Lab 3 - Data Cleaning

Learning Outcomes

At the end of the session, you will be able to:

- Impute missing values using R code.
- Normalize data using scaling methods.
- Encode categorical variables.

Activity 1 – Basic Imputation Methods

1. We'll start with basic method to replace missing values.

- Replace the column's missing value with zero.

```
df<-data.frame(Product = c('A','B', 'C','D','E'),Price=c(612,447,NA,374,831))
```

```
df
  Product Price
1       A   612
2       B   447
3       C    NA
4       D   374
5       E   831
```

```
df$Price[is.na(df$Price)] <- 0
```

- Replace the column's missing value with the mean.

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

- Replace the column's missing value with the median.

```
df$Price[is.na(df$Price)]<- median(df$Price,na.rm = TRUE)
```

2. Retrieve data from titanic library. Install the titanic package and call the library. View the dataset.

```
library(titanic)
summary(titanic)

titanic_train$Age
```

3. View the Age distribution using histogram.

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

ggplot(titanic_train, aes(Age)) +
  geom_histogram(color = "#000000", fill = "#0099F8") +
  ggtitle("Variable distribution") +
  theme_classic() +
  theme(plot.title = element_text(size = 18))
```

4. Perform simple value imputation and view the data.

```
value_imputed <- data.frame(
  original = titanic_train$Age,
  imputed_zero = replace(titanic_train$Age,
is.na(titanic_train$Age), 0),
  imputed_mean = replace(titanic_train$Age,
is.na(titanic_train$Age), mean(titanic_train$Age, na.rm = TRUE)),
  imputed_median = replace(titanic_train$Age,
is.na(titanic_train$Age), median(titanic_train$Age, na.rm =
TRUE))
)

value_imputed
```

5. Create histograms after imputation.

```
h1 <- ggplot(value_imputed, aes(x = original)) +
  geom_histogram(fill = "#ad1538", color = "#000000", position =
"identity") +
  ggtitle("Original distribution") +
  theme_classic()
h2 <- ggplot(value_imputed, aes(x = imputed_zero)) +
  geom_histogram(fill = "#15ad4f", color = "#000000", position =
"identity") +
  ggtitle("Zero-imputed distribution") +
  theme_classic()
h3 <- ggplot(value_imputed, aes(x = imputed_mean)) +
  geom_histogram(fill = "#1543ad", color = "#000000", position =
"identity") +
  ggtitle("Mean-imputed distribution") +
  theme_classic()
h4 <- ggplot(value_imputed, aes(x = imputed_median)) +
  geom_histogram(fill = "#ad8415", color = "#000000", position =
"identity") +
  ggtitle("Median-imputed distribution") +
  theme_classic()

plot_grid(h1, h2, h3, h4, nrow = 2, ncol = 2)
```

All imputation methods severely impact the distribution. There are a lot of missing values, so setting a single constant value doesn't make much sense.

Zero imputation is the worst, as it's highly unlikely for close to 200 passengers to have the age of zero.

Activity 2 – Impute Missing Values in R with MICE

MICE stands for Multivariate Imputation via Chained Equations, and it's one of the most common packages for R users. It assumes the missing values are missing at random (MAR). The basic idea behind the algorithm is to treat each variable that has missing values as a dependent variable in regression and treat the others as independent (predictors).

1. Install the mice package and test with `md.pattern()` function. Observe the visual representation of missing values.

```
library(mice)

titanic_numeric <- titanic_train %>%
  select(Survived, Pclass, SibSp, Parch, Age)

md.pattern(titanic_numeric)
```

2. Perform MICE imputation methods.

   pmm: Predictive mean matching.
   cart: Classification and regression trees.
   laso.norm: Lasso linear regression.

```
mice_imputed <- data.frame(
original = titanic_train$Age,
 imputed_pmm = complete(mice(titanic_numeric, method = "pmm"))$Age,
imputed_cart = complete(mice(titanic_numeric, method = "cart"))$Age,
  imputed_lasso = complete(mice(titanic_numeric, method = "lasso.norm"))$Age)

mice_imputed
```

3. Visualize all imputed data using a grid of histograms. Copy and modify the code from the previous section.

   Which imputed distributions are much closer to the original one?

Activity 3 – Imputation with R missForest Package

The Miss Forest imputation technique is based on the Random Forest algorithm. It's a non-parametric imputation method, which means it doesn't make explicit assumptions about the function form, but instead tries to estimate the function in a way that's closest to the data points.

1. Install and import **missForest** library.
2. Impute missing values in Age attribute.

```
 library(missForest)

missForest_imputed <- data.frame(
  original = titanic_numeric$Age,
  imputed_missForest = missForest(titanic_numeric)$ximp$Age
)
missForest_imputed
```

3. Visualize the distribution. Compare the imputed values with the values generated by MICE in previous activity.

Activity 4: Normalize data with scaling methods

Feature Scaling is an essential step prior to modeling while solving prediction problems in Data Science. Machine Learning algorithms work well with data that belongs to a smaller and standard scale. Normalization techniques enable us to reduce the scale of the variables and thus it affects the statistical distribution of the data in a positive manner.

1. Normalize data using Log Transformation

   ```
   log_scale = log(as.data.frame(titanic$Fare))
   ```

2. Normalize data using Min-Max Scaling

   With Min-Max Scaling, we scale the data values between a range of **0 to 1 only**. Due to this, the effect of outliers on the data values suppresses to a certain extent. Moreover, it helps us have a smaller value of the standard deviation of the data scale.

   ```
   library(caret)
   process <- preProcess(as.data.frame(titanic$Fare),
   method=c("range"))

   norm_scale <- predict(process, as.data.frame(titanic$Fare))
   ```

3. Normalize data using standard scaling in R.

   ```
   scale_data <- as.data.frame(scale(titanic$Fare))
   ```

Explore more scaling methods >
https://medium.com/swlh/data-normalisation-with-r-6ef1d1947970

Activity 5: Feature Encoding

There are several powerful machine learning algorithms in R. However, to make the best use of these algorithms, it is imperative that we transform the data into the desired format.

1. Label Encoding

In simple terms, label encoding is the process of replacing the different levels of a categorical variable with dummy numbers.

```
gender_encode <- ifelse(titanic_train$Sex  == "male",1,0)

table(gender_encode)
```

Observe the output that shows the label encoding. This is easy when you have two levels in the categorical variable.

Try with other nominal or categorical variables that contain more than two labels. Observe the output.

2. One-Hot Encoding

In this technique, one-hot (dummy) encoding is applied to the features, creating a binary column for each category level and returning a sparse matrix. In each dummy variable, the label “1” will represent the existence of the level in the variable, while the label “0” will represent its non-existence.

Let us create new data frame based on the selected variables from **titanic_train** dataset; **Fare, Sex** and **Embarked**.

```
new_dat =
data.frame(titanic_train$Fare,titanic_train$Sex,titanic_train$Emba
rked)

summary(new_dat)
```

Call **`caret`** library and transform the categorical variables using **`predict()`** function.

Use the **`dummyVars()`** function to create a full set of dummy variables. The **`dummyVars()`** method works on the categorical variables. It is to be noted that the second line contains the argument **`fullrank=T`**, which will create **n-1** columns for a categorical variable with **n** unique levels.

```
library(caret)

dmy <- dummyVars(" ~ .", data = new_dat, fullRank = T)
dat_transformed <- data.frame(predict(dmy, newdata = new_dat))

glimpse(dat_transformed)
```

Observe the output.

3. Encoding Continuous (or Numeric) Variables

We will consider the Fare variable as an example. Let's look at the summary statistics of this variable.

```
summary(new_dat$titanic_train.Fare)
```

Output:

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.00    7.91   14.45   32.20   31.00  512.33
```

The first step is to create a vector of cut-off points based on 1st Quarter value and 3rd Quarter values.

```
bins <- c(-Inf, 7.91, 31.00, Inf)
```

The second step gives the respective names to these cut-off points.

```
bin_names <- c("Low", "Mid50", "High")
```

The third step uses the **`cut()`** function to break the vector using the cut-off points.

```
new_dat$new_Fare <- cut(new_dat$titanic_train.Fare, breaks =
bins, labels = bin_names)
```

Finally, we compare the original **`Fare`** variable with the binned **`new_Fare`** variable using the **`summary()`** function.

```
summary(new_dat$titanic_train.Fare)

summary(new_dat$new_Fare)
```

<u>Week 5 Lab Submission</u>

1. Publish your work to GitHub and share it with GA.
2. Submit data pre-processing report in PDF format through ULearn. Use chunk dataset from Week 4 Activity. Select appropriate variables for data pre-processing.

Deadline: 10 Oct 2023