PROJECT REPORT

AMERICAN INTERNATIONAL UNIVERSITY BANGLADESH

Submitted By

Name: AHMED IMTIAZ

ID: 20-42933-1

Section: C

Course: INTRODUCTION TO DATA SCIENCE

Semester: SUMMER 2022-2023

Dataset Description

The dataset is chosen from the Kaggle website. The National Institute of Diabetes and Digestive and Kidney Diseases is the source of this dataset. The goal is to determine whether a patient has diabetes based on diagnostic parameters. Here all the patients in this dataset are female. There are 9 attributes and 768 instances in the dataset. The details of the attributes of this dataset are provided below.

- Pregnancies: Number of times pregnant
- Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
- Blood Pressure: Diastolic blood pressure (mm Hg)
- Skin Thickness: Triceps skin fold thickness (mm)
- Insulin: 2-Hour serum insulin (mu U/ml)
- BMI: Body mass index (weight in kg/ (height in m) ^2)
- Diabetes Pedigree Function: Diabetes pedigree function
- Age: Age (years)
- Outcome: Class variable (0 or 1)

Data Preparation

Code Segment: -

```
data <- read.csv("D:/INTRODUCTION TO DATA SCIENCE/Final Term/Project/diabetes.csv")

str(data)

summary(data)
```

Output Segment: -

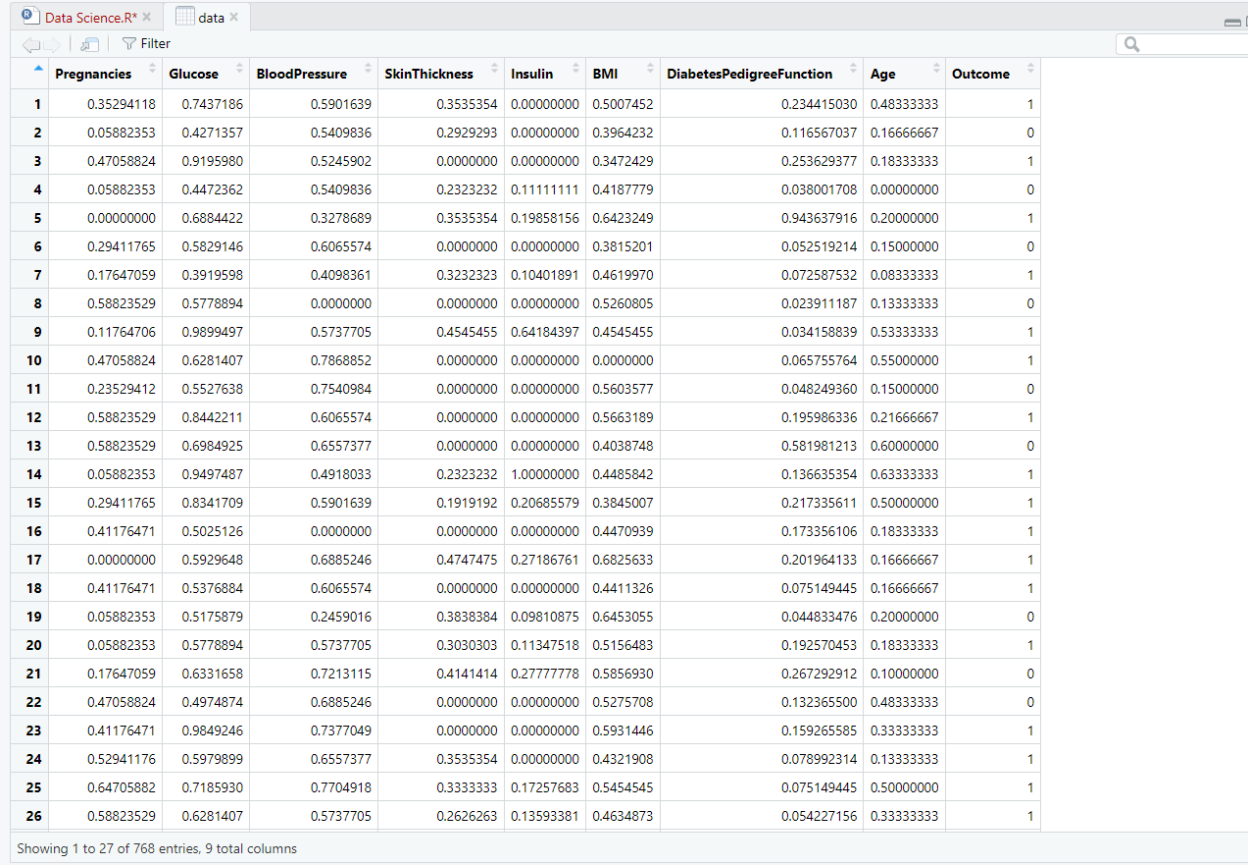
```
> str(data)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : num  0.3529 0.0588 0.4706 0.0588 0 ...
 $ Glucose                 : num  0.744 0.427 0.92 0.447 0.688 ...
 $ BloodPressure           : num  0.59 0.541 0.525 0.541 0.328 ...
 $ SkinThickness           : num  0.354 0.293 0 0.232 0.354 ...
 $ Insulin                 : num  0 0 0 0.111 0.199 ...
 $ BMI                     : num  0.501 0.396 0.347 0.419 0.642 ...
 $ DiabetesPedigreeFunction: num  0.234 0.117 0.254 0.038 0.944 ...
 $ Age                     : num  0.483 0.167 0.183 0 0.2 ...
 $ Outcome                 : num  1 0 1 0 1 0 1 0 1 1 ...
>
> summary(data)
  Pregnancies         Glucose       BloodPressure    SkinThickness       Insulin             BMI
 Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
 1st Qu.:0.05882   1st Qu.:0.4975   1st Qu.:0.5082   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.4069
 Median :0.17647   Median :0.5879   Median :0.5902   Median :0.2323   Median :0.03605   Median :0.4769
 Mean   :0.22618   Mean   :0.6075   Mean   :0.5664   Mean   :0.2074   Mean   :0.09433   Mean   :0.4768
 3rd Qu.:0.35294   3rd Qu.:0.7048   3rd Qu.:0.6557   3rd Qu.:0.3232   3rd Qu.:0.15041   3rd Qu.:0.5455
 Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
 DiabetesPedigreeFunction      Age            Outcome
 Min.   :0.00000          Min.   :0.0000   Min.   :0.000
 1st Qu.:0.07077          1st Qu.:0.0500   1st Qu.:0.000
 Median :0.12575          Median :0.1333   Median :0.000
 Mean   :0.16818          Mean   :0.2040   Mean   :0.349
 3rd Qu.:0.23409          3rd Qu.:0.3333   3rd Qu.:1.000
 Max.   :1.00000          Max.   :1.0000   Max.   :1.000
```

