Documentation

The data is of the form:

- Column 1: age (int)
- Column 2: Workclass (string), 8 options:
  - Private
  - Self-emp-not-inc
  - Self-emp-inc
  - Federal-gov
  - Local-gov
  - State-gov
  - Without-pay
  - Never-worked
- Column 3: final weight (int)
  - will be excluded from our model because this value has no effect on whether the person makes above 50k
- Column 4: Education (string), 16 options:
  - Bachelors
  - Some-college
  - 11th
  - HS-grad
  - Prof-school
  - Assoc-acdm
  - Assoc-voc
  - 9th
  - 7th-8th
  - 12th
  - Masters
  - 1st-4th
  - 10th
  - Doctorate
  - 5th-6th
  - Preschool
- Column 5: Education number (int)
- Column 6: Marital status (string), 7 options:
  - Married-civ-spouse
  - Divorced
  - Never-married
  - Separated
  - Widowed
  - Married-spouse-absent
  - Married-AF-spouse
- Column 7: Occupation (string), 14 options:
  - Tech-support
  - Craft-repair

    - Other-service
    - Sales
    - Exec-managerial
    - Prof-specialty
    - Handlers-cleaners
    - Machine-op-inspct
    - Adm-clerical
    - Farming-fishing
    - Transport-moving
    - Priv-house-serv
    - Protective-serv
    - Armed-Forces
- Column 8: relationship (string), 6 options:
    - Wife
    - Own-child
    - Husband
    - Not-in-family
    - Other-relative
    - Unmarried
- Column 9: race (string), 5 options:
    - White
    - Asian-Pac-Islander
    - Amer-Indian-Eskimo
    - Other
    - Black
- Column 10: sex (string), 2 options:
    - male
    - female
- Column 11: capital-gain (int)
- Column 12: capital-loss (int)
- Column 13: hours per week (int)
- Column 14: country of origin (string), 41 options:
    - United-States
    - Cambodia
    - England
    - Puerto-Rico
    - Canada
    - Germany
    - Outlying-US(Guam-USVI-etc)
    - India
    - Japan
    - Greece
    - South
    - China
    - Cuba
    - Iran
    - Honduras
    - Philippines

    - Italy
    - Poland
    - Jamaica
    - Vietnam
    - Mexico
    - Portugal
    - Ireland
    - France
    - Dominican-Republic
    - Laos
    - Ecuador
    - Taiwan
    - Haiti
    - Columbia
    - Hungary
    - Guatemala
    - Nicaragua
    - Scotland
    - Thailand
    - Yugoslavia
    - El-Salvador
    - Trinadad&Tobago
    - Peru
    - Hong
    - Holand-Netherlands
- Column 15: whether the person made above of below 50k, (string)

NOTE: all columns have empty data cells, filled with '?' by the researchers

Resulting Matrix:

We will need to do some feature engineering to make the data opperable for the qr decomposition processes to get our model. Thus, the resulting matrix should look like the following:

- Column 1: a ones vector
- Column 2: (*Directly from the data*)
    - age
- Column 3 - 10: (*entries will be either 1 for true or 0 for false*)
    - Private
    - Self-emp-not-inc
    - Self-emp-inc
    - Federal-gov
    - Local-gov
    - State-gov
    - Without-pay
    - Never-worked
- Column 11 - 26: (*entries will be either 1 for true or 0 for false*)
    - Bachelors
    - Some-college

