# Managing Big Data:
# 3rd Assignment

February 2, 2023


Dimakopoulos Andreas (1067668)
Kountakis Marinos (1069158)
Mantzari Eleni (1067704)
Moustakas Andreas (1067656)
Fragkoulia Maria(1069868)

Supervisor: Tzagkarakis Emmanouil

School of Economics and Business Administration
Department of Economics
Master of Science:
"Applied Economics and Data Analysis"

# Task 1

This section contains the summaries of the given articles. The relevant articles are as follows, "A hybrid approach for generating reputation based on opinions fusion and sentiment analysis" by Abdessamad Benlahbib and El Habib Nfaoui, "The Determinants of Bitcoin's Price: Utilization of GARCH and Machine Learning Approaches" by Ting-Hsuan Chen · Mu-Yen Chen and Guan-Ting Du, "Recession Forecasting Using Bayesian Classification" by Troy Davig and Aaron Smalter Hall. The goal of this task is to answer the following questions:

1. What is the objective of the work?
2. Does the approach adopted present anything innovative? If so what?
3. Which models/algorithms (machine and non-machine learning) have been used and why?
4. What data sets have been used and what variables they contained this data?

## Summary of the first paper — A hybrid approach for generating reputation based on opinions fusion and sentiment analysis

The paper *"A hybrid approach for generating reputation based on opinions fusion and sentiment analysis"* by Abdessamad Benlahbib and El Habib Nfaoui presents a method for generating a reputation score for a product or service based on a combination of sentiment analysis and opinion fusion. The aim of the research paper is to propose a hybrid approach for generating reputation scores for a product or service based on a combination of sentiment analysis and opinion fusion. The approach adopted in the paper presents something innovative, it proposes a hybrid approach that combines two methods: the first one is a lexicon-based method and the second one is a machine learning-based method. The lexicon-based method uses a set of predefined words and

expressions (positive, negative, neutral) to classify the reviews, the machine learning-based method uses a set of reviews and their corresponding classes (positive, negative, neutral) to train a classifier.

The models/algorithms used in the paper are lexicon-based and machine learning-based methods. The lexicon-based method uses a set of predefined words and expressions (positive, negative, neutral) to classify the reviews, the machine learning-based method uses a set of reviews and their corresponding classes (positive, negative, neutral) to train a classifier. Both methods are used to classify reviews based on their sentiment and then opinions fusion is used to combine the sentiment scores from each review to generate a final reputation score for the entity.

The authors used a dataset of online reviews of hotels and restaurants. The dataset contains variables such as the opinion, the sentiment, and the source of the opinion. The algorithm models used were evaluated by comparing the results of the proposed approach to other existing methods for reputation generation, using precision, recall, and F1-score as evaluation metrics.Their main findings of this publication indicate that the proposed hybrid approach is effective in generating reputation scores for entities. The research reaches the conclusion that the proposed approach can be applied to a wide range of domains, such as social media and e-commerce to help people make better decisions about products or services by providing them with an overall reputation score for each entity.

## Summary of the second paper — The Determinants of Bitcoin's Price: Utilization of GARCH and Machine Learning Approaches

The purpose of the work is to find the determinants of Bitcoin's price (2010-2018). In terms of the approach used in this particular paper, one of the techniques is the SVM, a supervised machine learning technique. This technique is innovative and hasn't been used before. SVM divides a set of data into two more precise categories without uniformizing the distribution of the data. The models developed in the study include a GARCH model, an SVM model, and a third model with the KKT condition. A time series model called the GARCH (Generalized Autoregressive Conditional Heteroskedastic Model) is frequently used to forecast future price variations. In order to investigate the factors influencing price trends that affect Bitcoin, this study evaluates GARCH results under various orderings. An even data distribution is produced using SVM by determining the best decision boundary to optimize the margins. The KKT condition is used in the study to maximize margins in

accordance with the optimization concept. The World Gold Council website, Federal Reserve Economic Data (FRED), investing.com, bitcoinity.org, and other websites were used to gather the data for this study. The Bitcoin pricing was gathered by Bitcoinity. org. The websites of Investing.com, FRED, and the World Gold Council are used to obtain additional variables, such as stock market indices, oil prices, exchanged rates, interest rates, and gold prices. The sample period covers the dates of July 19, 2010, and December 31, 2018. Bitcoin was first exchanged on July 19, 2010. All information is gathered as daily values. Days that are unavailable or not trading are not included in the analysis. The 1955 observations make up the final sample.

ARCH-LM test is used in this study to evaluate the series' volatility. It turns out that the GARCH model accurately predicted the volatility. The ARCH-LM test's null hypothesis is disregarded because the p value was less than 1% of significant. The F-statistic also disproved the null hypothesis in the identical circumstance. The ARCH LM test on the model's residuals revealed the existence of the ARCH effect, indicating that the conditional heteroscedasticity was present in the price series for Bitcoin. For the classification prediction, this study used the classification regression tree C4.5 model. Table 5 demonstrates that this model successfully predicts Bitcoin prices with an accuracy rate of up to 98%, and the values obtained in the confusion matrix are made up of the categorization and prediction outcomes. This study may see that it should be categorised as L when comparing prices that are below (L) and above (H) the median Bitcoin price. Only 22 of the 977 (L) data points were incorrectly classified, yielding a 90% accuracy rate; similarly, only 17 of the 978 (H) data points were incorrectly classified, yielding a 90% accuracy rate. Therefore, it is clear from this study that the decision tree model for categorization is quite effective.

To sum up everything that has been stated so far, this study shows that the price of bitcoin is positively impacted by exchange rates. Due to the politicization of US monetary policy in recent years, which has caused investors to lose faith in the government and turn to unrestricted Bitcoin, it can be concluded from the decision tree model that the Fed funds rate has the most impact on the price of Bitcoin. And lastly, this study finds that, aside from the actual Bitcoin price, potential Bitcoin investors can use the abovementioned USD/ Euro, USD/GBP, USD/CHF, and Euro/GBP exchange rates, along with the Fed funds rate, the FTSE 100, the US dollar index, the DAX, the Nikkei 225 and the price of gold as reference indicators.

