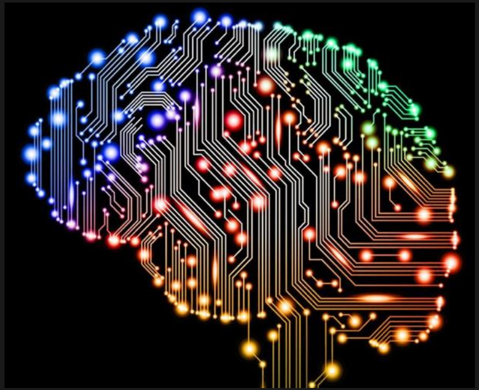
Predictive Modeling Prototyping with the R caret Package

Prof L

PURDUE
KRANNERT
SCHOOL OF MANAGEMENT



About Me

Clinical Assistant Professor at Purdue Universities' Krannert School of Management and Co-Founder/Chief Data Scientist of Biz Analytics Lab, LLC in Lafayette, IN.

- Teach and mentor students (Fall/Spring semesters)
- Work with a couple fantastic partners (Summer)

At Krannert, course coordinator and teacher for:

- MGMT 571 Data Mining (Fall semester)
- MGMT 590 Using R for Analytics (Fall semester)
- MGMT 590 Predictive Analytics (Spring semester)
- MGMT 690 MS BAIM Industry Practicum (Spring semester)

Spend most of my time obtaining and mentoring experimental learning projects for students within Purdue's M.S. in Business Analytics & Information Management (BAIM) program.

Key items to take away

- How **caret** is designed to work
- Key **caret** functions
- Purdue’s MS BAIM is awesome!

Presentation and R script link

https://github.com/MatthewALanham/informsba2018

Data Mining (MS BAIM fall core)

- Following a process (e.g. CRISP-DM, INFORMS CAP framework)
- Relational Databases/SQL
- Supervised vs. Unsupervised Learning
- Regression vs. Classification Problems
- Cross-validation Designs (validation set, k-fold, LOOCV, bias-variance tradeoff)
- Exploratory Data Analysis & Visualization with Tableau
- Data Pre-Processing (multicollinearity, binning, feature engineering)
- Linear Models
- Dimension Reduction via PCA & Stepwise Approaches
- Clustering Analysis (Hierarchical, k-Means, PAM, SOMs, Silhouettes)
- Classification and Regression Trees
- Feed Forward Neural Networks
- Association Rules/Market Basket Analysis

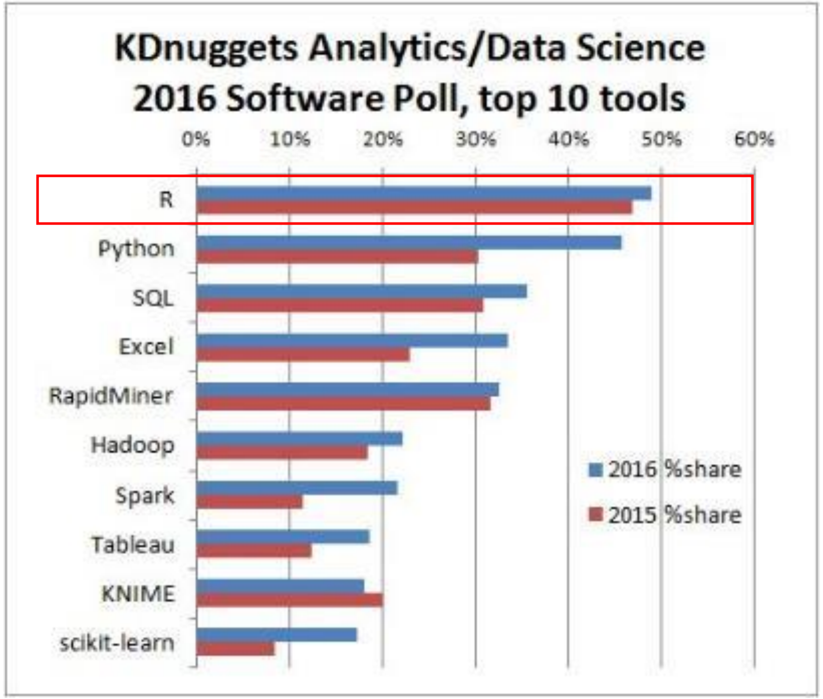
* Can easily do using caret

Predictive Analytics (MS BAIM spring elective)

- INFORMS CAP framework
- Decision Model Basics (LP, IP, MIP)
- Designing Solutions & Integrating Analytics (Descriptive, Predictive, Prescriptive)
- Review Cross-validation Designs, Bias-variance Tradeoff
- KNN, Bayes Classifier, Naïve Bayes
- Linear & Quadratic Discriminant Analysis (LDA/QDA)
- Support Vector Machines, Factorization Machines
- Multi-classification Modeling & Evaluation - Multinomial Logit, SVM
- Cost-based Learning & Evaluation
- Ensembling (Voting, Propensity Averaging, Random Forest/Bagging, AdaBoost/ Boosting, Gradient Boosting Machines, Meta-Modeling)
- Recurrent, LSTM, & Convolution Neural Nets to Deep learning
- Text Mining

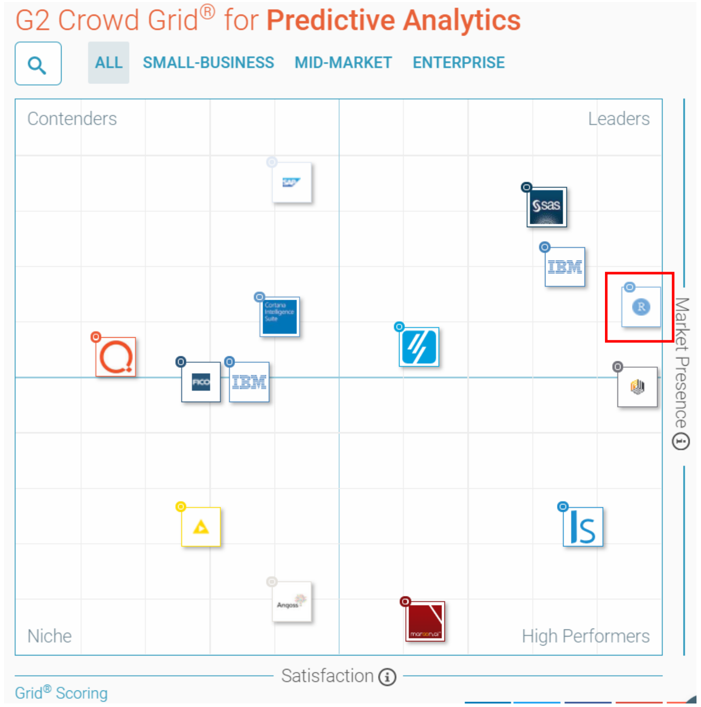
* Can easily do using caret

R/RStudio continues to be popular in practice



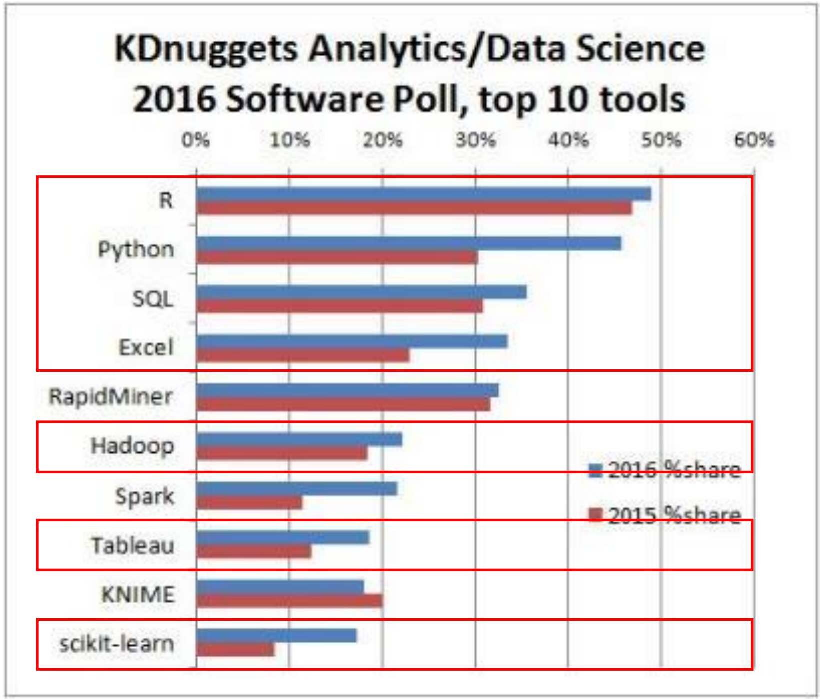
Source: KDnuggets.com (2017)

“Best Predictive Analytics Software”



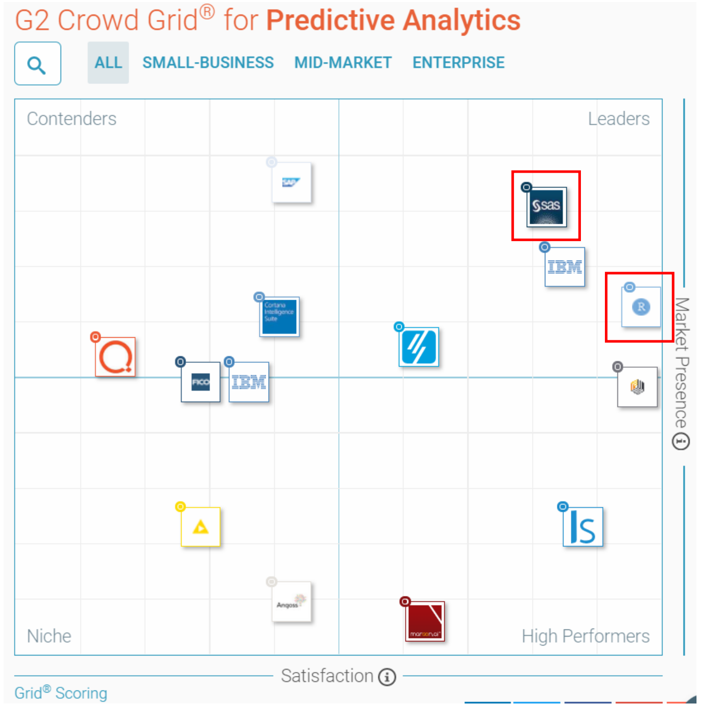
Source: G2crowd.com (2018)

MS BAIM students use other software too…



Source: KDnuggets.com (2017)

"Best Predictive Analytics Software"

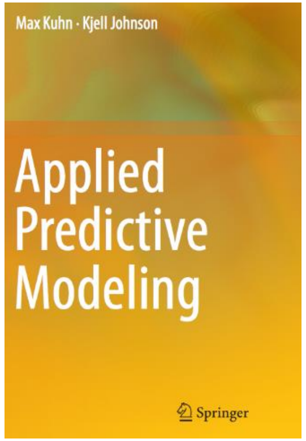


Source: G2crowd.com (2018)

caret Package

The **C**lassification **A**nd **RE**gression **T**raining **caret** package was developed by Max Kuhn as a tool to streamline the predictive modeling process for R users.

His book published in 2013 and has many nice examples.



Additional information here:
http://topepo.github.io/caret/index.html

Most of the functionality can be broken down into 5 areas

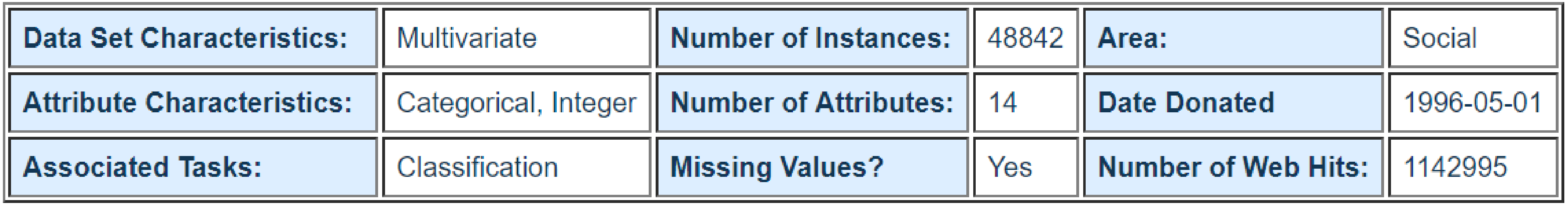
Data splitting	**Pre-processing**	**Feature selection**	**Model tuning using resampling**	**Variable importance estimation**

Example: Classic Adult/Census Income dataset

Problem type: Binary Classification

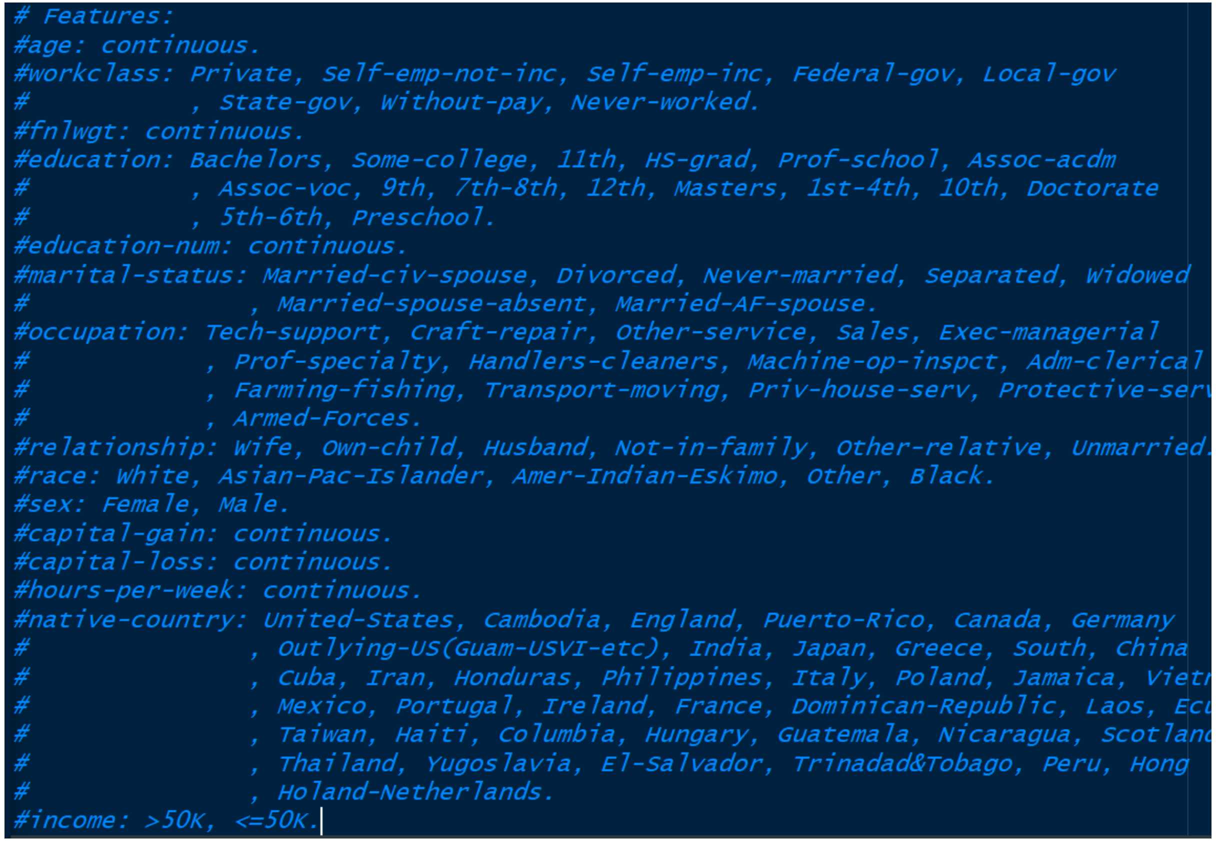
Objective: Predict whether income exceeds $50K/yr for an individual based on1994 census database.

