509906718
Intercept: 0.0004451063778896855

Iteration: 8
Gradient: 0.0003292508019669663
Intercept: 0.0005086224434484525

Iteration: 9
Gradient: 0.0003703579192873498
Intercept: 0.0005721208835555267

Iteration: 10
Gradient: 0.0004114541059472809
Intercept: 0.0006356017030448656

Iteration: 11
Gradient: 0.0004525393649413971
```

```
Intercept: 0.000699064906749101

Iteration: 12
Gradient: 0.0004936136992635148
Intercept: 0.0007625104994995404

Iteration: 13
Gradient: 0.0005346771119066294
Intercept: 0.0008259384861261657

Iteration: 14
Gradient: 0.000575729605862916
Intercept: 0.0008893488714576352

Iteration: 15
Gradient: 0.0006167711841237291
Intercept: 0.000952741660321283

Iteration: 16
Gradient: 0.0006578018496796031
Intercept: 0.0010161168575431197

Iteration: 17
Gradient: 0.0006988216055202526
Intercept: 0.0010794744679478326

Iteration: 18
Gradient: 0.0007398304546345723
Intercept: 0.0011428144963587863

Iteration: 19
Gradient: 0.0007808284000106376
Intercept: 0.001206136947598023

Iteration: 20
Gradient: 0.0008218154446357044
Intercept: 0.0012694418264862627

Iteration: 21
Gradient: 0.0008627915914962099
Intercept: 0.0013327291378429035

Iteration: 22
Gradient: 0.0009037568435777722
Intercept: 0.0013959988864860222

Iteration: 23
Gradient: 0.0009447112038651911
```

```
Intercept: 0.0014592510772323747

Iteration: 24
Gradient: 0.0009856546753424479
Intercept: 0.0015224857148973963

Iteration: 25
Gradient: 0.0010265872609927056
Intercept: 0.0015857028042952017

Iteration: 26
Gradient: 0.0010675089637983096
Intercept: 0.0016489023502385858

Iteration: 27
Gradient: 0.0011084197867407878
Intercept: 0.001712084357539024

Iteration: 28
Gradient: 0.00114931973280085
Intercept: 0.0017752488310066724

Iteration: 29
Gradient: 0.0011902088049583891
Intercept: 0.0018383957754503682

Iteration: 30
Gradient: 0.001231087006192481
Intercept: 0.0019015251956776304

Iteration: 31
Gradient: 0.0012719543394813851
Intercept: 0.0019646370964946595

Iteration: 32
Gradient: 0.0013128108078025436
Intercept: 0.0020277314827063385

Iteration: 33
Gradient: 0.001353656414132583
Intercept: 0.002090808359116233

Iteration: 34
Gradient: 0.0013944911614473133
Intercept: 0.002153867730526591

Iteration: 35
Gradient: 0.0014353150527217286
```

```
Intercept: 0.0022169096017383445

Iteration: 36
Gradient: 0.0014761280909300078
Intercept: 0.0022799339775511087

Iteration: 37
Gradient: 0.0015169302790455138
Intercept: 0.0023429408627631833

Iteration: 38
Gradient: 0.0015577216200407945
Intercept: 0.0024059302621715524

Iteration: 39
Gradient: 0.0015985021168875829
Intercept: 0.0024689021805718845