## Summary of the third paper — Recession Forecasting Using Bayesian Classification

In their paper "Recession Forecasting Using Bayesian Classification" Troy Davig and Aaron Smalter Hall used Naïve Bayes model as a recession forecasting tool. They used Markov-switching models and logistic regression (such as Neftci (1982) and Diebold and Rudebusch (1989)). However, in Markov-switching, Naïve Bayes treats National Bureau of Economic Research business cycle turning points as data, rather than hidden states to be inferred by the model. In this paper Naïve Bayes has a larger asymptotic error rate, but converges to the error rate faster than logistic regression, resulting in more accurate recession forecasts with limited data. The biggest difference in relation to previous studies is that the data set is richer, lag structure and capture the persistence of business cycle phases by using Markov-switching transition probabilities. Although, their approach is closely connect to Markov-switching time-series models, except they treat NBER turning points as data when identifying past recessions and expansions rather than something to be inferred. Models and algorithms (machine learning and non-machine learning) have been used in this paper are Naïve Bayes, Markov-switching models and logistic regression, probit and logit.

Data that they used was 135 macroeconomic variables at monthly intervals, recorded in the FRED-MD data set, - the observation period was from January 1959 through June 2016. However, they create the following "core" set from four variables from these 135 variables: ISM Manufacturing: Production Index (NAPMPI), Total non-farm payroll growth (PAYEMS), 10-year treasury rate minus Fed funds rate (T10YFFM) and Sand P's Common Stock Price Index: Composite (SP500). For the "core" set they make transformation only the total non-farm payroll growth and S and P's Common Stock Price Index, namely they took first difference of consecutive values. To supplement this data, they use the Federal Reserve Bank of Philadelphia's Real-time Data Set for Macroeconomists for real-time values of total non-farm payrolls in the core set of four variables. To evaluate forecasting performance, they used the F-measure under a zero-one loss. This approaches penalizes trivial classifiers. Would be correct most of the time, but lacks true predictive ability. They also evaluate outcomes under mean absolute error and weighting each observation uniformly.

They came to the conclusion that Naïve Bayes (NB) for business cycle turning point forecasting are better than logistic regression in almost every situation. One reason, is that NB converges to its asymptotic error rate faster than logistic regression. Another reason, is NB can easily incorporate a large amount of data. So, recession forecasting accuracy diverges across the two

approaches, with NB being considerably better under a range of criteria.

# Task 2

## R Code— Subtask i

```
# Install
install.packages("tm")  # for text mining
install.packages("SnowballC") # for text stemming
install.packages("syuzhet") # for sentiment analysis
install.packages("stringr") # replace all occurrences of a specific
#string or character within another string
install.packages("caTools") # install and load the caTools library
install.packages("e1071") # contains the Naive Bayes function
install.packages("caret") #for confusion matrix

# Load
library(tm)# Library "tm" is used to remove stopwords
#(We use stopwords for english)
library(SnowballC)# Library "SnonwballC" is used to stem words
library(stringr)
library(syuzhet)
library(dplyr)
library(caTools)
library(e1071)
library(caret)

graphics.off() ; rm(list = ls(all = TRUE)) ; cat("\014");

# Pre-processing
# Create the functions to be used for the pre-processing of data

# Clean up any HTML element
cleantext <- function(text) {

        # Remove the HTML elements
        return(gsub("<.*?>", "", text))
}

# Remove words that contain symbols that
# are not letters or numbers
is_spec <- function(text){
```

```
        # Remove the symbols
        text = str_replace_all(text, "[^[:alnum:]]", " ")

        return (text)
}

# Convert to lowercase
to_lower <- function(text) {
        text <- tolower(text)
        return(text)
}


#Remove stopwords
rem_stopwords <- function(text) {
        # Convert the input to a Corpus
        text <- Corpus(VectorSource(text))

        # Remove the stopwords
        text <- tm_map(text, removeWords, stopwords("SMART"))

        # Convert the Corpus back to character vector
        text <- unlist(sapply(text, as.character))

        return(text)
}


#Stem words
stem_txt <- function(text) {
        # Convert the input to a Corpus
        text <- Corpus(VectorSource(text))

        # Stem the words
        text <- tm_map(text, stemDocument, language = "english")

        # Convert the Corpus back to character vector
        text <- unlist(sapply(text, as.character))

        return(text)
}
```

```r
remove_numbers <- function(text) {
        # Use gsub() to replace all digits with an empty string
        text <- gsub("[0-9]", "", text)
        return(text)
}

# Read the text file
data <- read.csv('IMDBDataset.csv', stringsAsFactors = FALSE)

data$sentiment = as.factor( data$sentiment)

# Preprocessing
data$review <- cleantext(data$review)

data$review <- is_spec(data$review)

data$review <- to_lower(data$review)

data$review <- rem_stopwords(data$review)

data$review <- stem_txt(data$review)

data$review <- remove_numbers(data$review)


corpus <- Corpus(VectorSource(data$review))

dtm <- DocumentTermMatrix(corpus)

inspect(dtm)

#Create a vector saving the number of
#those dtm terms that
#occur at least 1000 times
freqTerms <- findFreqTerms(dtm, 1000)

#Create a new data frame keeping all rows
#and only the frequent terms
dtm_freq <- dtm[ , freqTerms]

#Convert the dtm with frequent terms to matrix
dtm_freq <- as.matrix(dtm_freq)
```

```
#Create a vector containing the values
#of the class attribute
y <- data$sentiment

#Create a df containing the frequent term outcomes and
#the class attribute
newData = data.frame(dtm_freq, y)


# Build a term-document matrix
TextDoc_dtm <- DocumentTermMatrix(data)
dtm_m <- as.matrix(TextDoc_dtm)


# create a data frame for training and testing
#critics_df <- data.frame(review = data$review,
# sentiment = as.factor(data$sentiment))


# split the data into training and testing sets
set.seed(1234) # for reproducibility

split_index <- sample(1:nrow(data), round(nrow(data) * 0.8))
train_data <- data[split_index, "review"]
test_data <- data[-split_index, "review"]

# Create a vector containing the values of
#the class attribute
y_train <- data[split_index, "sentiment"]

# Create the model #"naiveBayes" function
NaiveBayesModel <- naiveBayes(train_data, y_train)


#"predict" function to predict the class of new data
predictions <- predict(NaiveBayesModel, test_data)

#calculate the accuracy by taking the mean of the
#correct_predictions variable,
#which will give the proportion of correctly
#classified observations.
accuracy <- mean(predictions == data[-split_index, "sentiment"])
print(accuracy)
```

```
#Alternative way for accuracy
confusionMatrix(predictions, data[-split_index, "sentiment"])
```

## Hypothetical Question— Subtask ii

To use the classifier trained in question I to answer the question of: "How user comments on social networks affect the price of a particular product found in the supermarket".

