



FINAL-TERM PROJECT REPORT

Introduction To Data Science

Faculty

TOHEDUL ISLAM

Assistant Professor, Computer Science Dept.

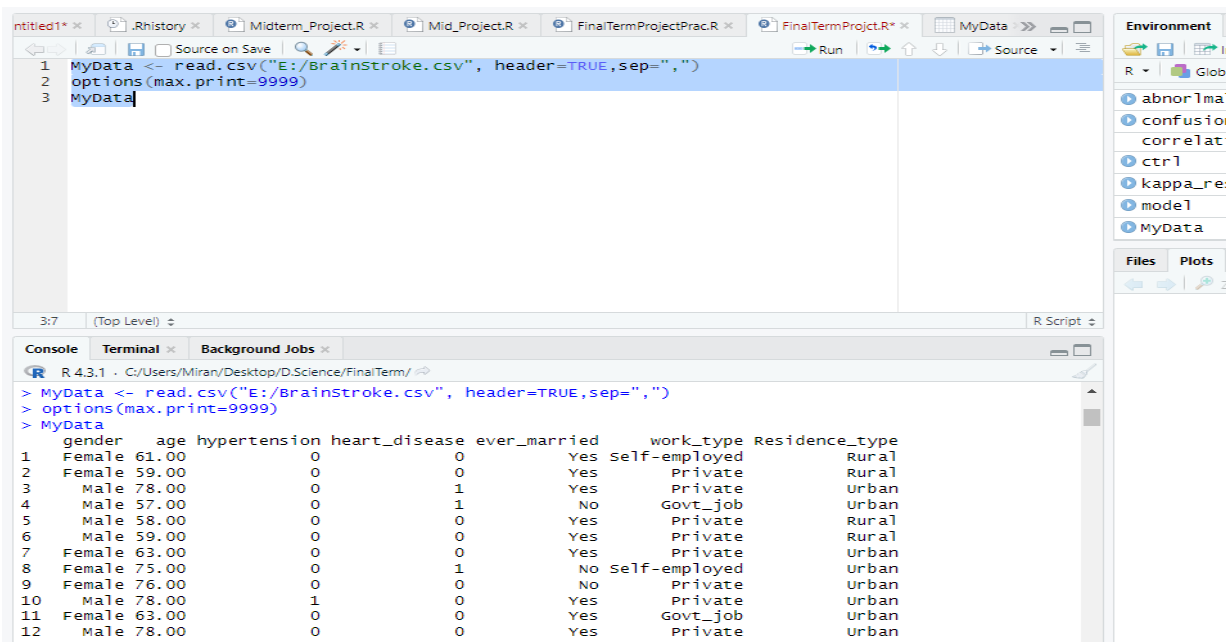
MD. ABDULLAH AL NASIR

19-41052-2

About Dataset: Here, the dataset named "Brain Stroke" has 202 instances and 11 attributes. Attributes are "Gender", "age", "hypertension", "heart_disease", "ever_married", "work_type", "Residence_type", "avg_glucose_level", "bmi", "smoking_status", "stroke".

Here, five attributes are categorical. Those are: "gender", "ever_married", "work_type", "Residence_type", "smoking_status". The attribute names: "age", "hypertension", "heart_disease", "avg_glucose_level", "bmi", "stroke" are numeric. The target attribute of this dataset is "stroke".

Step- 01: In the very first step, the dataset has been imported from the drive in a data frame named MyData using the read function. Using the options () function, we have imported all the data values with the help of max.print=9999.



The screenshot displays the RStudio interface. The top pane shows the R script with the following code:

```
1 MyData <- read.csv("E:/BrainStroke.csv", header=TRUE, sep=",")
2 options(max.print=9999)
3 MyData
```

The bottom pane shows the console output, which displays the first 12 rows of the dataset:

| | gender | age | hypertension | heart_disease | ever_married | work_type | Residence_type |
|----|--------|-------|--------------|---------------|--------------|---------------|----------------|
| 1 | Female | 61.00 | 0 | 0 | Yes | Self-employed | Rural |
| 2 | Female | 59.00 | 0 | 0 | Yes | Private | Rural |
| 3 | Male | 78.00 | 0 | 1 | Yes | Private | Urban |
| 4 | Male | 57.00 | 0 | 1 | No | Govt_job | Urban |
| 5 | Male | 58.00 | 0 | 0 | Yes | Private | Rural |
| 6 | Male | 59.00 | 0 | 0 | Yes | Private | Rural |
| 7 | Female | 63.00 | 0 | 0 | Yes | Private | Urban |
| 8 | Female | 75.00 | 0 | 1 | No | Self-employed | Urban |
| 9 | Female | 76.00 | 0 | 0 | No | Private | Urban |
| 10 | Male | 78.00 | 1 | 0 | Yes | Private | Urban |
| 11 | Female | 63.00 | 0 | 0 | Yes | Govt_job | Urban |
| 12 | Male | 78.00 | 0 | 0 | Yes | Private | Urban |

Step- 02: The syntax of the function `convertCategoricalToNumeric` indicates the beginning of the code. This function expects MyData to be a data frame containing the dataset to be processed.

The function includes a list called `categorical_cols` that lists the column names that are required to be changed from categorical to numeric format. "Gender," "ever_married," "work_type," "Residence_type," and "smoking_status" are the columns listed here.

Each of the categorical columns listed in `categorical_cols` are iterated over in the code's loop. The actions listed below are carried out for each column:

- Factor Conversion:** The factor function converts the values of the column into factors. This step is essential because it assigns a numerical representation to each distinct value in the column.
- Numeric Conversion:** The `as.numeric` function is then used to convert the newly created factors into numeric values. This basically gives every individual factor value a numerical label.

```

1 MyData <- read.csv("E:/BrainStroke.csv", header=TRUE, sep=",")
2 options(max.print=9999)
3 MyData
4
5 convertCategoricalToNumeric <- function(MyData)
6 {
7   categorical_cols <- c("gender", "ever_married", "work_type", "Residence_type", "smoking_status")
8   for(col in categorical_cols)
9   {
10    MyData[[col]] <- as.numeric(factor(MyData[[col]]))
11  }
12  return(MyData)
13 }
14 MyData <- convertCategoricalToNumeric(MyData)
15 MyData

```

Console output:

```

R 4.3.1 - C:/Users/Miran/Desktop/D.Science/FinalTerm/
+ MyData[[col]] <- as.numeric(factor(MyData[[col]]))
+ }
+ return(MyData)
+ }
> MyData <- convertCategoricalToNumeric(MyData)
> MyData
  gender    age hypertension heart_disease ever_married work_type Residence_type
1      1  61.00           0              0           2         4             1
2      1  59.00           0              0           2         3             1
3      2  78.00           0              1           2         3             2
4      2  57.00           0              1           1         2             2
5      2  58.00           0              0           2         3             1
6      2  59.00           0              0           2         3             1
7      1  63.00           0              0           2         3             2

```

Step- 03: The function `colSums ()` is used to calculate the column sums of a matrix or data frame. The function `is.na ()` is used to identify null values (NA) in a vector, matrix, or data frame. To calculate the number of missing values in each column, the function used here is `colSums (is.na ())`, keeping `MyData` as a parameter. Here we got that there are 0 null values in dataset.

```

1 MyData <- read.csv("E:/BrainStroke.csv", header=TRUE, sep=",")
2 options(max.print=9999)
3 MyData
4
5 convertCategoricalToNumeric <- function(MyData)
6 {
7   categorical_cols <- c("gender", "ever_married", "work_type", "Residence_type", "smoking_status")
8   for(col in categorical_cols)
9   {
10    MyData[[col]] <- as.numeric(factor(MyData[[col]]))
11  }
12  return(MyData)
13 }
14 MyData <- convertCategoricalToNumeric(MyData)
15 MyData
16
17 colSums(is.na(MyData))

```

Console output:

```

R 4.3.1 - C:/Users/Miran/Desktop/D.Science/FinalTerm/
197 70.15 29.75663 1 0
198 191.15 31.12417 3 0
199 95.02 31.79830 3 0
200 83.94 29.95130 3 0
201 83.75 29.09742 2 0
> colSums(is.na(MyData))
  gender    age hypertension heart_disease ever_married
0      0           0              0           0             0
  work_type Residence_type avg_glucose_level    bmi smoking_status
0      0           0              0           0             0
  stroke
0

```

Step- 04: Finding the outliers using Z-scores function. The `scale ()` function is used to identify outliers by examining the standardized values of age, `abs ()` return the absolute value of a near. Outliers we found here are 1.32 3.00 6.00 7.00 0.48 5.00 1.88 1.08 1.80 3.00 1.32.

Now the outliers in the array named [age_outlier_indicates] are replaced with NA.

```

8 categorical_cols <- c(gender, ever_married, work_type, residence_type, smoking_status)
9 for(col in categorical_cols)
10 { MyData[[col]] <- as.numeric(factor(MyData[[col]]))
11 }
12 return(MyData)
13 }
14 MyData <- convertCategoricalToNumeric(MyData)
15 MyData
16 colSums(is.na(MyData))
17
18 z_scores <- scale(MyData$age)
19 age_outlier_indices <- which(abs(z_scores) > 2)
20 age_outlier_values <- MyData$age[age_outlier_indices]
21 age_outlier_values
22
23 MyData$age[age_outlier_indices] <- NA
24 MyData
25

```

Console output:

```

> z_scores <- scale(MyData$age)
> age_outlier_indices <- which(abs(z_scores) > 2)
> age_outlier_values <- MyData$age[age_outlier_indices]
> age_outlier_values
[1] 1.32 3.00 6.00 7.00 0.48 5.00 1.88 1.08 1.80 3.00 1.32
> MyData$age[age_outlier_indices] <- NA
> MyData
  gender age hypertension heart_disease ever_married work_type Residence_type avg_glucose_level
1     61      0             0             0           2           4             1             202.21
2     59      0             0             0           2           3             1             76.15
3     78      0             0             1           2           2             2             219.84
4     57      0             0             1           1           2             2             217.08

```

Step- 05: Another way to reduce outliers' values is to replace the most frequent or average value in the entire column. Using the table () function to calculate the most frequencies and which.max () to find the index of that value by using these functions together in the age attribute, we got the most frequent age, which is 78.

Now the data frame myFrequencyAgeData comparing with MyData and null values of age is replaced by 78.

```

27
28 MyData$age[age_outlier_indices] <- NA
29 MyData
30
31 which(is.na(MyData$age))
32
33 most_frequent_age <- names(which.max(table(MyData$age)))
34 MyData$age[is.na(MyData$age)] <- most_frequent_age
35 most_frequent_age
36 myFrequencyAgeData = MyData
37 myFrequencyAgeData
38
39
40
41
42 MyData$age <- as.numeric(MyData$age)
43 MyData$stroke <- as.numeric(MyData$stroke)
44 MyData$stroke <- as.numeric(MyData$hypertension)
45
46

```

Console output:

```

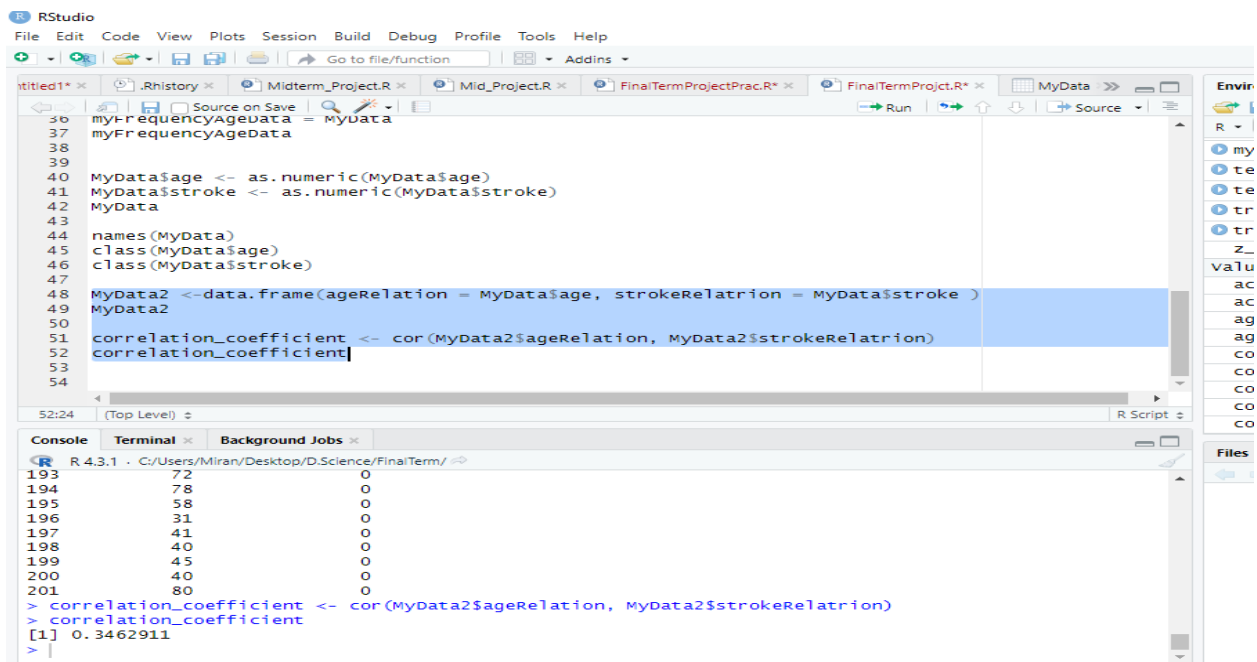
> most_frequent_age <- names(which.max(table(MyData$age)))
> MyData$age[is.na(MyData$age)] <- most_frequent_age
> most_frequent_age
[1] "78"
> myFrequencyAgeData = MyData
> myFrequencyAgeData
  gender age hypertension heart_disease ever_married work_type Residence_type avg_glucose_level
1     61      0             0             0           2           4             1             202.21
2     59      0             0             0           2           3             1             76.15
3     78      0             0             1           2           2             2             219.84
4     57      0             0             1           1           2             2             217.08

```

Step- 06: Two additional columns are being added to MyData2: The values from the age column of the original MyData data frame are entered into the first column, ageRelation. The values from the stroke column of the initial MyData data frame are placed in the second column, strokeRelation.

To find the correlation coefficient between two variables, use the `cor ()` function. In this instance, it will be applied to the MyData2 data frame's ageRelation and strokeRelation columns.

The variable `correlation_coefficient` contains the correlation calculation's outcome.



The screenshot shows the RStudio interface. The script editor contains the following code:

```
36 myFrequencyAgeData = MyData
37 myFrequencyAgeData
38
39
40 MyData$age <- as.numeric(MyData$age)
41 MyData$stroke <- as.numeric(MyData$stroke)
42 MyData
43
44 names(MyData)
45 class(MyData$age)
46 class(MyData$stroke)
47
48 MyData2 <- data.frame(ageRelation = MyData$age, strokeRelation = MyData$stroke)
49 MyData2
50
51 correlation_coefficient <- cor(MyData2$ageRelation, MyData2$strokeRelation)
52 correlation_coefficient
53
54
```

The console shows the output of the code:

```
R 4.3.1 > C:/Users/Miran/Desktop/D.Science/FinalTerm/
193      72      0
194      78      0
195      58      0
196      31      0
197      41      0
198      40      0
199      45      0
200      40      0
201      80      0
> correlation_coefficient <- cor(MyData2$ageRelation, MyData2$strokeRelation)
> correlation_coefficient
[1] 0.3462911
>
```

Step- 07: The R function `cor ()` is used to find the correlation matrix between variables in a dataset.

MyData: It is believed that this is a dataset (data frame or matrix) with a number of different variables. The `cor ()` function will determine how closely these variables are correlated.

"Pearson" is the correlation coefficient calculating method, according to this argument. The "Pearson" approach determines the Pearson correlation coefficient, which analyses the linear relationship between the variables. Another R function, **round ()**, rounds the values of a matrix or data frame to a predetermined number of digits.

`digits = 2:` This parameter indicates how many decimal places the correlation coefficients will be rounded to.

The correlation matrix will contain values between -1 and 1, where:

Values close to 1 indicate a strong positive correlation.

Values close to -1 indicate a strong negative correlation.

Values close to 0 indicate little or no linear correlation.

A strong positive correlation is found the output of this dataset.

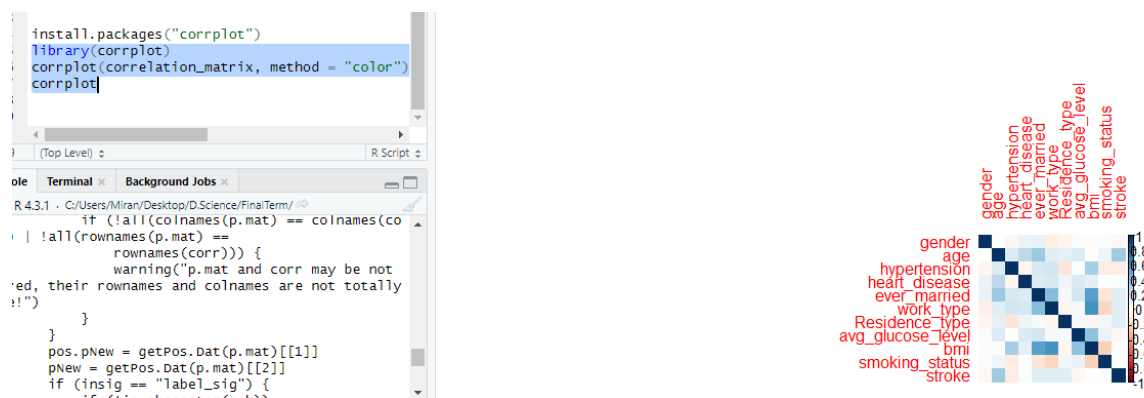
```

59 correlation_coefficient
60
61
62
63 correlation_matrix <- round(cor(MyData, method = "pearson"), digits = 2)
64 correlation_matrix
65
66 install.packages("corrplot")
67 library(corrplot)
68

```

| | age | hypertension | heart_disease | ever_married | work_type | Residence_type | avg_glucose_level | bmi | smoking_status | stroke |
|-------------------|-------|--------------|---------------|--------------|-----------|----------------|-------------------|-------|----------------|--------|
| age | 1.00 | 0.15 | 0.26 | 0.36 | 0.13 | 0.10 | | | | |
| hypertension | -0.05 | 1.00 | -0.05 | 0.17 | 0.18 | -0.14 | | | | |
| heart_disease | 0.08 | 0.26 | 1.00 | 0.14 | 0.17 | 0.07 | | | | |
| ever_married | 0.09 | 0.36 | 0.17 | 1.00 | 0.42 | 0.03 | | | | |
| work_type | -0.09 | 0.13 | 0.18 | 0.17 | 1.00 | -0.06 | | | | |
| Residence_type | -0.06 | 0.10 | -0.14 | 0.07 | 0.03 | 1.00 | | | | |
| avg_glucose_level | 0.02 | 0.15 | 0.00 | 0.16 | 0.18 | 0.03 | 1.00 | | | |
| bmi | -0.02 | -0.01 | 0.33 | 0.11 | 0.54 | 0.59 | -0.05 | 1.00 | | |
| smoking_status | 0.02 | 0.06 | -0.10 | 0.00 | -0.14 | -0.22 | 0.05 | -0.14 | 1.00 | |
| stroke | -0.04 | 0.35 | -0.10 | 0.01 | 0.12 | 0.12 | 0.12 | 0.12 | 0.12 | 1.00 |

Step- 08: The code `corrplot (correlation_matrix, method = "color")` generates a color-coded visualization of the correlation matrix stored in `correlation_matrix`. It uses the "corrplot" function, likely from the "corrplot" package, to create a plot where colors represent the strength and direction of correlations between variables.



Step- 09: `install. packages("class") library(class)` - The "class" package provides functions for k-nearest neighbors (KNN) classification and regression.

In this section we creating two data frame named training and test dataset. Here the code creates the `train_data` dataframe by extracting the "age" and "stroke" columns from the dataset, assuming there's one called `MyData`. The same columns are used to generate the test data dataframe as well.

The KNN algorithm's parameter `k`, which specifies how many neighbors to take into consideration when producing predictions, is set in this line. `K` in this case is set to 3.

The output ends with "0 0 0 0 0 0", indicating the predicted class labels for the last few data points in the test dataset.

Step- 10: `install.packages("caTools")` `library(caTools)`- "caTools" package, which provides various functions for data manipulation and splitting datasets.

The `train_data` dataframe has a total of 161 rows, according to this output. Based on the outcome of the splitting procedure, which allocated about 80% of the data to the training set, these rows were chosen from the original dataset.

The test_data dataframe has 40 rows in total, according to this output. Based on the outcome of the splitting procedure, which allocated about 20% of the data to the test set, these rows were also chosen from the original dataset.

This balance between precision and recall depends on the specific problem and the trade-offs you are willing to make between minimizing false positives and false negatives.

```
87
88
89 precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
90 precision
91 recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
92 recall
93
```

93:1 (Top Level) ⚡

Console Terminal × Background Jobs ×

R 4.3.1 · C:/Users/Miran/Desktop/D.Science/FinalTerm/ ↗

```
print(paste("Precision:", precision))
1] "Precision: 1"
print(paste("Recall:", recall))
1] "Recall: 0.85"
precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
recall <- conf_matrix[2, 2] / sum(conf_matrix[2, ])
precision <- conf_matrix[2, 2] / sum(conf_matrix[, 2])
precision
1] 1
recall
1] 0.85
```