Description: - Here, str code is used to check the structure of the dataset, and in the output, we can see that there is no character-related data; all data is in the numeric or number format. On the other hand, the summary code is used to check if there is a null value in the dataset. In the output, we can see no null value in the dataset.

Code Segment: -

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
data <- as.data.frame(lapply(data, normalize))
```

Output Segment: -



Data Science.R* | data | Filter

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	0.35294118	0.7437186	0.5901639	0.3535354	0.00000000	0.5007452	0.234415030	0.48333333	1
2	0.05882353	0.4271357	0.5409836	0.2929293	0.00000000	0.3964232	0.116567037	0.16666667	0
3	0.47058824	0.9195980	0.5245902	0.0000000	0.00000000	0.3472429	0.253629377	0.18333333	1
4	0.05882353	0.4472362	0.5409836	0.2323232	0.11111111	0.4187779	0.038001708	0.00000000	0
5	0.00000000	0.6884422	0.3278689	0.3535354	0.19858156	0.6423249	0.943637916	0.20000000	1
6	0.29411765	0.5829146	0.6065574	0.0000000	0.00000000	0.3815201	0.052519214	0.15000000	0
7	0.17647059	0.3919598	0.4098361	0.3232323	0.10401891	0.4619970	0.072587532	0.08333333	1
8	0.58823529	0.5778894	0.0000000	0.0000000	0.00000000	0.5260805	0.023911187	0.13333333	0
9	0.11764706	0.9899497	0.5737705	0.4545455	0.64184397	0.4545455	0.034158839	0.53333333	1
10	0.47058824	0.6281407	0.7868852	0.0000000	0.00000000	0.0000000	0.065755764	0.55000000	1
11	0.23529412	0.5527638	0.7540984	0.0000000	0.00000000	0.5603577	0.048249360	0.15000000	0
12	0.58823529	0.8442211	0.6065574	0.0000000	0.00000000	0.5663189	0.195986336	0.21666667	1
13	0.58823529	0.6984925	0.6557377	0.0000000	0.00000000	0.4038748	0.581981213	0.60000000	0
14	0.05882353	0.9497487	0.4918033	0.2323232	1.00000000	0.4485842	0.136635354	0.63333333	1
15	0.29411765	0.8341709	0.5901639	0.1919192	0.20685579	0.3845007	0.217335611	0.50000000	1
16	0.41176471	0.5025126	0.0000000	0.0000000	0.00000000	0.4470939	0.173356106	0.18333333	1
17	0.00000000	0.5929648	0.6885246	0.4747475	0.27186761	0.6825633	0.201964133	0.16666667	1
18	0.41176471	0.5376884	0.6065574	0.0000000	0.00000000	0.4411326	0.075149445	0.16666667	1
19	0.05882353	0.5175879	0.2459016	0.3838384	0.09810875	0.6453055	0.044833476	0.20000000	0
20	0.05882353	0.5778894	0.5737705	0.3030303	0.11347518	0.5156483	0.192570453	0.18333333	1
21	0.17647059	0.6331658	0.7213115	0.4141414	0.27777778	0.5856930	0.267292912	0.10000000	0
22	0.47058824	0.4974874	0.6885246	0.0000000	0.00000000	0.5275708	0.132365500	0.48333333	0
23	0.41176471	0.9849246	0.7377049	0.0000000	0.00000000	0.5931446	0.159265585	0.33333333	1
24	0.52941176	0.5979899	0.6557377	0.3535354	0.00000000	0.4321908	0.078992314	0.13333333	1
25	0.64705882	0.7185930	0.7704918	0.3333333	0.17257683	0.5454545	0.075149445	0.50000000	1
26	0.58823529	0.6281407	0.5737705	0.2626263	0.13593381	0.4634873	0.054227156	0.33333333	1

Showing 1 to 27 of 768 entries, 9 total columns

Description: - At first, a function is created by writing code normalize function (x); the data values are given then normalization is done.

Pearson's Correlation Coefficient

Code Segment: -