Iteration: 40
Gradient: 0.001639271772556797
Intercept: 0.002531856622758533

Iteration: 41
Gradient: 0.0016800305900185406
Intercept: 0.0025947935935245374

Iteration: 42
Gradient: 0.0017207785722421027
Intercept: 0.002657713097661623

Iteration: 43
Gradient: 0.0017615157221959588
Intercept: 0.002720615139960201

Iteration: 44
Gradient: 0.00180224204284777
Intercept: 0.0027834997252093687

Iteration: 45
Gradient: 0.001842957537164384
Intercept: 0.0028463668581969115

Iteration: 46
Gradient: 0.001883662208111835
Intercept: 0.002909216543709301

Iteration: 47
Gradient: 0.001924356058655344
```

```
Intercept: 0.0029720487865316975

Iteration: 48
Gradient: 0.001965039091759319
Intercept: 0.003034863591447948

Iteration: 49
Gradient: 0.002005711310387355
Intercept: 0.0030976609632405888

Iteration: 50
Gradient: 0.0020463727175022354
Intercept: 0.003160440906690845

Iteration: 51
Gradient: 0.0020870233160659303
Intercept: 0.0032232034265786296

Iteration: 52
Gradient: 0.002127663109039598
Intercept: 0.003285948527682547

Iteration: 53
Gradient: 0.0021682920993835843
Intercept: 0.0033486762147798893

Iteration: 54
Gradient: 0.002208910290057425
Intercept: 0.0034113864926466403

Iteration: 55
Gradient: 0.002249517684019843
Intercept: 0.0034740793660574735

Iteration: 56
Gradient: 0.00229011428422875
Intercept: 0.0035367548397857535

Iteration: 57
Gradient: 0.002330700093641248
Intercept: 0.0035994129186035367

Iteration: 58
Gradient: 0.002371275115213627
Intercept: 0.0036620536072815703

Iteration: 59
Gradient: 0.0024118393519013672
```

```
Intercept: 0.0037246769105892933

Iteration: 60
Gradient: 0.0024523928066591378
Intercept: 0.003787282833294838

Iteration: 61
Gradient: 0.0024929354824407986
Intercept: 0.0038498713801650283

Iteration: 62
Gradient: 0.002533467382199399
Intercept: 0.003912442555965382

Iteration: 63
Gradient: 0.002573988508887179
Intercept: 0.00397499636546011

Iteration: 64
Gradient: 0.0026144988654555694
Intercept: 0.004037532813412116

Iteration: 65
Gradient: 0.0026549984548551913
Intercept: 0.004100051904583

Iteration: 66
Gradient: 0.0026954872800358572
Intercept: 0.004162553643733055

Iteration: 67
Gradient: 0.0027359653439465704
Intercept: 0.004225038035621268

Iteration: 68
Gradient: 0.002776432649535526
Intercept: 0.004287505085005322

Iteration: 69
Gradient: 0.0028168891997501114
Intercept: 0.004349954796641597

Iteration: 70
Gradient: 0.002857334997536904
Intercept: 0.004412387175285167

Iteration: 71
Gradient: 0.002897770045841676
```

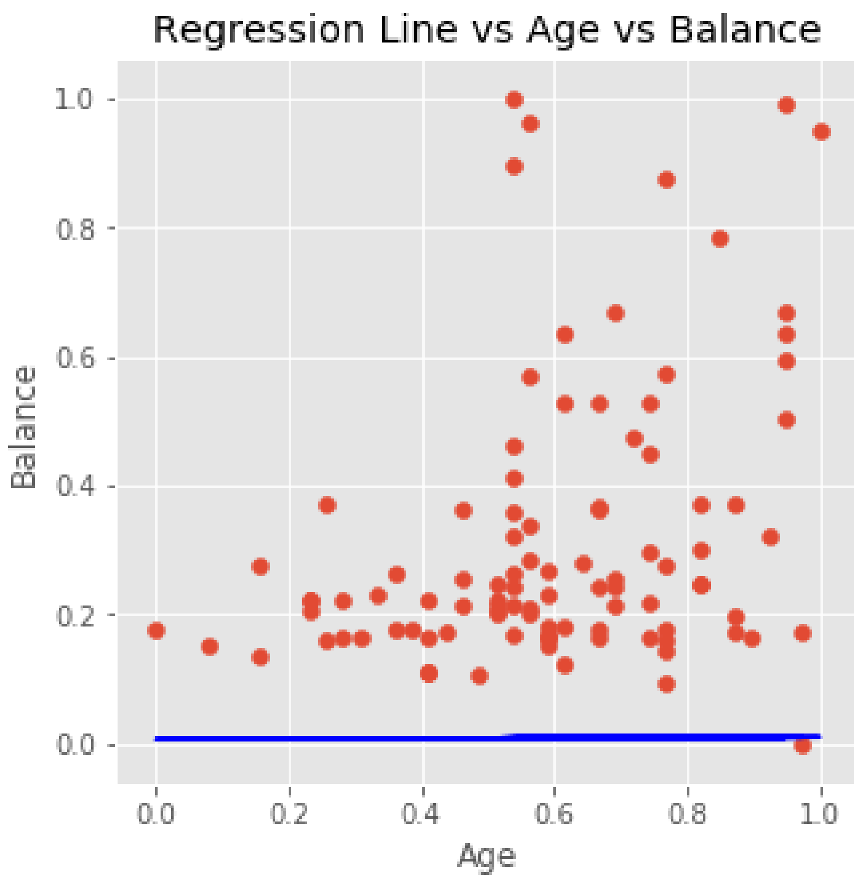
```
Intercept: 0.004474802225689802