    - 11th
    - HS-grad
    - Prof-school
    - Assoc-acdm
    - Assoc-voc
    - 9th
    - 7th-8th
    - 12th
    - Masters
    - 1st-4th
    - 10th
    - Doctorate
    - 5th-6th
    - Preschool
- Column 27: (*Directly from the data*)
    - Education number
- Column 28 - 34: (*entries will be either 1 for true or 0 for false*)
    - Married-civ-spouse
    - Divorced
    - Never-married
    - Separated
    - Widowed
    - Married-spouse-absent
    - Married-AF-spouse
- Column 35 - 48: (*entries will be either 1 for true or 0 for false*)
    - Tech-support
    - Craft-repair
    - Other-service
    - Sales
    - Exec-managerial
    - Prof-specialty
    - Handlers-cleaners
    - Machine-op-inspct
    - Adm-clerical
    - Farming-fishing
    - Transport-moving
    - Priv-house-serv
    - Protective-serv
    - Armed-Forces
- Column 49 - 54: (*entries will be either 1 for true or 0 for false*)
    - Wife
    - Own-child
    - Husband
    - Not-in-family
    - Other-relative
    - Unmarried
- Column 55 - 59: (*entries will be either 1 for true or 0 for false*)
    - White

    - Asian-Pac-Islander
    - Amer-Indian-Eskimo
    - Other
    - Black
- Column 60 - 61: (*entries will be either 1 for true or 0 for false*)
    - male
    - female
- Column 62: (*Directly from the data*)
    - capital-gain
- Column 63: (*Directly from the data*)
    - capital-loss
- Column 64: (*Directly from the data*)
    - hours per week
- Column 65 - 105: (*entries will be either 1 for true or 0 for false*)
    - United-States
    - Cambodia
    - England
    - Puerto-Rico
    - Canada
    - Germany
    - Outlying-US(Guam-USVI-etc)
    - India
    - Japan
    - Greece
    - South
    - China
    - Cuba
    - Iran
    - Honduras
    - Philippines
    - Italy
    - Poland
    - Jamaica
    - Vietnam
    - Mexico
    - Portugal
    - Ireland
    - France
    - Dominican-Republic
    - Laos
    - Ecuador
    - Taiwan
    - Haiti
    - Columbia
    - Hungary
    - Guatemala
    - Nicaragua
    - Scotland

- Thailand
- Yugoslavia
- El-Salvador
- Trinadad&Tobago
- Peru
- Hong
- Holand-Netherlands

Methodology:

We will use QR decomposition to obtain the x hats for the equation:

$$f(a) = 1 + \hat{x}_1 x_1 + \hat{x}_2 x_2 + \ldots + \hat{x}_{105} x_{105}$$

Only the first 80% of the data will be used to create the model, and the model will be tested on the final 20% of the data.

In [28]:
```
using DataFrames
using CSV
dataSet = CSV.read("CensusData1994.csv", DataFrame; header = false)
X = Matrix(dataSet[:,1:15])
m,n=size(X)
#80% of the data is 26,048.8 rows, round up for 26,049.
#20% is therefore 6512
m1 = 26049
```

Out[28]: 26049

In [29]:
```
employmentDict = ["Private", "Self-emp-not-inc", "Self-emp-inc", "Federal-gov", "Local-gov", "Sta
educationDict = ["Bachelors", "Some-college", "11th", "HS-grad", "Prof-school", "Assoc-acdm", "As
maritalStatusDict = ["Married-civ-spouse", "Divorced", "Never-married", "Separated", "Widowed", '
occupationDict = ["Tech-support", "Craft-repair", "Other-service", "Sales", "Exec-managerial", "P
relationshipDict = ["Wife", "Own-child", "Husband", "Not-in-family", "Other-relative", "Unmarriec
raceDict = ["White", "Asian-Pac-Islander", "Amer-Indian-Eskimo", "Other", "Black"];
sexDict = ["Male","Female"];
countryDict = ["United-States", "Cambodia", "England", "Puerto-Rico", "Canada", "Germany", "Outly
```

In [30]:
```
firstCol = ones(m1);
age = X[1:m1,1];
educationNum = X[1:m1,5];
capGain = X[1:m1, 11];
capLoss = X[1:m1, 12];
hours = X[1:m1,13];
income = X[1:m1,15];
y = zeros(m1);
for k in 1:m1
   if strip(income[k]) == ">50K"
        y[k] = 1
    end
end
```

