AMERICAN INTERNATIONAL UNIVERSITY–BANGLADESH (AIUB)

FACULTY OF SCIENCE & TECHNOLOGY

INTRODUCTION TO DATA SCIENCE [A]

FINAL TERM PROJECT REPORT ON

Taste Trios Dataset

Submitted By

Sl No.	Name	ID
1	AYON GHOSH	21-44703-1
2	SANUAR HOSSAIN	21-45298-2

Submitted To

TOHEDUL ISLAM

Assistant Professor,

Department of Computer Science,

Faculty of Science and Technology,

American International University-Bangladesh (AIUB)

2023-2024, Fall

Date of Submission: December 25, 2023

Table of content:

About Dataset:

The Taste Trios dataset is a captivating compilation that delves into the world of perfect ingredient combinations, focusing on main ingredients harmonizing with two sub-ingredients. This dataset is meticulously categorized into three compatibility levels: Highly Compatible, Moderately Compatible, and Compatible. Each entry represents a delightful trio of ingredients, offering culinary enthusiasts and professionals a trove of inspiration for delicious crafting recipes and uncovering new flavor profiles.

Data Preparation:

Finding Missing Values:

colSums(is.na(finalproject))

```
> colSums(is.na(finalproject))
        Ingredient1          Ingredient2          Ingredient3 ClassificationOutput 
                  0                    0                    0                    0 
> 
```

This function returns the total number of missing values per column.
This dataset has no missing values.

Correlation:

Pearson's Chi-squared test:

```
tableIng1 <- table(finalproject$Ingredient1,finalproject$ClassificationOutput)
chi_1<-chisq.test(tableIng1)
chi_1
pVal_1<-chi$p.value
if(pVal_1<0.05){

  print("Ingredient 1 is Singnificat")

}else{

  print("Ingredient 1 is Not Singnificat ")

}

tableIng2 <- table(finalproject$Ingredient2,finalproject$ClassificationOutput)
chi_2<-chisq.test(tableIng1)
chi_2
pVal_2<-chi$p.value
if(pVal_2<0.05){

  print("Ingredient 2 is Singnificat")

}else{

  print("Ingredient 2 is Not Singnificat ")

}

tableIng3 <- table(finalproject$Ingredient3,finalproject$ClassificationOutput)
chi_3<-chisq.test(tableIng3)
```

```
chi_3
pVal_3<-chi$p.value
if(pVal_3<0.05){

  print("Ingredient 3 is Singnificat")

}else{


  print("Ingredient3 is Not Singnificat ")

}
```

```
data:  tableIng1
X-squared = 87.18, df = 36, p-value = 3.917e-06

> pVal_1<-chi$p.value
> if(pVal_1<0.05){
+ 
+   print("Ingredient 1 is Singnificat")
+ 
+ }else{
+ 
+ 
+   print("Ingredient 1 is Not Singnificat ")
+ 
+ 
+ 
+ }
[1] "Ingredient 1 is Singnificat"
> chi_2

        Pearson's Chi-squared test

data:  tableIng1
X-squared = 87.18, df = 36, p-value = 3.917e-06

> pVal_2<-chi$p.value
> if(pVal_2<0.05){
+ 
+   print("Ingredient 2 is Singnificat")
+ 
+ }else{
+ 
+ 
+   print("Ingredient 2 is Not Singnificat ")
+ 
+ 
+ 
+ }
[1] "Ingredient 2 is Singnificat"
```

```
> chi_3

        Pearson's Chi-squared test

data:  tableIng3
X-squared = 280.27, df = 302, p-value = 0.8103

> pval_3<-chi$p.value
> if(pval_3<0.05){
+
+   print("Ingredient 3 is Singnificat")
+
+ }else{
+
+
+   print("Ingredient3 is Not Singnificat ")
+
+ }
[1] "Ingredient 3 is Singnificat"
```

Here, we perform **Pearson's Chi-squared test to** find the correlation between independent and dependent attributes. Code shows that all attributes are significant because the value od p is less than or equal to 0.05.