Dataset: http://archive.ics.uci.edu/ml/datasets/Adult



Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	1142995

Features



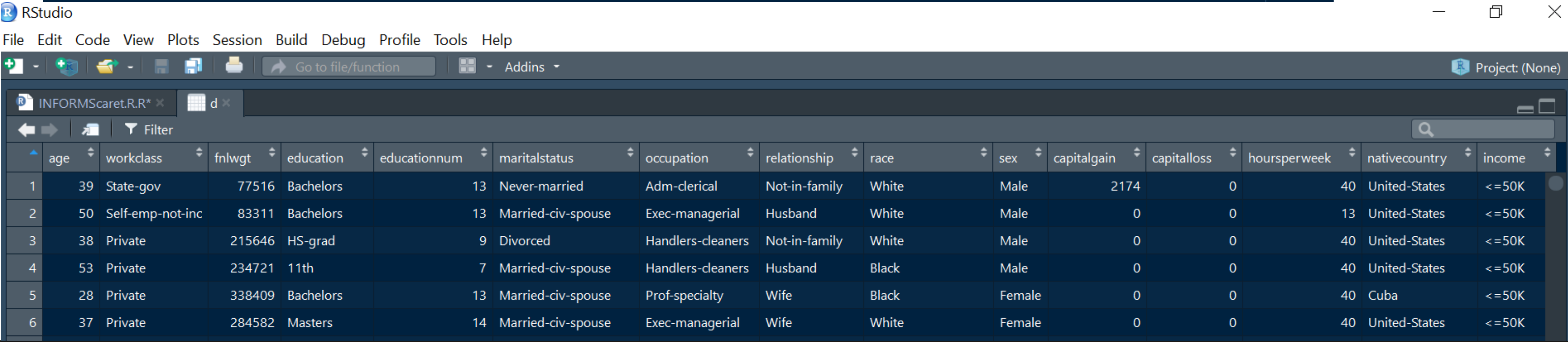
```
# Features:
#age: continuous.
#workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov
#           , State-gov, Without-pay, Never-worked.
#fnlwgt: continuous.
#education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm
#           , Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate
#           , 5th-6th, Preschool.
#education-num: continuous.
#marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed
#               , Married-spouse-absent, Married-AF-spouse.
#occupation: 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-ser
#            , Armed-Forces.
#relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
#race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
#sex: Female, Male.
#capital-gain: continuous.
#capital-loss: continuous.
#hours-per-week: continuous.
#native-country: 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, Viet
#               , Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ec
#               , Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotlan
#               , Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong
#               , Holand-Netherlands.
#income: >50K, <=50K.
```

Potential features

Target

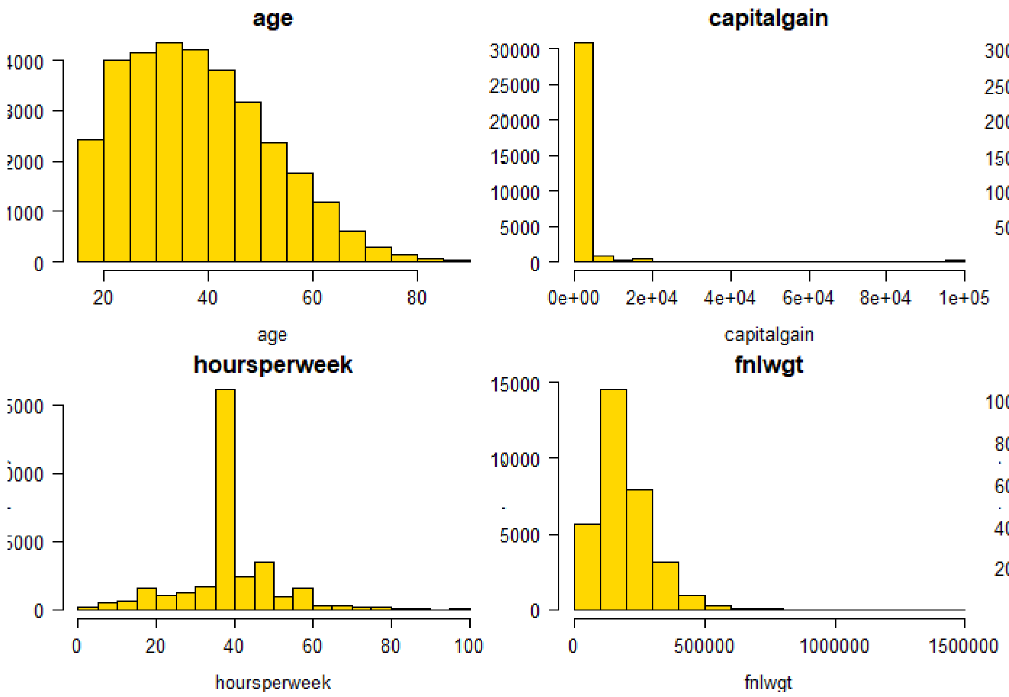
Load dataset

```r
# Load data from the web
myUrl <- "http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
d <- read.table(file=myUrl, header=F, sep=",", quote="",
                colClasses=c("numeric","factor","numeric","factor","numeric"
                             ,rep("factor",5),rep("numeric",3),rep("factor",2)))
# specify column names
names(d) <- c("age","workclass","fnlwgt","education","educationnum",
              "maritalstatus","occupation","relationship","race","sex",
              "capitalgain","capitalloss","hoursperweek","nativecountry",
              "income")
# examine data structure
str(d)
```

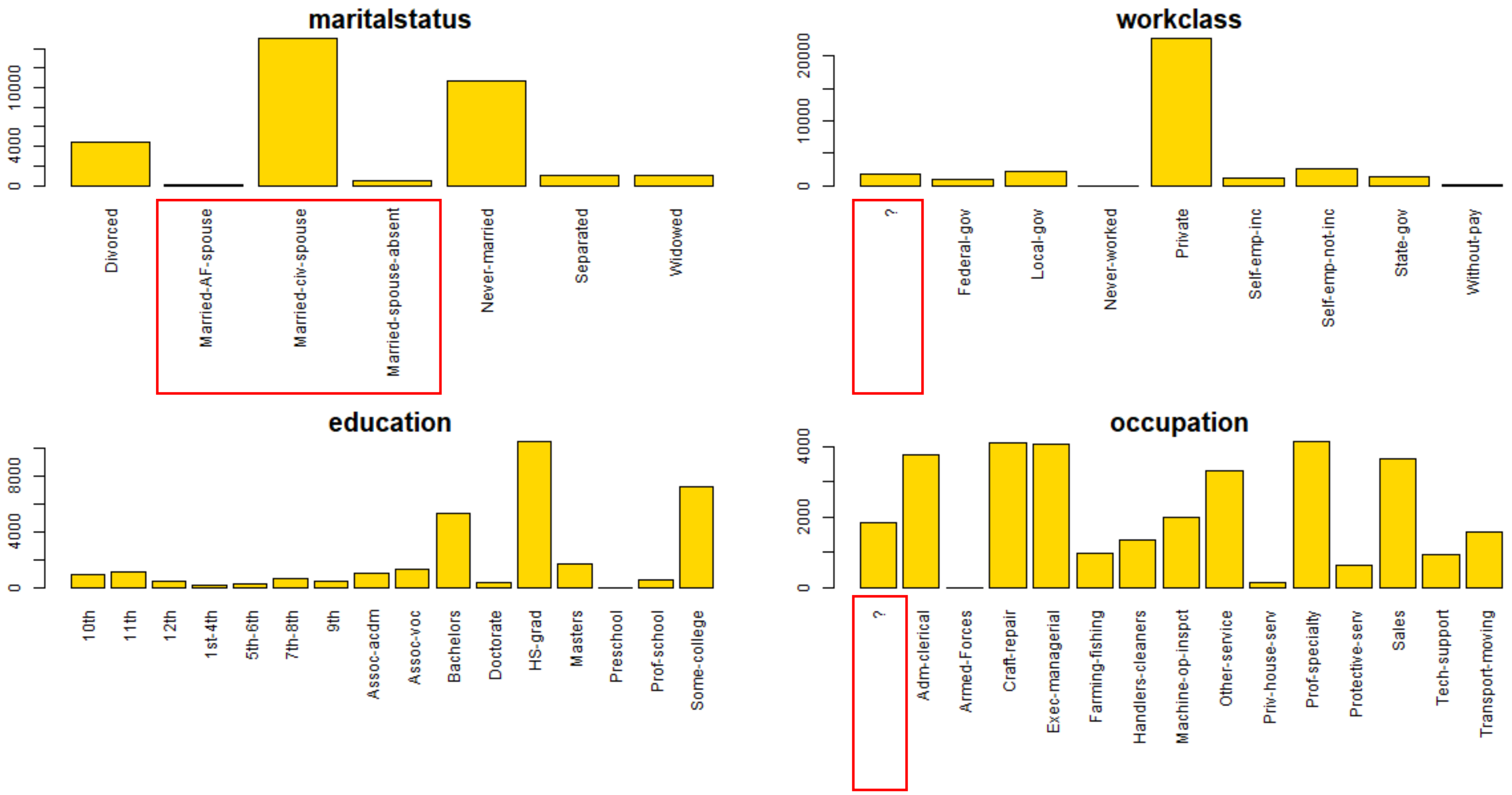


	age	workclass	fnlwgt	education	educationnum	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	nativecountry	income
1	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
2	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
3	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
4	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
5	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
6	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

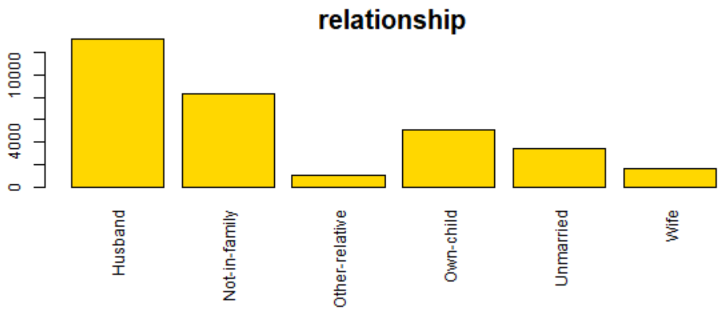
EDA

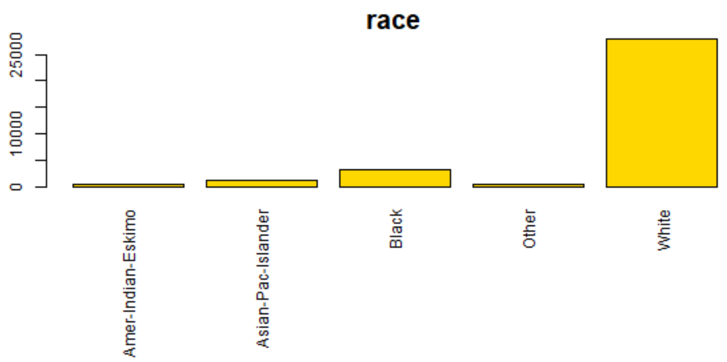

age
capitalgain
capitalloss
hoursperweek
fnlwgt
educationnum

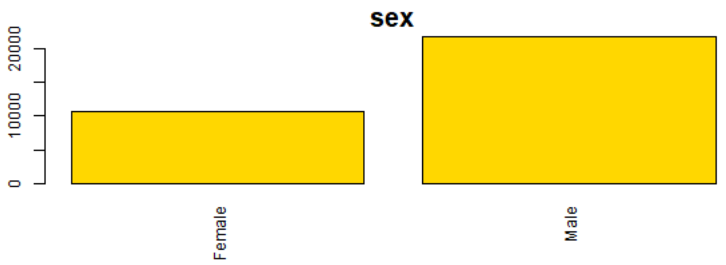
EDA

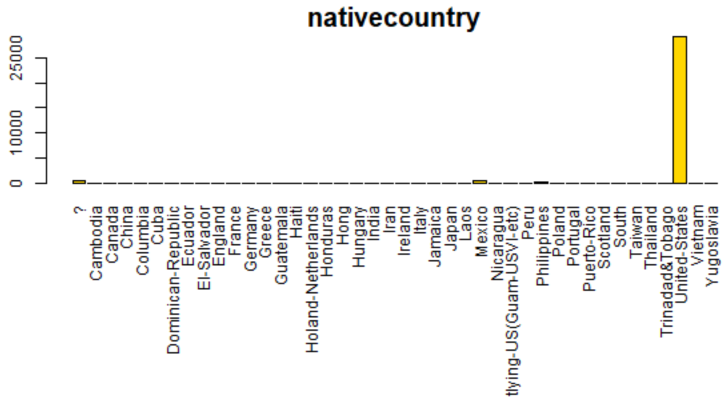

maritalstatus
0
4000
10000
Divorced
Married-AF-spouse
Married-civ-spouse
Married-spouse-absent
Never-married
Separated
Widowed
workclass
0
10000
20000
?
Federal-gov
Local-gov
Never-worked
Private
Self-emp-inc
Self-emp-not-inc
State-gov
Without-pay
education
0
4000
8000
10th
11th
12th
1st-4th
5th-6th
7th-8th
9th
Assoc-acdm
Assoc-voc
Bachelors
Doctorate
HS-grad
Masters
Preschool
Prof-school
Some-college
occupation
0
2000
4000
?
Adm-clerical
Armed-Forces
Craft-repair
Exec-managerial
Farming-fishing
Handlers-cleaners
Machine-op-inspct
Other-service
Priv-house-serv
Prof-specialty
Protective-serv
Sales
Tech-support
Transport-moving

EDA

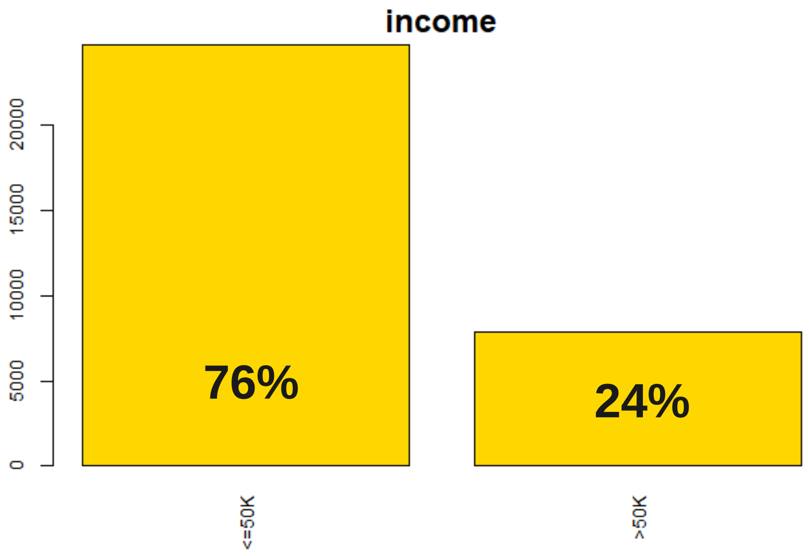

relationship
0
4000
10000
Husband
Not-in-family
Other-relative
Own-child
Unmarried
Wife


race
0
10000
25000
Amer-Indian-Eskimo
Asian-Pac-Islander
Black
Other
White


sex
0
10000
20000
Female
Male

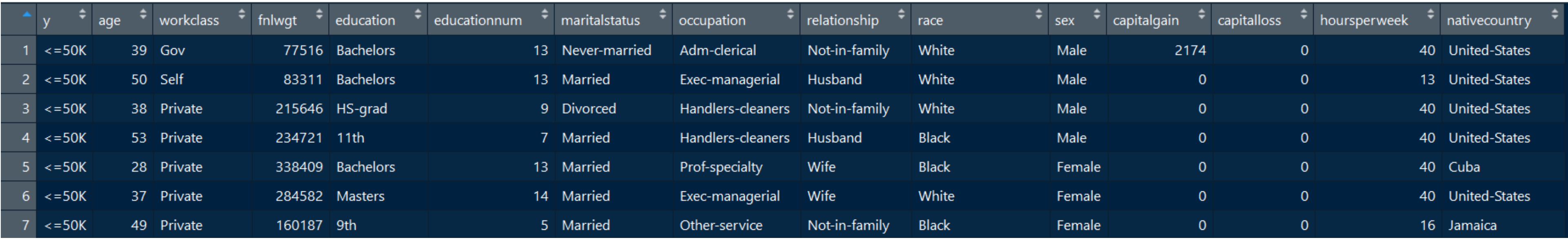

nativecountry
0
10000
25000
?
Cambodia
Canada
China
Columbia
Cuba
Dominican-Republic
Ecuador
El-Salvador
England
France
Germany
Greece
Guatemala
Haiti
Holand-Netherlands
Honduras
Hong
Hungary
India
Iran
Ireland
Italy
Jamaica
Japan
Laos
Mexico
Nicaragua
tlying-US(Guam-USVI-etc)
Peru
Philippines
Poland
Portugal
Puerto-Rico
Scotland
South
Taiwan
Thailand
Trinadad&Tobago
United-States
Vietnam
Yugoslavia

EDA



EDA

- In the R script I do some data cleaning. Will not cover for this presentation.



	y	age	workclass	fnlwgt	education	educationnum	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	nativecountry
1	<=50K	39	Gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States
2	<=50K	50	Self	83311	Bachelors	13	Married	Exec-managerial	Husband	White	Male	0	0	13	United-States
3	<=50K	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States
4	<=50K	53	Private	234721	11th	7	Married	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States
5	<=50K	28	Private	338409	Bachelors	13	Married	Prof-specialty	Wife	Black	Female	0	0	40	Cuba
6	<=50K	37	Private	284582	Masters	14	Married	Exec-managerial	Wife	White	Female	0	0	40	United-States
7	<=50K	49	Private	160187	9th	5	Married	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica

caret preprocessing functions

1) **dummyVars ()** function creates a full set of dummy variables (i.e. less than full rank parameterization)

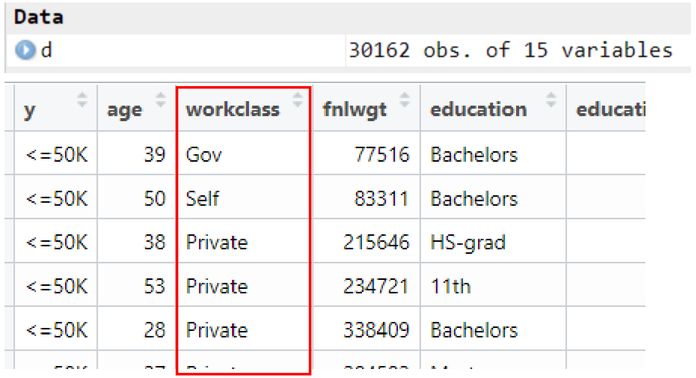
2) **findCorrelation()** function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations

3) **findLinearCombos()** function enumerates and resolves the linear combinations in a numeric matrix

4) **preProcess()** function performs transformations (centering, scaling etc.) estimated from the training data and applied to any data set with the same variables

dummyVars()

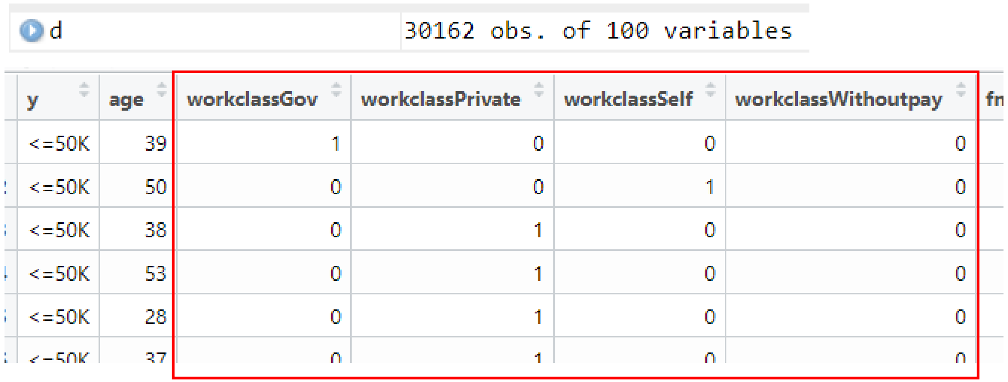
1) **dummyVars ()** function creates a full set of dummy variables (i.e. less than full rank parameterization)



Data

d 30162 obs. of 15 variables

y	age	workclass	fnlwgt	education	educati
<=50K	39	Gov	77516	Bachelors	
<=50K	50	Self	83311	Bachelors	
<=50K	38	Private	215646	HS-grad	
<=50K	53	Private	234721	11th	
<=50K	28	Private	338409	Bachelors	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

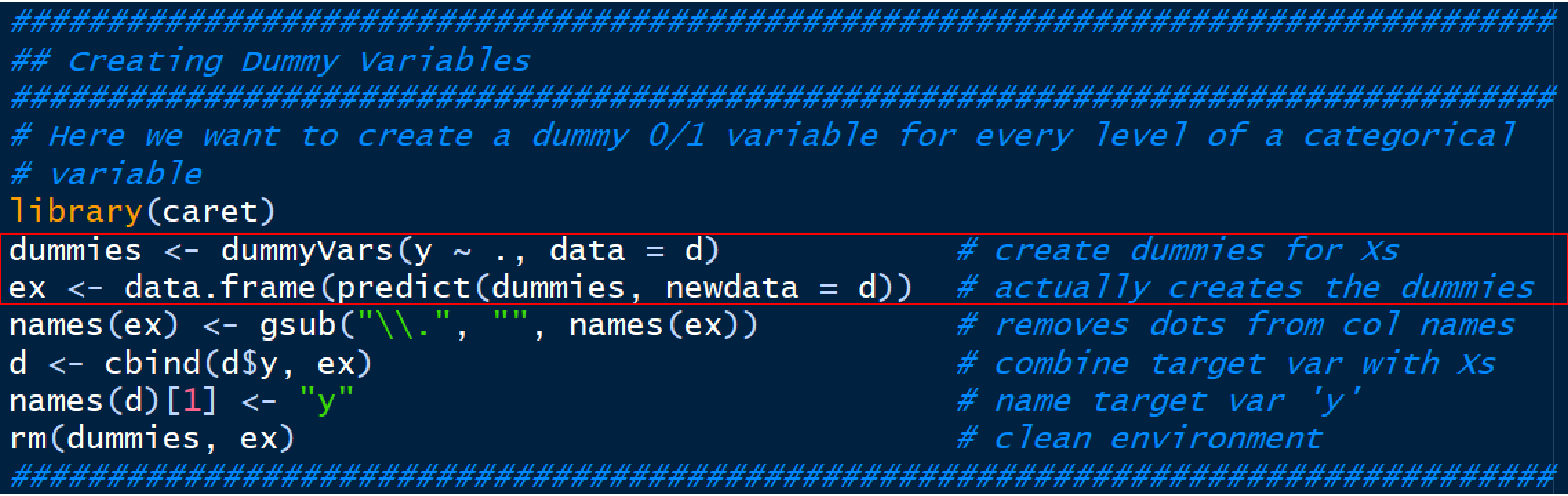


d 30162 obs. of 100 variables

y	age	workclassGov	workclassPrivate	workclassSelf	workclassWithoutpay	fn
<=50K	39	1	0	0	0	
<=50K	50	0	0	1	0	
<=50K	38	0	1	0	0	
<=50K	53	0	1	0	0	
<=50K	28	0	1	0	0	
<=50K	37	0	1	0	0	

dummyVars()

1) **dummyVars ()** function creates a full set of dummy variables (i.e. less than full rank parameterization)

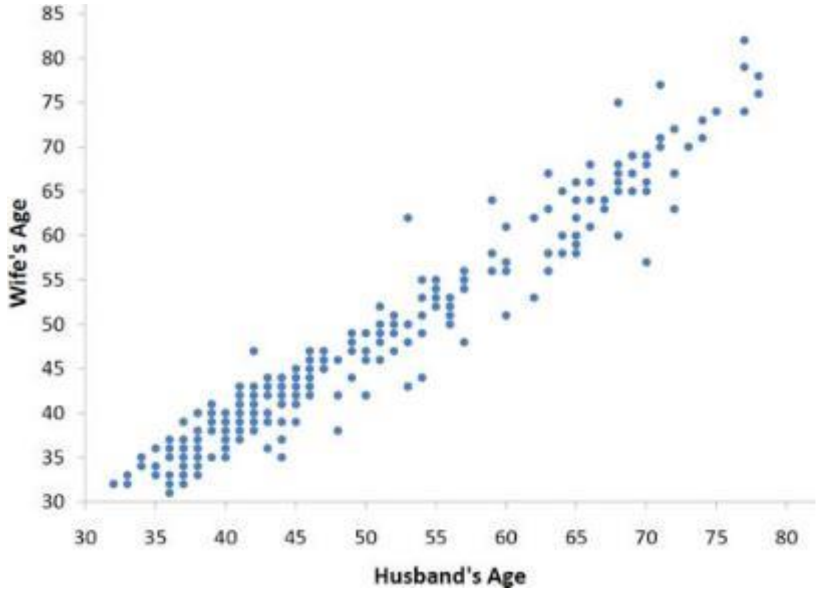


```
################################################################################
## Creating Dummy Variables
################################################################################
# Here we want to create a dummy 0/1 variable for every level of a categorical
# variable
library(caret)
dummies <- dummyVars(y ~ ., data = d)              # create dummies for Xs
ex <- data.frame(predict(dummies, newdata = d))   # actually creates the dummies
names(ex) <- gsub("\\.", "", names(ex))           # removes dots from col names
d <- cbind(d$y, ex)                               # combine target var with Xs
names(d)[1] <- "y"                                # name target var 'y'
rm(dummies, ex)                                   # clean environment
################################################################################
```

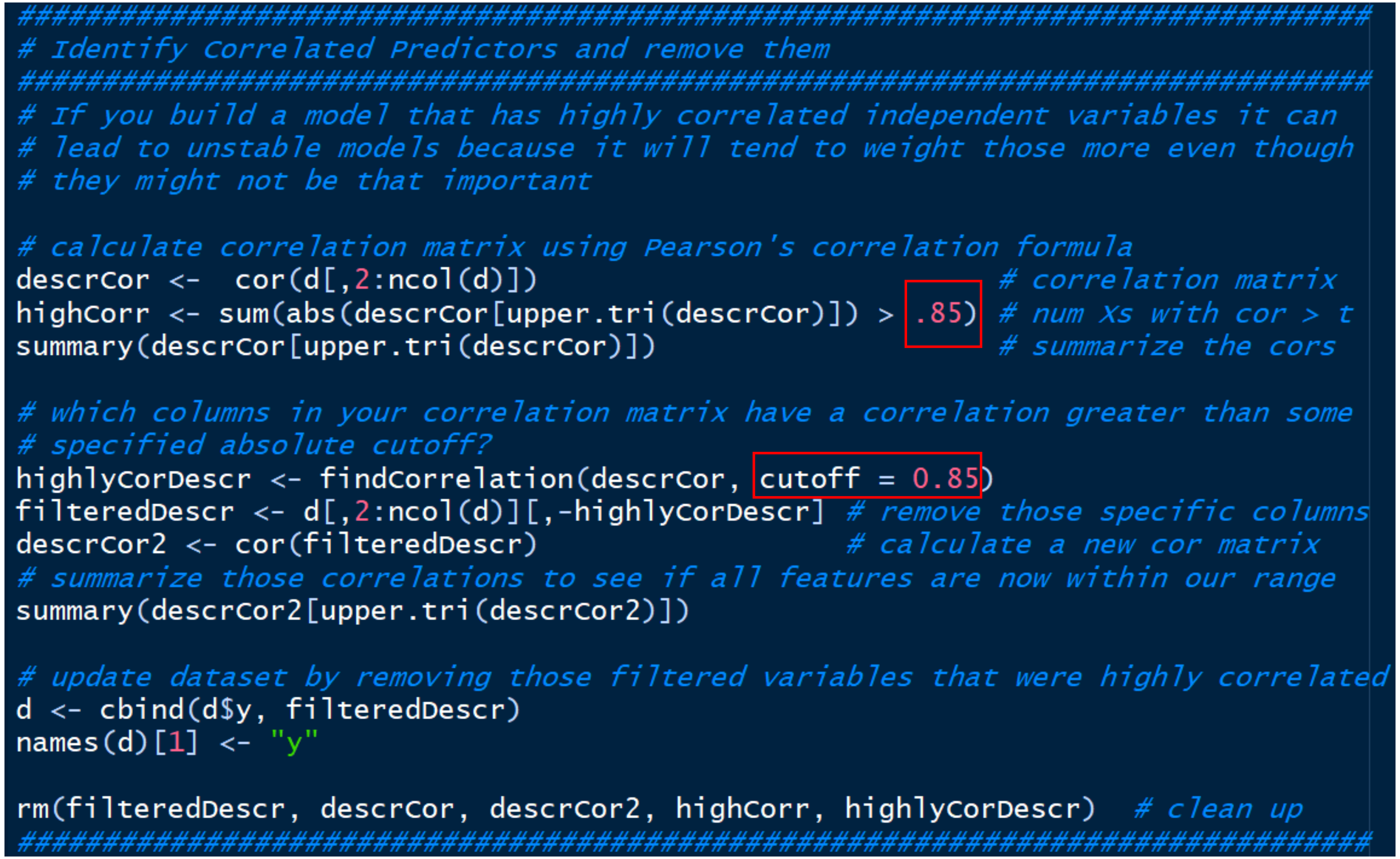
findCorrelation()

2) findCorrelation() function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations.

If you build a model that has highly correlated independent variables it can lead to unstable models because it will tend to weight those more even though they might not be that important.



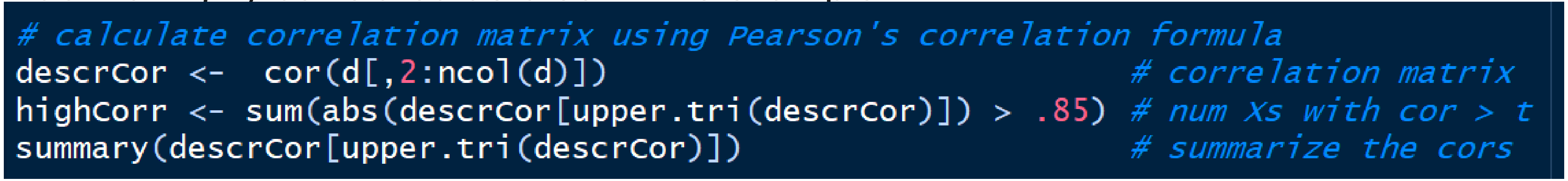
findCorrelation()



```
################################################################################
# Identify Correlated Predictors and remove them
################################################################################
# If you build a model that has highly correlated independent variables it can
# lead to unstable models because it will tend to weight those more even though
# they might not be that important

# calculate correlation matrix using Pearson's correlation formula
descrCor <-  cor(d[,2:ncol(d)])                            # correlation matrix
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .85) # num Xs with cor > t
summary(descrCor[upper.tri(descrCor)])                     # summarize the cors

# which columns in your correlation matrix have a correlation greater than some
# specified absolute cutoff?
highlyCorDescr <- findCorrelation(descrCor, cutoff = 0.85)
filteredDescr <- d[,2:ncol(d)][,-highlyCorDescr] # remove those specific columns
descrCor2 <- cor(filteredDescr)                  # calculate a new cor matrix
# summarize those correlations to see if all features are now within our range
summary(descrCor2[upper.tri(descrCor2)])

# update dataset by removing those filtered variables that were highly correlated
d <- cbind(d$y, filteredDescr)
names(d)[1] <- "y"

rm(filteredDescr, descrCor, descrCor2, highCorr, highlyCorDescr)  # clean up
################################################################################
```

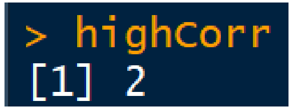
findCorrelation()

I define highly correlated as 0.85 in this example

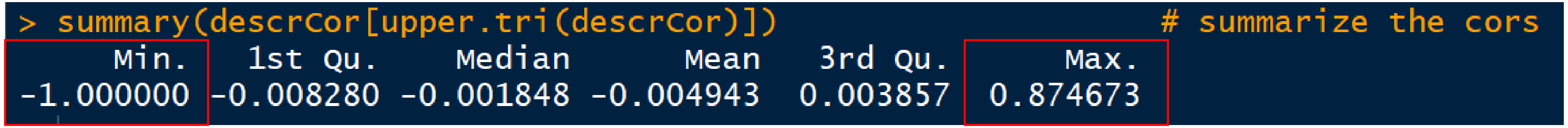


```
# calculate correlation matrix using Pearson's correlation formula
descrCor <-  cor(d[,2:ncol(d)])                          # correlation matrix
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .85) # num Xs with cor > t
summary(descrCor[upper.tri(descrCor)])                    # summarize the cors
```

highCorr tells me how many variables have a correlation +/- 0.85



```
> highCorr
[1] 2
```

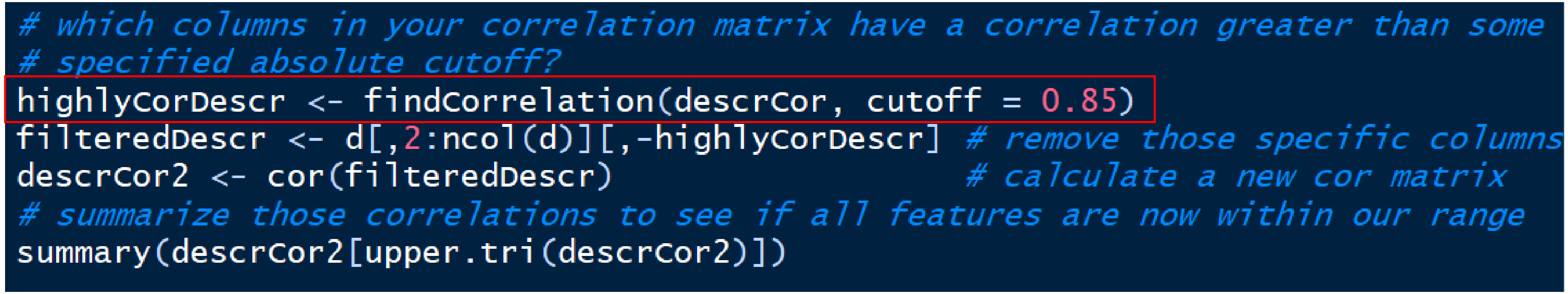


```
> summary(descrCor[upper.tri(descrCor)])                    # summarize the cors
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
-1.000000 -0.008280 -0.001848 -0.004943  0.003857  0.874673 
```

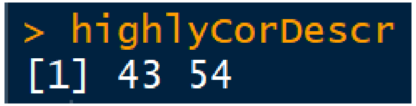
Looks like we have high correlations

findCorrelation()

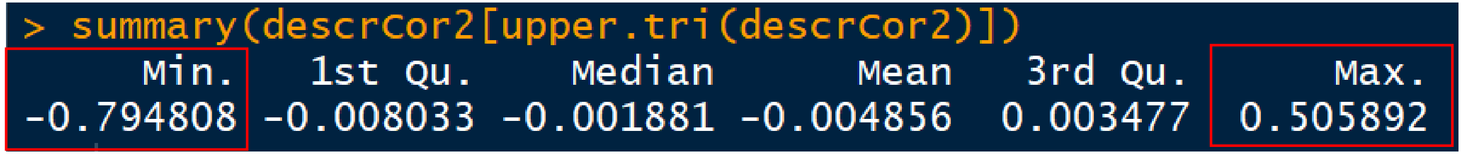
findCorrelation() identifies those columns in the data that are highly correlated



```r
# which columns in your correlation matrix have a correlation greater than some
# specified absolute cutoff?
highlyCorDescr <- findCorrelation(descrCor, cutoff = 0.85)
filteredDescr <- d[,2:ncol(d)][,-highlyCorDescr] # remove those specific columns
descrCor2 <- cor(filteredDescr)                   # calculate a new cor matrix
# summarize those correlations to see if all features are now within our range
summary(descrCor2[upper.tri(descrCor2)])
```



```
> highlyCorDescr
[1] 43 54
```

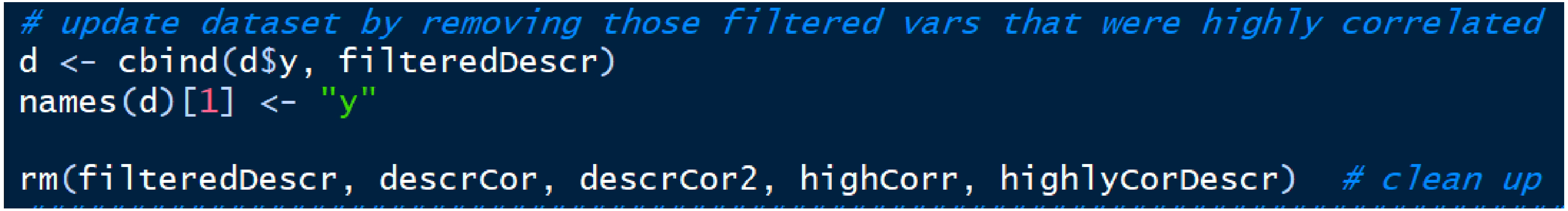


```
> summary(descrCor2[upper.tri(descrCor2)])
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
-0.794808 -0.008033 -0.001881 -0.004856  0.003477  0.505892 
```

All correlations are now below our threshold

findCorrelation()

Assuming you can justify dropping those features in respect to the problem, here we combine our target variable with the set of features are not “highly correlated.”



```r
# update dataset by removing those filtered vars that were highly correlated
d <- cbind(d$y, filteredDescr)
names(d)[1] <- "y"

rm(filteredDescr, descrCor, descrCor2, highCorr, highlyCorDescr)  # clean up
```

findLinearCombos()

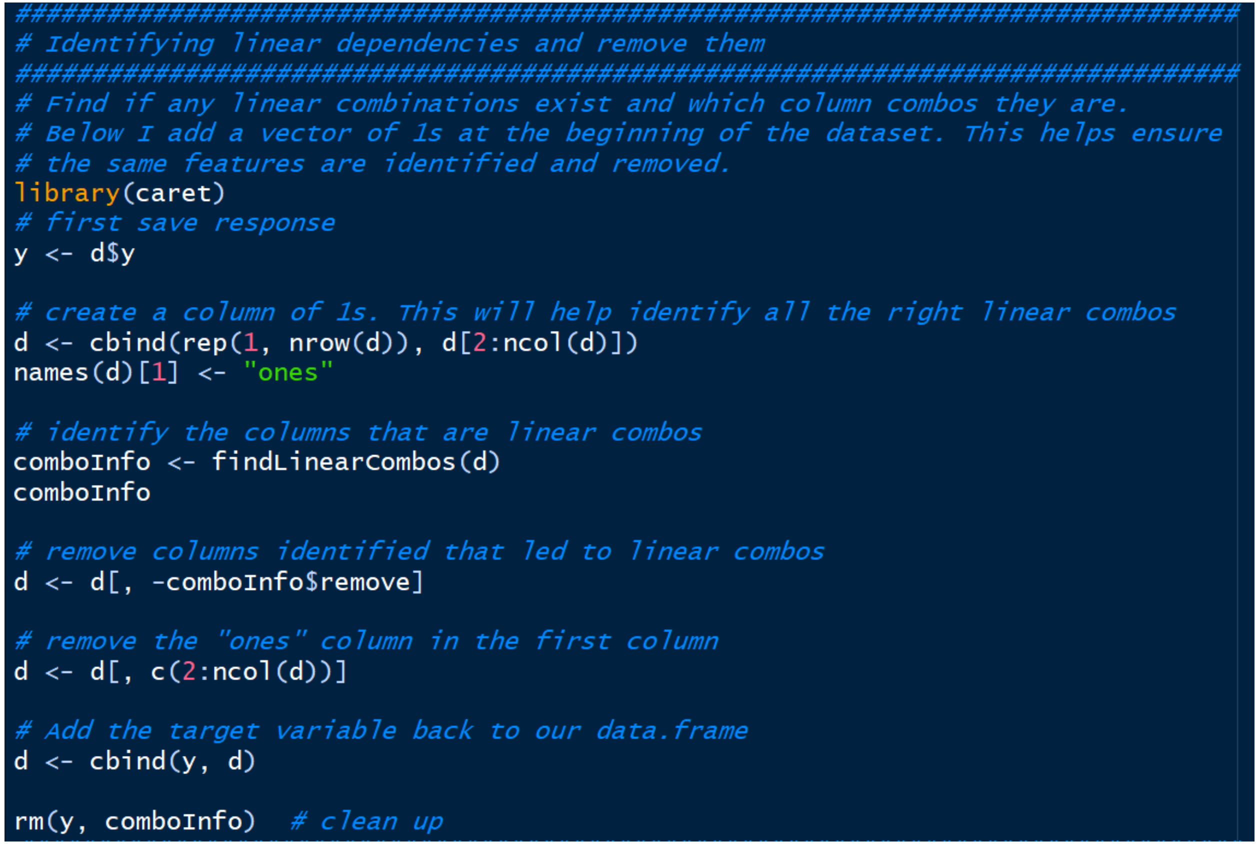
3) findLinearCombos() function enumerates and resolves the linear combinations in a numeric matrix

Y	Age	workclassGov	workclassPrivate	workclassSelf	workclassWithoutpay	row sum
1	18	1	0	0	0	1
1	22	1	0	0	0	1
0	45	1	0	0	0	1
1	33	0	0	1	0	1
0	56	0	0	1	0	1
1	43	0	1	0	0	1
0	51	0	1	0	0	1
0	25	0	0	0	1	1

Drop a column

Y	Age	workclassGov	workclassPrivate	workclassSelf	workclassWithoutpay	row sum
1	18	1	0	0	0	0
1	22	1	0	0	0	0
0	45	1	0	0	0	0
1	33	0	0	1	0	1
0	56	0	0	1	0	1
1	43	0	1	0	0	1
0	51	0	1	0	0	1
0	25	0	0	0	1	1

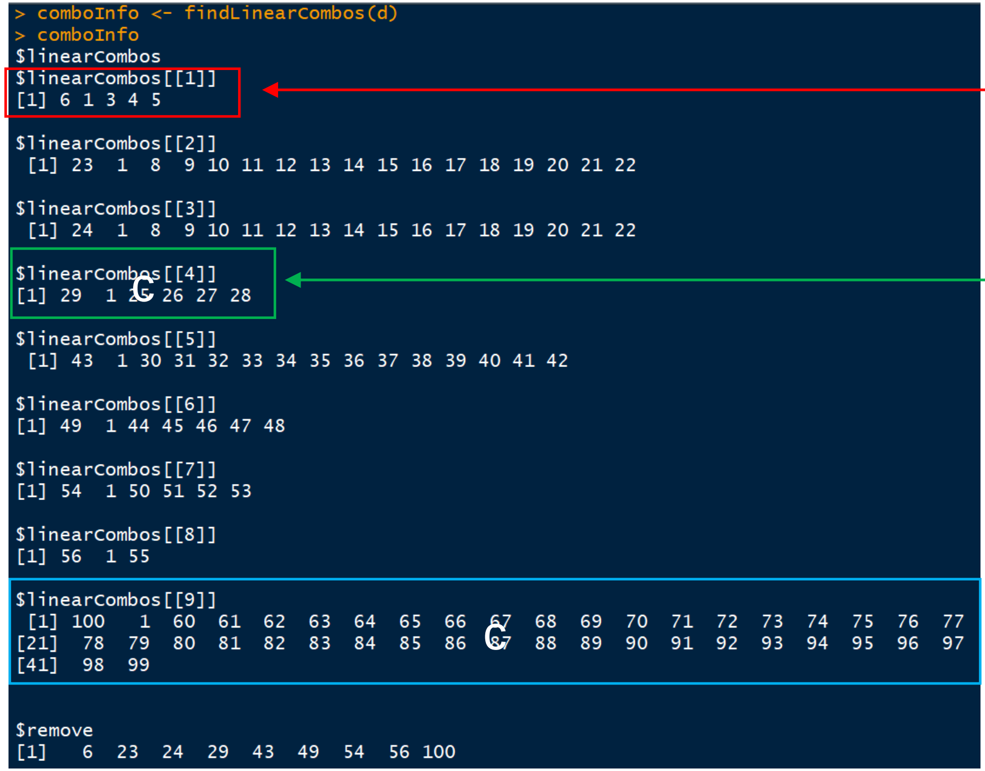
findLinearCombos()



```
##################################################################################
# Identifying linear dependencies and remove them
##################################################################################
# Find if any linear combinations exist and which column combos they are.
# Below I add a vector of 1s at the beginning of the dataset. This helps ensure
# the same features are identified and removed.
library(caret)
# first save response
y <- d$y

# create a column of 1s. This will help identify all the right linear combos
d <- cbind(rep(1, nrow(d)), d[2:ncol(d)])
names(d)[1] <- "ones"

# identify the columns that are linear combos
comboInfo <- findLinearCombos(d)
comboInfo

# remove columns identified that led to linear combos
d <- d[, -comboInfo$remove]

# remove the "ones" column in the first column
d <- d[, c(2:ncol(d))]

# Add the target variable back to our data.frame
d <- cbind(y, d)

rm(y, comboInfo)  # clean up
```

findLinearCombos()

Here we can set the different groups of columns that form linear combinations



```
> comboInfo <- findLinearCombos(d)
> comboInfo
$linearCombos
$linearCombos[[1]]
[1] 6 1 3 4 5

$linearCombos[[2]]
 [1] 23  1  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22

$linearCombos[[3]]
 [1] 24  1  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22

$linearCombos[[4]]
[1] 29  1 25 26 27 28

$linearCombos[[5]]
 [1] 43  1 30 31 32 33 34 35 36 37 38 39 40 41 42

$linearCombos[[6]]
[1] 49  1 44 45 46 47 48

$linearCombos[[7]]
[1] 54  1 50 51 52 53

$linearCombos[[8]]
[1] 56  1 55

$linearCombos[[9]]
 [1] 100   1  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
[21]  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
[41]  98  99


$remove
[1]   6  23  24  29  43  49  54  56 100
```

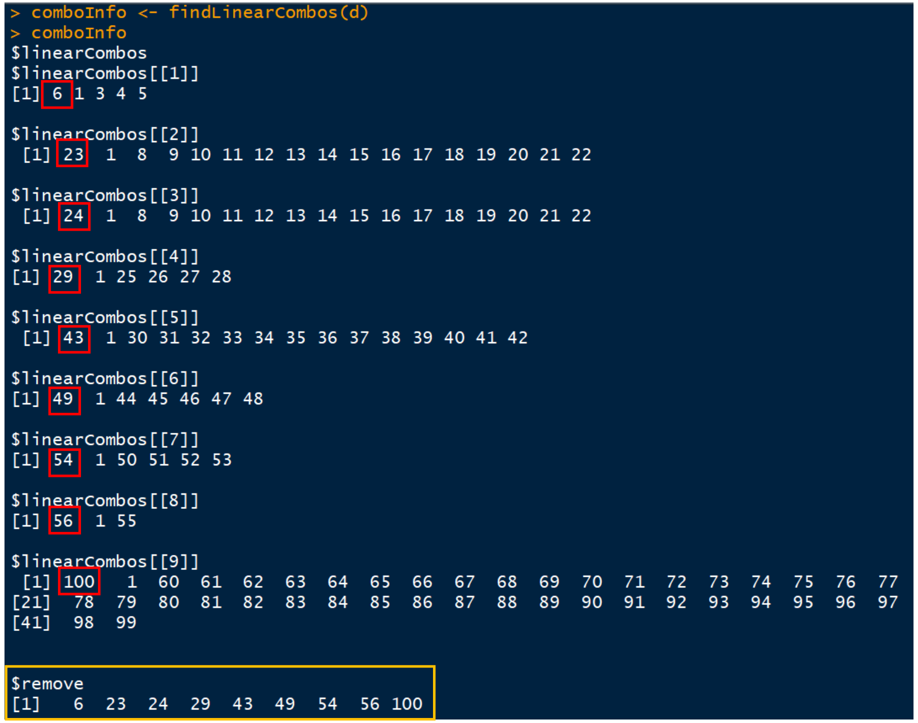
Work class dummies

Marital status dummies

Native country dummies

findLinearCombos()

Here we can see the columns to drop. By default it will drop the first column among a set of linear combos.



```
> comboInfo <- findLinearCombos(d)
> comboInfo
$linearCombos
$linearCombos[[1]]
[1] 6 1 3 4 5

$linearCombos[[2]]
 [1] 23  1  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22

$linearCombos[[3]]
 [1] 24  1  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22

$linearCombos[[4]]
[1] 29  1 25 26 27 28

$linearCombos[[5]]
 [1] 43  1 30 31 32 33 34 35 36 37 38 39 40 41 42

$linearCombos[[6]]
[1] 49  1 44 45 46 47 48

$linearCombos[[7]]
[1] 54  1 50 51 52 53

$linearCombos[[8]]
[1] 56  1 55

$linearCombos[[9]]
 [1] 100   1  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
[21]  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
[41]  98  99


$remove
[1]   6  23  24  29  43  49  54  56 100
```

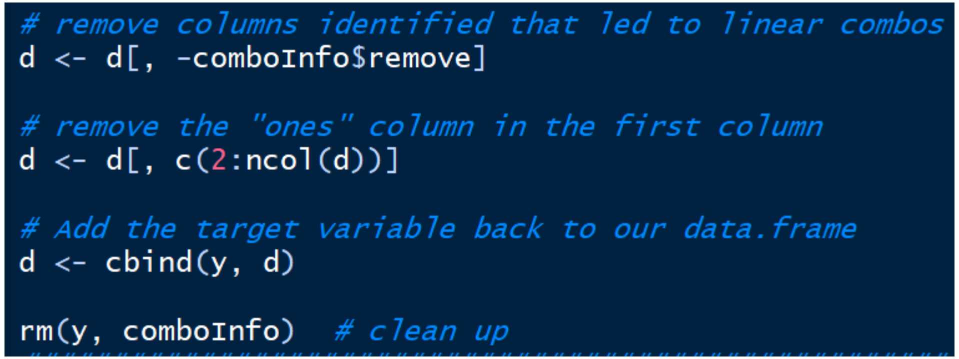
However, you can specify which columns you want to drop manually if you choose.

You'll usually have some justification for keeping or dropping some of them.

- Using a linear model and want a particular feature to serve as baseline
- Some dummies are too sparse

findLinearCombos()

Lastly, we will go ahead and remove those columns indicated for removal. We also add back our target variable Y.

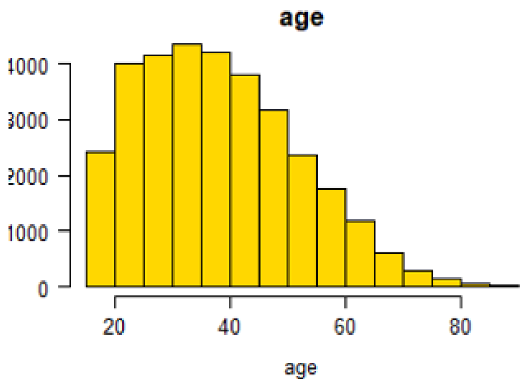


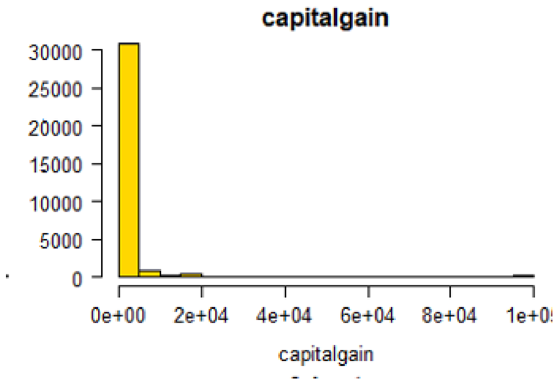
```
# remove columns identified that led to linear combos
d <- d[, -comboInfo$remove]

# remove the "ones" column in the first column
d <- d[, c(2:ncol(d))]

# Add the target variable back to our data.frame
d <- cbind(y, d)

rm(y, comboInfo)  # clean up
```

preProcess()

4) **preProcess()** function performs transformations (centering, scaling etc.) estimated from the training data and applied to any data set with the same variables

Variables tend to have ranges different from each other:





Some data mining algorithms are adversely affected by differences in variable ranges, where greater ranges tend to have larger influence on data model's results.

preProcess()

There are various transformations available in preProcess. Some commonly used ones are:

Goal: Put your features on the same scale
- Z-score standardization
- Min-max normalization

```
method = c("center","scale")

method = c("range")
```

Goal: Make your features more bell-shaped
- Box-Cox
- Yeo-Johnson

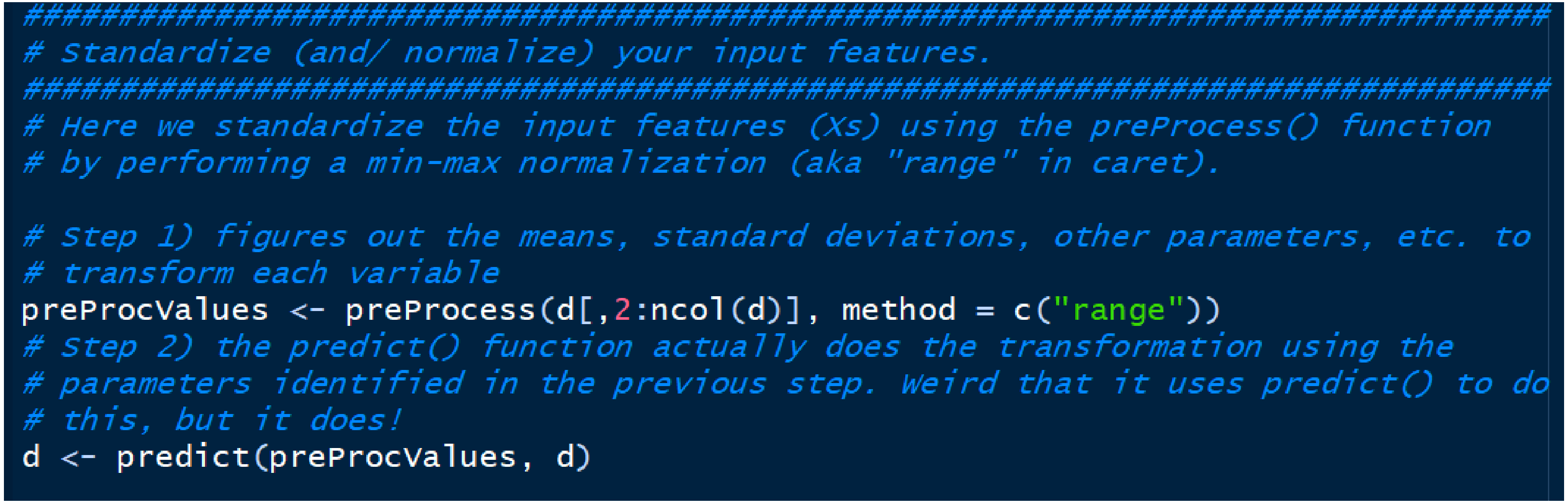
```
method = c("BoxCox")

method = c("YeoJohnson")
```

Goal: Do a combination of both
- Z-score & Yeo-Johnson
- Min-max & Yeo-Johnson

```
method = c("center","scale","YeoJohnson")

method = c("range","YeoJohnson")
```

preProcess()

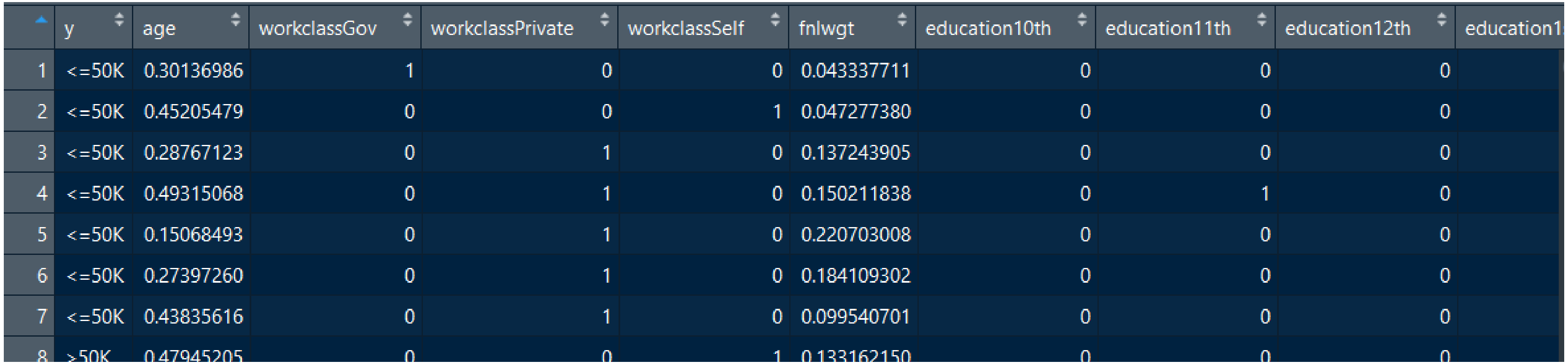
Here we transform our features using the min-max normalization (a.k.a. “range”)



```
###################################################################
# Standardize (and/ normalize) your input features.
###################################################################
# Here we standardize the input features (Xs) using the preProcess() function
# by performing a min-max normalization (aka "range" in caret).

# Step 1) figures out the means, standard deviations, other parameters, etc. to
# transform each variable
preProcValues <- preProcess(d[,2:ncol(d)], method = c("range"))
# Step 2) the predict() function actually does the transformation using the
# parameters identified in the previous step. Weird that it uses predict() to do
# this, but it does!
d <- predict(preProcValues, d)
```

preProcess()

Viewing the data post-transformation, you can see that the fnlwgt features is between 0 and 1.

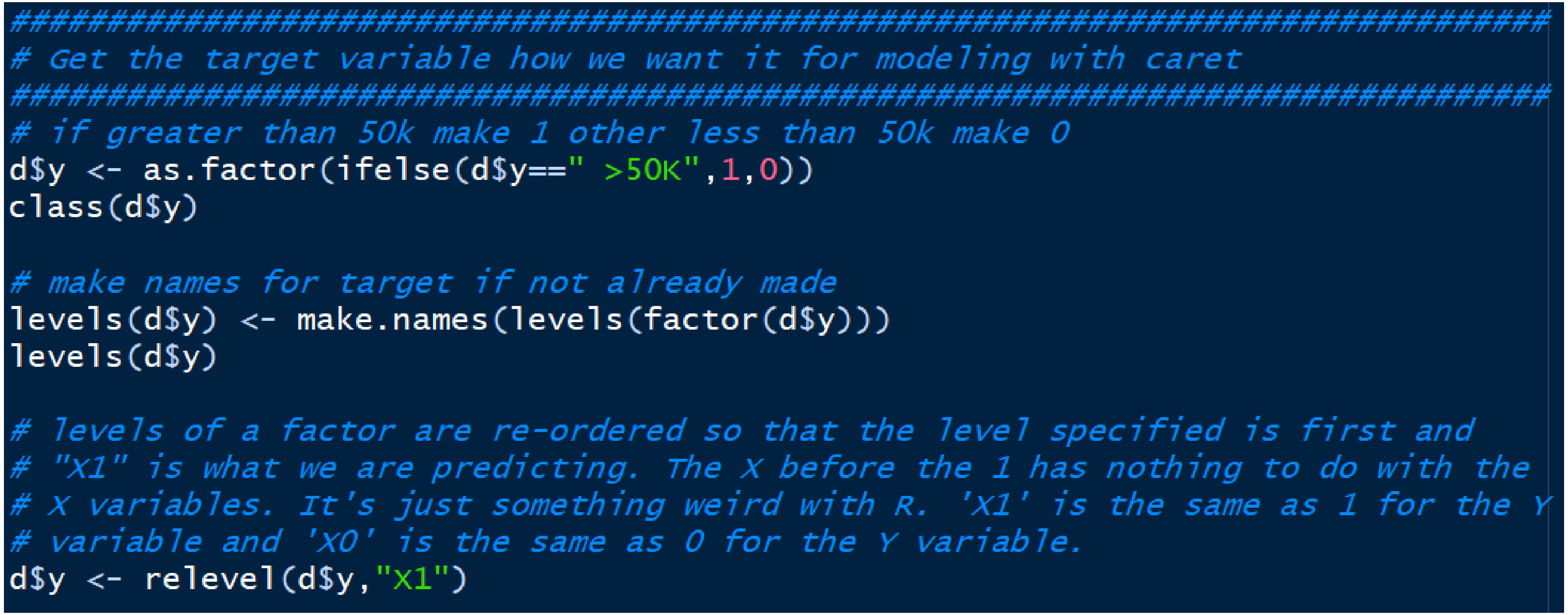


	y	age	workclassGov	workclassPrivate	workclassSelf	fnlwgt	education10th	education11th	education12th	education1
1	<=50K	0.30136986	1	0	0	0.043337711	0	0	0	
2	<=50K	0.45205479	0	0	1	0.047277380	0	0	0	
3	<=50K	0.28767123	0	1	0	0.137243905	0	0	0	
4	<=50K	0.49315068	0	1	0	0.150211838	0	1	0	
5	<=50K	0.15068493	0	1	0	0.220703008	0	0	0	
6	<=50K	0.27397260	0	1	0	0.184109302	0	0	0	
7	<=50K	0.43835616	0	1	0	0.099540701	0	0	0	
8	>50K	0.47945205	0	0	1	0.133162150	0	0	0	

Previously, this feature was between 18 and 80+

Couple things I do for classification probs…

Some algorithms wrapped in caret for classification require your target variable to be “named”. I always do this to avoid possible errors later in training.



```r
######################################################################################
# Get the target variable how we want it for modeling with caret
######################################################################################
# if greater than 50k make 1 other less than 50k make 0
d$y <- as.factor(ifelse(d$y==" >50K",1,0))
class(d$y)

# make names for target if not already made
levels(d$y) <- make.names(levels(factor(d$y)))
levels(d$y)

# levels of a factor are re-ordered so that the level specified is first and
# "X1" is what we are predicting. The X before the 1 has nothing to do with the
# X variables. It's just something weird with R. 'X1' is the same as 1 for the Y
# variable and 'X0' is the same as 0 for the Y variable.
d$y <- relevel(d$y,"X1")
```

caret model design and training functions

5) **createDataPartition()** function allows one to easily partition their data into training and test sets that are distributed (or imbalanced) similar to one another.

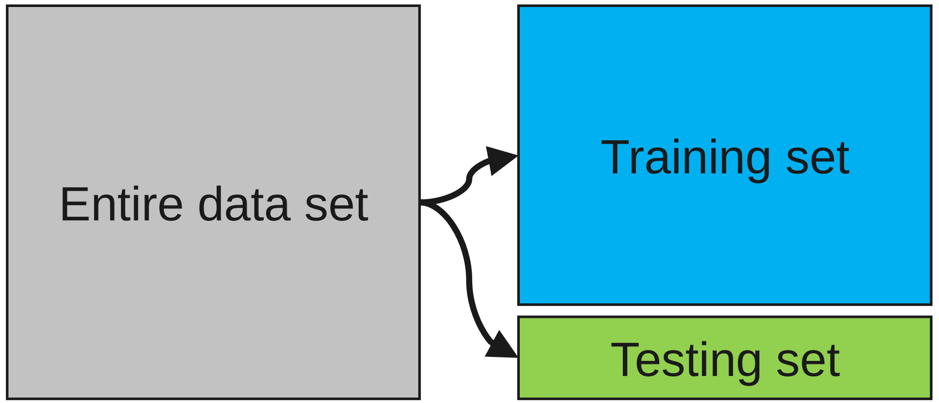
6) **upSample(), downSample()** allows you to up or down sample your training data if it is severely unbalanced

7) **trainControl()** is where you specify how you want to design your run

8) **train()** is the bread and butter of what makes the caret package so great. It's a wrapper for essentially every (not all) regression or classification modeling technique

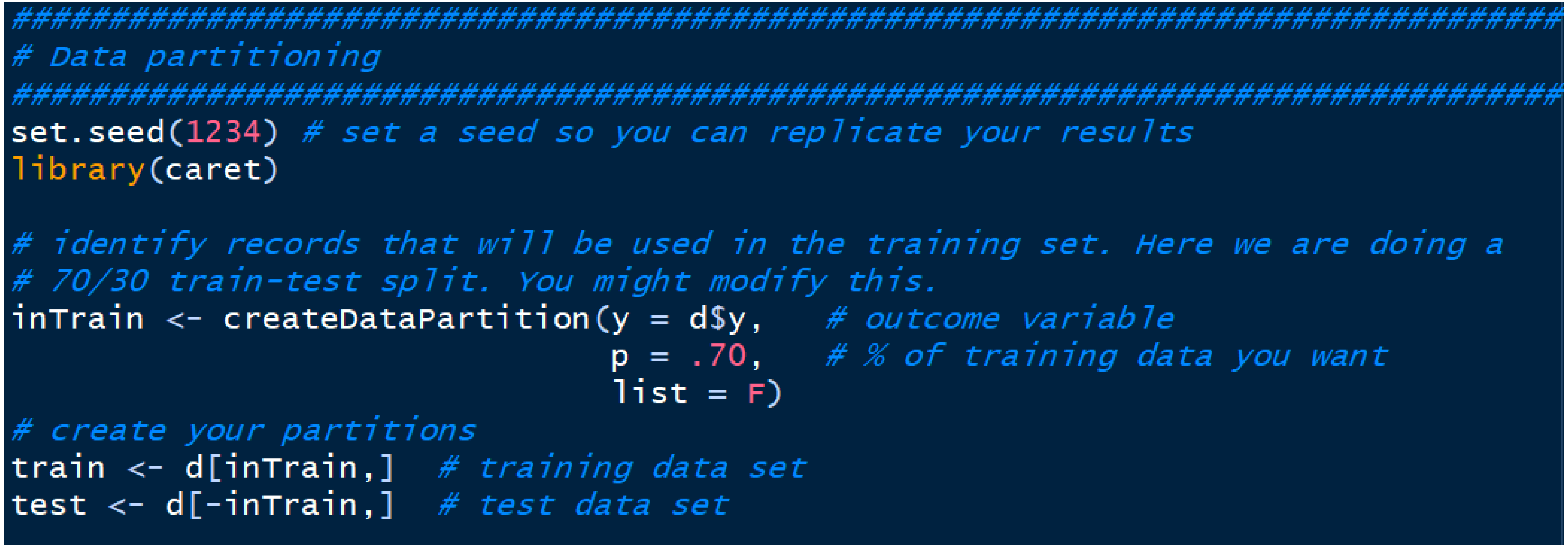
createDataPartition()

5) createDataPartition() function allows one to easily partition their data into training and test sets that are distributed (or imbalanced) similar to one another.



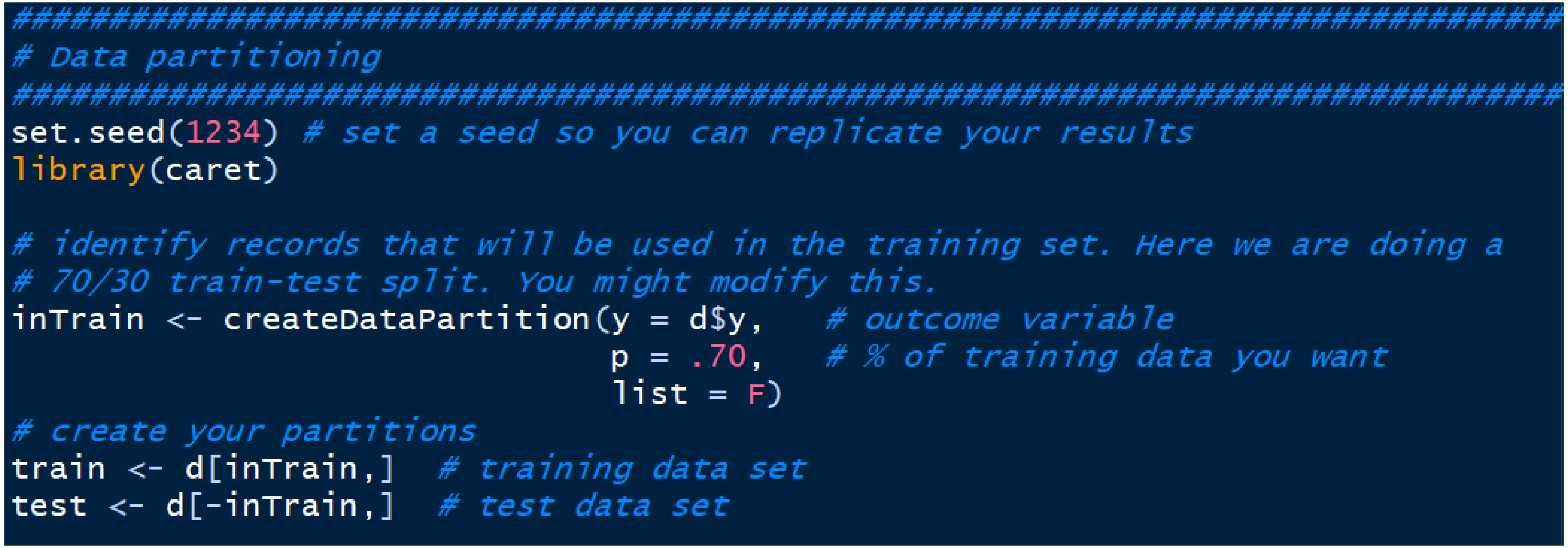
createDataPartition()

5) createDataPartition() function allows one to easily partition their data into training and test sets that are distributed (or imbalanced) similar to one another.

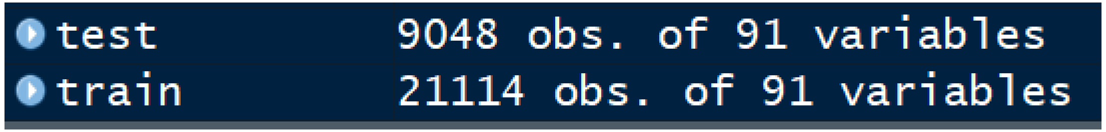


```
#################################################################################
# Data partitioning
#################################################################################
set.seed(1234) # set a seed so you can replicate your results
library(caret)

# identify records that will be used in the training set. Here we are doing a
# 70/30 train-test split. You might modify this.
inTrain <- createDataPartition(y = d$y,   # outcome variable
                               p = .70,   # % of training data you want
                               list = F)
# create your partitions
train <- d[inTrain,]  # training data set
test <- d[-inTrain,]  # test data set
```

createDataPartition()



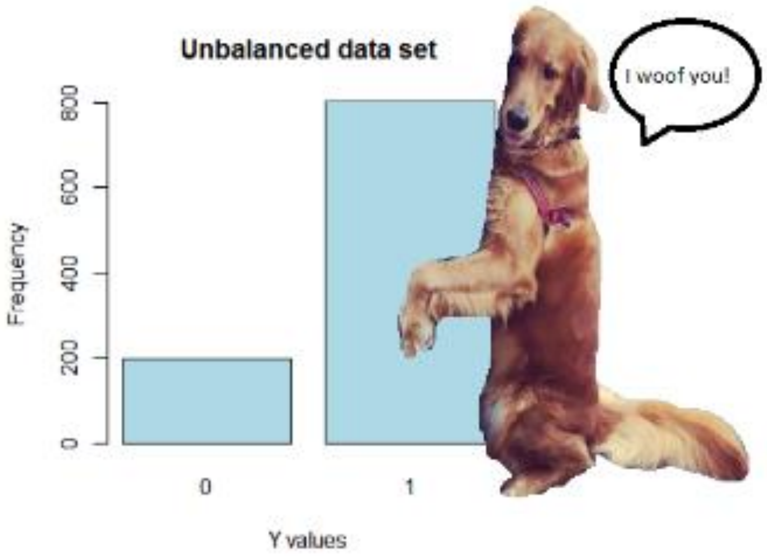
```
###########################################################################
# Data partitioning
###########################################################################
set.seed(1234) # set a seed so you can replicate your results
library(caret)

# identify records that will be used in the training set. Here we are doing a
# 70/30 train-test split. You might modify this.
inTrain <- createDataPartition(y = d$y,   # outcome variable
                               p = .70,   # % of training data you want
                               list = F)
# create your partitions
train <- d[inTrain,]  # training data set
test <- d[-inTrain,]  # test data set
```



```
test         9048 obs. of 91 variables
train        21114 obs. of 91 variables
```

upSample(), downSample()

6) upSample(), downSample() allows you to up or down sample your training data if it is severely unbalanced



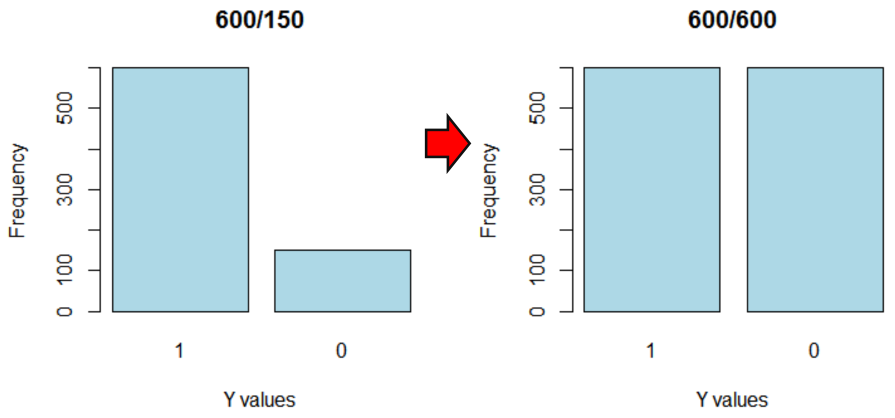
Algorithms love the majority class like Golden Retrievers love tennis balls

upSample(), downSample()

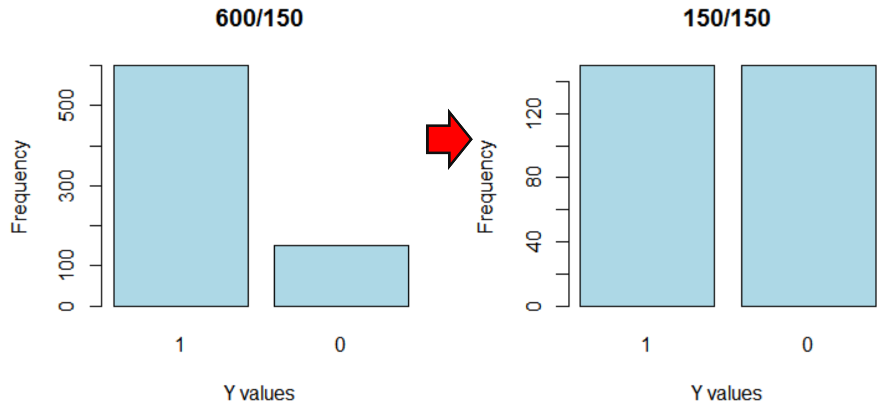
Up-sampling resamples the minority class with replacement until the same number of records exist for the minority class as the majority class

Down-sampling samples the majority class until the same number of records exist for the majority class as the minority class

Up-sampling

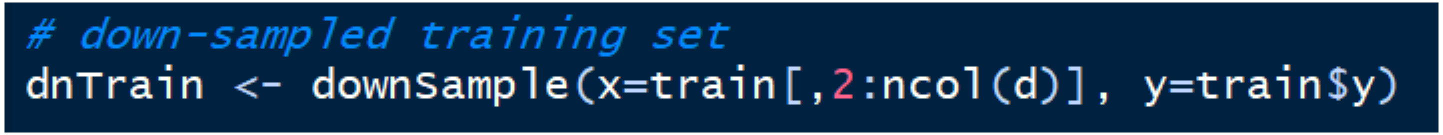


Down-sampling

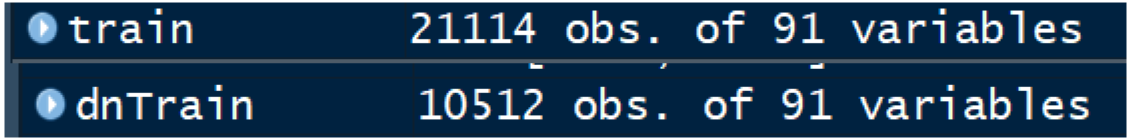


upSample(), downSample()

Here is our down-sampled training set



```r
# down-sampled training set
dnTrain <- downSample(x=train[,2:ncol(d)], y=train$y)
```



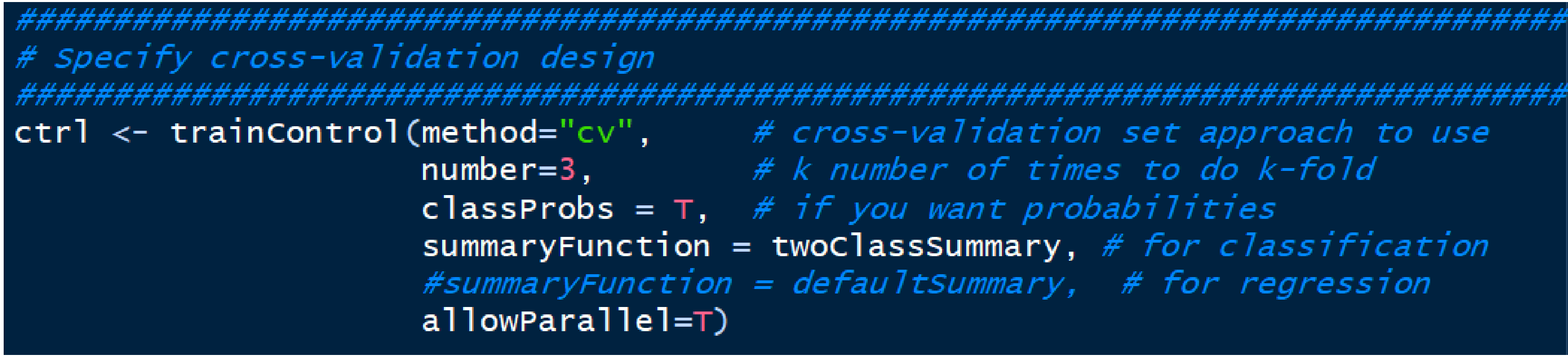
```
train      21114 obs. of 91 variables
dnTrain    10512 obs. of 91 variables
```

trainControl()

7) trainControl() is where you specify how you want to design your run

Here you specify if your problem is (1) a regression or (2) classification

Also, how you want to train your model (e.g. validation set approach, k-Fold cv, LOOCV, etc.)



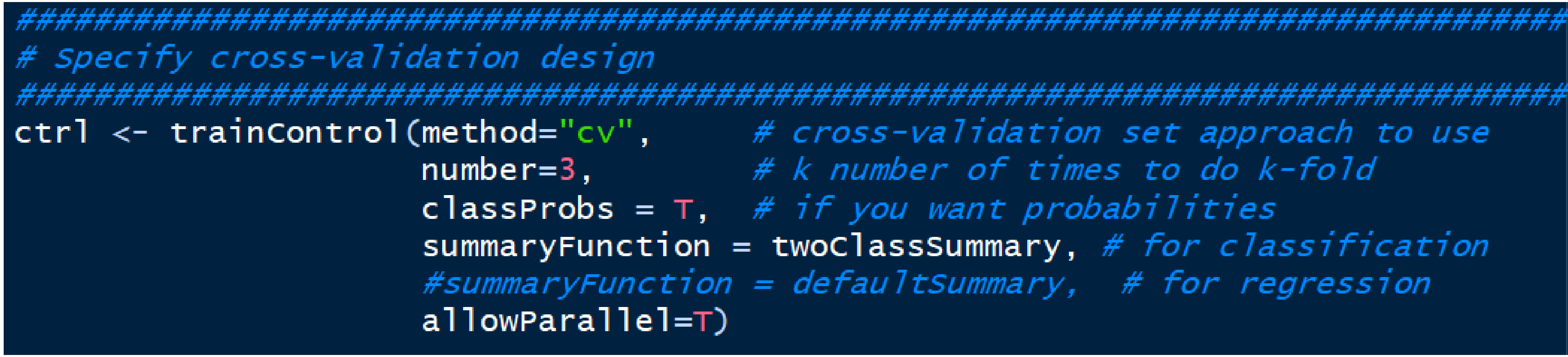
```
############################################################################
# Specify cross-validation design
############################################################################
ctrl <- trainControl(method="cv",    # cross-validation set approach to use
                    number=3,       # k number of times to do k-fold
                    classProbs = T, # if you want probabilities
                    summaryFunction = twoClassSummary, # for classification
                    #summaryFunction = defaultSummary,  # for regression
                    allowParallel=T)
```

trainControl()

7) trainControl() is where you specify how you want to design your run

Here you specify if your problem is (1) a regression or (2) classification

Also, how you want to train your model (e.g. validation set approach, k-Fold cv, LOOCV, etc.)



```
################################################################################
# Specify cross-validation design
################################################################################
ctrl <- trainControl(method="cv",     # cross-validation set approach to use
                     number=3,        # k number of times to do k-fold
                     classProbs = T,  # if you want probabilities
                     summaryFunction = twoClassSummary, # for classification
                     #summaryFunction = defaultSummary,  # for regression
                     allowParallel=T)
```

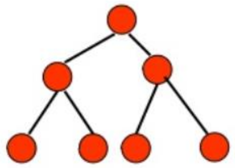
train()

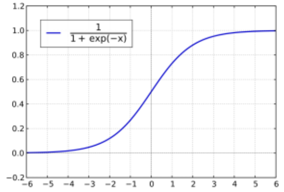
8) train() is the bread and butter of what makes the caret package so great. It's a wrapper for essentially every (not all) regression or classification modeling technique

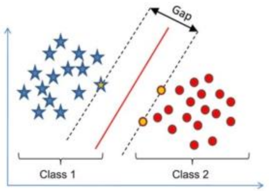
"Wrapper"?

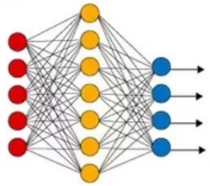
train() idea…

train() idea…

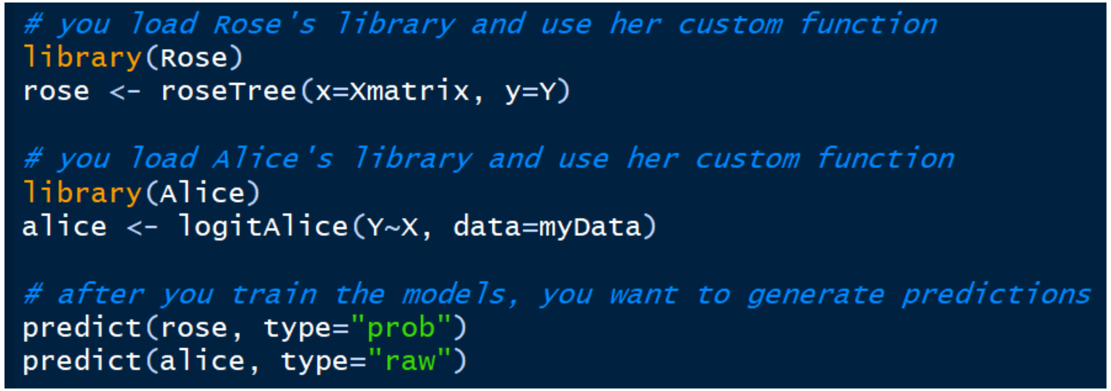




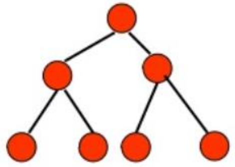


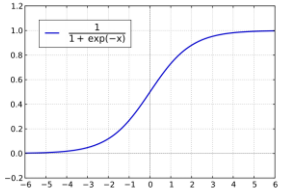


train() idea…



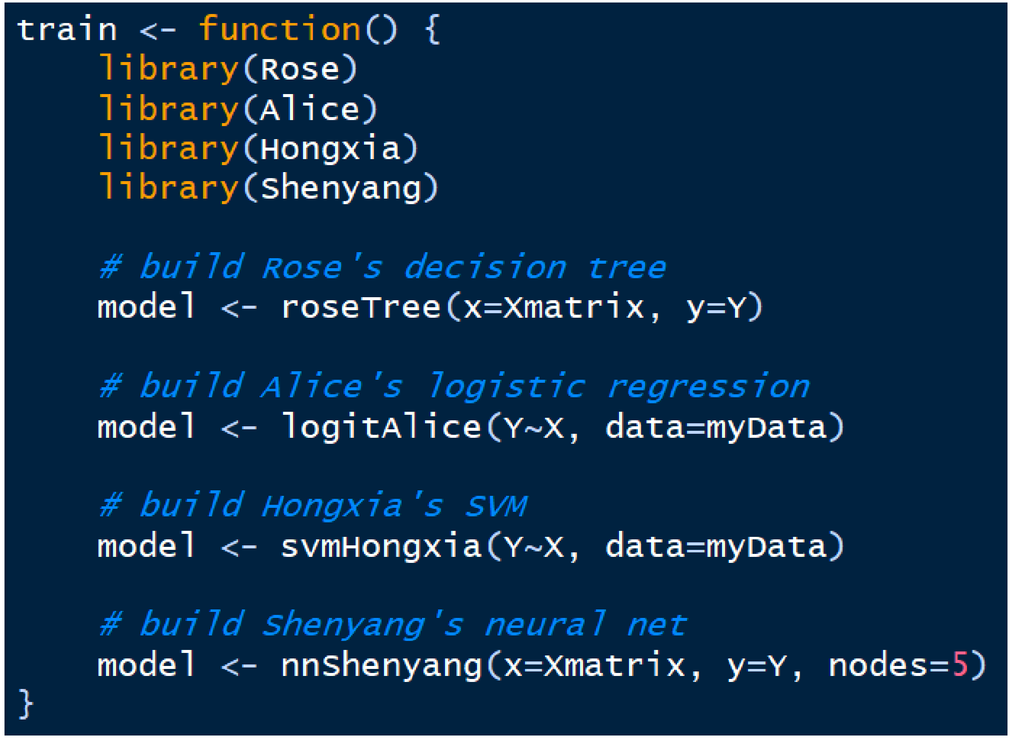
```r
# you load Rose's library and use her custom function
library(Rose)
rose <- roseTree(x=Xmatrix, y=Y)

# you load Alice's library and use her custom function
library(Alice)
alice <- logitAlice(Y~X, data=myData)

# after you train the models, you want to generate predictions
predict(rose, type="prob")
predict(alice, type="raw")
```





No pre-specified design pattern so there are inconsistencies in how one sets functional inputs and what is returned in the output

train() idea…

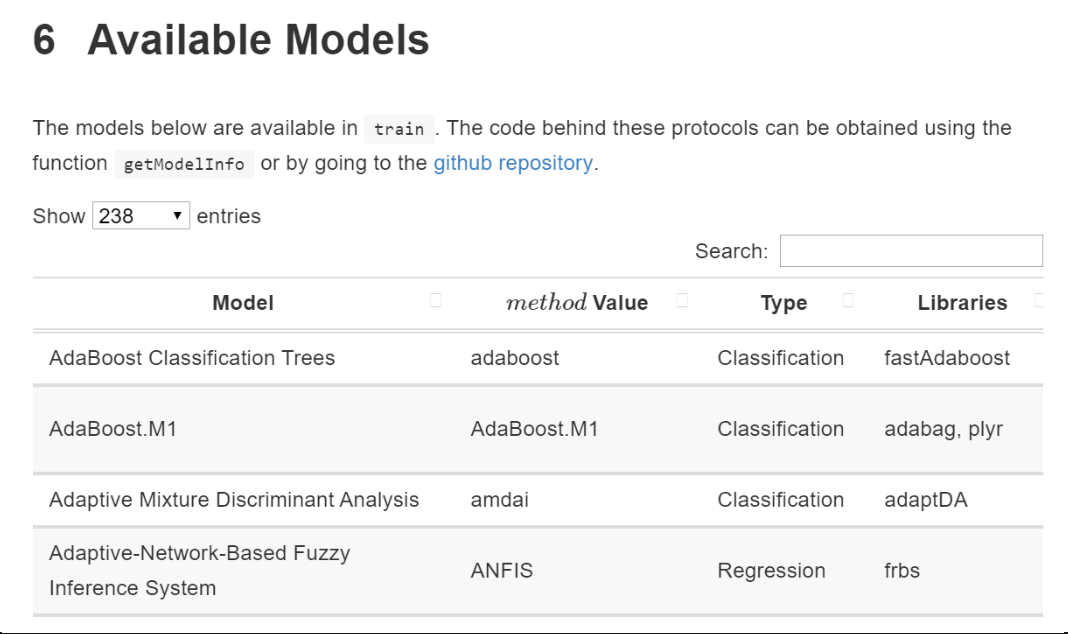


```r
train <- function() {
    library(Rose)
    library(Alice)
    library(Hongxia)
    library(Shenyang)

    # build Rose's decision tree
    model <- roseTree(x=Xmatrix, y=Y)

    # build Alice's logistic regression
    model <- logitAlice(Y~X, data=myData)

    # build Hongxia's SVM
    model <- svmHongxia(Y~X, data=myData)

    # build Shenyang's neural net
    model <- nnShenyang(x=Xmatrix, y=Y, nodes=5)
}
```

train() essentially wraps the packages (or writes a function around them) so there is a consistent way to input the arguments

train()

All the available wrapped models for train are here:
https://topepo.github.io/caret/available-models.html



6 Available Models

The models below are available in `train`. The code behind these protocols can be obtained using the function `getModelInfo` or by going to the github repository.

Show 238 entries

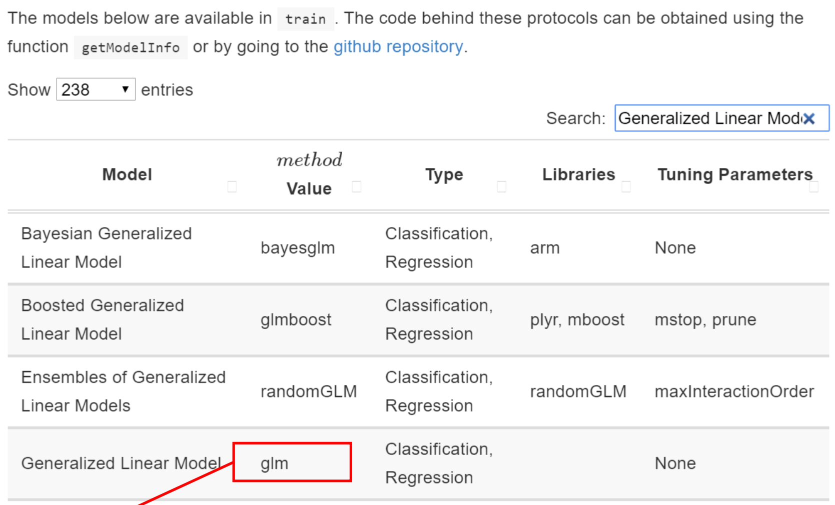
Search:

Model	*method* Value	Type	Libraries
AdaBoost Classification Trees	adaboost	Classification	fastAdaboost
AdaBoost.M1	AdaBoost.M1	Classification	adabag, plyr
Adaptive Mixture Discriminant Analysis	amdai	Classification	adaptDA
Adaptive-Network-Based Fuzzy Inference System	ANFIS	Regression	frbs

train()

Our first model is a logistic regression training on the train set (not down-sampled dataset)

Because glm is a family of different models, you need to specify which one (e.g. binomial/logit link) as an additional argument.

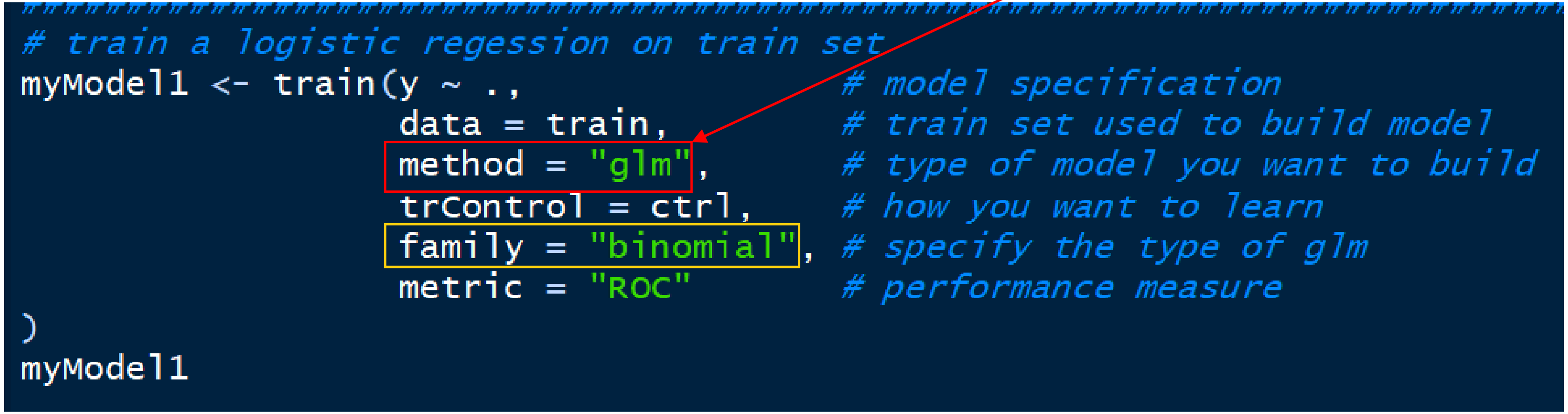


The models below are available in `train`. The code behind these protocols can be obtained using the function `getModelInfo` or by going to the github repository.

Show 238 entries

Search: Generalized Linear Mod

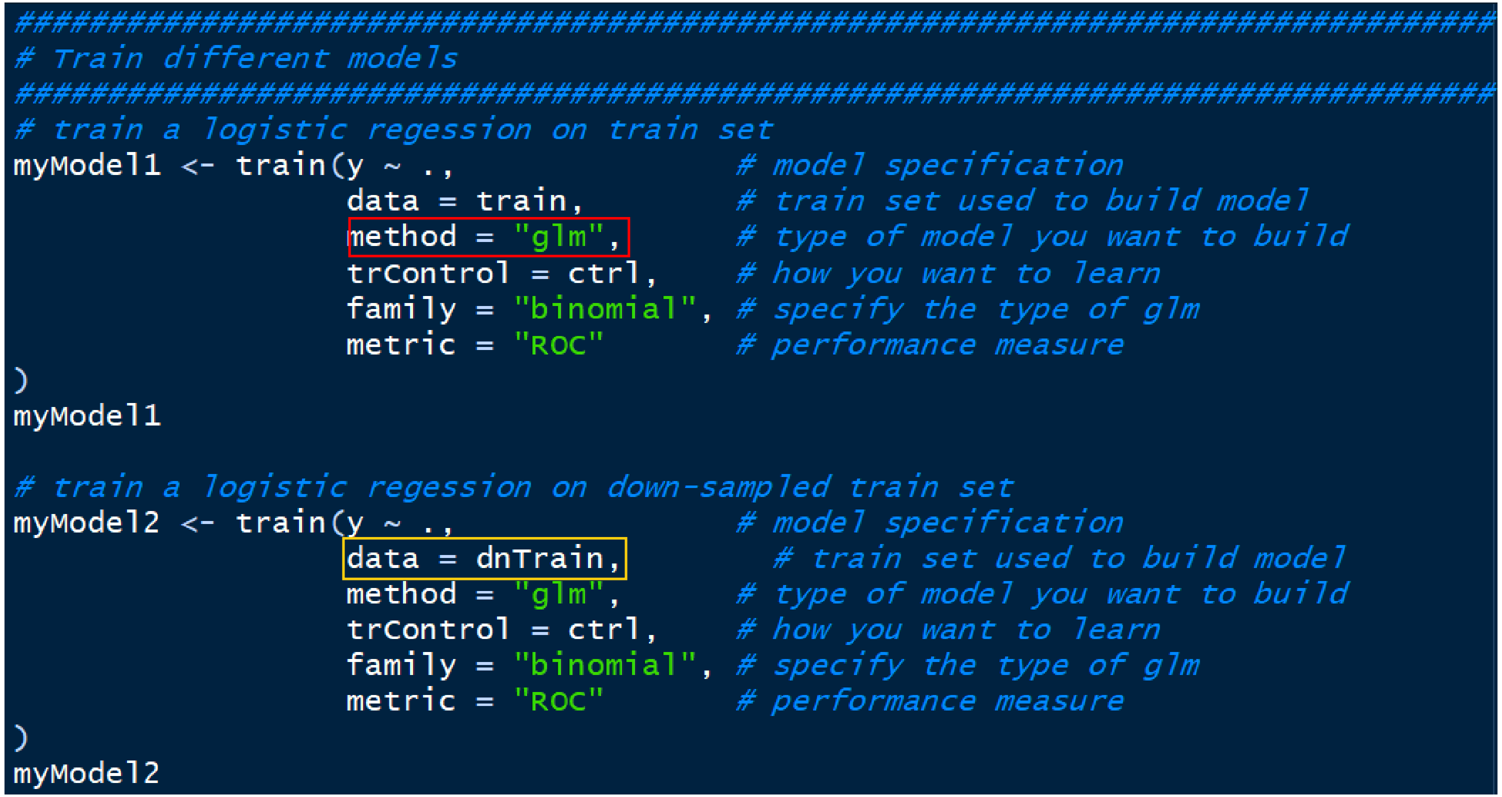
Model	method Value	Type	Libraries	Tuning Parameters
Bayesian Generalized Linear Model	bayesglm	Classification, Regression	arm	None
Boosted Generalized Linear Model	glmboost	Classification, Regression	plyr, mboost	mstop, prune
Ensembles of Generalized Linear Models	randomGLM	Classification, Regression	randomGLM	maxInteractionOrder
Generalized Linear Model	glm	Classification, Regression		None



```
# train a logistic regession on train set
myModel1 <- train(y ~ .,                # model specification
                  data = train,         # train set used to build model
                  method = "glm",       # type of model you want to build
                  trControl = ctrl,     # how you want to learn
                  family = "binomial",  # specify the type of glm
                  metric = "ROC"        # performance measure
)
myModel1
```

train()

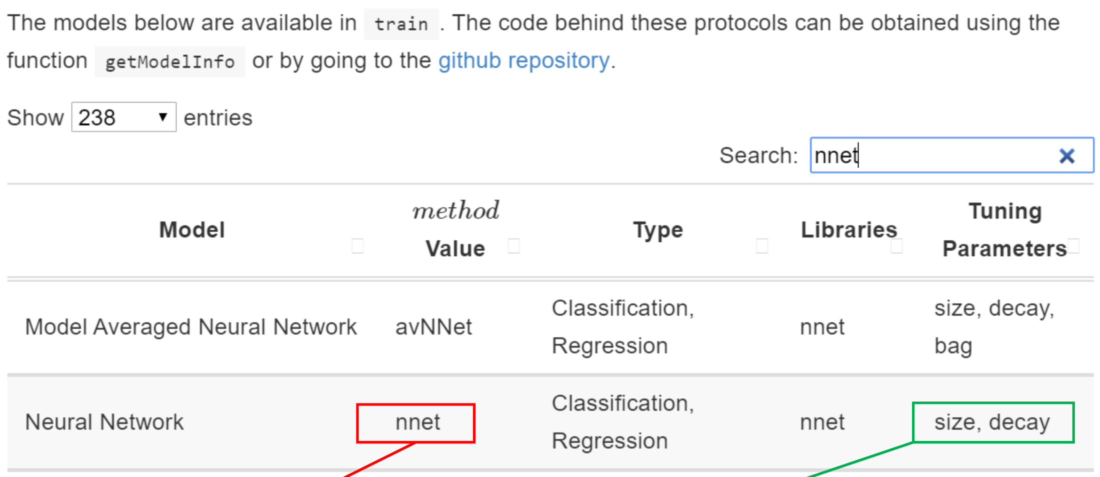
Specify method



```
################################################################################
# Train different models
################################################################################
# train a logistic regession on train set
myModel1 <- train(y ~ .,                  # model specification
                  data = train,           # train set used to build model
                  method = "glm",         # type of model you want to build
                  trControl = ctrl,       # how you want to learn
                  family = "binomial",    # specify the type of glm
                  metric = "ROC"          # performance measure
)
myModel1

# train a logistic regession on down-sampled train set
myModel2 <- train(y ~ .,                  # model specification
                  data = dnTrain,           # train set used to build model
                  method = "glm",         # type of model you want to build
                  trControl = ctrl,       # how you want to learn
                  family = "binomial",    # specify the type of glm
                  metric = "ROC"          # performance measure
)
myModel2
```

train()

Maybe we'll try a feed-forward neural net.

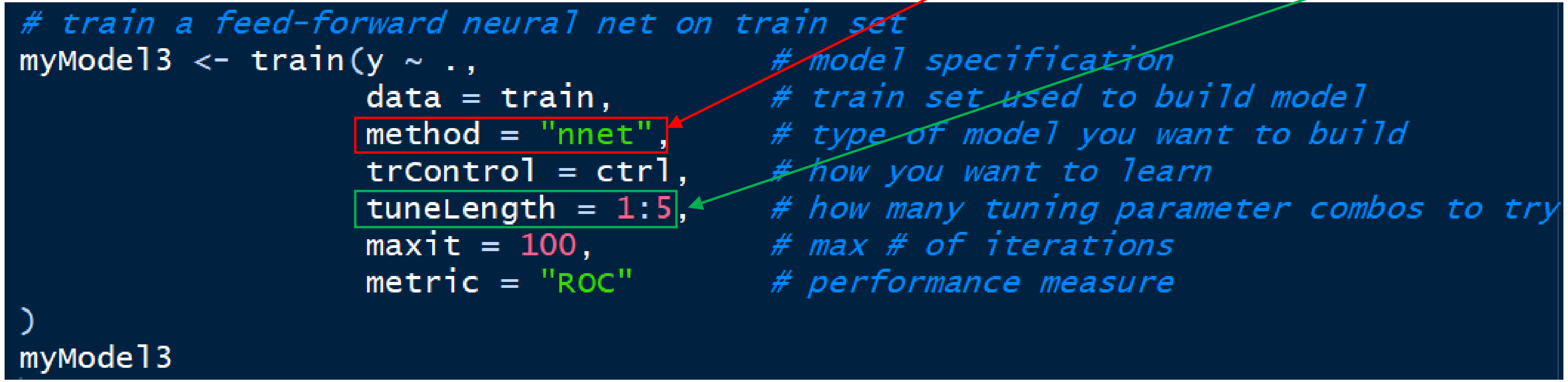


The models below are available in `train`. The code behind these protocols can be obtained using the function `getModelInfo` or by going to the github repository.

Show 238 entries

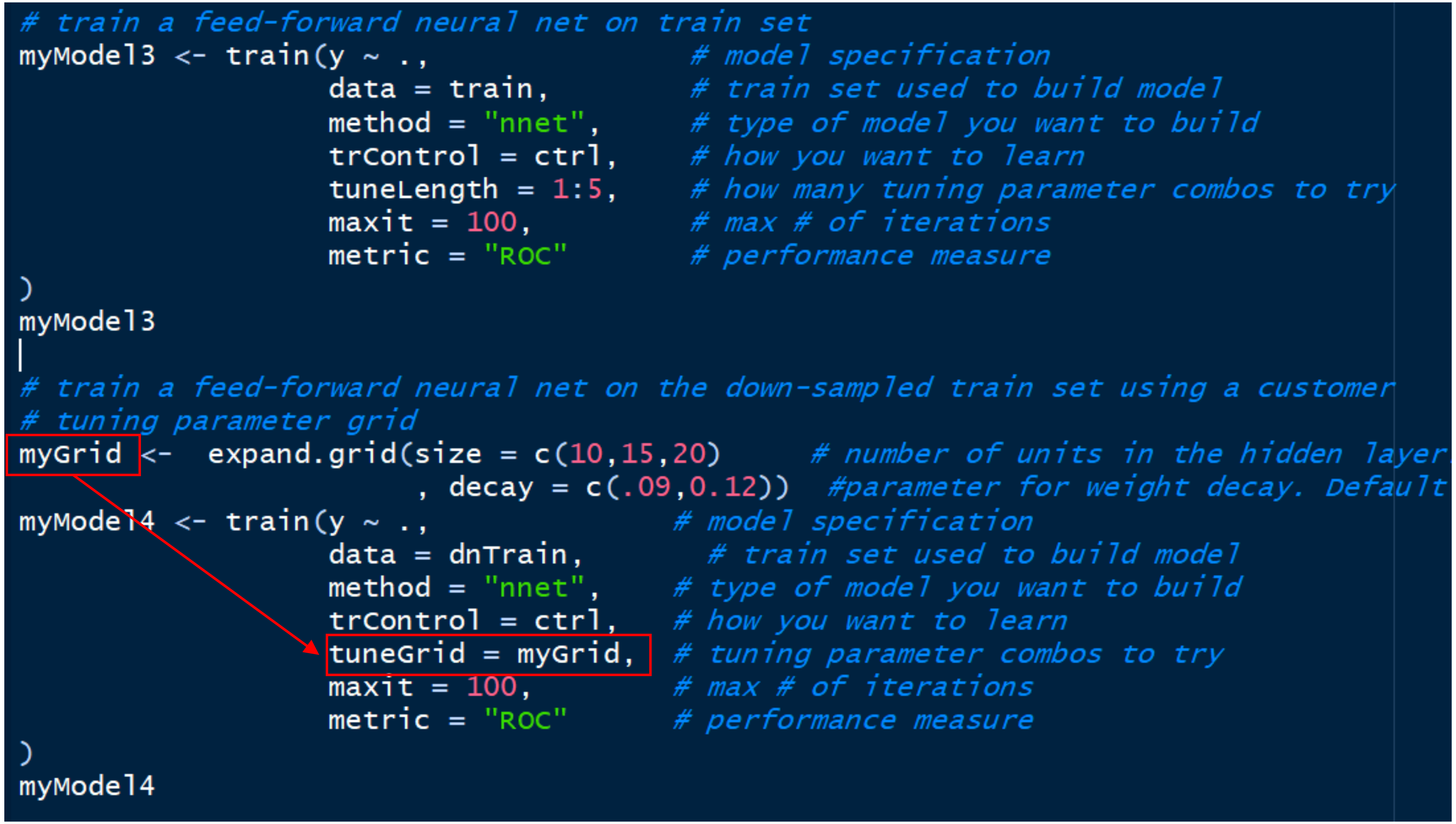
Search: nnet

Model	*method* Value	Type	Libraries	Tuning Parameters
Model Averaged Neural Network	avNNet	Classification, Regression	nnet	size, decay, bag
Neural Network	nnet	Classification, Regression	nnet	size, decay



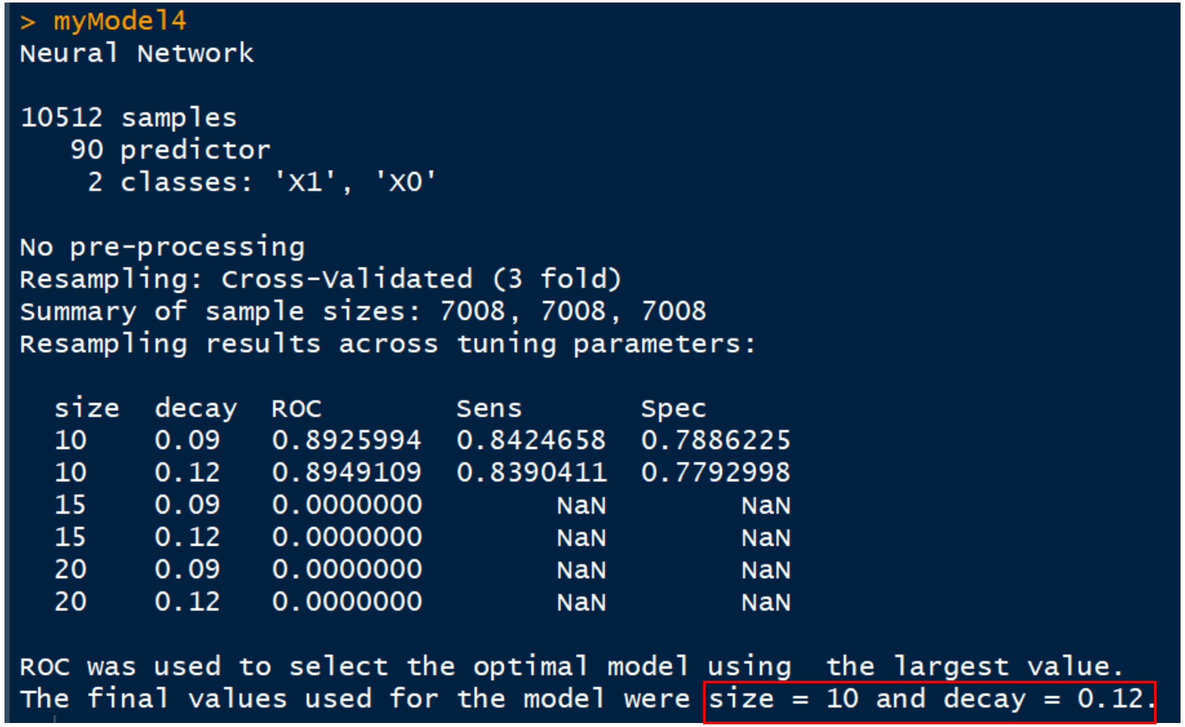
```
# train a feed-forward neural net on train set
myModel3 <- train(y ~ .,                 # model specification
                  data = train,          # train set used to build model
                  method = "nnet",       # type of model you want to build
                  trControl = ctrl,      # how you want to learn
                  tuneLength = 1:5,      # how many tuning parameter combos to try
                  maxit = 100,           # max # of iterations
                  metric = "ROC"         # performance measure
)
myModel3
```

train()



```
# train a feed-forward neural net on train set
myModel3 <- train(y ~ .,                   # model specification
                  data = train,            # train set used to build model
                  method = "nnet",         # type of model you want to build
                  trControl = ctrl,        # how you want to learn
                  tuneLength = 1:5,        # how many tuning parameter combos to try
                  maxit = 100,             # max # of iterations
                  metric = "ROC"           # performance measure
)
myModel3

# train a feed-forward neural net on the down-sampled train set using a customer
# tuning parameter grid
myGrid <-  expand.grid(size = c(10,15,20)      # number of units in the hidden layer
                    , decay = c(.09,0.12))   #parameter for weight decay. Default
myModel4 <- train(y ~ .,                  # model specification
                  data = dnTrain,           # train set used to build model
                  method = "nnet",        # type of model you want to build
                  trControl = ctrl,       # how you want to learn
                  tuneGrid = myGrid,      # tuning parameter combos to try
                  maxit = 100,            # max # of iterations
                  metric = "ROC"          # performance measure
)
myModel4
```

train()



```
> myModel4
Neural Network

10512 samples
   90 predictor
    2 classes: 'X1', 'X0'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 7008, 7008, 7008
Resampling results across tuning parameters:

  size  decay  ROC        Sens       Spec
  10    0.09   0.8925994  0.8424658  0.7886225
  10    0.12   0.8949109  0.8390411  0.7792998
  15    0.09   0.0000000        NaN        NaN
  15    0.12   0.0000000        NaN        NaN
  20    0.09   0.0000000        NaN        NaN
  20    0.12   0.0000000        NaN        NaN

ROC was used to select the optimal model using  the largest value.
The final values used for the model were size = 10 and decay = 0.12.
```

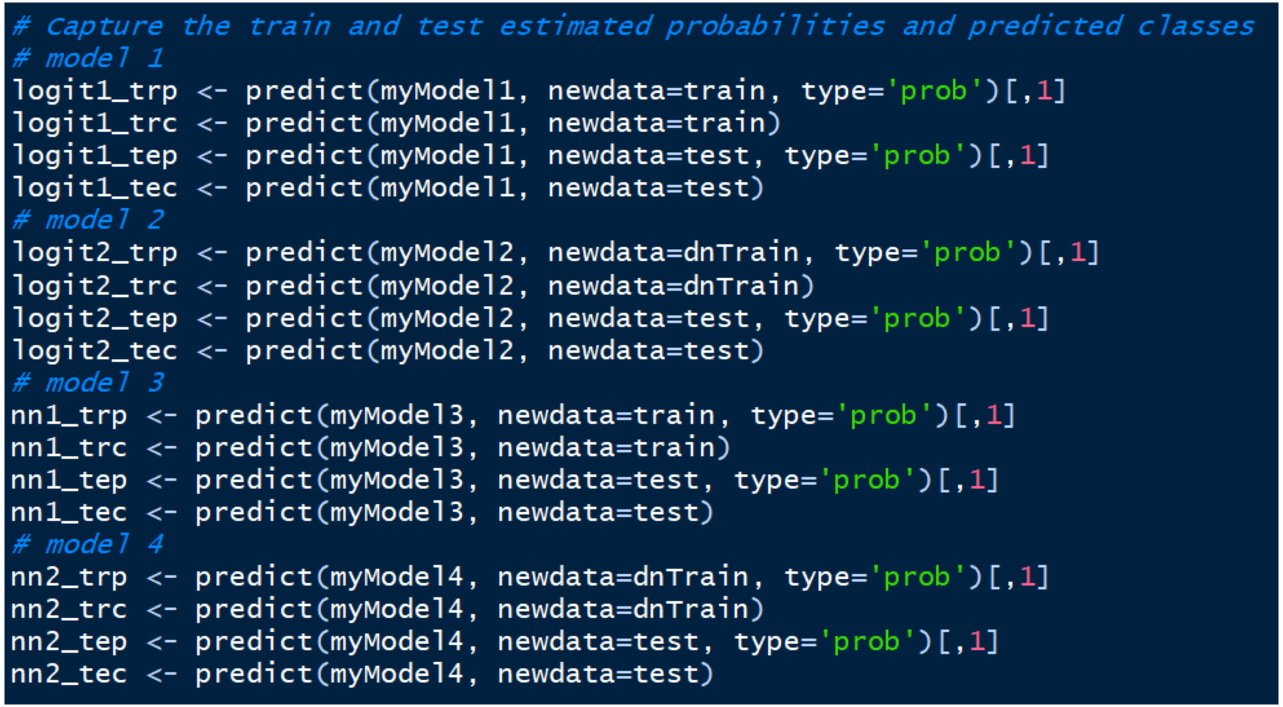
Shows the best set of tuning parameters at the bottom.

caret evaluation functions

9) **predict()** allows you to generate your predictions from a training set or testing set
10) **confusionMatrix()** provides a convenient function to assess classification model performance

predict()

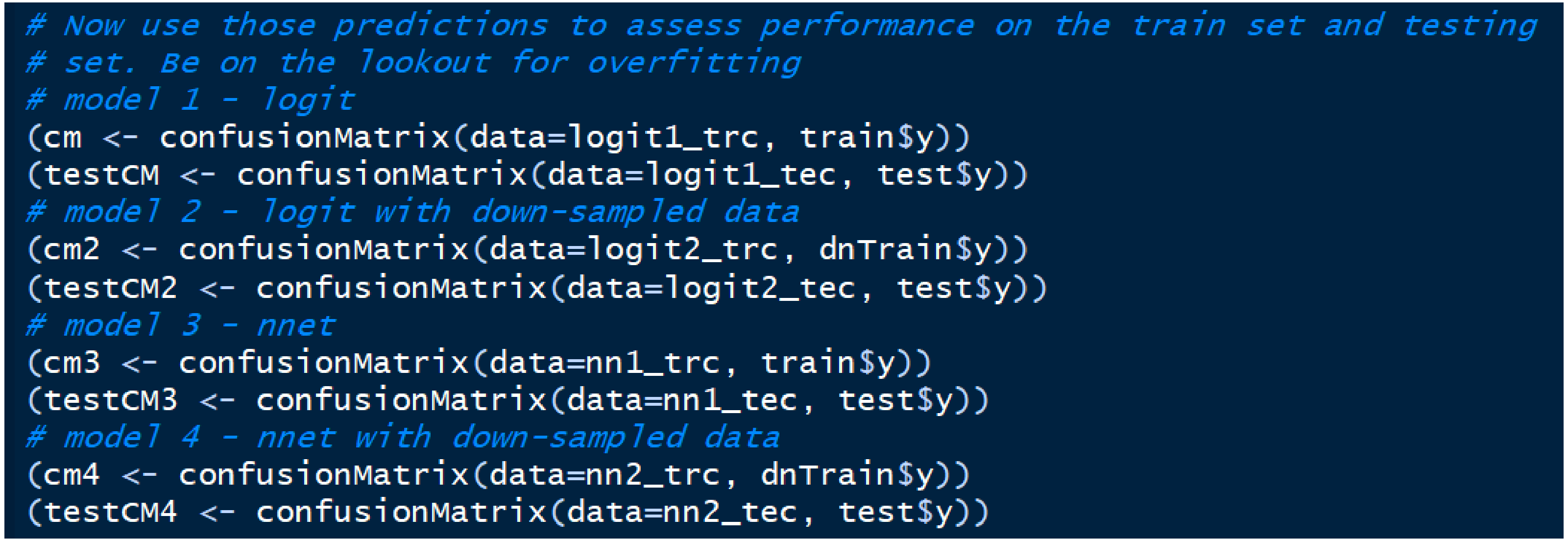
9) **predict()** allows you to generate your predictions from a training set or testing set



```
# Capture the train and test estimated probabilities and predicted classes
# model 1
logit1_trp <- predict(myModel1, newdata=train, type='prob')[,1]
logit1_trc <- predict(myModel1, newdata=train)
logit1_tep <- predict(myModel1, newdata=test, type='prob')[,1]
logit1_tec <- predict(myModel1, newdata=test)
# model 2
logit2_trp <- predict(myModel2, newdata=dnTrain, type='prob')[,1]
logit2_trc <- predict(myModel2, newdata=dnTrain)
logit2_tep <- predict(myModel2, newdata=test, type='prob')[,1]
logit2_tec <- predict(myModel2, newdata=test)
# model 3
nn1_trp <- predict(myModel3, newdata=train, type='prob')[,1]
nn1_trc <- predict(myModel3, newdata=train)
nn1_tep <- predict(myModel3, newdata=test, type='prob')[,1]
nn1_tec <- predict(myModel3, newdata=test)
# model 4
nn2_trp <- predict(myModel4, newdata=dnTrain, type='prob')[,1]
nn2_trc <- predict(myModel4, newdata=dnTrain)
nn2_tep <- predict(myModel4, newdata=test, type='prob')[,1]
nn2_tec <- predict(myModel4, newdata=test)
```

confusionMatrix()

10) confusionMatrix() provides a convenient function to assess classification model performance

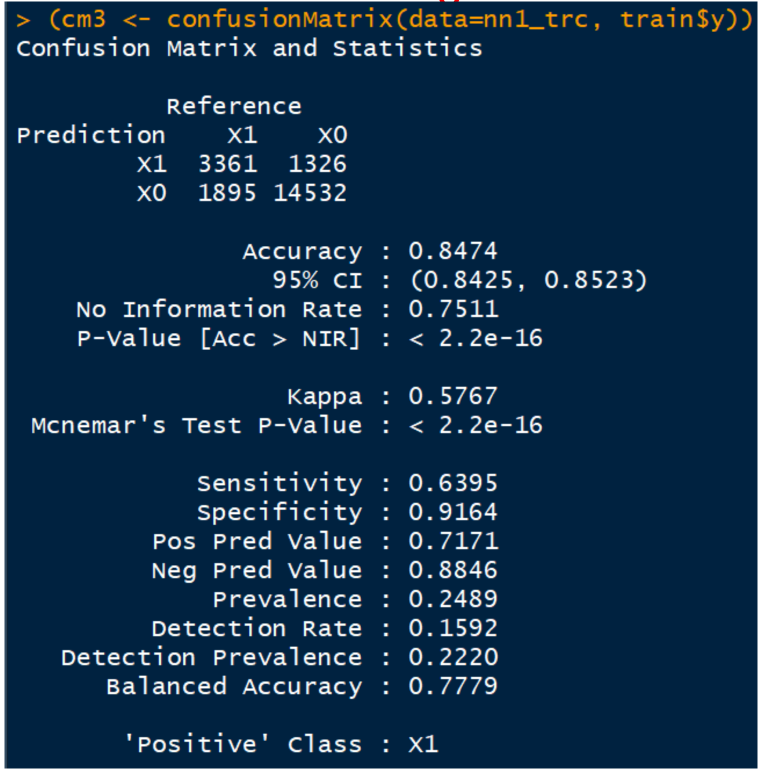


```
# Now use those predictions to assess performance on the train set and testing
# set. Be on the lookout for overfitting
# model 1 - logit
(cm <- confusionMatrix(data=logit1_trc, train$y))
(testCM <- confusionMatrix(data=logit1_tec, test$y))
# model 2 - logit with down-sampled data
(cm2 <- confusionMatrix(data=logit2_trc, dnTrain$y))
(testCM2 <- confusionMatrix(data=logit2_tec, test$y))
# model 3 - nnet
(cm3 <- confusionMatrix(data=nn1_trc, train$y))
(testCM3 <- confusionMatrix(data=nn1_tec, test$y))
# model 4 - nnet with down-sampled data
(cm4 <- confusionMatrix(data=nn2_trc, dnTrain$y))
(testCM4 <- confusionMatrix(data=nn2_tec, test$y))
```

confusionMatrix()

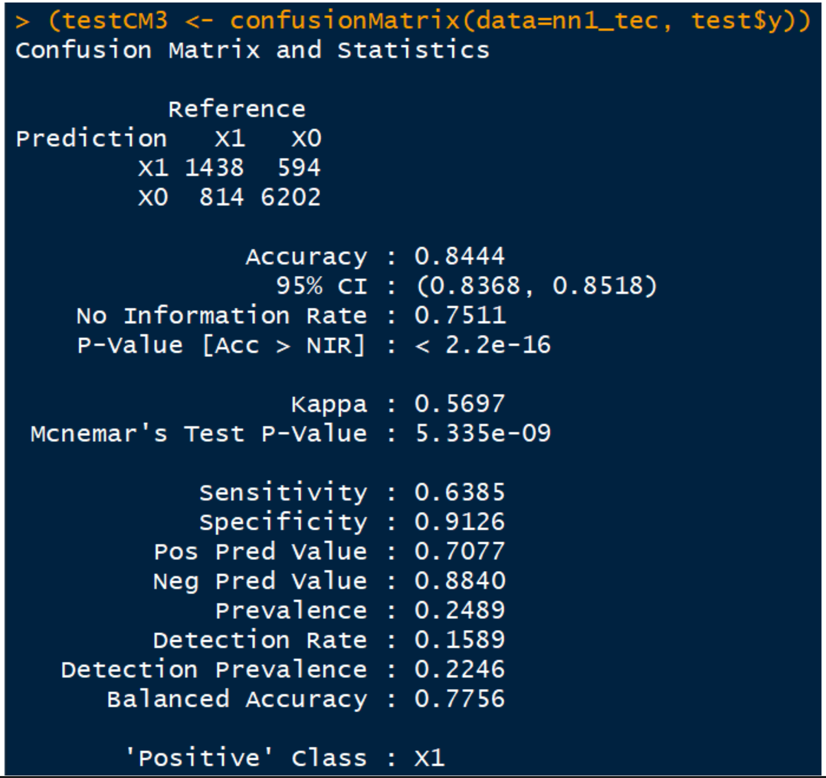
Neural net model without rebalancing

training



```
> (cm3 <- confusionMatrix(data=nn1_trc, train$y))
Confusion Matrix and Statistics

          Reference
Prediction    X1    X0
        X1  3361  1326
        X0  1895 14532

               Accuracy : 0.8474
                 95% CI : (0.8425, 0.8523)
    No Information Rate : 0.7511
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5767
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6395
            Specificity : 0.9164
         Pos Pred Value : 0.7171
         Neg Pred Value : 0.8846
             Prevalence : 0.2489
         Detection Rate : 0.1592
   Detection Prevalence : 0.2220
      Balanced Accuracy : 0.7779

       'Positive' Class : X1
```

test



```
> (testCM3 <- confusionMatrix(data=nn1_tec, test$y))
Confusion Matrix and Statistics

          Reference
Prediction   X1   X0
        X1 1438  594
        X0  814 6202

               Accuracy : 0.8444
                 95% CI : (0.8368, 0.8518)
    No Information Rate : 0.7511
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5697
 Mcnemar's Test P-Value : 5.335e-09

            Sensitivity : 0.6385
            Specificity : 0.9126
         Pos Pred Value : 0.7077
         Neg Pred Value : 0.8840
             Prevalence : 0.2489
         Detection Rate : 0.1589
   Detection Prevalence : 0.2246
      Balanced Accuracy : 0.7756

       'Positive' Class : X1
```

Take Away

- caret provides
  - A nice wrapper to many (not all) R packages for predictive modeling and machine learning
  - Functions to easily pre-process your data before modeling
  - Ability to tune hyperparameters with ease by specifying **tuneLength=** or **tuneGrid=**
  - Can incorporate other specific package arguments, but they might not be part of tuning parameters list
  - Easily evaluate regression and classification-type problems