Iteration: 72
Gradient: 0.0029381943476093895
Intercept: 0.00453719995260797

Iteration: 73
Gradient: 0.0029786079057842006
Intercept: 0.004599580360790836

Iteration: 74
Gradient: 0.003019010723309458
Intercept: 0.0046619434549882615

Iteration: 75
Gradient: 0.0030594028031277025
Intercept: 0.004724289239948805

Iteration: 76
Gradient: 0.0030997841481806695
Intercept: 0.004786617720419725

Iteration: 77
Gradient: 0.003140154761409287
Intercept: 0.004848928901146975

Iteration: 78
Gradient: 0.0031805146457536764
Intercept: 0.004911222786875212

Iteration: 79
Gradient: 0.0032208638041531542
Intercept: 0.00497349938234779

Iteration: 80
Gradient: 0.00326120223954623
Intercept: 0.00503575869230676

Iteration: 81
Gradient: 0.003301529954870608
Intercept: 0.005098000721492877

Iteration: 82
Gradient: 0.003341846953063187
Intercept: 0.005160225474645593

Iteration: 83
Gradient: 0.00338215323706006
```

```
Intercept: 0.005222432956503062

Iteration: 84
Gradient: 0.0034224488097965165
Intercept: 0.005284623171802138

Iteration: 85
Gradient: 0.00346273367420704
Intercept: 0.005346796125278377

Iteration: 86
Gradient: 0.0035030078332253093
Intercept: 0.005408951821666037

Iteration: 87
Gradient: 0.0035432712897841994
Intercept: 0.005471090265698077

Iteration: 88
Gradient: 0.0035835240468157815
Intercept: 0.005533211462106159

Iteration: 89
Gradient: 0.003623766107251322
Intercept: 0.005595315415620646

Iteration: 90
Gradient: 0.0036639974740212845
Intercept: 0.005657402130970607

Iteration: 91
Gradient: 0.0037042181500553284
Intercept: 0.0057194716128838126

Iteration: 92
Gradient: 0.0037444281382823107
Intercept: 0.005781523866086736

Iteration: 93
Gradient: 0.0037846274416302845
Intercept: 0.0058435588953045575

Iteration: 94
Gradient: 0.0038248160630265002
Intercept: 0.0059055767052611586

Iteration: 95
Gradient: 0.0038649940053974065
```

```
Intercept: 0.005967577300679128

Iteration: 96
Gradient: 0.0039051612716686493
Intercept: 0.0060295606862797585

Iteration: 97
Gradient: 0.0039453178647650715
Intercept: 0.006091526866783049

Iteration: 98
Gradient: 0.003985463787610715
Intercept: 0.006153475846907703

Iteration: 99
Gradient: 0.004025599043128821
Intercept: 0.006215407631371132

Iteration: 100
Gradient: 0.004065723634241827
Intercept: 0.006277322224889452


Process Complete... after 100 epochs
