starter

October 4, 2019

0.1 Homework 06 - Instructions

0.1.1 This is a 30 point assignment graded from answers to questions and automated tests that should be run at the bottom. Be sure to clearly label all of your answers and commit final tests at the end.

0.1.2 For the exercise below you should import the Iris Dataset as a dataframe called iris.df.

(1). Create a variable called sepal_width.mean that contains the mean of the sepal_width column of iris.df.

[1]:
```
# import the Iris dataset as dataframe called iris.df
iris.df = read.csv(file = 'iris.csv', header = TRUE, sep = ',')

# create a variable sepal_width.mean
sepal_width.mean <- mean(iris.df[, 'sepal_width'])

sepal_width.mean
```

3.054

(2). Create a new column of iris.df called `sepalArea` that is equal to the sepal_width times sepal_length.

[2]:
```
iris.df['sepalArea'] <- iris.df$sepal_width * iris.df$sepal_length

head(iris.df)
```

A data.frame: 6 x 6

sepal_length <dbl>	sepal_width <dbl>	petal_length <dbl>	petal_width <dbl>	species <fct>	sepalArea <dbl>
5.1	3.5	1.4	0.2	setosa	17.85
4.9	3.0	1.4	0.2	setosa	14.70
4.7	3.2	1.3	0.2	setosa	15.04
4.6	3.1	1.5	0.2	setosa	14.26
5.0	3.6	1.4	0.2	setosa	18.00
5.4	3.9	1.7	0.4	setosa	21.06

(3). Create a new dataframe `iristrain.df` that includes the first 75 rows of the iris dataframe.

[3]:
```
iristrain.df <- iris.df[1:75, ]

dim(iristrain.df)
```

1. 75 2. 6

(4). Create a new dataframe iristest.df that includes the last 75 rows of the iris dataframe.

[4]:
```
iristest.df <- iris.df[76:nrow(iris.df), ]

dim(iristest.df)
```

1. 75 2. 6

(5). Create a new vector `sepal_length` from the `sepal_length` column of the iris dataframe.

[5]:
```
sepal_length <- iris.df$sepal_length

sepal_length[1:10]
```

1. 5.1 2. 4.9 3. 4.7 4. 4.6 5. 5 6. 5.4 7. 4.6 8. 5 9. 4.4 10. 4.9

(6). While we can submit our answer to Kaggle to see how it will perform, we can also utilize our test data to assess accuracy. Accuracy is the percentage of predictions made correctly-i.e., the percentage of people in which our prediction regarding their survival. a. Create columns in the training dataset PredEveryoneDies and PredGender with the same predictions from above. b. Create variables AccEveryoneDies and AccGender using a calculation of accuracy of predictions for the training dataset.

[6]:
```
# create dataframe in R from test.csv and train.csv
train = read.csv(file = 'train.csv', header = TRUE, sep = ',')
test = read.csv(file = 'test.csv', header = TRUE, sep = ',')

# explore these two dataframe
#dim(train)
#names(train)

#dim(test)
#names(test)

# a. Create columns in the training dataset PredEveryoneDies and PredGender with␣
 ↪the same predictions from above
train['PredEveryoneDies'] <- 0

train['PredGender'] <- 0
train[train$Sex == 'female', 'PredGender'] <- 1

# b. Create variables AccEveryoneDies and AccGender using a calculation of␣
 ↪accuracy of predictions for the training dataset.
Calculate_Accuracy <- function(prediction, actual) {
    accuracy <- 0
    correct_predictions <- 0
    num_predictions <- length(prediction)

    for (i in 1:num_predictions) {
        if (prediction[i] == actual[i]) {
            correct_predictions <- correct_predictions + 1
        }
    }
```

```
    accuracy <- correct_predictions/num_predictions*100
}

AccEveryoneDies <- Calculate_Accuracy(train$PredEveryoneDies, train$Survived)
AccGender <- Calculate_Accuracy(train$PredGender, train$Survived)

AccEveryoneDies
AccGender
```

61.6161616161616

78.675645342312

(7). Notice how we are utilizing the code to select out the passengerID and the Survived column and generating a submission file over and over? This is in need of a function. Create a generate_submission function that accepts a DataFrame, a target column, and a filename and writes out the submission file with just the passengerID and the Survived columns, where the survived column is equal to the target column. It should then return a DataFrame with the passengerID and the Survived columns.

Executeing the following should return a dataframe with just passengerID and the Survived column:

submitdie <- generate_submission(train, "PredEveryoneDies", "submiteveryonedies.csv")

[7]:
```
generate_submission <- function(df, target_col, filename) {
    submission <- df[, c('PassengerId', 'Survived')]
    submission$Survived <- target_col
    write.csv(submission, file = filename)
}

submitdie <- generate_submission(train, "PredEveryoneDies", "submiteveryonedies.
 ↪csv")
```

(8). In according to the women and children first protocol we hypothesize that our model could be improved by including whether the individual was a child in addition to gender. After coding survival based on gender, update your recommendation to prediction in the training dataset survival based on age. `train['PredGenderAge13']` should be the prediction incorporating both Gender and whether `Age` < 13. train['PredGenderAge18'] should be the prediction incorporating both Gender and whether `Age` < 18. `AccGenderAge13` should be the accuracy of the age prediction, based on `train['PredGenderAge13']`. `AccGenderAge18` should be the accuracy of the age prediction, based on `train['PredGenderAge18']`.

[8]:
```
# deal with missing values in Age, since "missing values are not allowed in␣
 ↪subscripted assignments of data frames"
average_age <- mean(train$Age, na.rm = TRUE)
train[is.na(train$Age), 'Age'] <- average_age

average_age_test <- mean(test$Age, na.rm = TRUE)
test[is.na(test$Age), 'Age'] <- average_age

# Make new predictions
```

```
train['PredGenderAge13'] <- 0
train[(train$Sex == 'female')|(train$Age <13), 'PredGenderAge13'] <- 1

train['PredGenderAge18'] <- 0
train[(train$Sex == 'female')|(train$Age <18), 'PredGenderAge18'] <- 1

# Calculate Accuracy
AccGenderAge13 <- Calculate_Accuracy(train$PredGenderAge13, train$Survived)
AccGenderAge18 <- Calculate_Accuracy(train$PredGenderAge18, train$Survived)

AccGenderAge13
AccGenderAge18
```

79.2368125701459

77.3288439955107

(9). You should find that the AccGenderAge13 is better than AccGenderAge18. Create a new column child in the test and train DataFrames that is 1 if Age < 13 and 0 otherwise. This is a feature.

[9]:
```
#head(train)

train['child'] <- 0
train[train$Age < 13, 'child'] <- 1

test['child'] <- 0
test[test$Age < 13, 'child'] <- 1

#head(train[train$child==1,])
```

0.1.3 Run Tests

This is a bit different but it will run each of the associated tests for this assignment.

[10]:
```
install.packages('testthat', lib='/home/nbuser/R')
```

[11]:
```
sink("test.md")
library('testthat')
test_file("test.intro-r-exercises.R", reporter = "tap")
sink()
```

```
1..20
# Context Homework 5
ok 1 Q1 Create a variable called sepal_width.mean
ok 2 Q2Create a new column of iris.df called sepalArea
ok 3 Q3 Create a new dataframe iristrain.df that includes the first 75 rows
ok 4 Q3 Create a new dataframe iristrain.df that includes the first 75 rows
ok 5 Q4 Create a new dataframe iristest.df that includes the last 75 rows of the
iris dataframe.
ok 6 Q4 Create a new dataframe iristest.df that includes the last 75 rows of the
iris dataframe.
```

```
ok 7 (5). Create a new vector sepal_length from the sepal_length column of the
iris dataframe.
ok 8 (5). Create a new vector sepal_length from the sepal_length column of the
iris dataframe.
ok 9 (5). Create a new vector sepal_length from the sepal_length column of the
iris dataframe.
ok 10 (5). Create a new vector sepal_length from the sepal_length column of the
iris dataframe.
ok 11 (6). AccEveryoneDies and AccGender
ok 12 (6). AccEveryoneDies and AccGender
ok 13 (6). AccEveryoneDies and AccGender
ok 14 (6). AccEveryoneDies and AccGender
ok 15 (8). PredGenderAge13 and PredGenderAge18
ok 16 (8). PredGenderAge13 and PredGenderAge18
ok 17 (8). PredGenderAge13 and PredGenderAge18
ok 18 (8). PredGenderAge13 and PredGenderAge18
ok 19 (9). Child
ok 20 (9). Child
```

[ ]: