Session 4: Data Cleaning and Preparation in R

Objectives:

By the end of this session, participants will be able to perform following tasks:

- Handling missing values
- Handling duplicate values
- Discretizing continuous variables
- Creating new columns in the dataset
- Applying logarithmic transformation
- Recoding categorical variables
- Writing data in csv file

Data cleaning and preparation are the essential steps in the data analysis process. In this session, we will explore various techniques and tools available in R for cleaning and preparing data. By following this manual, you will learn how to handle missing values, deal with outliers, perform data transformations, and more. This session assumes basic familiarity with R programming and data manipulation concepts.

The first step is to read data available in csv file. Using the command below, we will first load the dataset. The data set contains laboratory values of blood donors and Hepatitis C patients and demographic values like age.

```
dataHep<- read.csv("hcvdat0.csv")
```

The next step is to understand the structure of file i.e., columns and rows present in the data using the set of commands covered in Session 2.

Exercise 4.1

Write the command to get the structure of data.

4.1 Handling missing Values

Unfortunately, most of the real-world datasets contain missing values. Missing value (R special value NA– not available) means that we do not know the actual value for whatever reason.

As the first step we probably want to know the distribution of missing values across the variables. The `summary` command below will report the number of missing values for each variable.

```
summary(dataHep)
```

For individual variables we can use the command below to get the count of missing values.

```
sum(is.na(dataHep$PROT))
```

To get the first row of the data frame, type the following command.

Exercise 4.2

Check the total number of missing values in 'CHOL' column

After we have identified the missing values, the next step is to either **remove** those, or **replace** those with other values (imputed values).

4.1.1 Removing Missing Values

In R, `na.omit` is used to remove the missing value. However, this will remove all rows that contain a missing value, even for cases where we do want to preserve those.

```
data_nomissing <- na.omit(dataHep)
```

4.1.2 Replacing Missing Values

Another common strategy, instead of removing missing values, is to replace those to with a certain value. This is a form of *imputing*, that is inferring the value for a missing from the observable data.

In this section, we will see how we can fill or populate missing values in a dataset using mean and median. We will use the apply method to get the mean and median of missing columns. The very first step is to get the list of columns that contain at least one missing value (NA) value.

```
listMissingColumns <- colnames(dataHep)[apply(dataHep, 2, anyNA)]

print(listMissingColumns)
```

Now we are required to compute the mean and median of the corresponding columns. Since we need to omit NA values in the missing columns, therefore, we can pass **"na.rm = True"** argument to the **apply()** function. Another way to isolate a column in a data frame using the following syntax employing the symbol **$:**

```
meanMissing <- apply(data[,colnames(dataHep) %in% listMissingColumns],2,
mean, na.rm =  TRUE)

print(meanMissing)
```

Exercise 4.3

Compute the median of columns present in `listMissingColumns.`

The next step is to replace the missing values with the column means using the command below.

```
dataHep$ALB[is.na(dataHep$ALB)] <- mean(dataHep$ALB,na.rm = TRUE)
```

Exercise 4.4

Replace the missing values with the mean of columns ALP, ALT, CHOL and PROT.

4.2 Handling duplicate data

The R function `duplicated()` returns a logical vector where TRUE specifies which elements of a vector or data frame are duplicates.

```
duplicated(dataHep)
```

If you want to remove duplicated elements, use `!duplicated(),` where ! is a logical negation.

```
dataHep<- dataHep [!duplicated(dataHep),]
```

4.3 Discretizing Continuous Variables

When analyzing how human income or health changes through the life cycle we usually want to create simple and easily understandable age groups. For instance, “below 25”, “25-35”, “35-50”, and “over 50”. This can be achieved with `cut` function:

```
ageGroup <- cut(dataHep$Age,breaks=c(-Inf, 25, 35, 50, Inf))
```

4.4 Creating News Columns in the Data Frame

There are multiple ways to add new column to the existing dataframe. The simplest way to add a new column to a data frame is with the $-Operator.

The command below will add a new column AgeGroup in the `data`.

```
dataHep$AgeGroup <- ageGroup

dataHep
```

Use the table function to get the count of records in each age group.

```
table(dataHep$AgeGroup)
```

Exercise 4.5

Discretize the variable ALB into the following three categories

- Low Albumin: Below 3.5 g/L
- Normal Albumin: 3.5 g/L to 5.0 g/L
- High Albumin: Above 5.0 g/L

Create a new column ‘ALBCategory’ to store the categorical value of ALB

4.5 Logarithmic Transformation

The log-transformation is widely used to deal with skewed data. The code below shows how to perform common a base-10 log transformation.

```
dataHep$ALT_Log <- log(dataHep$ALT)
```

After analysis, you should back-transform the results to get them on the original scale. In this case, the reverse transformation is done with the exp function. Here is how it looks with previously log-transformed data.

```
exp(dataHep$ALT_Log)
```

Exercise 4.6

Apply log transformation on column CREA

4.6 Recoding Categorical Variables

Recoding a categorical variable to another variable can be done in R though several methods. Using base R, recoding can be done with the match() function.

The code below will replace "0=Blood Donor" and "0s=suspect Blood Donor" with "**Healthy**" category and "1=Hepatitis", "2=Fibrosis" and "3=Cirrhosis" with "**HepC**".

```
#Recoding categorical variable

oldvals <- c("0=Blood Donor", "0s=suspect Blood Donor",
"1=Hepatitis","2=Fibrosis","3=Cirrhosis")

newvals <- factor(c("Healthy", "Healthy", "HepC","HepC","HepC"))

dataHep["Disease_Status"] <- newvals[ match(dataHep$Category, oldvals) ]
```

4.7 Write a csv file

Once the data is cleaned and prepared for analysis, use `write.csv()` function to export data frame to csv file for later use.

```
#writing file on csv

write.csv(dataHep, "hepCdata_clean.csv", row.names=FALSE)
```

The above code is using the `write.csv()` function to write a data frame called dataHep to a CSV (Comma-Separated Values) file named "hepCdata_clean.csv". Let's break down the code and understand its components:

a) write.csv() is an R function that writes a data frame to a CSV file. It takes multiple arguments, including the data frame to be written, the file path or name, and other optional parameters.
b) dataHep is the data frame being written to the CSV file. It contains the data that you want to export.
c) "hepCdata_clean.csv" is the name of the CSV file that will be created. It is enclosed in double quotation marks.
d) row.names=FALSE is an optional parameter that specifies whether to include row names in the CSV file. In this case, it is set to FALSE, which means the row names of the data frame will not be included in the first column of the CSV file.