In [31]:
```
function createDataMatrix(dict, m, n, col, X)
    matrix = zeros(m,n);
    for i in 1:m
        for j in 1:n
            if strip(X[i,col]) == dict[j]
                matrix[i,j] = 1
            end
        end
```

```
    end
    return matrix
end
```

Out[31]: createDataMatrix (generic function with 1 method)

In [32]:
```
employmentMatrix = createDataMatrix(employmentDict, m1, length(employmentDict), 2, X);
educationMatrix = createDataMatrix(educationDict, m1, length(educationDict), 4, X);
maritalStatusMatrix = createDataMatrix(maritalStatusDict, m1, length(maritalStatusDict), 6, X);
occupationMatrix = createDataMatrix(occupationDict, m1, length(occupationDict), 7, X);
relationshipMatrix = createDataMatrix(relationshipDict, m1, length(relationshipDict), 8, X);
raceMatrix = createDataMatrix(raceDict, m1, length(raceDict), 9, X);
sexMatrix = createDataMatrix(sexDict, m1, length(sexDict), 10, X);
countryMatrix = createDataMatrix(countryDict, m1, length(countryDict), 14, X);
```

In [33]:
```
employmentMatrix
```

Out[33]:
```
26049×8 Matrix{Float64}:
 0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                        ⋮
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

In [34]:
```
a = [firstCol age employmentMatrix educationMatrix educationNum maritalStatusMatrix occupationMat
A = convert(Matrix{Float64}, a)
```

Out[34]:
```
26049×105 Matrix{Float64}:
 1.0  39.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  50.0  0.0  1.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  38.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  53.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  28.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  37.0  1.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  49.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  52.0  0.0  1.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  31.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  42.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  37.0  1.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  30.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  23.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 ⋮                        ⋮        ⋱           ⋮
 1.0  29.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  65.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  33.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  29.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  18.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  43.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  42.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  42.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  32.0  1.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  36.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  41.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  30.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

In [35]:
```
using LinearAlgebra
Q,R = qr(A); Q = Matrix(Q)
xhat = R\(Q'*y)
```

Out[35]:
```
105-element Vector{Float64}:
  2.0810773877263105e12
  0.0024652077065681183
  1.4720418421792422e11
  1.472041842178606e11
  1.472041842179807e11
  1.4720418421798953e11
  1.4720418421789096e11
  1.4720418421788544e11
  1.4720418421781647e11
  0.0987890974708189
 -4.1694477573199615e11
 -5.4221479994124e11
 -6.674848241504244e11
  ⋮
 -0.038810637503083195
 -0.06602105504624921
  0.05791058244568107
 -0.0024808268466938693
 -0.05641142718242209
  0.025582033401511092
  0.1382223461642015
  0.05138051060452354
 -0.05327900839839017
 -0.005465225154119248
  0.05412840043737516
 -0.1223583717288264
```

In [36]:
```
#test on seen data
seenTest = ones(m1).*xhat[1]
for k in 2:105
```

```
    seenTest += xhat[k].*A[:,k]
end
seenTest
```

Out[36]:
```
26049-element Vector{Float64}:
  0.17951734111687026
  0.4938213143060502
 -0.016540815836026332
  0.27545137166397365
  0.5464271103317623
  0.7539669966639737
 -0.15720437819396724
  0.4242124355265521
  0.4204733720430383
  0.653496358216227
  0.5861453513146011
  0.38959710534656056
  0.09462846268881682
  ⋮
  0.35699434041397365
  0.7732857383635066
  0.03326387166397367
 -0.030212690836026332
 -0.04746138106118318
  0.49078340291397365
  0.16412324666397365
  0.28717012166397365
  0.2152339314388168
  0.12164277791397367
  0.34185762166397365
  6.0746663973668125e-5
```