Apply Naïve Bayes:

```
install.packages("e1071")
library(e1071)
nb <- naiveBayes(ClassificationOutput ~ Ingredient1 + Ingredient2 + Ingredient3, data = finalproject)

multi_unknown_instance <- data.frame(
  Ingredient1 = c("Mushroom", "Pumpkin", "Mango"),
  Ingredient2 = c("Butter", "Apples", "Honey"),
  Ingredient3 = c("Sage", "Curry", "Lime")
)
predicted_class <- predict(nb, newdata = multi_unknown_instance)
print("Predicted Class:")
print(predicted_class)
```

```
> print("Predicted Class:")
[1] "Predicted Class:"
> print(predicted_class)
[1] Highly Compatible     Moderately Compatible Compatible
```

Here, we apply Naïve Bayse algorithm for some unknown instances to predict their class. Output shows the classes of them.

```
nbs <- naiveBayes(ClassificationOutput ~ Ingredient1 + Ingredient2 + Ingredient3, data = finalproject)
single_unknown_instance <- data.frame(
  Ingredient1 = "Mushroom",
  Ingredient2 = "Butter",
  Ingredient3 = "Sage"
)
predicted_class <- predict(nbs, newdata = single_unknown_instance)
print("Predicted Class:")
print(predicted_class)
finalproject
```

```
> nbs <- naiveBayes(ClassificationOutput ~ Ingredient1 + Ingredient2 + Ingredient3, data = finalproject)
> single_unknown_instance <- data.frame(
+   Ingredient1 = "Mushroom",
+   Ingredient2 = "Butter",
+   Ingredient3 = "Sage"
+ )
> predicted_class <- predict(nbs, newdata = single_unknown_instance)
> print("Predicted Class:")
[1] "Predicted Class:"
> print(predicted_class)
[1] Highly Compatible
```

Here, we apply the Naïve Bayse algorithm for an unknown instance to predict this class. Output shows the class of this.

Dividing the data into training and test set:

```
install.packages("caret")
library(caret)
train_index <- sample(1:nrow(finalproject), size = 0.8 * nrow(finalproject))
train_data <- finalproject[train_index, ]
test_data <- finalproject[-train_index, ]
train_data
test_data

num_of_test_data <- nrow(test_data)
print("No of Test Data set: ")
num_of_test_data


num_of_train_data <- nrow(train_data)
print("No of Tranning Data set: ")
num_of_train_data

Test_dataset <- read.csv("D:/Data Science_Ayon/test_data.csv", header=TRUE, sep=",")
Test_dataset

Train_csv_data <- read.csv("D:/Data Science_Ayon/train_data.csv", header=TRUE,
sep=",")
Train_csv_data
```

```
110 Blueberries   Vanilla Extract      Coconut Milk Moderately Compatible
111       Lemon          Cilantro            Garlic            Compatible
112       Lemon             Lemon       Blueberries            Compatible
113    Chickpeas          Avocado           Spinach            Compatible
114    Chickpeas       Bell Pepper           Hummus            Compatible
115       Shrimp             Lime          Cilantro            Compatible
116 Blueberries      Greek Yogurt             Honey            Compatible
117 Blueberries           Spinach     Almond Butter            Compatible
118      Potato   Parmesan Cheese          Broccoli     Highly Compatible
119      Potato    Cheddar Cheese             Bacon     Highly Compatible
120      Potato          Rosemary             Lemon     Highly Compatible
121      Potato            Tomato           Oregano Moderately Compatible
122      Potato            Tomato       Bell Pepper            Compatible
```

Our test dataset has 122 instances.

```
470      Walnuts      Cherry Tomatoes      Mozzarella          Compatible
471      Avocado         Feta Cheese        Olive Oil   Highly Compatible
472     Mushroom               Honey          Mustard Moderately Compatible
473       Tomato         Goat Cheese            Honey Moderately Compatible
474        Mango              Papaya     Coconut Milk Moderately Compatible
475       Salmon        Coconut Milk             Lime Moderately Compatible
476  Blueberries               Lemon             Mint   Highly Compatible
477         Pork               Apple             Sage   Highly Compatible
478    Chickpeas             Spinach            Curry   Highly Compatible
479        Mango          Watermelon      Feta Cheese Moderately Compatible
480       Shrimp               Lemon           Garlic   Highly Compatible
481       Tomato              Capers        Anchovies Moderately Compatible
482  Blueberries   Pomegranate Seeds           Quinoa          Compatible
483        Mango            Cucumber           Yogurt Moderately Compatible
484       Tomato        Caprese Salad   Balsamic Glaze   Highly Compatible
485      Walnuts                Pear          Arugula   Highly Compatible
```

Our training dataset has 485 instances.