Gradient(m): 0.004065723634241827 Intercept(c): 0.006277322224889452
```

In [19]:
```
new_Y_pred = new_m*new_X + new_c
```

In [20]:
```
plt.figure(figsize=(5,5))
plt.scatter(new_X, new_Y)
plt.plot(new_X, new_Y_pred, color='blue') # regression line
plt.xlabel('Age')
plt.ylabel('Balance')
plt.title("Regression Line vs Age vs Balance")
plt.show()
```



As expected the linear regression model performs well once the data has been standardized, unlike the unstandardized data. The model was unable to properly converge which resulted in almost random gradient values as it attempted to find the global/local minima.

A learning rate of 0.0001 was used, with 100 iterations.

8 Using a higher learning rate with a higher iteration

- alpha = 0.005 & epochs = 1000

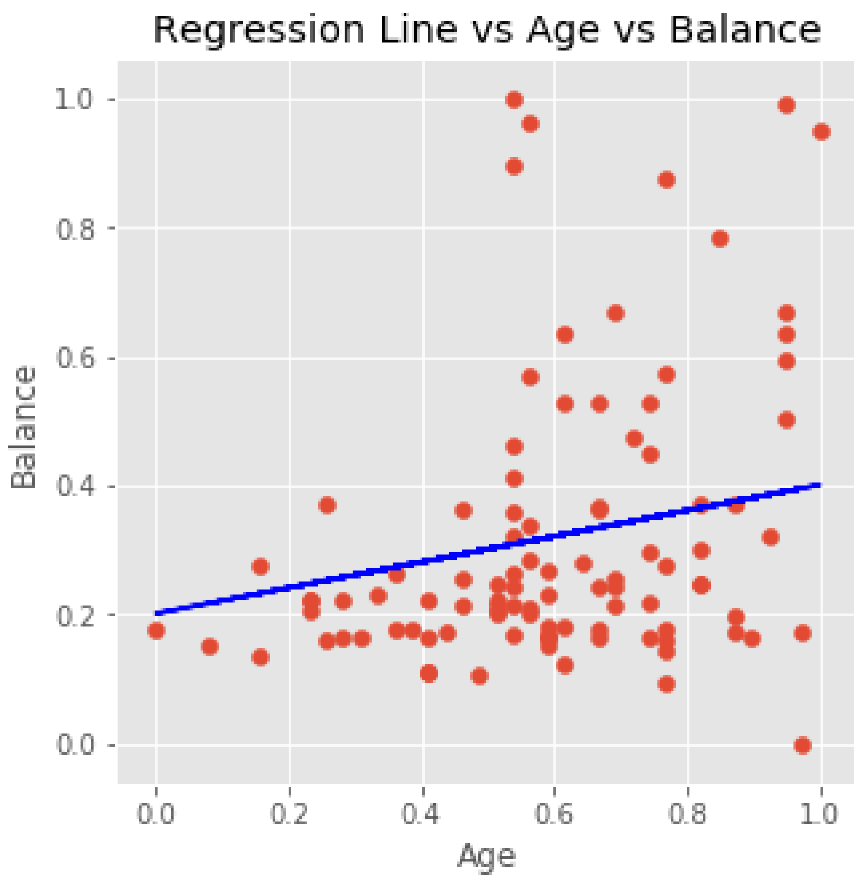
```
In [21]: new_m = 0 # gradient
         new_c = 0 # Intercept

         new_alpha = 0.005  # The learning Rate
         new_epochs = 1000  # The number of iterations to perform gradient descent

         new_n = float(len(new_X)) # Number of elements in X
```

```
# Performing Gradient Descent
for i in range(new_epochs):
    new_Y_pred = new_m*new_X + new_c  # The current predicted value of Y
    new_D_m = (-2/new_n) * sum(new_X * (new_Y - new_Y_pred))  # Partial derivative wr
    new_D_c = (-2/new_n) * sum(new_Y - new_Y_pred)  # Partial derivative wrt c
    new_m = new_m - new_alpha * new_D_m  # Update m
    new_c = new_c - new_alpha * new_D_c  # Update c

    ### commented out on purpose
    #print("Iteration: {}".format(i+ 1), "\nGradient: {}".format(new_m),
    #"\nIntercept: {}".format(new_c), "\n")

print("\nProcess Complete... after {} epochs".format(new_epochs))
print("Gradient(m): {}".format(new_m), "Intercept(c): {}".format(new_c))
```

```
Process Complete... after 1000 epochs
Gradient(m): 0.19951142488472115 Intercept(c): 0.20158307777849563
```

In [22]:
```
new_Y_pred = new_m*new_X + new_c
```

In [23]:
```
plt.figure(figsize=(5,5))
plt.scatter(new_X, new_Y)
plt.plot(new_X, new_Y_pred, color='blue') # regression line
plt.xlabel('Age')
plt.ylabel('Balance')
plt.title("Regression Line vs Age vs Balance")
plt.show()
```



9 Using a lower learning rate with a higher iteration

- alpha = 0.001 & epochs = 2000

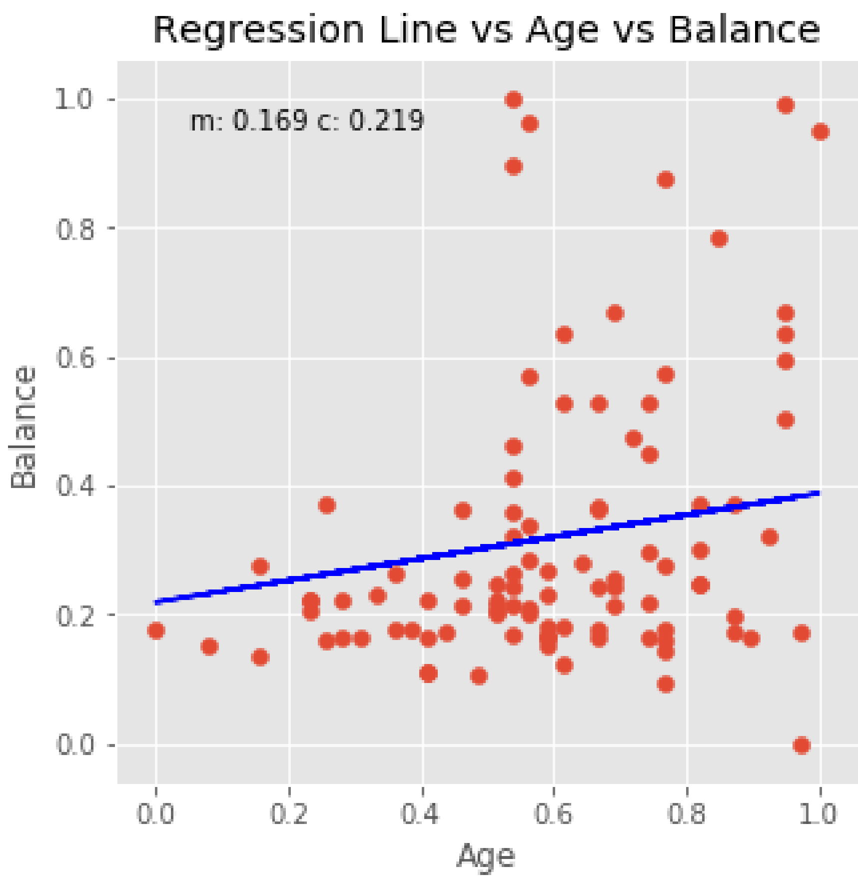
In [24]:
```
new_m = 0 # gradient
new_c = 0 # Intercept