In [37]:
```
seenResult = zeros(m1)
for i in 1:m1
    if seenTest[i] > 0.5
        seenResult[i] = 1
    end
end
seenResult
```

Out[37]:
```
26049-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
 1.0
 1.0
 0.0
 0.0
 0.0
 1.0
 1.0
 0.0
 0.0
 ⋮
 0.0
 1.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
```

In [38]:
```
#comparison between seen data and actual data
totalSeenCorrect = 0
for i in 1:m1
    if seenResult[i] - y[i] == 0
        totalSeenCorrect += 1
    end
end
totalSeenCorrect
```

Out[38]:
```
21883
```

In [39]:
```
function createUnseenDataMatrix(dict, m1, m, n, col, X)
    matrix = zeros(m-m1,n);
    for i in m1:m
        for j in 1:n
            if strip(X[i,col]) == dict[j] && i < m
                matrix[i-m1+1,j] = 1
            end
        end
    end
    return matrix
end
```

Out[39]:
```
createUnseenDataMatrix (generic function with 1 method)
```

In [40]:
```
ageU
```

Out[40]:
```
6512-element Vector{Any}:
 21
 33
 19
 19
 35
 43
 45
 25
 39
 26
 60
 52
 22
  ⋮
 43
 43
 32
 43
 32
 53
 22
 27
 40
 58
 22
 52
```

In [41]:
```
firstColU = ones(m-m1);
ageU = X[m1+1:m,1];
educationNumU = X[m1+1:m,5];
capGainU = X[m1+1:m, 11];
capLossU = X[m1+1:m, 12];
hoursU = X[m1+1:m,13];
incomeU = X[m1+1:m,15];
employmentMatrixU = createUnseenDataMatrix(employmentDict, m1, m, length(employmentDict), 2, X);
educationMatrixU = createUnseenDataMatrix(educationDict, m1, m, length(educationDict), 4, X);
maritalStatusMatrixU = createUnseenDataMatrix(maritalStatusDict, m1, m, length(maritalStatusDict)
occupationMatrixU = createUnseenDataMatrix(occupationDict, m1, m, length(occupationDict), 7, X);
relationshipMatrixU = createUnseenDataMatrix(relationshipDict, m1, m, length(relationshipDict), 8
raceMatrixU = createUnseenDataMatrix(raceDict, m1, m, length(raceDict), 9, X);
sexMatrixU = createUnseenDataMatrix(sexDict, m1, m, length(sexDict), 10, X);
countryMatrixU = createUnseenDataMatrix(countryDict, m1, m, length(countryDict), 14, X);
```

In [42]:
```
#test on unseen data
test = ones(m-m1).*xhat[1]
testD = [firstColU ageU employmentMatrixU educationMatrixU educationNumU maritalStatusMatrixU occ
testData = convert(Matrix{Float64}, testD);
for k in 2:105
    test += xhat[k].*testData[:,k]
end
test
```

Out[42]:
```
6512-element Vector{Float64}:
  4.175667473634007e10
 -4.175667473635736e10
  0.19766167735108153
  4.175667473634626e10
 -0.006286909586026332
  0.0012020265513952438
  0.0363300298037227
 -1.6702669894524197e11
  0.33843965291397365
  1.6702669894601422e11
 -2.0878337368192767e11
 -8.351334947218402e10
  3.340533978917089e11
  ⋮
  2.0878337368224567e11
  0.0593016243891305
  1.6702669894577545e11
 -2.0878337368165637e11
 -1.2527002420880084e11
  0.3464293736396347
  1.6702669894616022e11
 -8.3513349472632e10
  1.2527002420965485e11
  0.31256074666397365
 -0.07887022753634004
  0.20543607011628207
```