We would first collect data on user comments about the product on social networks. This data would come from various sources such as Twitter, Facebook, Instagram, etc. The data would take the form of text comments, along with any relevant metadata such as the data and time the comment was posted and the user's name. The data would be preprocessed to clean it and remove any irrelevant information. This might include removing special characters, stop words and stemming or lemmatizing the text. Additionally, we might also, use sentiment analysis techniques such as sentiment dictionaries to extract sentiment information from the comments.

To classify the sentiment of the comments, we would use the Naïve Bayes algorithm, which is a probabilistic algorithm that is efficient, easy to implement and can handle high-dimensional datasets, which is why we would use it. After training the classifier using the training data, we would use the "predict" function to predict the sentiment of the comments. After collecting the sentiment of each comment, we would collect data on the prices of the product at the supermarket over time. To understand the relationship between the sentiment of the comments and the prices of the product, we would use statistical methods such as correlation or regression analysis. Correlation analysis would be used to determine the strength of a linear relationship between two variables, while regression analysis would be used to model the relationship between two variables.

Finally, we would interpret the results of the analysis to draw conclusions about how user comments on social networks affect the price of the product. It is important to highlight that this is a simplified scenario and in real-world scenario many other factors also affect the prices of the products and this analysis alone may not be enough to draw solid conclusions.

# Task 3

## R Code— Subtask i

```r
# install.packages("rpart")
# install.packages("rpart.plot")
library(rpart)
library(rpart.plot)

setwd("C:/Master/BD/Projects/Project3")

# Reading the mushroom data
musdata <- read.table("agaricuslepiota.data", sep=",", header=FALSE)
musdata[, 1] <- factor(musdata[, 1], levels = c("e", "p"),
                       labels = c("edible", "poisonous"))
colnames(musdata)[1] <- "classes"

#write.csv(musdata, file = "musdata.csv", row.names = FALSE)
# --> if i want to export this dataset

# Splitting the data into training and testing sets
set.seed(123)
splitindex <- sample(1:nrow(musdata), size = 0.8*nrow(musdata))
training <- musdata[splitindex,]
testing <- musdata[-splitindex,]

# Creating and visualizing the decision tree
treemodel <- rpart(classes ~ ., data = training, method = "class")
rpart.plot(treemodel, extra = 104)

# Predictions using the model
predictions <- predict(treemodel, newdata = testing, type = "class")
print(predictions)

# Confusion Matrix
confmatrix <- table(predictions, testing$classes)
print(confmatrix)

# Accuracy
accuracy <- sum(diag(confmatrix))/sum(confmatrix)
print(accuracy)
```

## Results R— Subtask i

This R code builds a decision tree using the "rpart" library to predict whether a mushroom is edible or poisonous based on various features of the mushroom. The code first loads the "rpart" and "rpart.plot" libraries, sets the working directory to the folder where the mushroom data is stored, and then reads the mushroom data into a dataframe called "musdata." The "musdata" is then split into a training set and a testing set.

A decision tree model is built on the training set using the "rpart" function, and the tree structure is visualized using the "rpart.plot" function. The predictions on the testing set are obtained using the "predict" function, and the accuracy of the model is calculated based on the confusion matrix created by comparing the predicted and actual values. The accuracy is calculated as the ratio of the number of correct predictions to the total number of predictions.

The confusion matrix below is showing the results of the predictions made by the decision tree model on the testing data set. The columns represent the actual class of the samples and the rows represent the predicted class of the samples.

| predictions | ***edible*** | ***poisonous*** |
|---|---|---|
| ***edible*** | 864 | 12 |
| ***poisonous*** | 0 | 749 |

From the matrix, it can be seen that:

864 samples were correctly classified as edible,

12 samples were misclassified as poisonous while they were actually edible,

0 samples were misclassified as edible while they were actually poisonous,

749 samples were correctly classified as poisonous.

From the confusion matrix we can generate the accuracy score of the prediction. The accuracy score is a measure of how well the model's predictions match the actual outcomes - in our example, the true class labels of the mushrooms in the testing data set.

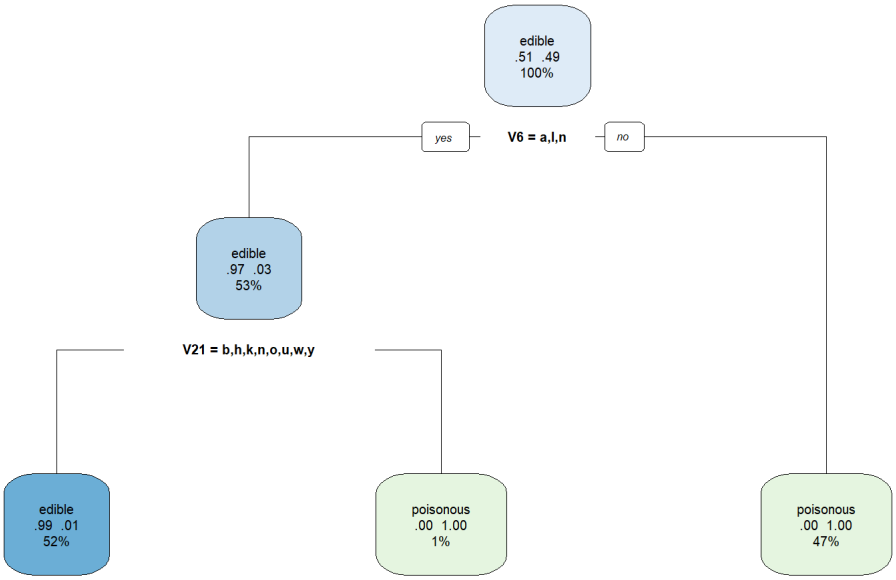
$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

The number of correct predictions is $864 + 749 = 1613$, and the total number of predictions is $876 + 761 = 1637$. So, the accuracy score would be calculated as:

$$\text{Accuracy} = \frac{1613}{1637} = 0.9926154$$

For our example, the accuracy score of 0.9926154 means that the model correctly predicted the class labels for 99.26% of the mushrooms in the testing data set.

Lastly, we visualized the results in a decision tree.




## Python Code— Subtask i

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt

# load the dataset
mushroom_data = pd.read_table("agaricuslepiota.data", header
                              = None, delimiter = ',')

# column names
```

```
column_names = ['edibility', 'cap-shape', 'cap-surface', 'cap
                                -color', 'bruises', 'odor', '
                                gill-attachment', 'gill-
                                spacing', 'gill-size', 'gill-
                                color', 'stalk-shape', 'stalk-
                                root', 'stalk-surface-above-
                                ring', 'stalk-surface-below-
                                ring', 'stalk-color-above-ring
                                ', 'stalk-color-below-ring', '
                                veil-type', 'veil-color', '
                                ring-number', 'ring-type', '
                                spore-print-color', '
                                population', 'habitat']
mushroom_data.columns = column_names
print (mushroom_data.columns)
print (mushroom_data.head())

# convert categorical variables to numerical variables with
                                One-hot encoding.
# a list of the names of our categorical variables.
categoricalVariables=mushroom_data.select_dtypes([object]).
                                columns

for var in categoricalVariables:

print("\tOne-Hot-Encoding variable ",var, " .....", sep="",
                                end="")
if var == "edibility":
print("Ignored")
continue
mushroom_data[var]=pd.Categorical(mushroom_data[var])
varDummies = pd.get_dummies(mushroom_data[var], prefix = var)
mushroom_data = pd.concat([mushroom_data, varDummies], axis=1
                                )
mushroom_data=mushroom_data.drop([var], axis=1)
print("Done")

print("\n\tVariables of DataFrame bankData after One-hot
                                encoding:")
print("\t", mushroom_data.columns)

# We add a new column to the end of DataFrame mushroom_data
                                named newY, containing these
                                new values of 0 and 1.
# Once we have done this, the original column/variable y is
                                not needed anymore and can be
                                dropped.
mushroom_data['newY'] = ( mushroom_data['edibility'].map( {'p
                                ':0, 'e':1}) )
```

```
mushroom_data=mushroom_data.drop(['edibility'], axis=1)
print("\n\nPreprocessing done.")

# target variable
y = mushroom_data["newY"]

# feature variables
X = mushroom_data.iloc[:, :-1]

# split the dataset into training and test sets
trainingSetFeatures, testingSetFeatures, trainingSetClass,
                                testingSetClass =
                                train_test_split(X, y,
                                test_size = 0.2, random_state
                                = 0)

# build the decision tree model
model = DecisionTreeClassifier(criterion="entropy")
model_fit = model.fit(trainingSetFeatures, trainingSetClass)

# make predictions on the test set
predictions = model.predict(testingSetFeatures)

# calculate the confusion matrix and accuracy
cm = confusion_matrix(testingSetClass, predictions)
acc = accuracy_score(testingSetClass, predictions)

# print the confusion matrix and accuracy
print("Confusion matrix:")
print(cm)
print("Accuracy:", acc)

# visualize the decision tree
plt.figure(figsize=(15,5))
tree.plot_tree(model, filled=True, feature_names=X.columns,
                                class_names=["edible", "
                                poisonous"])
plt.show()
```

## Results Python— Subtask i

The above Python code aims to build a decision tree model to classify the edibility of mushrooms based on their various features, evaluate the model's performance using a confusion matrix and accuracy score, and visualize the decision tree. The data is loaded from a file and processed to prepare it for modeling, including converting categorical variables to numerical variables

using one-hot encoding, splitting the data into training and test sets, and fitting the decision tree model using the training data. The model's performance is evaluated using the test data, and the confusion matrix and accuracy score are calculated. Finally, the decision tree is visualized to provide a visual representation of how the model makes its predictions.

Before commenting on the results it is important to mention that there is a difference in results between the Python and R code. This difference likely arose from the fact that it was necessary to convert the categorical variables to numerical variables (by making dummies) before building the decision tree in Python, but not in R. This step is required in Python because decision trees "work" with numerical input and cannot handle categorical data directly. On the other hand, some implementations of decision trees in R can handle categorical data natively.

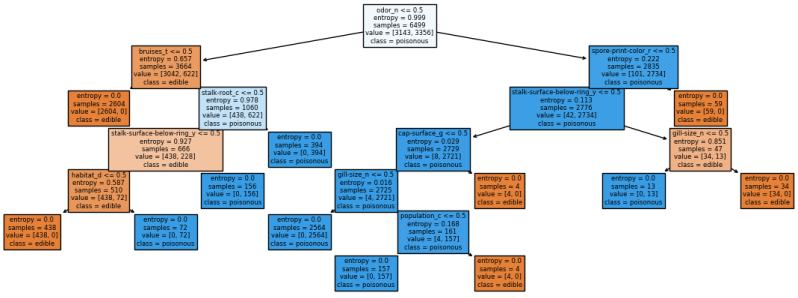
By converting the categorical data to numerical data in Python, we effectively transformed the original data into a different representation that can be used as input to the decision tree algorithm. This can lead to different results in the decision tree structure, performance metrics, and visualizations, compared to running the same algorithm on the original categorical data in R.

Thus, the generated confusion matrix from Python configures as:

| predictions | ***edible*** | ***poisonous*** |
|---|---|---|
| ***edible*** | 773 | 0 |
| ***poisonous*** | 0 | 852 |

The accuracy score of 1 suggests that the model may be overfitted. The high accuracy and perfect confusion matrix could be due to various factors such as the nature of the data, the algorithms used, the model parameters, etc. It is important to consider other factors such as the class distribution in the data, the model parameters, and the evaluation metrics used to assess the model performance.

Lastly, we visualized the results in a decision tree.



## Entropy— Subtask ii

To calculate the Entropy gain of the feature "habitat" using the first 30 records of the Mushroom Data Set, we followed these steps:

1. We calculated the entropy of the target variable "edibility" for the entire dataset (all 30 records).

2. For each unique value of the feature "habitat" in the dataset (e.g. "u", "g", "d", "l", "m", "p", "w"), we calculatet the entropy of the target variable "edibility" for the subset of records that have that value of "habitat".

3. For each unique value of the feature "habitat" in the dataset, we calculated the weighted average of the entropies calculated in step 2.

4. Lastly, we calculated the Entropy gain of the feature "habitat" by subtracting the weighted average of the entropies calculated in step 3 from the entropy of the target variable "edibility" calculated in step 1.

| | | |
|---|---|---|
| ***Number of poisonous: "p"*** | 9 | 0,3 |
| ***Number of edible, "e"*** | 21 | 0,7 |
| ***Total:*** | ***30*** | ***1*** |

The logarithmic calculations are generated as follow:

log(9/30) = -0,522878745

log(21/30) = -0,15490196

log(2) = 0,301029996

$log_2(9/30)$ = -1,736965594

$log_2(21/30)$ = -0,514573173

Before generating the entropy measures we calculated the number of times each unique value appear in the 30 first observations.

| ***Values*** | ***Class = p*** | ***Class = e*** | ***Sum*** | ***Perc.*** |
|---|---|---|---|---|
| *grasses* | 4 | 7 | 11 | 0,367 |
| *leaves* | 0 | 0 | 0 | 0 |
| *meadows* | 0 | 11 | 11 | 0,367 |
| *paths* | 0 | 0 | 0 | 0 |
| *urban* | 5 | 2 | 7 | 0,23 |
| *waste* | 0 | 0 | 0 | 0 |
| *wood* | 0 | 1 | 1 | 0,03 |
| ***Total*** | | | 30 | 1 |

Lastly, we calculated the required logarithmic values.

| | ***for "p"*** | ***log*** | ***log2*** | ***for "e"*** | ***log*** | ***log2*** |
|---|---|---|---|---|---|---|
| *grasses* | 0,36 | -0,44 | -1,46 | 0,64 | -0,20 | -0,65 |
| *meadows* | 0 | - | - | 1 | 0 | - |
| *urban* | 0,71 | -0,15 | -0,49 | 0,29 | -0,54 | -1,81 |
| *wood* | 0 | - | - | 1 | 0 | - |

The Entropy measures are:

$$\text{Entropy Node} = -0.3 * (-1.74) - 0.7 * (-0.51) = 0.881$$
$$\text{Entropy(Habitat)} = 0.37 * (-(0.36 * (-1.46)) - (0.64 * (-0.65))) + 0.37 * 0 + 0.23 * (-(0.71 * (-0.49)) - (0.29 * (-1.81))) + 0.03 * 0 = 0.558$$

As a result, the entropy gain of the "habitat" feature would be the entropy of the "edible/poisonous" categorization feature minus the weighted average of the entropy for each possible value of the "habitat" feature:

$$\text{Entropy Gain} = 0.881 - 0.558 = 0.333$$

An entropy gain of 0.333 for the "habitat" feature indicates that the feature has reduced the uncertainty or entropy in the "edible/poisonous" categorization by 0.333, after taking into account all possible values of the feature. Meaning that, the feature provides some useful information for predicting the "edible/poisonous" categorization.

## Python Code— Subtask iii

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.naive_bayes import GaussianNB, BernoulliNB,
                                MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split

# load the dataset
mushroom_data = pd.read_table("agaricuslepiota.data", header
                                = None, delimiter = ',')

# column names
column_names = ['edibility', 'cap-shape', 'cap-surface', 'cap
                                -color', 'bruises', 'odor', '
                                gill-attachment', 'gill-
                                spacing', 'gill-size', 'gill-
                                color', 'stalk-shape', 'stalk-
                                root', 'stalk-surface-above-
                                ring', 'stalk-surface-below-
                                ring', 'stalk-color-above-ring
                                ', 'stalk-color-below-ring', '
                                veil-type', 'veil-color', '
                                ring-number', 'ring-type', '
                                spore-print-color', '
                                population', 'habitat']
mushroom_data.columns = column_names

# replacing any ? for missing values
mushroom_data = mushroom_data.replace('?', np.NaN)

# droppingthe missing values
mushroom_data = mushroom_data.dropna(0, how='any')

mushroom_data.info()

# Now we have to tell our dataset that the values 'y', 'n'
                                are categorical (or factors in
                                 R parlance).
# This is called labelling in pandas. This will resule in one
                                 value 'y' to be encoded as 1
                                and 'n' as 0.
mushroom_data = mushroom_data.apply(preprocessing.
                                LabelEncoder().fit_transform)

trainSet, testSet = train_test_split(mushroom_data, test_size
                                =0.2, random_state = 0)
```

```
# trainClass will contain only the first column of the
                                dataset (iloc enables integer-
                                based indexing)
trainClass = trainSet.iloc[:,0] # -> all rows (:), first
                                column (0). IMPORTANT: Python
                                uses 0-based indexing!

# All other attributes except first one will form our
                                training data
trainData = trainSet.iloc[:, 1:16]

# Slice the testing set into 2 parts:  one that contains only
                                 the first row which is the
                                class attribute and
# another one that contains only all the other variables. We
                                do this in order to calculate
                                the confusion matrix
# at the end
testClass = testSet.iloc[:,0]
testData = testSet.iloc[:,1:16]

# Create the Multinomial Naive Bayes classifier object to
                                properly use a probability
mNB = MultinomialNB()
# Train the Naive Bayes classifier using the training set
NBModel = mNB.fit(trainData, trainClass)

#Predict the class attribute
testResults = NBModel.predict(testData)

# Done. Let's claculate and display the confusion matrix
cmatrix = confusion_matrix(testClass, testResults)
print(cmatrix)


# Let's see the accuracy of the Naive Bayes classifier we
                                just built
accS = accuracy_score(testClass, testResults)
print("Accuracy score = ", accS )
```

## Results Python— Subtask iii

In this task we used the Multinomial Naive Bayes classification algorithm. The purpose of task is to build a Naive Bayes classifier to predict the edibility of mushrooms based on various features. Multinomial Naive Bayes is a type of Naive Bayes classifier that is suited for discrete data, such as text data, which are count-based. In this case, the data consists of categorical variables, so the Multinomial Naive Bayes is appropriate.

We load a mushroom dataset stored in a text file and perform several preprocessing tasks to prepare the data for modeling. The preprocessing tasks include replacing any missing values, encoding categorical variables, and splitting the dataset into a training set and a testing set. The Naive Bayes model is trained using the training data, and the accuracy of the model is evaluated using the test data. Then, we calculate the confusion matrix and the accuracy score to measure the performance of the model, which provide information about how well the model classifies the test data.

The confusion matrix below displays the results of a binary classification model, where the values on the left diagonal represent the number of true positive (edible) and true negative (poisonous) predictions, and the values off the diagonal represent the number of false positive (predicted as edible but actually poisonous) and false negative (predicted as poisonous but actually edible) predictions.

| predictions | ***edible*** | ***poisonous*** |
|---|---|---|
| ***edible*** | 623 | 73 |
| ***poisonous*** | 120 | 309 |

From the matrix, it can be seen that:

623 samples were correctly classified as edible,

73 samples were misclassified as poisonous while they were actually edible,

120 samples were misclassified as edible while they were actually poisonous,

309 samples were correctly classified as poisonous.

The accuracy score of 0.83, which is calculated as (true positive + true negative) / total, suggests that the model is able to correctly classify the mushrooms into edible or poisonous categories 83% of the time.

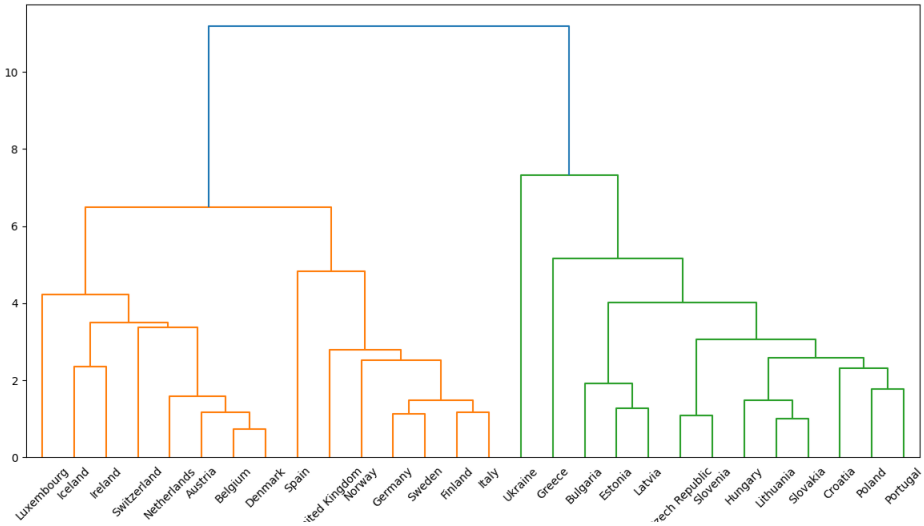
# Task 4

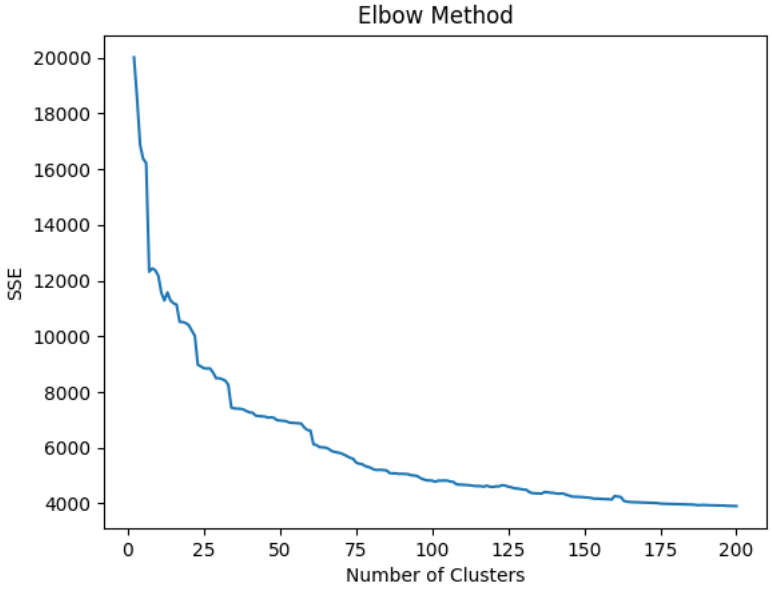
## Python Code

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

movies = pd.read_csv('/Users/dhmako/Documents/Assignments in
                                Managing Big Data/Third
                                Assignment/movies.csv')

from kmodes.kmodes import KModes

cat = movies.drop(['movieId','title'],axis=1)

clusters = 200
sse = []

############ VERY LONG TO EXECUTE ############

for k in range(2, clusters+1):
print("\t>>> Executing Kmode with k=", k, "clusters.....",
                                end="")
kmode = KModes(n_clusters=k, max_iter=500, n_init=10)
# .fit() method actually does the kmeans clustering of
                                wineData
kmode.fit(cat)
# .inertia_ contains the value of the objective function
                                which is the sum of squared
                                distances
# across all clusters (SSE). Add the sum of squared distances
                                 into the list, so that
# we can later diplay it
sse.append(kmode.cost_)
print("Done.")

print('\n K-finding iterations done...')

kmode.cluster_centroids_

print('\n Display plot to apply the Elbow method')

plt.plot(range(2,clusters+1),sse)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
```

```
plt.show()

# k = 100 looks like a good value...

print("Executing KModes with k=100 (100 clusters) to get
                                FINAL clustering of the data
                                ...", end="")
kmode = KModes(n_clusters=100, max_iter=500, n_init=10)
clusters = kmode.fit(cat)
print("\nDone")

clusters.labels_

ratings = pd.read_csv('/Users/dhmako/Documents/Assignments in
                                 Managing Big Data/Third
                                Assignment/ratings.csv')


ratings[ratings['userId']==135]['movieId']

len(clusters.cluster_centroids_)

movies['ClusterID'] = clusters.labels_

movies.head(5)

movies[movies['ClusterID']==5]['title']
ratings['ClusterID'] = clusters.labels_

# Assign column ClusterID to ratings dataframe where ratings.
                                movieid = movies.movieid

movies.reset_index()
movies.set_index('ClusterID')
merged_df = ratings.merge(movies[['movieId','ClusterID']], on
                                ='movieId', how='left')

merged_df.isna().sum(0)

# Mean rating for every movie
ratings.groupby('movieId')['rating'].mean()

# Mean rating for every cluster that has a movie user=198
                                rated
user198 = merged_df[merged_df['userId']==198].groupby('
                                ClusterID')['rating'].mean().
                                sort_values(ascending=False)

user198 = pd.DataFrame(user198)
```

```
user198.reset_index()

# Drop clusters with less than 3.5 mean rating
mask = user198['rating']<3.5

user198 = user198.drop(user198[mask].index)
user198.head()
user198 = user198.reset_index()

if len(user198)==0:
print("Sorry, no recommendations for you! The wise speak only
                                of what they know")
else:
print("See recommendations below...")

# Extract the clusters with average rating >= 3.5 for user
                                198
clusters_high_rating = user198[user198['rating'] >= 3.5]['
                                ClusterID']

# Create an empty list to store the movie recommendations
movie_recs = []

# Iterate over clusters with average rating >= 3.5 for user
                                198
for cluster in clusters_high_rating:
# Extract the movies from the merged_df dataframe that belong
                                 to the current cluster and
                                that user 198 has not seen
cluster_movies = merged_df[(merged_df['ClusterID'] == cluster
                                ) & (merged_df['userId'] !=
                                198)]
# Sort the movies by rating in descending order
cluster_movies = cluster_movies.sort_values(by='rating',
                                ascending=False)
# Get the top 2 movies
top_2_movies = cluster_movies.head(2)
# Extract the movie titles
top_2_titles = top_2_movies.merge(movies, on='movieId')['
                                title']
# Append the titles to the movie_recs list
movie_recs.extend(top_2_titles)

# Print the message and the movie recommendations
print("You may also like the following movies:")
for title in movie_recs:
print(title)

print("If you are in doubt, follow your nose...")
```

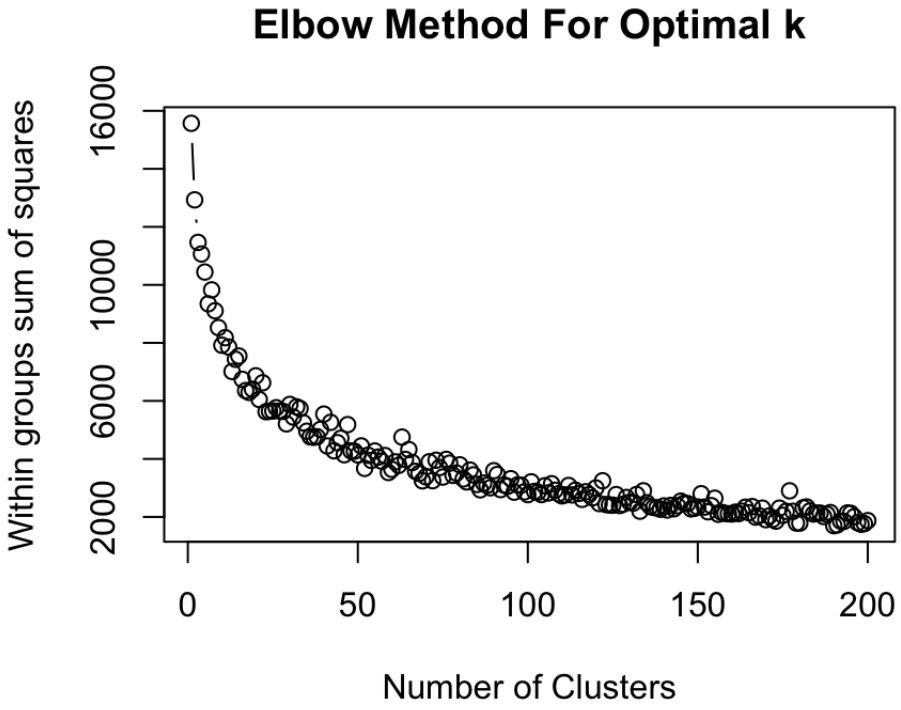

10
8
6
4
2
0
Luxembourg
Iceland
Ireland
Switzerland
Netherlands
Austria
Belgium
Denmark
Spain
Norway
Germany
Sweden
Finland
Italy
Ukraine
Greece
Bulgaria
Estonia
Latvia
Slovenia
Hungary
Lithuania
Slovakia
Croatia
Poland
Portugal


Elbow Method
SSE
20000
18000
16000
14000
12000
10000
8000
6000
4000
0
25
50
75
100
125
150
175
200
Number of Clusters

## R Code

```
graphics.off() ; rm(list = ls(all = TRUE)) ; cat("\014");

library("MASS")
library("klaR")
library(cluster)
library(dplyr)

movies <- read.csv("movies.csv")
ratings <- read.csv("ratings.csv")

cat <- movies[3:22]

#Find the appropriate k based on sum of squared errors

wss <- (nrow(cat)-1)*sum(apply(cat,2,var))
for (i in 2:200) wss[i] <- sum(kmeans(cat, centers=i)$withinss)
plot(1:200, wss, type="b", xlab="Number␣of␣Clusters",
ylab="Within␣groups␣sum␣of␣squares",
main="Elbow␣Method␣For␣Optimal␣k")