m_list = []
c_list = []

new_alpha = 0.001  # The learning Rate
new_epochs = 2000  # The number of iterations to perform gradient descent

new_n = float(len(new_X)) # Number of elements in X

# Performing Gradient Descent
for i in range(new_epochs):
    # append current m and c into a list
    m_list.append(new_m)
```

```
        c_list.append(new_c)

        new_Y_pred = new_m*new_X + new_c  # The current predicted value of Y
        new_D_m = (-2/new_n) * sum(new_X * (new_Y - new_Y_pred))  # Partial derivative wr
        new_D_c = (-2/new_n) * sum(new_Y - new_Y_pred)  # Partial derivative wrt c
        new_m = new_m - new_alpha * new_D_m  # Update m
        new_c = new_c - new_alpha * new_D_c  # Update c

        ### Commented out to reduce number of pages
        #print("Iteration: {}".format(i+ 1), "\nGradient: {}".format(new_m),
              #"\nIntercept: {}".format(new_c), "\n")

    print("\nProcess Complete... after {} epochs".format(new_epochs))
    print("Gradient(m): {}".format(new_m), "Intercept(c): {}".format(new_c))
```

```
Process Complete... after 2000 epochs
Gradient(m): 0.16878962845968176 Intercept(c): 0.21927932474659825
```

```
In [25]: new_Y_pred = new_m*new_X + new_c
```

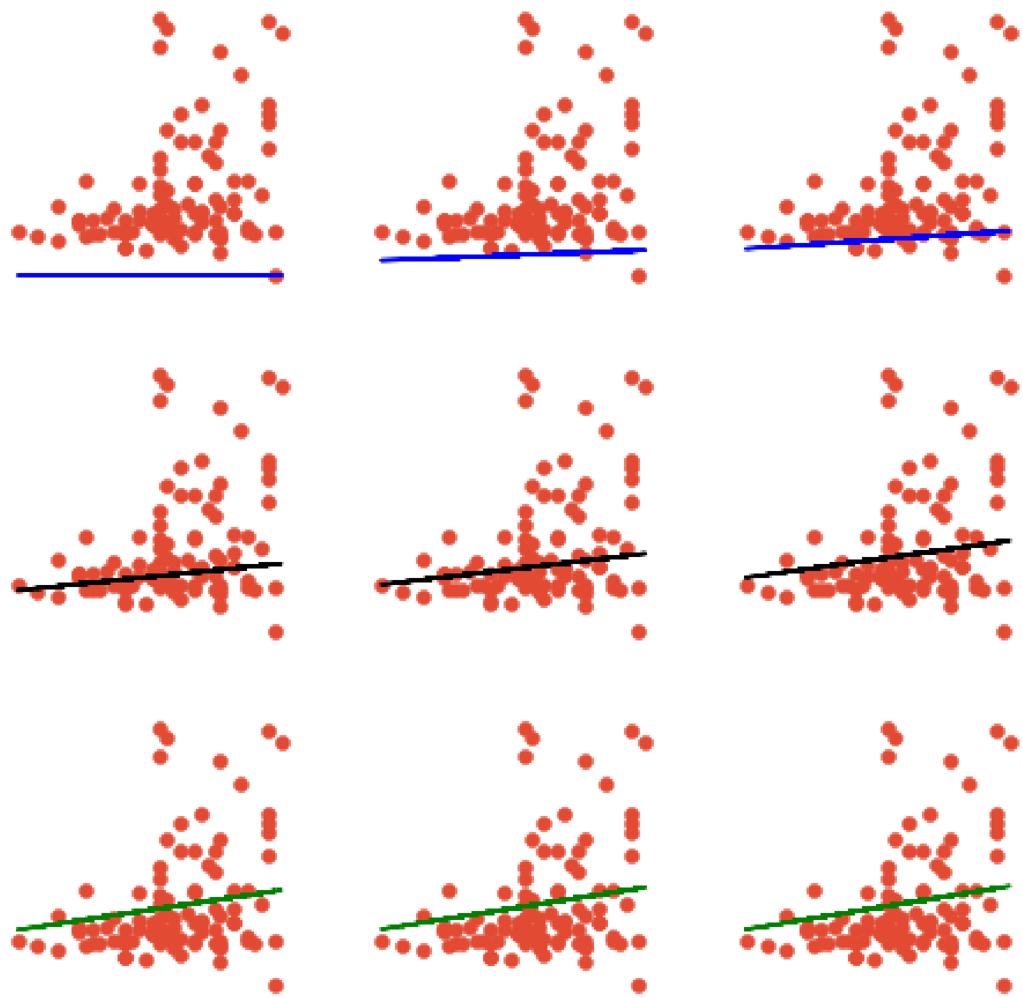
```
In [26]: plt.figure(figsize=(5,5))
         plt.scatter(new_X, new_Y)
         plt.plot(new_X, new_Y_pred, color='blue') # regression line
         plt.xlabel('Age')
         plt.ylabel('Balance')
         plt.text(0.1,0.9,"m: {} c: {}".format(round(new_m, 3), round(new_c, 3)),transform=plt
         plt.title("Regression Line vs Age vs Balance")
         plt.show()
```



10 The algorithm in action as it finds the line of best fit

```
In [27]: y_pred_list = [] # list to store predictions

         # Store every prediction for each cobination of m and c
         for i in range(len(m_list)):
             pred = m_list[i]*new_X + c_list[i]
             y_pred_list.append(pred)

In [28]: fig, ax = plt.subplots(3, 3, figsize =(8,8)) # 5 rows, 5 columns

         ax[0, 0].scatter(new_X,new_Y) #row=0, col=0
         ax[0, 0].plot(new_X, y_pred_list[0], 'b')
         ax[0, 1].scatter(new_X,new_Y)
         ax[0, 1].plot(new_X, y_pred_list[112], 'b')
         ax[0, 2].scatter(new_X,new_Y)
         ax[0, 2].plot(new_X, y_pred_list[224], 'b')
         ax[1, 0].scatter(new_X,new_Y) #row=1, col=0
         ax[1, 0].plot(new_X, y_pred_list[420], 'k')
```

```
ax[1, 1].scatter(new_X,new_Y)
ax[1, 1].plot(new_X, y_pred_list[560], 'k')
ax[1, 2].scatter(new_X,new_Y)
ax[1, 2].plot(new_X, y_pred_list[888], 'k')
ax[2, 0].scatter(new_X,new_Y) #row=2, col=0
ax[2, 0].plot(new_X, y_pred_list[1200], 'g')
ax[2, 1].scatter(new_X,new_Y)
ax[2, 1].plot(new_X, y_pred_list[1600], 'g')
ax[2, 2].scatter(new_X,new_Y)
ax[2, 2].plot(new_X, y_pred_list[1999], 'g')
[axi.set_axis_off() for axi in ax.ravel()]
plt.show()
```



11 New Prediction

```
In [29]: value = int(input("Enter Age: "))
         # Standardize value first
         my_x = (value - new_X.min()) / (new_X.max() - new_X.min())


         my_pred = new_m*my_x + new_c
         # value had been standardized so to get original value
         pred_balance = my_pred * 100
         print("Predicted Balance: {}".format(pred_balance))
```

```
Enter Age: 60
Predicted Balance: 1034.6657032327503
```