In [43]:
```
yU = zeros(m-m1)
for i in 1:m-m1
    if income[i] == ">50K"
        yU[i] = 1
    end
end

unseenResult = zeros(m-m1)
for i in 1:m-m1
    if test[i] > 0.5
        unseenResult[i] = 1
    end
end

#comparison between unseen data and actual data
totalUnseenCorrect = 0
for i in 1:m-m1
    if unseenResult[i] - yU[i] == 0
        totalUnseenCorrect += 1
    end
end
totalUnseenCorrect
```

Out[43]:
```
3709
```

In [44]:
```
println("The percentage of correct predictions on seen data was ", (totalSeenCorrect/m1)*100)
```

```
The percentage of correct predictions on seen data was 84.00706361088717
```

In [45]:
```
println("The percentage of correct predictions on unseen data was ", (totalUnseenCorrect/(m-m1))*
```

```
The percentage of correct predictions on unseen data was 56.95638820638821
```

In [46]:
```
dataSet = CSV.read("CensusData2023.csv", DataFrame; header = false)
X = Matrix(dataSet[:,1:15])
```

```
m,n = size(X)
```

Out[46]: (9, 15)

In [47]:
```
employmentMatrix23 = createDataMatrix(employmentDict, m, length(employmentDict), 2, X);
educationMatrix23 = createDataMatrix(educationDict, m, length(educationDict), 4, X);
maritalStatusMatrix23 = createDataMatrix(maritalStatusDict, m, length(maritalStatusDict), 6, X);
occupationMatrix23 = createDataMatrix(occupationDict, m, length(occupationDict), 7, X);
relationshipMatrix23 = createDataMatrix(relationshipDict, m, length(relationshipDict), 8, X);
raceMatrix23 = createDataMatrix(raceDict, m, length(raceDict), 9, X);
sexMatrix23 = createDataMatrix(sexDict, m, length(sexDict), 10, X);
countryMatrix23 = createDataMatrix(countryDict, m, length(countryDict), 14, X);
```

In [48]:
```
firstCol = ones(m);
age = X[1:m,1];
educationNum = zeros(m);
capGain = X[1:m, 11];
capLoss = X[1:m, 12];
hours = X[1:m,13];
income = X[1:m,15];
y23 = zeros(m);
```

In [49]:
```
for i in 1:m
    if income[i] == ">=100k"
        y23[i] = 1
    end
end
y23
```

Out[49]:
```
9-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
 1.0
 0.0
 0.0
 0.0
 1.0
```

In [50]:
```
age
```

Out[50]:
```
9-element Vector{Any}:
 22
 23
  0
 20
 50
 49
 24
 22
 23
```

In [51]:
```
age = convert(Vector{Float64}, age)
```

Out[51]:
```
9-element Vector{Float64}:
 22.0
 23.0
  0.0
 20.0
 50.0
 49.0
 24.0
 22.0
 23.0
```

In [52]:
```
a = [firstCol age employmentMatrix23 educationMatrix23 educationNum maritalStatusMatrix23 occupat
A = convert(Matrix{Float64}, a)
```

Out[52]:
```
9×105 Matrix{Float64}:
 1.0  22.0  1.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  23.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0   0.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  20.0  0.0  0.0  1.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  50.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  49.0  0.0  0.0  0.0  0.0  0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  24.0  0.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  22.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 1.0  23.0  1.0  0.0  0.0  0.0  0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

In [54]:
```
test23 = ones(m)
for k in 2:105
    test23 += xhat[k].*A[:,k]
end
test23
```

Out[54]:
```
9-element Vector{Float64}:
 1.6777535107320286e12
 1.803023534941368e12
 1.8030235349413464e12
 1.6777535107320579e12
 1.8447802096780593e12
 1.8447802096781409e12
 1.6777535107321216e12
 1.8030235349414182e12
 1.8030235349414778e12
```

In [58]:
```
result23 = zeros(m)
for i in 1:m
    if test23[i] > 0.5
        result23[i] = 1
    end
end
result23
```

Out[58]:
```
9-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
```

The model guesses that every person from our survey makes more than the modern equivilent of 50k (which is 100k)

In [ ]: