Logistic Regression Project - Solutions

In this project we will be working with the UCI adult dataset. We will be attempting to predict if people in the data set belong in a certain class by salary, either making <=50k or >50k per year.

Typically most of your time is spent cleaning data, not running the few lines of code that build your model, this project will try to reflect that by showing different issues that may arise when cleaning data.

Get the Data

Read in the adult_sal.csv file and set it to a data frame called adult.

In [1]:

```
adult <- read.csv('adult_sal.csv')
```

Check the head of adult

In [2]:

```
head(adult)
```

Out[2]:

	X	age	type_employer	fnlwgt	education	education_num	marital	occupation	relationship	race	sex	capital_gai
1	1	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
2	2	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
3	3	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
4	4	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
5	5	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0
6	6	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0

You should notice the index has been repeated. Drop this column.

In [3]:

```
library(dplyr)
adult <- select(adult,-X)
```

```
Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:
```

intersect, setdiff, setequal, union

Check the head,str, and summary of the data now.

In [4]:

```
head(adult)
```

Out[4]:

	age	type_employer	fnlwgt	education	education_num	marital	occupation	relationship	race	sex	capital_gain
1	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174
2	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0
3	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0
4	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0
5	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0
6	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0

In [5]:

```
str(adult)
```

```
'data.frame': 32561 obs. of  15 variables:
 $ age          : int  39 50 38 53 28 37 49 52 31 42 ...
 $ type_employer: Factor w/ 9 levels "?","Federal-gov",..: 8 7 5 5 5 5 5 7 5 5 ...
 $ fnlwgt       : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
 $ education    : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
 $ education_num: int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital      : Factor w/ 7 levels "Divorced","Married-AF-spouse",..: 5 3 1 3 3 3 4 3 5 3 ...
 $ occupation   : Factor w/ 15 levels "?","Adm-clerical",..: 2 5 7 7 11 5 9 5 11 5 ...
 $ relationship : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
 $ race         : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
 $ sex          : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
 $ capital_gain : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital_loss : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hr_per_week  : int  40 13 40 40 40 40 16 45 50 40 ...
 $ country      : Factor w/ 42 levels "?","Cambodia",..: 40 40 40 40 6 40 24 40 40 40 ...
 $ income       : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
```

In [6]:

```
summary(adult)
```

Out[6]:

```
      age                type_employer       fnlwgt
 Min.   :17.00   Private         :22696   Min.   :  12285
 1st Qu.:28.00   Self-emp-not-inc: 2541   1st Qu.: 117827
 Median :37.00   Local-gov       : 2093   Median : 178356
 Mean   :38.58   ?               : 1836   Mean   : 189778
 3rd Qu.:48.00   State-gov       : 1298   3rd Qu.: 237051
 Max.   :90.00   Self-emp-inc    : 1116   Max.   :1484705
                 (Other)         :  981
       education     education_num                    marital
 HS-grad     :10501   Min.   : 1.00   Divorced             : 4443
```

```
Some-college: 7291   1st Qu.: 9.00   Married-AF-spouse     :   23
Bachelors    : 5355   Median :10.00   Married-civ-spouse    :14976
Masters      : 1723   Mean   :10.08   Married-spouse-absent:  418
Assoc-voc    : 1382   3rd Qu.:12.00   Never-married         :10683
11th         : 1175   Max.   :16.00   Separated             : 1025
(Other)      : 5134                   Widowed               :  993
          occupation           relationship                 race
Prof-specialty :4140   Husband       :13193   Amer-Indian-Eskimo:  311
Craft-repair   :4099   Not-in-family : 8305   Asian-Pac-Islander: 1039
Exec-managerial:4066   Other-relative:  981   Black             : 3124
Adm-clerical   :3770   Own-child     : 5068   Other             :  271
Sales          :3650   Unmarried     : 3446   White             :27816
Other-service  :3295   Wife          : 1568
(Other)        :9541
     sex          capital_gain    capital_loss      hr_per_week
Female:10771   Min.   :    0   Min.   :   0.0   Min.   : 1.00
Male  :21790   1st Qu.:    0   1st Qu.:   0.0   1st Qu.:40.00
               Median :    0   Median :   0.0   Median :40.00
               Mean   : 1078   Mean   :  87.3   Mean   :40.44
               3rd Qu.:    0   3rd Qu.:   0.0   3rd Qu.:45.00
               Max.   :99999   Max.   :4356.0   Max.   :99.00

          country         income
United-States:29170   <=50K:24720
Mexico       :  643   >50K : 7841
?            :  583
Philippines  :  198
Germany      :  137
Canada       :  121
(Other)      : 1709
```

Data Cleaning

Notice that we have a lot of columns that are cateogrical factors, however a lot of these columns have too many factors than may be necessary. In this data cleaning section we'll try to clean these columns up by reducing the number of factors.

type_employer column

Use table() to check out the frequency of the type_employer column.

In [7]:

```
table(adult$type_employer)
```

Out[7]:

```
               ?      Federal-gov        Local-gov     Never-worked
            1836              960             2093                7
         Private     Self-emp-inc Self-emp-not-inc        State-gov
           22696             1116             2541             1298
     Without-pay
              14
```

How many Null values are there for type_employer? What are the two smallest groups?

In [8]:

```
# 1836 Null Values
#
# Never-worked and Without-pay
```

Combine these two smallest groups into a single group called "Unemployed". There are lots of ways to do this, so feel free to get creative. Hint: It may be helpful to convert these objects into character data types (as.character() and then use sapply with a custom function)

In [9]:

```
unemp <- function(job){
  job <- as.character(job)
  if (job=='Never-worked' | job=='Without-pay'){
```

```
        return('Unemployed')
    }else{
        return(job)
    }
}
```

In [10]:

```
adult$type_employer <- sapply(adult$type_employer,unemp)
```

In [11]:

```
table(adult$type_employer)
```

Out[11]:

```
           ?      Federal-gov        Local-gov          Private
        1836              960             2093            22696
    Self-emp-inc Self-emp-not-inc        State-gov       Unemployed
        1116             2541             1298               21
```

What other columns are suitable for combining? Combine State and Local gov jobs into a category called SL-gov and combine self-employed jobs into a category called self-emp.

In [12]:

```
group_emp <- function(job){
    if (job=='Local-gov' | job=='State-gov'){
        return('SL-gov')
    }else if (job=='Self-emp-inc' | job=='Self-emp-not-inc'){
        return('self-emp')
    }else{
        return(job)
    }
}
```

In [13]:

```
adult$type_employer <- sapply(adult$type_employer,group_emp)
```

In [14]:

```
table(adult$type_employer)
```

Out[14]:

```
          ? Federal-gov     Private    self-emp      SL-gov  Unemployed
       1836         960       22696        3657        3391          21
```

Marital Column

Use table() to look at the marital column

In [15]:

```
table(adult$marital)
```

Out[15]:

```
             Divorced     Married-AF-spouse    Married-civ-spouse
                 4443                    23                 14976
Married-spouse-absent         Never-married             Separated
                  418                 10683                  1025
              Widowed
                  993
```

Reduce this to three groups:

Reduce this to three groups:

- Married
- Not-Married
- Never-Married

In [16]:

```
group_marital <- function(mar){
    mar <- as.character(mar)

    # Not-Married
    if (mar=='Separated' | mar=='Divorced' | mar=='Widowed'){
        return('Not-Married')

    # Never-Married
    }else if(mar=='Never-married'){
        return(mar)

     #Married
    }else{
        return('Married')
    }
}
```

In [17]:

```
adult$marital <- sapply(adult$marital,group_marital)
table(adult$marital)
```

Out[17]:

```
      Married Never-married   Not-Married 
        15417         10683          6461 
```

Country Column

Check the country column using table()

In [18]:

```
table(adult$country)
```

Out[18]:

```
                         ?                   Cambodia 
                       583                         19 
                    Canada                      China 
                       121                         75 
                  Columbia                       Cuba 
                        59                         95 
        Dominican-Republic                    Ecuador 
                        70                         28 
               El-Salvador                    England 
                       106                         90 
                    France                    Germany 
                        29                        137 
                    Greece                  Guatemala 
                        29                         64 
                     Haiti         Holand-Netherlands 
                        44                          1 
                  Honduras                       Hong 
                        13                         20 
                   Hungary                      India 
                        13                        100 
                      Iran                    Ireland 
                        43                         24 
                     Italy                    Jamaica 
                        73                         81 
                     Japan                       Laos 
                        62                         18 
                    Mexico                  Nicaragua 
                       643                         34 
```

```
Outlying-US(Guam-USVI-etc)                  Peru
                14                     31
         Philippines                 Poland
               198                     60
           Portugal             Puerto-Rico
                37                    114
           Scotland                  South
                12                     80
             Taiwan                Thailand
                51                     18
     Trinadad&Tobago          United-States
                19                  29170
            Vietnam              Yugoslavia
                67                     16
```

Group these countries together however you see fit. You have flexibility here because there is no right/wrong way to do this, possibly group by continents. You should be able to reduce the number of groups here significantly though.