```
install.packages("corrplot")
library(corrplot)

correlation_matrix <- cor(data, method = "pearson")
correlation_with_outcome <- correlation_matrix[, "outcome"]
print(correlation_with_outcome)
```

Output Segment: -

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.00000000	0.12945867	0.14128198	-0.08167177	-0.07353461	0.01768309	-0.03352267	0.54434123	0.22189815
Glucose	0.12945867	1.00000000	0.15258959	0.05732789	0.33135711	0.22107107	0.13733730	0.26351432	0.46658140
BloodPressure	0.14128198	0.15258959	1.00000000	0.20737054	0.08893338	0.28180529	0.04126495	0.23952795	0.06506836
SkinThickness	-0.08167177	0.05732789	0.20737054	1.00000000	0.43678257	0.39257320	0.18392757	-0.11397026	0.07475223
Insulin	-0.07353461	0.33135711	0.08893338	0.43678257	1.00000000	0.19785906	0.18507093	-0.04216295	0.13054795
BMI	0.01768309	0.22107107	0.28180529	0.39257320	0.19785906	1.00000000	0.14064695	0.03624187	0.29269466
DiabetesPedigreeFunction	-0.03352267	0.13733730	0.04126495	0.18392757	0.18507093	0.14064695	1.00000000	0.03356131	0.17384407
Age	0.54434123	0.26351432	0.23952795	-0.11397026	-0.04216295	0.03624187	0.03356131	1.00000000	0.23835598
Outcome	0.22189815	0.46658140	0.06506836	0.07475223	0.13054795	0.29269466	0.17384407	0.23835598	1.00000000

Description: - Here, a built-in library is installed first, then a correlation matrix is created, and the Pearson method is used here; the outcome is the target values. Correlation is calculated against the targeted value with all other values.

KNN Find Accuracy 10-Fold Cross Validation

Code Segment: -

```r

install.packages(caret)
library(caret)

if (!"Outcome" %in% colnames(data)) {
  stop("'Outcome' column not found in the dataset.")
}

trainIndex <- createDataPartition(data$Outcome, p = 0.7, list = FALSE)
training_data <- data[trainIndex, ]
testing_data <- data[-trainIndex, ]

k_values <- expand.grid(k = c(5))

knn_model <- train(
  Outcome ~ .,
  data = training_data,
  method = "knn",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = k_values
)

print(knn_model)

predictions <- predict(knn_model, newdata = testing_data)

accuracy <- sum(predictions == testing_data$Outcome) / nrow(testing_data)
cat("Accuracy with optimal k:", accuracy, "\n")

```

Output Segment: -

```
k-Nearest Neighbors

538 samples
  8 predictor

Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 484, 484, 484, 484, 484, 484, ...
Resampling results:

  RMSE       Rsquared   MAE
  0.4322101  0.2118599  0.3169672

Tuning parameter 'k' was held constant at a value of 5
>
> # Predict on the testing set using the optimal model
> predictions <- predict(knn_model, newdata = testing_data)
>
> # Calculate the accuracy
> accuracy <- sum(predictions == testing_data$Outcome) / nrow(testing_data)
> cat("Accuracy with optimal k:", accuracy, "\n")
Accuracy with optimal k: 0.3173913
```

Description: - At first, the outcome column is checked, then data is split in the ratio of train 70% and test 30%, then the model knn is applied, then the accuracy is found out by using 10-fold cross-validation.

Confusion Matrix

Code Segment: -

```
53
54 install.packages("Metrics")
55 library(Metrics)
56
57 pred_values <- factor(data$Outcome)
58 actual_values <- factor(data$Outcome)
59
60 cf <- caret::confusionMatrix(data=pred_values,reference=actual_values)
61 print(cf)
62
63 predicted <- c(data$Outcome)
64 actual <- c(data$Outcome)
65
66 Metrics::precision(predicted, actual)
67 Metrics::recall(predicted, actual)
68
```

Output Segment: -

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 500   0
         1   0 268

               Accuracy : 1
                 95% CI : (0.9952, 1)
    No Information Rate : 0.651
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.651
         Detection Rate : 0.651
   Detection Prevalence : 0.651
      Balanced Accuracy : 1.000

       'Positive' Class : 0

>
> predicted <- c(data$Outcome)
> actual <- c(data$Outcome)
>
> Metrics::precision(predicted, actual)
[1] 1
> Metrics::recall(predicted, actual)
[1] 1
```

Description: - First of all, the metrics library is installed, then the predicted value and the actual value are tabled, then the confusion matrix is created, and the recall value and precision value are calculated against the outcome column.