We have split our whole dataset into 20% as test dataset and 80% as training dataset.

```
> num_of_test_data <- nrow(test_data)
> print("No of Test Data set: ")
[1] "No of Test Data set: "
> num_of_test_data
[1] 122
> num_of_train_data <- nrow(train_data)
> print("No of Tranning Data set: ")
[1] "No of Tranning Data set: "
> num_of_train_data
[1] 485
```

Create a Machine Learning Model:

install.packages("naivebayes")
library(naivebayes)

ML_nb_model <- naive_bayes(ClassificationOutput ~ ., data = train_data)
ML_nb_model

```
=================================================================== Naive Bayes ========

 Call:
naive_bayes.formula(formula = ClassificationOutput ~ ., data = train_data)

-----------------------------------------------------------------------------------------

Laplace smoothing: 0

-----------------------------------------------------------------------------------------

 A priori probabilities:

         Compatible     Highly Compatible Moderately Compatible
          0.2536082             0.4412371             0.3051546
```

Here, we have created a machine learning model using a training dataset.

predicted_class <- predict(ML_nb_model, newdata = Test_dataset)
correctly_classified <- sum(predicted_class == Test_dataset$ClassificationOutput)
cat("Number of correctly classified instances :", correctly_classified, "\n")

accuracy <- correctly_classified / nrow(Test_dataset)
cat("accuracy is:")
accuracy

```
> correctly_classified <- sum(predicted_class == Test_dataset$ClassificationOutput)
> cat("Number of correctly classified instances :", correctly_classified, "\n")
Number of correctly classified instances : 49
> accuracy <- correctly_classified / nrow(Test_dataset)
> cat("accuracy is:")
accuracy is:
> accuracy
[1] 0.4016393
>
```

Here, the number of correctly classified instances is 49 out of 122 of test dataset and accuracy is 0.4016393.

10-Fold cross validation:

CrossValidation <- trainControl(method = "cv", number = 10)
CrossValidation
cv_model <- train(ClassificationOutput ~ ., data = finalproject, method = "naive_bayes", trControl = CrossValidation)
cv_model

all_fold <- cv_model$resample
all_fold

accuracy <- cv_model$results$Accuracy
accuracy

```
> all_fold <- cv_model$resample
> all_fold
    Accuracy Kappa Resample
1  0.4500000     0   Fold01
2  0.4333333     0   Fold02
3  0.4500000     0   Fold03
4  0.4500000     0   Fold04
5  0.4426230     0   Fold05
6  0.4426230     0   Fold06
7  0.4426230     0   Fold07
8  0.4354839     0   Fold08
9  0.4354839     0   Fold09
10 0.4500000     0   Fold10
> accuracy <- cv_model$results$Accuracy
> accuracy
[1] 0.2388516 0.4432170
```

Here, we apply (k=10) 10-Fold cross validation to measure the machine learning model performance. Output shows the specific values of each fold and mean accuracy of those. Which is 0.4432170.

Confusion Matrix:

cm <- confusionMatrix(cv_model)
cm

```
                        Reference
Prediction               Compatible Highly Compatible Moderately Compatible
  Compatible                    0.0               0.0                   0.0
  Highly Compatible            23.9              44.3                  31.8
  Moderately Compatible         0.0               0.0                   0.0

 Accuracy (average) : 0.4432
```

Here, we created a confusion matrix using machine learning model.

TP, TN, FN, FP and the Recall, Precision and F- measure:

```
cat("For  Compatible  Class (TP,FN,FP,TN): ")
TP_Compatible <- cm$table[1, 1]
cat("True Positives (TP):", TP, "\n")


FN_Compatible <- cm$table[1, 2]+ cm$table[1, 3]
cat("False Negatives (FN):", FN, "\n")


FP_Compatible <- cm$table[2, 1]+cm$table[3, 1]
cat("False Positives (FP):", FP, "\n")


TN_Compatible <- cm$table[2, 2]+cm$table[2,3]+cm$table[3,2]+cm$table[3, 3]
cat("True Negatives (TN):", TN, "\n")


cat("Precision For Compatible  Class :")
Precision_Compatible = TP_Compatible / (TP_Compatible + FP_Compatible)
Precision_Compatible

cat("Recall For Compatible  Class :")
recall_Compatible=TP_Compatible/(TP_Compatible+FN_Compatible)
recall_Compatible

cat("F-Measure Compatible  Class :")
F_Compatible = 2 * (Precision_Compatible * recall_Compatible) / (Precision_Compatible +
recall_Compatible)
F_Compatible
```

```
> cat("For  Compatible  Class (TP,FN,FP,TN): ")
For  Compatible  Class (TP,FN,FP,TN): > TP_Compatible <- cm$table[1, 1]
> cat("For  Compatible  Class (TP,FN,FP,TN): ")
For  Compatible  Class (TP,FN,FP,TN):
> TP_Compatible <- cm$table[1, 1]
> cat("True Positives (TP):", TP, "\n")
True Positives (TP): 0
> FN_Compatible <- cm$table[1, 2]+ cm$table[1, 3]
> cat("False Negatives (FN):", FN, "\n")
False Negatives (FN): 0
> FP_Compatible <- cm$table[2, 1]+cm$table[3, 1]
> cat("False Positives (FP):", FP, "\n")
False Positives (FP): 31.79572
> TN_Compatible <- cm$table[2, 2]+cm$table[2,3]+cm$table[3,2]+cm$table[3, 3]
> cat("True Negatives (TN):", TN, "\n")
True Negatives (TN): 68.20428
> cat("Precision For Compatible  Class :")
Precision For Compatible  Class :
> Precision_Compatible = TP_Compatible / (TP_Compatible + FP_Compatible)
> Precision_Compatible
[1] 0
> cat("Recall For Compatible  Class :")
Recall For Compatible  Class :
> recall_Compatible=TP_Compatible/(TP_Compatible+FN_Compatible)
> recall_Compatible
[1] NaN
> cat("F-Measure Compatible  Class :")
F-Measure Compatible  Class :
> F_Compatible = 2 * (Precision_Compatible * recall_Compatible) / (Precision_Compatible + recall_Compatible)
> F_Compatible
[1] NaN
```

Here we find out the TP, TN, FN, FP , Recall, Precision and F- measure for Compatible class.

```
cat("For  Highly Compatible  Class (TP,FN,FP,TN): ")
TP_HighlyCompatible <- cm$table[2, 2]
cat("True Positives (TP):", TP, "\n")


FN_HighlyCompatible <- cm$table[2, 1]+ cm$table[2, 3]
cat("False Negatives (FN):", FN, "\n")


FP_HighlyCompatible <- cm$table[1, 2]+cm$table[3, 2]
cat("False Positives (FP):", FP, "\n")


TN_HighlyCompatible <- cm$table[1, 1]+cm$table[1,3]+cm$table[3,1]+cm$table[3, 3]
cat("True Negatives (TN):", TN, "\n")


cat("Precision For Highly Compatible  Class :")
Precision_HighlyCompatible = TP_HighlyCompatible / (TP_HighlyCompatible +
FP_HighlyCompatible)
Precision_HighlyCompatible
```

```
cat("Recall For Highly Compatible  Class :")
recall_HighlyCompatible=TP_HighlyCompatible/(TP_HighlyCompatible+FN_HighlyCompa
tible)
recall_HighlyCompatible

cat("F-Measure Highly Compatible  Class :")
F_HighlyCompatible = 2 * (Precision_HighlyCompatible * recall_HighlyCompatible) /
(Precision_HighlyCompatible + recall_HighlyCompatible)
F_HighlyCompatible
```

```
> cat("For  Highly Compatible  Class (TP,FN,FP,TN): ")
For  Highly Compatible  Class (TP,FN,FP,TN):
> TP_HighlyCompatible <- cm$table[2, 2]
> cat("True Positives (TP):", TP, "\n")
True Positives (TP): 0
> FN_HighlyCompatible <- cm$table[2, 1]+ cm$table[2, 3]
> cat("False Negatives (FN):", FN, "\n")
False Negatives (FN): 0
> FP_HighlyCompatible <- cm$table[1, 2]+cm$table[3, 2]
> cat("False Positives (FP):", FP, "\n")
False Positives (FP): 31.79572
> TN_HighlyCompatible <- cm$table[1, 1]+cm$table[1,3]+cm$table[3,1]+cm$table[3, 3]
> cat("True Negatives (TN):", TN, "\n")
True Negatives (TN): 68.20428
> cat("Precision For Highly Compatible  Class :")
Precision For Highly Compatible  Class :
> Precision_HighlyCompatible = TP_HighlyCompatible / (TP_HighlyCompatible + FP_HighlyCompatible)
> Precision_HighlyCompatible
[1] 1
> cat("Recall For Highly Compatible  Class :")
Recall For Highly Compatible  Class :
> recall_HighlyCompatible=TP_HighlyCompatible/(TP_HighlyCompatible+FN_HighlyCompatible)
> recall_HighlyCompatible
[1] 0.4431631
> cat("F-Measure Highly Compatible  Class :")
F-Measure Highly Compatible  Class :
> F_HighlyCompatible = 2 * (Precision_HighlyCompatible * recall_HighlyCompatible) / (Precision_HighlyCompatible + recall_HighlyCompatible)
> F_HighlyCompatible
[1] 0.6141553
```

Here we find out the TP, TN, FN, FP , Recall, Precision and F- measure for Highly Compatible class.

```
cat("For  Moderately  Compatible  Class (TP,FN,FP,TN): ")
TP_ModeratelyCompatible <- cm$table[3, 3]
cat("True Positives (TP):", TP,"\n")


FN_ModeratelyCompatible <- cm$table[3, 1]+ cm$table[3, 2]
cat("False Negatives (FN):", FN, "\n")


FP_ModeratelyCompatible <- cm$table[1, 3]+cm$table[2, 3]
cat("False Positives (FP):", FP, "\n")


TN_ModeratelyCompatible <- cm$table[1, 1]+cm$table[1,2]+cm$table[2,1]+cm$table[2, 2]
cat("True Negatives (TN):", TN, "\n")
```

```
cat("Precision For Compatible  Class :")
Precision = TP_ModeratelyCompatible / (TP_ModeratelyCompatible +
FP_ModeratelyCompatible)

cat("Recall For Compatible  Class :")
recall_ModeratelyCompatible=TP_ModeratelyCompatible/(TP_ModeratelyCompatible+FN
_ModeratelyCompatible)

cat("F-Measure Compatible  Class :")
F1_ModeratelyCompatible = 2 * (Precision_ModeratelyCompatible *
recall_ModeratelyCompatible) / (Precision_ModeratelyCompatible +
recall_ModeratelyCompatible)
F1_ModeratelyCompatible
```

```
> cat("For  Moderately  Compatible  Class (TP,FN,FP,TN): ")
For  Moderately  Compatible  Class (TP,FN,FP,TN):
> TP_ModeratelyCompatible <- cm$table[3, 3]
> cat("True Positives (TP):", TP,"\n")
True Positives (TP): 0
> FN_ModeratelyCompatible <- cm$table[3, 1]+ cm$table[3, 2]
> cat("False Negatives (FN):", FN, "\n")
False Negatives (FN): 0
> FP_ModeratelyCompatible <- cm$table[1, 3]+cm$table[2, 3]
> cat("False Positives (FP):", FP, "\n")
False Positives (FP): 31.79572
> TN_ModeratelyCompatible <- cm$table[1, 1]+cm$table[1,2]+cm$table[2,1]+cm$table[2, 2]
> cat("True Negatives (TN):", TN, "\n")
True Negatives (TN): 68.20428
> cat("Precision For Compatible  Class :")
Precision For Compatible  Class :
> Precision = TP_ModeratelyCompatible / (TP_ModeratelyCompatible + FP_ModeratelyCompatible)
> cat("Recall For Compatible  Class :")
Recall For Compatible  Class :
> recall_ModeratelyCompatible=TP_ModeratelyCompatible/(TP_ModeratelyCompatible+FN_ModeratelyCompatible)
> cat("F-Measure Compatible  Class :")
F-Measure Compatible  Class :
> F1_ModeratelyCompatible = 2 * (Precision_ModeratelyCompatible * recall_ModeratelyCompatible) / (Precision_ModeratelyCompatible + recall_ModeratelyCompat
ble)
> F1_ModeratelyCompatible
[1] NaN
>
```

Here we find out the TP, TN, FN, FP, Recall, Precision and F- measure for Moderately Compatible class.