In [19]:

```r
levels(adult$country)
```

Out[19]:

'?' 'Cambodia' 'Canada' 'China' 'Columbia' 'Cuba' 'Dominican-Republic' 'Ecuador' 'El-Salvador' 'England' 'France' 'Germany' 'Greece' 'Guatemala' 'Haiti' 'Holand-Netherlands' 'Honduras' 'Hong' 'Hungary' 'India' 'Iran' 'Ireland' 'Italy' 'Jamaica' 'Japan' 'Laos' 'Mexico' 'Nicaragua' 'Outlying-US(Guam-USVI-etc)' 'Peru' 'Philippines' 'Poland' 'Portugal' 'Puerto-Rico' 'Scotland' 'South' 'Taiwan' 'Thailand' 'Trinadad&Tobago' 'United-States' 'Vietnam' 'Yugoslavia'

In [20]:

```r
Asia <- c('China','Hong','India','Iran','Cambodia','Japan', 'Laos' ,
        'Philippines' ,'Vietnam' ,'Taiwan', 'Thailand')

North.America <- c('Canada','United-States','Puerto-Rico' )

Europe <- c('England' ,'France', 'Germany' ,'Greece','Holand-Netherlands','Hungary',
          'Ireland','Italy','Poland','Portugal','Scotland','Yugoslavia')

Latin.and.South.America <- c('Columbia','Cuba','Dominican-Republic','Ecuador',
                             'El-Salvador','Guatemala','Haiti','Honduras',
                             'Mexico','Nicaragua','Outlying-US(Guam-USVI-etc)','Peru',
                            'Jamaica','Trinadad&Tobago')
Other <- c('South')
```

In [21]:

```r
group_country <- function(ctry){
    if (ctry %in% Asia){
        return('Asia')
    }else if (ctry %in% North.America){
        return('North.America')
    }else if (ctry %in% Europe){
        return('Europe')
    }else if (ctry %in% Latin.and.South.America){
        return('Latin.and.South.America')
    }else{
        return('Other')
    }
}
```

In [22]:

```r
adult$country <- sapply(adult$country,group_country)
```

Use table() to confirm the groupings

In [23]:

```
table(adult$country)
```

Out[23]:

```
          Asia             Europe Latin.and.South.America
           671               521                    1301
 North.America              Other
         29405               663
```

Check the str() of adult again. Make sure any of the columns we changed have factor levels with factor()

In [24]:

```
str(adult)
```

```
'data.frame':	32561 obs. of  15 variables:
 $ age          : int  39 50 38 53 28 37 49 52 31 42 ...
 $ type_employer: chr  "SL-gov" "self-emp" "Private" "Private" ...
 $ fnlwgt       : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
 $ education    : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
 $ education_num: int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital      : chr  "Never-married" "Married" "Not-Married" "Married" ...
 $ occupation   : Factor w/ 15 levels "?","Adm-clerical",..: 2 5 7 7 11 5 9 5 11 5 ...
 $ relationship : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
 $ race         : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
 $ sex          : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
 $ capital_gain : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital_loss : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hr_per_week  : int  40 13 40 40 40 40 16 45 50 40 ...
 $ country      : chr  "North.America" "North.America" "North.America" "North.America" ...
 $ income       : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
```

In [25]:

```
adult$type_employer <- sapply(adult$type_employer,factor)
adult$country <- sapply(adult$country,factor)
adult$marital <- sapply(adult$marital,factor)
```

You could have also done something like:

```
adult$type_employer <- factor(adult$type_employer)
```

In [26]:

```
str(adult)
```

```
'data.frame':	32561 obs. of  15 variables:
 $ age          : int  39 50 38 53 28 37 49 52 31 42 ...
 $ type_employer: Factor w/ 6 levels "SL-gov","self-emp",..: 1 2 3 3 3 3 3 2 3 3 ...
 $ fnlwgt       : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
 $ education    : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
 $ education_num: int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital      : Factor w/ 3 levels "Never-married",..: 1 2 3 2 2 2 2 2 1 2 ...
 $ occupation   : Factor w/ 15 levels "?","Adm-clerical",..: 2 5 7 7 11 5 9 5 11 5 ...
 $ relationship : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
 $ race         : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
 $ sex          : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
 $ capital_gain : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital_loss : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hr_per_week  : int  40 13 40 40 40 40 16 45 50 40 ...
 $ country      : Factor w/ 5 levels "North.America",..: 1 1 1 1 2 1 2 1 1 1 ...
 $ income       : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
```

We could still play around with education and occupation to try to reduce the number of factors for those columns, but let's go ahead and move on to dealing with the missing data. Feel free to group thos columns as well and see how they effect your model.

Missing Data

Notice how we have data that is missing.

Amelia

Install and load the Amelia package.

In [27]:

```
#install.packages('Amelia',repos = 'http://cran.us.r-project.org')
library(Amelia)
```

```
Loading required package: Rcpp
##
## Amelia II: Multiple Imputation
## (Version 1.7.4, built: 2015-12-05)
## Copyright (C) 2005-2016 James Honaker, Gary King and Matthew Blackwell
## Refer to http://gking.harvard.edu/amelia/ for more information
##
```

Convert any cell with a '?' or a ' ?' value to a NA value. Hint: is.na() may be useful here or you can also use brackets with a conditional statement. Refer to the solutions if you can't figure this step out.

In [28]:

```
adult[adult == '?'] <- NA
```

Using table() on a column with NA values should now not display those NA values, instead you'll just see 0 for ?. Optional: Refactor these columns (may take awhile). For example:

In [29]:

```
table(adult$type_employer)
```

Out[29]:

```
    SL-gov    self-emp     Private Federal-gov           ?  Unemployed 
      3391        3657       22696         960           0          21 
```

In [30]:

```
adult$type_employer <- sapply(adult$type_employer,factor)
adult$country <- sapply(adult$country,factor)
adult$marital <- sapply(adult$marital,factor)
adult$occupation <- sapply(adult$occupation,factor)
```

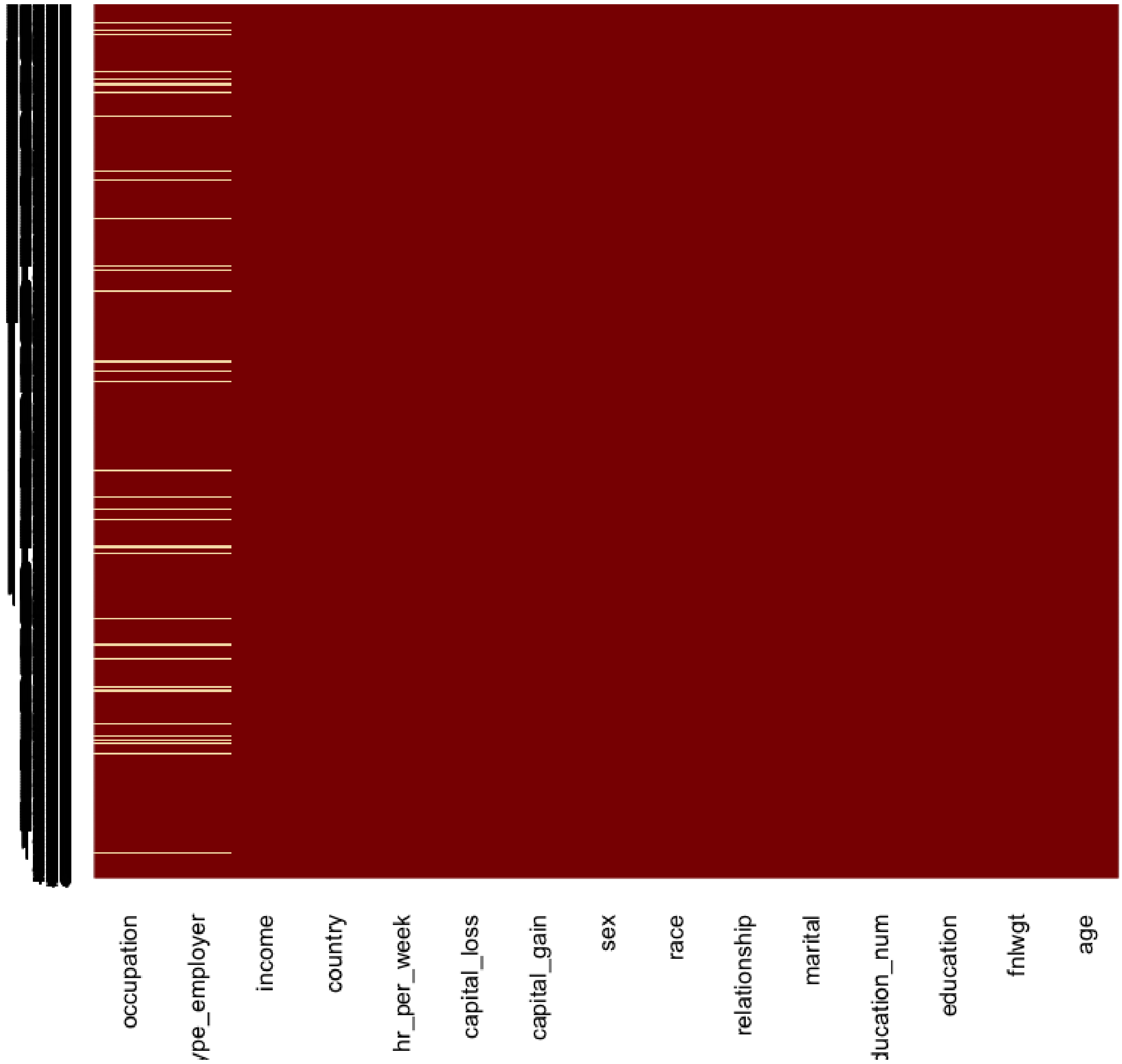
You could have also done something like:

```
adult$type_employer <- factor(adult$type_employer)
```

Play around with the missmap function from the Amelia package. Can you figure out what its doing and how to use it?

In [31]:

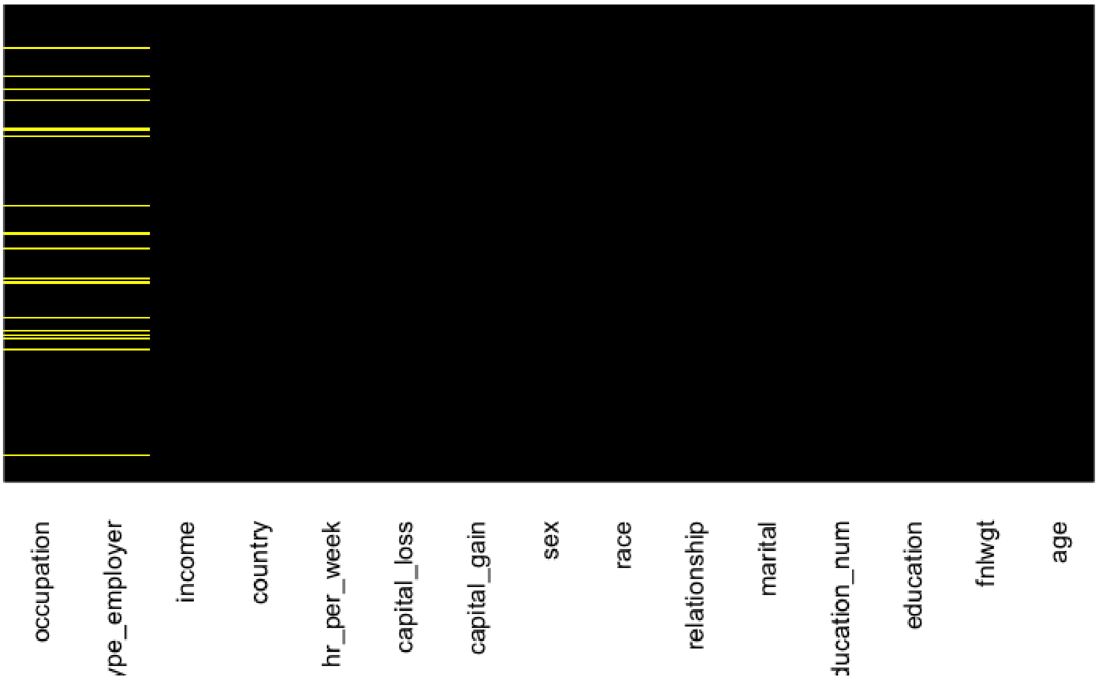
```
missmap(adult)
```



You should have noticed that using missmap(adult) is bascially a heatmap pointing out missing values (NA). This gives you a quick glance at how much data is missing, in this case, not a whole lot (relatively speaking). You probably also noticed that there is a bunch of y labels, get rid of them by running the command below. What is col=c('yellow','black') doing?

In [32]:

```
missmap(adult,y.at=c(1),y.labels = c(''),col=c('yellow','black'))
```



Use na.omit() to omit NA data from the adult data frame. Note, it really depends on the situation and your data to judge whether or not this is a good decision. You shouldn't always just drop NA values.

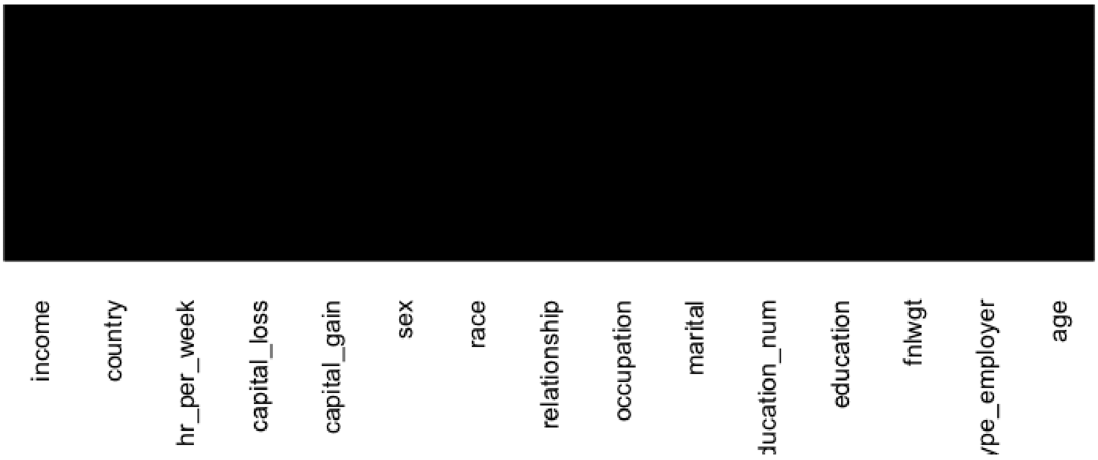
In [33]:

```
# May take awhile
adult <- na.omit(adult)
#str(adult)
```

Use missmap() to check that all the NA values were in fact dropped.

In [34]:

```
missmap(adult,y.at=c(1),y.labels = c(''),col=c('yellow','black'))
```



EDA

Although we've cleaned the data, we still have explored it using visualization.

Check the str() of the data.

In [35]:

```
str(adult)
```

```
'data.frame': 30718 obs. of  15 variables:
 $ age          : int  39 50 38 53 28 37 49 52 31 42 ...
 $ type_employer: Factor w/ 5 levels "SL-gov","self-emp",..: 1 2 3 3 3 3 3 2 3 3 ...
 $ fnlwgt       : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
 $ education    : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
 $ education_num: int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital      : Factor w/ 3 levels "Never-married",..: 1 2 3 2 2 2 2 2 1 2 ...
 $ occupation   : Factor w/ 14 levels "Adm-clerical",..: 1 2 3 3 4 2 5 2 4 2 ...
 $ relationship : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
 $ race         : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
 $ sex          : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
 $ capital_gain : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital_loss : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hr_per_week  : int  40 13 40 40 40 40 16 45 50 40 ...
 $ country      : Factor w/ 5 levels "North.America",..: 1 1 1 1 2 1 2 1 1 1 ...
 $ income       : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:1843] 28 62 70 78 107 129 150 155 161 188 ...
  .. ..- attr(*, "names")= chr [1:1843] "28" "62" "70" "78" ...
```

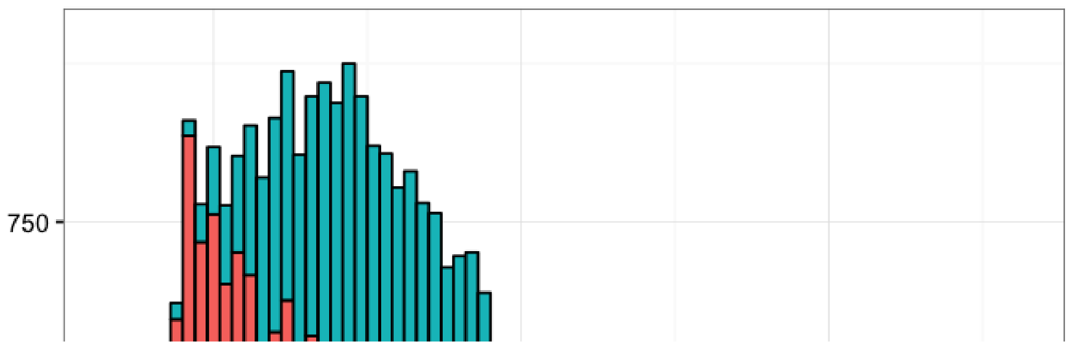
Use ggplot2 to create a histogram of ages, colored by income.

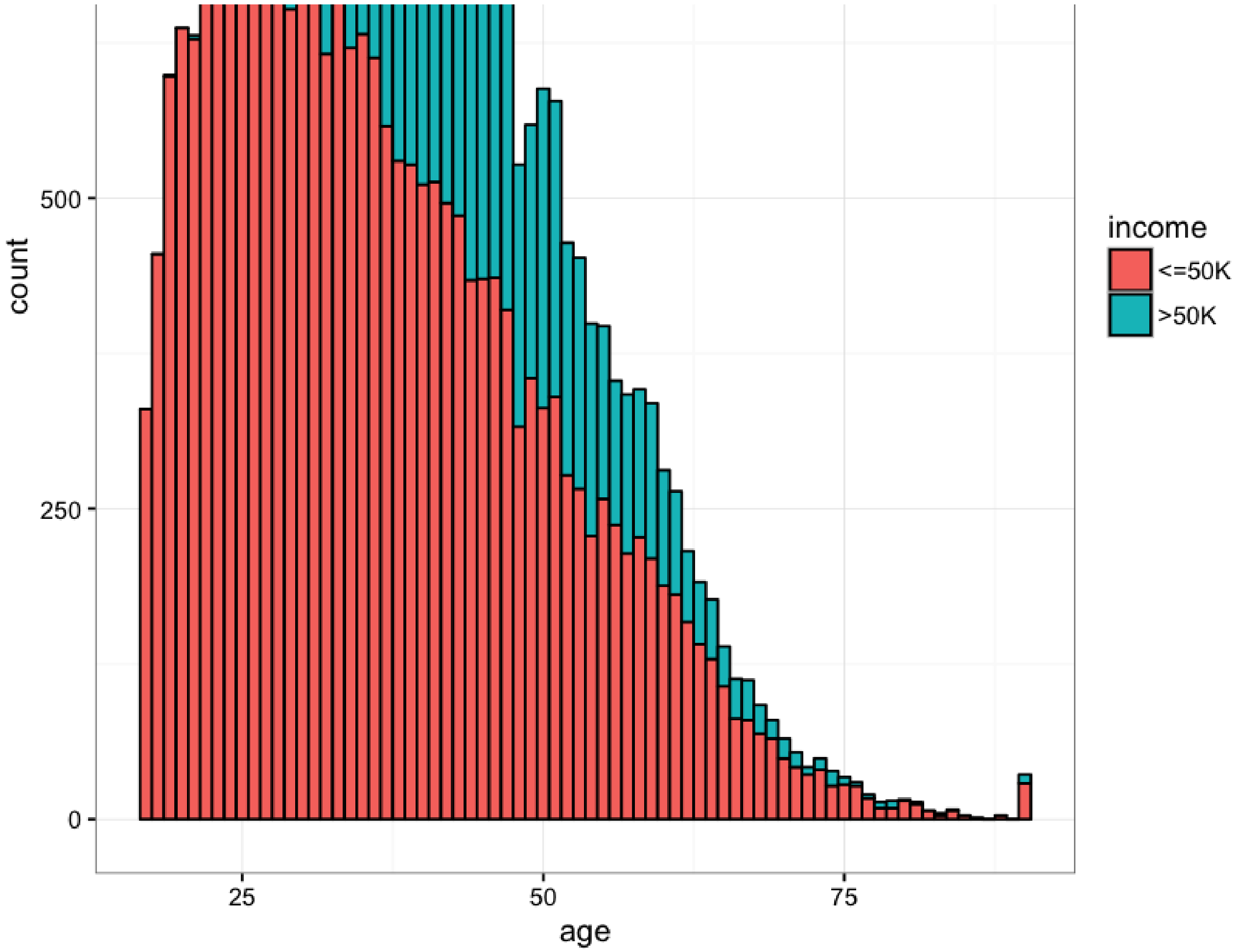
In [36]:

```
library(ggplot2)
library(dplyr)
```

In [37]:

```
ggplot(adult,aes(age)) + geom_histogram(aes(fill=income),color='black',binwidth=1) + theme_bw()
```



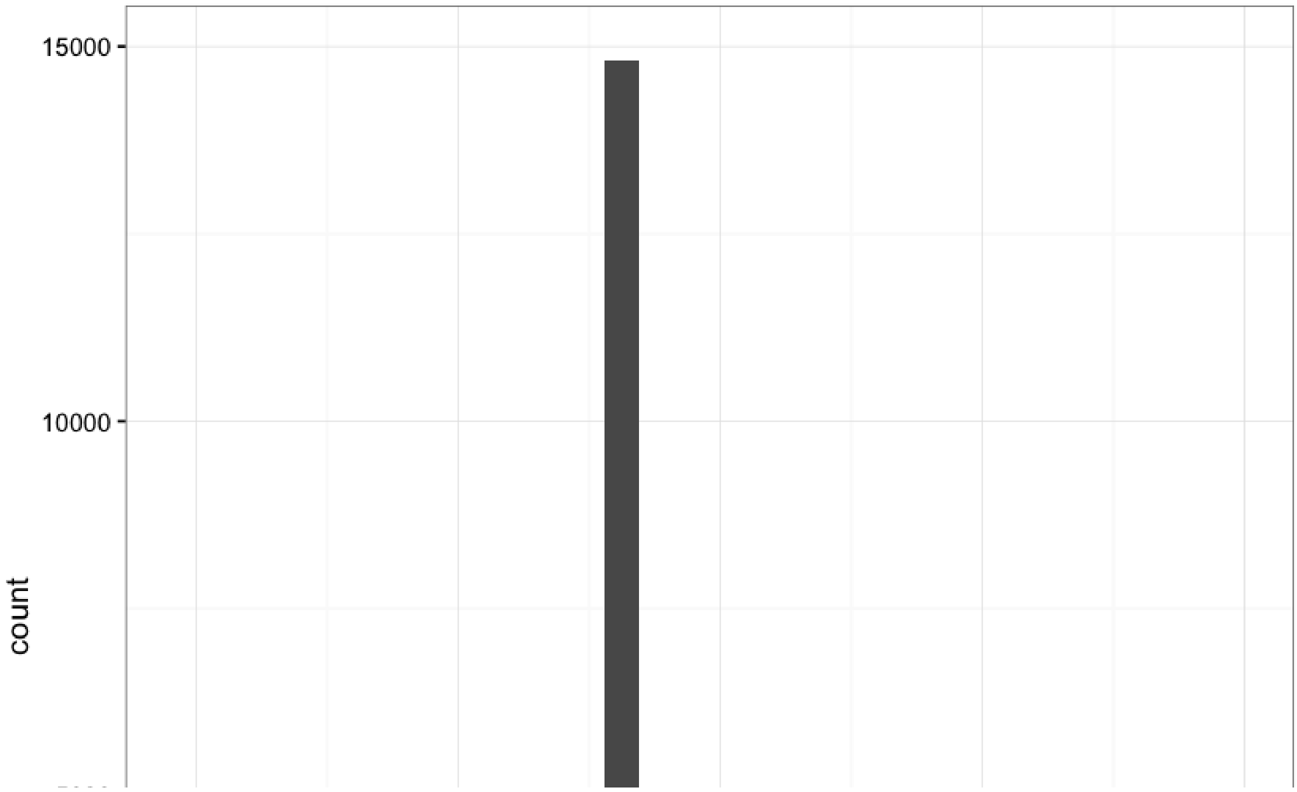


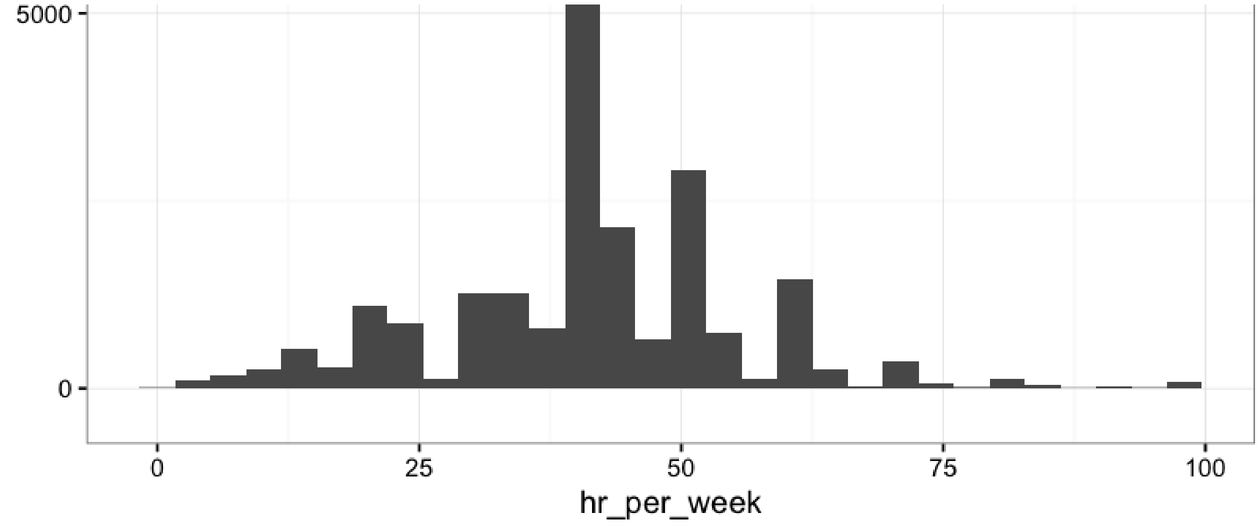
Plot a histogram of hours worked per week

In [38]:

```
ggplot(adult,aes(hr_per_week)) + geom_histogram() + theme_bw()
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.





Rename the country column to region column to better reflect the factor levels.

In [52]:

```
#Lots of ways to do this, could use dplyr as well
names(adult)[names(adult)=="country"] <- "region"
```

In [53]:

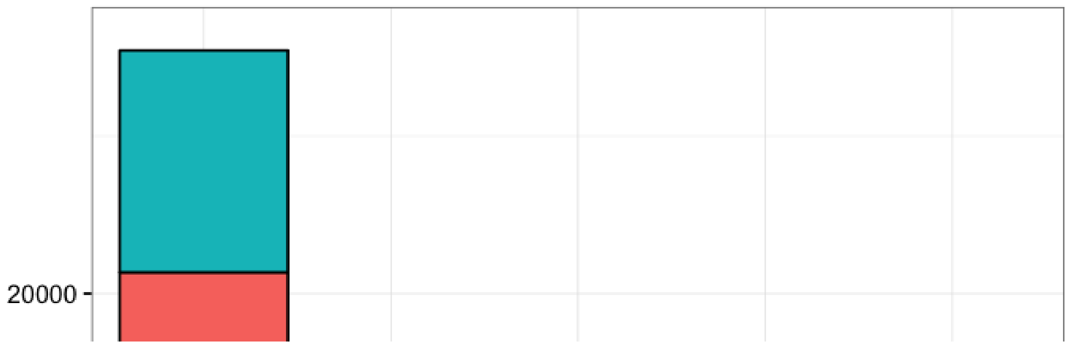
```
str(adult)
```

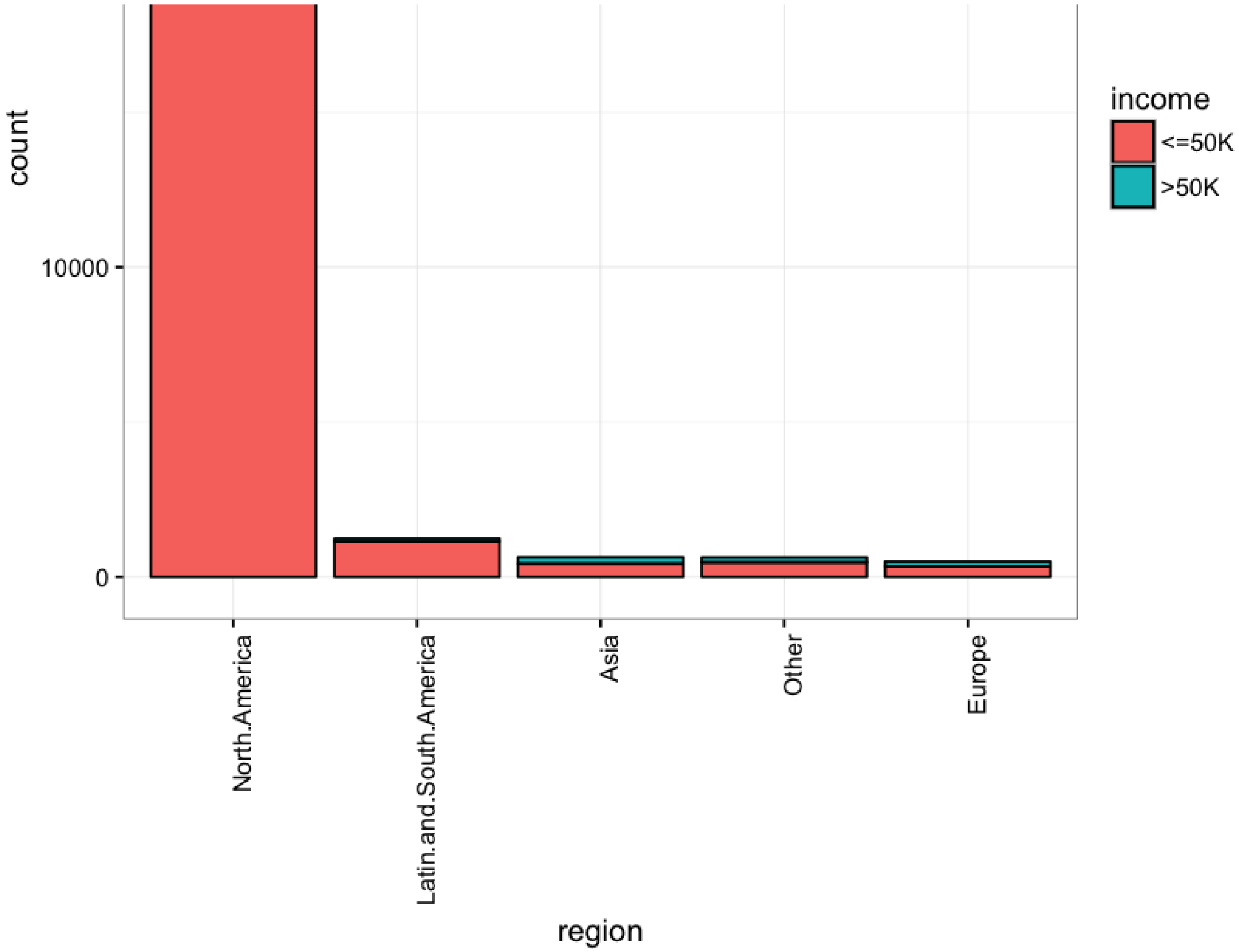
```
'data.frame':	30718 obs. of  15 variables:
 $ age          : int  39 50 38 53 28 37 49 52 31 42 ...
 $ type_employer: Factor w/ 5 levels "SL-gov","self-emp",..: 1 2 3 3 3 3 3 2 3 3 ...
 $ fnlwgt       : int  77516 83311 215646 234721 338409 284582 160187 209642 45781 159449 ...
 $ education    : Factor w/ 16 levels "10th","11th",..: 10 10 12 2 10 13 7 12 13 10 ...
 $ education_num: int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital      : Factor w/ 3 levels "Never-married",..: 1 2 3 2 2 2 2 2 1 2 ...
 $ occupation   : Factor w/ 14 levels "Adm-clerical",..: 1 2 3 3 4 2 5 2 4 2 ...
 $ relationship : Factor w/ 6 levels "Husband","Not-in-family",..: 2 1 2 1 6 6 2 1 2 1 ...
 $ race         : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5 5 5 3 3 5 3 5 5 5 ...
 $ sex          : Factor w/ 2 levels "Female","Male": 2 2 2 2 1 1 1 2 1 2 ...
 $ capital_gain : int  2174 0 0 0 0 0 0 0 14084 5178 ...
 $ capital_loss : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hr_per_week  : int  40 13 40 40 40 40 16 45 50 40 ...
 $ region       : Factor w/ 5 levels "North.America",..: 1 1 1 1 2 1 2 1 1 1 ...
 $ income       : Factor w/ 2 levels "<=50K",">50K": 1 1 1 1 1 1 1 2 2 2 ...
 - attr(*, "na.action")=Class 'omit'  Named int [1:1843] 28 62 70 78 107 129 150 155 161 188 ...
  .. ..- attr(*, "names")= chr [1:1843] "28" "62" "70" "78" ...
```

Create a barplot of region with the fill color defined by income class. Optional: Figure out how rotate the x axis text for readability

In [54]:

```
ggplot(adult,aes(region)) + geom_bar(aes(fill=income),color='black')+theme_bw()+
theme(axis.text.x = element_text(angle = 90, hjust = 1))
```





Building a Model

Now it's time to build a model to classify people into two groups: Above or Below 50k in Salary.

Logistic Regression

Refer to the Lecture or ISLR if you are fuzzy on any of this.

Logistic Regression is a type of classification model. In classification models, we attempt to predict the outcome of categorical dependent variables, using one or more independent variables. The independent variables can be either categorical or numerical.

Logistic regression is based on the logistic function, which always takes values between 0 and 1. Replacing the dependent variable of the logistic function with a linear combination of dependent variables we intend to use for regression, we arrive at the formula for logistic regression.

Take a quick look at the head() of adult to make sure we have a good overview before going into building the model!

In [55]:

```
head(adult)
```

Out[55]:

	age	type_employer	fnlwgt	education	education_num	marital	occupation	relationship	race	sex	capital_gain	c
1	39	SL-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0
2	50	self-emp	83311	Bachelors	13	Married	Exec-managerial	Husband	White	Male	0	0
3	38	Private	215646	HS-grad	9	Not-Married	Handlers-cleaners	Not-in-family	White	Male	0	0
4	53	Private	234721	11th	7	Married	Handlers-	Husband	Black	Male	0	0

cleaners

	age	type_employer	fnlwgt	education	education_num	marital	occupation	relationship	race	sex	capital_gain	c
5	28	Private	338409	Bachelors	13	Married	Prof-specialty	Wife	Black	Female	0	0
6	37	Private	284582	Masters	14	Married	Exec-managerial	Wife	White	Female	0	0

Train Test Split

Split the data into a train and test set using the caTools library as done in previous lectures. Reference previous solutions notebooks if you need a refresher.

In [59]:

```
# Import Library
library(caTools)

# Set a random see so your "random" results are the same as this notebook
set.seed(101)

# Split up the sample, basically randomly assigns a booleans to a new column "sample"
sample <- sample.split(adult$income, SplitRatio = 0.70) # SplitRatio = percent of sample==TRUE

# Training Data
train = subset(adult, sample == TRUE)

# Testing Data
test = subset(adult, sample == FALSE)
```

Training the Model

Explore the glm() function with help(glm). Read through the documentation.

In [60]:

```
help(glm)
```

Out[60]:

glm {stats}	R Documentation

Fitting Generalized Linear Models

Description

glm is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, contrasts = NULL, ...)

glm.fit(x, y, weights = rep(1, nobs),
        start = NULL, etastart = NULL, mustart = NULL,
        offset = rep(0, nobs), family = gaussian(),
        control = list(), intercept = TRUE)

## S3 method for class 'glm'
weights(object, type = c("prior", "working"), ...)
```

Arguments

	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
family	a description of the error distribution and link function to be used in the model. For glm this can be a character string naming a family function, a family function or the result of a call to a family function. For glm.fit only the third option is supported. (See family for details of family functions.)
data	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which glm is called.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
start	starting values for the parameters in the linear predictor.
etastart	starting values for the linear predictor.
mustart	starting values for the vector of means.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
control	a list of parameters for controlling the fitting process. For glm.fit this is passed to glm.control.
model	a logical value indicating whether *model frame* should be included as a component of the returned value.
method	the method to be used in fitting the model. The default method "glm.fit" uses iteratively reweighted least squares (IWLS): the alternative "model.frame" returns the model frame and does no fitting. User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as glm.fit. If specified as a character string it is looked up from within the stats namespace.
x, y	For glm: logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value. For glm.fit: x is a design matrix of dimension n * p, and y is a vector of observations of length n.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.
intercept	logical. Should an intercept be included in the *null* model?
object	an object inheriting from class "glm".
type	character, partial matching allowed. Type of weights to extract from the fitted model object. Can be abbreviated.
...	For glm: arguments to be used to form the default control argument if it is not supplied directly. For weights: further arguments passed to or from other methods.

Details

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. For binomial and quasibinomial families the response can also be specified as a factor (when the first level denotes failure and all others success) or as a two-column matrix with the columns giving the numbers of successes and failures. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with any duplicates removed.

A specification of the form first:second indicates the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the *cross* of first and second. This is the same as first + second + first:second.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers *w_i*, that each response *y_i* is the mean of *w_i* unit-weight observations. For a binomial GLM prior weights are used to give the number of trials when the response is the proportion of successes: they would rarely be used for a Poisson GLM.

glm.fit is the workhorse function: it is not normally called directly but can be more efficient where the response vector, design matrix glm.fit is the workhorse function: it is not normally called directly but can be more efficient where the response vector, design matrix and family have already been calculated.

If more than one of etastart, start and mustart is specified, the first in the list will be used. It is often advisable to supply starting values for a quasi family, and also for families with unusual links such as gaussian("log").

All of weights, subset, offset, etastart and mustart are evaluated in the same way as variables in formula, that is first in data and then in the environment of formula.

For the background to warning messages about 'fitted probabilities numerically 0 or 1 occurred' for binomial GLMs, see Venables & Ripley (2002, pp. 197–8).

Value

glm returns an object of class inheriting from "glm" which inherits from the class "lm". See later in this section. If a non-standard method is used, the object will also inherit from the class (if any) returned by that function.

The function summary (i.e., summary.glm) can be used to obtain or print a summary of the results and the function anova (i.e., anova.glm) to produce an analysis of variance table.

The generic accessor functions coefficients, effects, fitted.values and residuals can be used to extract various useful features of the value returned by glm.

weights extracts a vector of weights, one for each case in the fit (after subsetting and na.action).

An object of class "glm" is a list containing at least the following components:

coefficients	a named vector of coefficients
residuals	the *working* residuals, that is the residuals in the final iteration of the IWLS fit. Since cases with zero weights are omitted, their working residuals are NA.
fitted.values	the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
rank	the numeric rank of the fitted linear model.
family	the family object used.
linear.predictors	the linear fit on link scale.
deviance	up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero.
aic	A version of Akaike's *An Information Criterion*, minus twice the maximized log-likelihood plus twice the number of parameters, computed by the aic component of the family. For binomial and Poison families the dispersion is fixed at one and the number of parameters is the number of coefficients. For gaussian, Gamma and inverse gaussian families the dispersion is estimated from the residual deviance, and the number of parameters is the number of coefficients plus one. For a gaussian family the MLE of the dispersion is used so this is a valid value of AIC, but for Gamma and inverse gaussian families it is not. For families fitted by quasi-likelihood the value is NA.
null.deviance	The deviance for the null model, comparable with deviance. The null model will include the offset, and an intercept if there is one in the model. Note that this will be incorrect if the link function depends on the data other than through the fitted mean: specify a zero offset to force a correct calculation.
iter	the number of iterations of IWLS used.
weights	the *working* weights, that is the weights in the final iteration of the IWLS fit.
prior.weights	the weights initially supplied, a vector of 1s if none were.
df.residual	the residual degrees of freedom.
df.null	the residual degrees of freedom for the null model.
y	if requested (the default) the y vector used. (It is a vector even for a binomial model.)
x	if requested, the model matrix.
model	if requested (the default), the model frame.
converged	logical. Was the IWLS algorithm judged to have converged?
boundary	logical. Is the fitted value on the boundary of the attainable values?
call	the matched call.
formula	the formula supplied.
terms	the terms object used.
data	the data argument.

data	the data argument.
offset	the offset vector used.
control	the value of the control argument used.
method	the name of the fitter function used, currently always "glm.fit".
contrasts	(where relevant) the contrasts used.
xlevels	(where relevant) a record of the levels of the factors used in fitting.
na.action	(where relevant) information returned by model.frame on the special handling of NAs.

In addition, non-empty fits will have components qr, R and effects relating to the final weighted linear fit.

Objects of class "glm" are normally of class c("glm",
"lm"), that is inherit from class "lm", and well-designed methods for class "lm" will be applied to the weighted linear model at the final iteration of IWLS. However, care is needed, as extractor functions for class "glm" such as residuals and weights do **not** just pick out the component of the fit with the same name.

If a binomial glm model was specified by giving a two-column response, the weights returned by prior.weights are the total numbers of cases (factored by the supplied case weights) and the component y of the result is the proportion of successes.

Fitting functions

The argument method serves two purposes. One is to allow the model frame to be recreated with no fitting. The other is to allow the default fitting function glm.fit to be replaced by a function which takes the same arguments and uses a different fitting algorithm. If glm.fit is supplied as a character string it is used to search for a function of that name, starting in the stats namespace.

The class of the object return by the fitter (if any) will be prepended to the class returned by glm.

Author(s)

The original **R** implementation of glm was written by Simon Davies working for Ross Ihaka at the University of Auckland, but has since been extensively re-written by members of the R Core team.

The design was inspired by the S function of the same name described in Hastie & Pregibon (1992).

References

Dobson, A. J. (1990) *An Introduction to Generalized Linear Models.* London: Chapman and Hall.

Hastie, T. J. and Pregibon, D. (1992) *Generalized linear models.* Chapter 6 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

McCullagh P. and Nelder, J. A. (1989) *Generalized Linear Models.* London: Chapman and Hall.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* New York: Springer.

See Also

anova.glm, summary.glm, etc. for glm methods, and the generic functions anova, summary, effects, fitted.values, and residuals.

lm for non-generalized *linear* models (which SAS calls GLMs, for 'general' linear models).

loglin and loglm (package [MASS]) for fitting log-linear models (which binomial and Poisson GLMs are) to contingency tables.

bigglm in package [biglm] for an alternative way to fit GLMs to large datasets (especially those with many cases).

esoph, infert and predict.glm have examples of fitting binomial glms.

Examples

```
## Dobson (1990) Page 93: Randomized Controlled Trial :
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
anova(glm.D93)
summary(glm.D93)

## an example with offsets from Venables & Ripley (2002, p.189)
```

an example with offsets from Venables & Ripley (2002, p.189)
utils::data(anorexia, package = "MASS")

anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                family = gaussian, data = anorexia)
summary(anorex.1)

# A Gamma example, from McCullagh & Nelder (1989, pp. 300-2)
clotting <- data.frame(
    u = c(5,10,15,20,30,40,60,80,100),
    lot1 = c(118,58,42,35,27,25,21,19,18),
    lot2 = c(69,35,26,21,18,16,13,12,12))
summary(glm(lot1 ~ log(u), data = clotting, family = Gamma))
summary(glm(lot2 ~ log(u), data = clotting, family = Gamma))

Not run:
for an example of the use of a terms object as a formula
demo(glm.vr)

End(Not run)

[Package *stats* version 3.2.2]

Use all the features to train a glm() model on the training data set, pass the argument family=binomial(logit) into the glm function.

In [61]:

```
model = glm(income ~ ., family = binomial(logit), data = train)
```

```
Warning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

If you get a warning, this just means that the model may have guessed the probability of a class with a 0% or 100% chance of occuring.

Check the model summary

In [62]:

```
summary(model)
```

Out[62]:

```
Call:
glm(formula = income ~ ., family = binomial(logit), data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.1163  -0.5172  -0.1965   0.0000   3.6235

Coefficients: (1 not defined because of singularities)
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)              -7.364e+00  4.245e-01 -17.346  < 2e-16 ***
age                       2.534e-02  2.007e-03  12.627  < 2e-16 ***
type_employerself-emp     7.501e-03  8.999e-02   0.083 0.933571
type_employerPrivate      2.371e-01  7.321e-02   3.239 0.001198 **
type_employerFederal-gov  6.835e-01  1.266e-01   5.399 6.71e-08 ***
type_employerUnemployed  -1.346e+01  3.688e+02  -0.036 0.970888
fnlwgt                    5.424e-07  2.085e-07   2.601 0.009291 **
education11th             2.094e-01  2.570e-01   0.814 0.415384
education12th             3.925e-01  3.410e-01   1.151 0.249612
education1st-4th         -4.590e-01  6.067e-01  -0.757 0.449323
education5th-6th         -8.009e-02  3.980e-01  -0.201 0.840503
education7th-8th         -4.991e-01  2.880e-01  -1.733 0.083096 .
```

```
education9th              -1.229e-02  3.191e-01  -0.038 0.969292
educationAssoc-acdm        1.250e+00  2.165e-01   5.775 7.70e-09 ***
educationAssoc-voc         1.452e+00  2.084e-01   6.970 3.17e-12 ***
educationBachelors         2.003e+00  1.938e-01  10.337  < 2e-16 ***
educationDoctorate         2.874e+00  2.636e-01  10.902  < 2e-16 ***
educationHS-grad           8.359e-01  1.888e-01   4.426 9.58e-06 ***
educationMasters           2.347e+00  2.063e-01  11.374  < 2e-16 ***
educationPreschool        -1.879e+01  1.645e+02  -0.114 0.909053
educationProf-school       2.797e+00  2.468e-01  11.337  < 2e-16 ***
educationSome-college      1.203e+00  1.915e-01   6.283 3.33e-10 ***
education_num                     NA         NA      NA       NA
maritalMarried             1.280e+00  1.943e-01   6.588 4.45e-11 ***
maritalNot-Married         5.435e-01  9.953e-02   5.460 4.75e-08 ***
occupationExec-managerial  7.689e-01  9.095e-02   8.453  < 2e-16 ***
occupationHandlers-cleaners -7.944e-01  1.726e-01  -4.603 4.17e-06 ***
occupationProf-specialty   4.957e-01  9.626e-02   5.149 2.62e-07 ***
occupationOther-service   -8.248e-01  1.386e-01  -5.952 2.65e-09 ***
occupationSales            2.896e-01  9.749e-02   2.971 0.002972 **
occupationCraft-repair     4.151e-02  9.483e-02   0.438 0.661616
occupationTransport-moving -1.114e-01  1.189e-01  -0.937 0.348928
occupationFarming-fishing -1.120e+00  1.619e-01  -6.920 4.52e-12 ***
occupationMachine-op-inspct -2.194e-01  1.203e-01  -1.824 0.068080 .
occupationTech-support     6.829e-01  1.325e-01   5.153 2.56e-07 ***
occupationProtective-serv  6.029e-01  1.491e-01   4.044 5.24e-05 ***
occupationArmed-Forces    -6.252e-01  1.844e+00  -0.339 0.734504
occupationPriv-house-serv -3.600e+00  1.938e+00  -1.858 0.063179 .
relationshipNot-in-family -8.661e-01  1.907e-01  -4.541 5.60e-06 ***
relationshipOther-relative -1.086e+00  2.546e-01  -4.268 1.97e-05 ***
relationshipOwn-child     -1.797e+00  2.357e-01  -7.625 2.45e-14 ***
relationshipUnmarried     -1.031e+00  2.154e-01  -4.784 1.72e-06 ***
relationshipWife           1.476e+00  1.235e-01  11.949  < 2e-16 ***
raceAsian-Pac-Islander     6.073e-01  3.206e-01   1.894 0.058243 .
raceBlack                  4.528e-01  2.847e-01   1.590 0.111800
raceOther                  4.135e-02  4.217e-01   0.098 0.921902
raceWhite                  6.595e-01  2.711e-01   2.432 0.014997 *
sexMale                    8.855e-01  9.378e-02   9.442  < 2e-16 ***
capital_gain               3.192e-04  1.273e-05  25.076  < 2e-16 ***
capital_loss               6.549e-04  4.561e-05  14.358  < 2e-16 ***
hr_per_week                2.906e-02  1.987e-03  14.623  < 2e-16 ***
regionLatin.and.South.America -5.925e-01  1.595e-01  -3.714 0.000204 ***
regionAsia                -6.475e-02  2.044e-01  -0.317 0.751446
regionOther               -4.300e-01  1.651e-01  -2.604 0.009206 **
regionEurope               4.404e-02  1.552e-01   0.284 0.776660
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 24138  on 21502  degrees of freedom
Residual deviance: 14004  on 21449  degrees of freedom
AIC: 14112

Number of Fisher Scoring iterations: 14
```

We have still a lot of features! Some important, some not so much. R comes with an *awesome* function called step(). The step() function iteratively tries to remove predictor variables from the model in an attempt to delete variables that do not significantly add to the fit. How does it do this? It uses [AIC](). Read the wikipedia page for AIC if you want to further understand this, you can also check out help(step). This level of statistics is outside the scope of this project assignment so let's keep moving along

In [63]:

```
help(step)
```

Out[63]:

step {stats}	R Documentation

Choose a model by AIC in a Stepwise Algorithm

Description

Select a formula-based model by AIC.

Usage

step(object, scope, scale = 0,
    direction = c("both", "backward", "forward"),
    trace = 1, keep = NULL, steps = 1000, k = 2, ...)

Arguments

object	an object representing a model of an appropriate class (mainly "lm" and "glm"). This is used as the initial model in the stepwise search.
scope	defines the range of models examined in the stepwise search. This should be either a single formula, or a list containing components upper and lower, both formulae. See the details for how to specify the formulae and how they are used.
scale	used in the definition of the AIC statistic for selecting the models, currently only for lm, aov and glm models. The default value, 0, indicates the scale should be estimated: see extractAIC.
direction	the mode of stepwise search, can be one of "both", "backward", or "forward", with a default of "both". If the scope argument is missing the default for direction is "backward". Values can be abbreviated.
trace	if positive, information is printed during the running of step. Larger values may give more detailed information.
keep	a filter function whose input is a fitted model object and the associated AIC statistic, and whose output is arbitrary. Typically keep will select a subset of the components of the object and return them. The default is not to keep anything.
steps	the maximum number of steps to be considered. The default is 1000 (essentially as many as required). It is typically used to stop the process early.
k	the multiple of the number of degrees of freedom used for the penalty. Only k = 2 gives the genuine AIC: k = log(n) is sometimes referred to as BIC or SBC.
...	any additional arguments to extractAIC.

Details

step uses add1 and drop1 repeatedly; it will work for any method for which they work, and that is determined by having a valid method for extractAIC. When the additive constant can be chosen so that AIC is equal to Mallows' *Cp*, this is done and the tables are labelled appropriately.

The set of models searched is determined by the scope argument. The right-hand-side of its lower component is always included in the model, and right-hand-side of the model is included in the upper component. If scope is a single formula, it specifies the upper component, and the lower model is empty. If scope is missing, the initial model is used as the upper model.

Models specified by scope can be templates to update object as used by update.formula. So using . in a scope formula means 'what is already there', with .^2 indicating all interactions of existing terms.

There is a potential problem in using glm fits with a variable scale, as in that case the deviance is not simply related to the maximized log-likelihood. The "glm" method for function extractAIC makes the appropriate adjustment for a gaussian family, but may need to be amended for other cases. (The binomial and poisson families have fixed scale by default and do not correspond to a particular maximum-likelihood problem for variable scale.)

Value

the stepwise-selected model is returned, with up to two additional components. There is an "anova" component corresponding to the steps taken in the search, as well as a "keep" component if the keep= argument was supplied in the call. The "Resid. Dev" column of the analysis of deviance table refers to a constant minus twice the maximized log likelihood: it will be a deviance only in cases where a saturated model is well-defined (thus excluding lm, aov and survreg fits, for example).

Warning

The model fitting must apply the models to the same dataset. This may be a problem if there are missing values and **R**'s default of na.action = na.omit is used. We suggest you remove the missing values first.

Calls to the function nobs are used to check that the number of observations involved in the fitting process remains unchanged.

Note

This function differs considerably from the function in S, which uses a number of approximations and does not in general compute the correct AIC.

This is a minimal implementation. Use stepAIC in package MASS for a wider range of object classes.

Author(s)

B. D. Ripley: step is a slightly simplified version of stepAIC in package MASS (Venables & Ripley, 2002 and earlier editions).

The idea of a step function follows that described in Hastie & Pregibon (1992); but the implementation in **R** is more general.

References

Hastie, T. J. and Pregibon, D. (1992) *Generalized linear models.* Chapter 6 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S.* New York: Springer (4th ed).

See Also

stepAIC in MASS, add1, drop1

Examples

```
## following on from example(lm)

step(lm.D9)

summary(lm1 <- lm(Fertility ~ ., data = swiss))
slm1 <- step(lm1)
summary(slm1)
slm1$anova
```

[Package *stats* version 3.2.2]

Use new.model <- step(your.model.name) to use the step() function to create a new model.

In [65]:

```
new.step.model <- step(model)
```

```
Start:  AIC=14112.05
income ~ age + type_employer + fnlwgt + education + education_num +
    marital + occupation + relationship + race + sex + capital_gain +
    capital_loss + hr_per_week + region
```

```
Warning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
Step:  AIC=14112.05
income ~ age + type_employer + fnlwgt + education + marital +
    occupation + relationship + race + sex + capital_gain + capital_loss +
    hr_per_week + region
```

```
Warning message:
```

```
Warning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurredWarning message:
: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
               Df Deviance   AIC
<none>              14004 14112
- fnlwgt        1    14011 14117
- race          4    14019 14119
- region        4    14026 14126
- type_employer 4    14050 14150
- marital       2    14060 14164
- sex           1    14097 14203
- age           1    14165 14271
- capital_loss  1    14217 14323
- hr_per_week   1    14222 14328
- relationship  5    14288 14386
- occupation   13    14444 14526
- education    15    14718 14796
- capital_gain  1    15248 15354
```

You should get a bunch of messages informing you of the process. Check the new.model by using summary()

In [66]:

```
summary(new.step.model)
```

Out[66]:

```
Call:
glm(formula = income ~ age + type_employer + fnlwgt + education + 
    marital + occupation + relationship + race + sex + capital_gain + 
    capital_loss + hr_per_week + region, family = binomial(logit), 
    data = train)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-5.1163  -0.5172  -0.1965   0.0000   3.6235  

Coefficients:
                            Estimate Std. Error z value Pr(>|z|)    
(Intercept)               -7.364e+00  4.245e-01 -17.346  < 2e-16 ***
age                        2.534e-02  2.007e-03  12.627  < 2e-16 ***
type_employerself-emp      7.501e-03  8.999e-02   0.083 0.933571    
type_employerPrivate       2.371e-01  7.321e-02   3.239 0.001198 ** 
type_employerFederal-gov   6.835e-01  1.266e-01   5.399 6.71e-08 ***
type_employerUnemployed   -1.346e+01  3.688e+02  -0.036 0.970888    
fnlwgt                     5.424e-07  2.085e-07   2.601 0.009291 ** 
education11th              2.094e-01  2.570e-01   0.814 0.415384    
education12th              3.925e-01  3.410e-01   1.151 0.249612    
education1st-4th          -4.590e-01  6.067e-01  -0.757 0.449323    
education5th-6th          -8.009e-02  3.980e-01  -0.201 0.840503    
education7th-8th          -4.991e-01  2.880e-01  -1.733 0.083096 .  
education9th              -1.229e-02  3.191e-01  -0.038 0.969292    
educationAssoc-acdm        1.250e+00  2.165e-01   5.775 7.70e-09 ***
educationAssoc-voc         1.452e+00  2.084e-01   6.970 3.17e-12 ***
educationBachelors         2.003e+00  1.938e-01  10.337  < 2e-16 ***
educationDoctorate         2.874e+00  2.636e-01  10.902  < 2e-16 ***
educationHS-grad           8.359e-01  1.888e-01   4.426 9.58e-06 ***
educationMasters           2.347e+00  2.063e-01  11.374  < 2e-16 ***
educationPreschool        -1.879e+01  1.645e+02  -0.114 0.909053    
educationProf-school       2.797e+00  2.468e-01  11.337  < 2e-16 ***
educationSome-college      1.203e+00  1.915e-01   6.283 3.33e-10 ***
maritalMarried             1.280e+00  1.943e-01   6.588 4.45e-11 ***
maritalNot-Married         5.435e-01  9.953e-02   5.460 4.75e-08 ***
occupationExec-managerial  7.689e-01  9.095e-02   8.453  < 2e-16 ***
```

```
occupationHandlers-cleaners  -7.944e-01  1.726e-01  -4.603 4.17e-06 ***
occupationProf-specialty      4.957e-01  9.626e-02   5.149 2.62e-07 ***
occupationOther-service      -8.248e-01  1.386e-01  -5.952 2.65e-09 ***
occupationSales               2.896e-01  9.749e-02   2.971 0.002972 **
occupationCraft-repair        4.151e-02  9.483e-02   0.438 0.661616
occupationTransport-moving   -1.114e-01  1.189e-01  -0.937 0.348928
occupationFarming-fishing    -1.120e+00  1.619e-01  -6.920 4.52e-12 ***
occupationMachine-op-inspct  -2.194e-01  1.203e-01  -1.824 0.068080 .
occupationTech-support        6.829e-01  1.325e-01   5.153 2.56e-07 ***
occupationProtective-serv     6.029e-01  1.491e-01   4.044 5.24e-05 ***
occupationArmed-Forces       -6.252e-01  1.844e+00  -0.339 0.734504
occupationPriv-house-serv    -3.600e+00  1.938e+00  -1.858 0.063179 .
relationshipNot-in-family    -8.661e-01  1.907e-01  -4.541 5.60e-06 ***
relationshipOther-relative   -1.086e+00  2.546e-01  -4.268 1.97e-05 ***
relationshipOwn-child        -1.797e+00  2.357e-01  -7.625 2.45e-14 ***
relationshipUnmarried        -1.031e+00  2.154e-01  -4.784 1.72e-06 ***
relationshipWife              1.476e+00  1.235e-01  11.949  < 2e-16 ***
raceAsian-Pac-Islander        6.073e-01  3.206e-01   1.894 0.058243 .
raceBlack                     4.528e-01  2.847e-01   1.590 0.111800
raceOther                     4.135e-02  4.217e-01   0.098 0.921902
raceWhite                     6.595e-01  2.711e-01   2.432 0.014997 *
sexMale                       8.855e-01  9.378e-02   9.442  < 2e-16 ***
capital_gain                  3.192e-04  1.273e-05  25.076  < 2e-16 ***
capital_loss                  6.549e-04  4.561e-05  14.358  < 2e-16 ***
hr_per_week                   2.906e-02  1.987e-03  14.623  < 2e-16 ***
regionLatin.and.South.America -5.925e-01  1.595e-01  -3.714 0.000204 ***
regionAsia                   -6.475e-02  2.044e-01  -0.317 0.751446
regionOther                  -4.300e-01  1.651e-01  -2.604 0.009206 **
regionEurope                  4.404e-02  1.552e-01   0.284 0.776660
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 24138  on 21502  degrees of freedom
Residual deviance: 14004  on 21449  degrees of freedom
AIC: 14112

Number of Fisher Scoring iterations: 14
```

You should have noticed that the step() function kept all the features used previously! While we used the AIC criteria to compare models, there are other criteria we could have used. If you want you can try reading about the variable inflation factor (VIF) and vif() function to explore other options for comparison criteria. In the meantime let's continue on and see how well our model performed against the test set.

[Review what a confusion matrix is on wikipedia.](#)

Create a confusion matrix using the predict function with type='response' as an argument inside of that function.

In [74]:

```
test$predicted.income = predict(model, newdata=test, type="response")

table(test$income, test$predicted.income > 0.5)
```

```
Warning message:
In predict.lm(object, newdata, se.fit, scale = 1, type = ifelse(type == : prediction from a rank-deficient fit may be misleading
```

Out[74]:

```
        FALSE TRUE
  <=50K  6372  548
  >50K    872 1423
```

You'll notice we have a rank deficient fit. Find out more about what issues this may cause by reading this [stackexchange post](#).

What was the accuracy of our model?

In [76]:

In [76]:

```
(6372+1423)/(6372+1423+548+872)
```

Out[76]:

0.845903418339664

Calculate other measures of performance like, recall or precision.

In [77]:

```
#recall
6732/(6372+548)
```

Out[77]:

0.972832369942197

In [78]:

```
#precision
6732/(6372+872)
```

Out[78]:

0.929320817228051

So after executing the above mentioned steps the model should be able to predict the income class

We can also find the other evaluation parameters like recall, precision etc...

Do make the required changes as needed and make your predictions