#Appropriate k=100
kmodes_result <- kmodes(cat, modes = 100)

#New column with the clusterid (group number)
movies$clusterid <- kmodes_result$cluster

#Get movies that user198 has rated
user198_movies <- ratings %>% filter(userId == 198) %>% select
(movieId, rating, timestamp)

#Get the clusters that the movies user198 has rated belong to
user198_movies <- left_join(user198_movies, movies, by = "movieId")
%>% select(movieId, rating, timestamp, clusterid)

#Group clusters by mean rating
cluster_mean_rating <- user198_movies
%>% group_by(clusterid) %>% summarize(mean_rating = mean(rating))

#Drop clusters with mean rating less than 3.5
cluster_mean_rating <- cluster_mean_rating
```

```
%>% filter(mean_rating >= 3.5)

#If there are no clusters with mean rating over 3.5
#we can't recommend any movies!!
if (nrow(cluster_mean_rating) == 0)
{  print("Sorry, no recommendations for you! ")}
#Do or do not there is no try!
else {  print("See recommendations below")}

#Get movies title and movieid from
 #movies dataframe based on the same clusterid
 movies_mean_ratings <- inner_join(movies, cluster_mean_rating,
 by = "clusterid")

#drop the categories columns
movies_mean_ratings <- movies_mean_ratings %>% select(-c(3:22))

#sort by descending rating order
movies_mean_ratings <- movies_mean_ratings
%>% arrange(desc(mean_rating))

#top 2 rated movies of each cluster
top2_movies_per_cluster <- movies_mean_ratings
%>% group_by(clusterid) %>% slice_head(n=2)

#If there are any movies user198 has already seen, drop them!
top2_movies_per_cluster <- top2_movies_per_cluster
%>%  filter(!movieId %in% user198_movies)

print("You may also like the following movies")
# show the title of the moviestop2_movies_per_cluster$title
```




## Summary of the fourth paper — Cluster Analysis of Economic Data

The application of recently developed similarity measures for objects with nominal variable values is highlighted in the essay, which also introduces various cluster analysis methodologies. According to the studies done (Ulc et al., 2013), clustering using these measures can produce clusters with lower within-cluster variability than the overlap measure.

The study also notes that modern methods that take uncertainty into account are a viable tool for enhancing outcomes in comparison to conventional fuzzy cluster analysis. It also emphasizes how effective methods for handling huge data sets are, such those used in the EU-SILC survey. The paper does point out that commercial software solutions are lagging in their adoption of these cutting-edge techniques and frequently lack measures for nominal variables, fuzzy cluster analysis, and techniques for big data files. As a result, to carry out modern analyses, analysts might need to use various software programs.

# Task 5

## R Code

```
# install.packages("arules")
library(arules)
library(openxlsx)

# Setup our environment. Where is our data?
setwd("C:/Master/BD/Projects/Project3")

graphics.off() ; rm(list = ls(all = TRUE)) ; cat("\014");

# reading the files
data <- read.xlsx("fertility-diagnosis.xlsx")

# Discretize the data set
data <- discretizeDF(data)

# Remove the 2nd and 9th columns
data <- data[,-c(2,9)]

# Create the transaction data set
transactions <- as(data, "transactions")



# Subtask i
# Run the Apriori algorithm
rules <- apriori(transactions, parameter = list(target = "rules"))

# number of rules
nrules <- length(rules)

# print the number of rules
print(nrules)

# print the rules
inspect(rules)


# Subtask ii
# Apply the Apriori algorithm
```

```
altrules <- apriori(transactions, parameter = list(
supp = 0.02, conf = 1, target = "rules"),
appearance = list(rhs = "Diagnosis=0"))


# View the number and content of the returned rules
naltrules <- length(altrules)
print(naltrules)
print(inspect(altrules))

# For checking that alturles consists of support >= 0.02) | Optional
# altrule <- quality(altrules)
# altrule <- altrule %>% filter(support >= 0.02)

al = data.frame(
lhs = labels(lhs(altrules)),
rhs = labels(rhs(altrules)),
altrules@quality)

print(al)

# Subtask iii
# Remove redundant
finalrules <- altrules[!is.redundant(altrules)]

# Inspect
arules::inspect(finalrules)

# Create a dataframe
finalrules = data.frame(
lhs = labels(lhs(finalrules)),
rhs = labels(rhs(finalrules)),
finalrules@quality)
```

## Results R

In this task we perform association rule mining on a dataset of fertility diagnosis. At first, we discretized the data using the discretizeDF function. This function transforms the continuous variables in the dataset into categorical variables. For our data preparation we also remove the "Age at the time of analysis" (2nd) and "Number of hours spent sitting per day ene-16". (9th) columns. The final step in preparing the data is to convert it into a transaction dataset using the as function with the argument "transactions".

Subtask i performs association rule mining using the Apriori algorithm. The apriori function is used to find all the rules in the transaction dataset with the parameter argument set to "rules". The number of rules found is stored in the nrules variable, and the rules themselves are printed using the inspect function.

Subtask ii performs association rule mining using the Apriori algorithm with specific support and confidence values. Support is set to 0.02, which means that a rule should appear in at least 2% of the transactions in the dataset. Confidence is set to 1, meaning that 100% of the transactions that contain the antecedent of a rule should also contain the consequent. These values can be adjusted to control the size of the rule set and the strength of the rules, depending on the requirements of the analysis. Lastly, the right-hand side of the association rule to be Diagnosis=O, meaning that the algorithm will only consider rules where Diagnosis is "altered".

The output of the subtask i of 2080 rules means that the Apriori algorithm generated 2080 rules from the transaction data set. These rules are the association rules between different attributes of the data set, i.e., the relationship between the different items (columns) in the transactions. It is important to note that the number of rules generated is not always an indicator of the quality of the rules. The rules generated by the Apriori algorithm may be too general or too specific, and they may not necessarily capture the underlying patterns in the data. The rules generated in subtask i are generated without any specific support or confidence values.

On the other hand, subtask ii applies the Apriori algorithm with specific support (0.02) and confidence (1), and filters the rules with the right-hand side (rhs) item "Diagnosis=O". It generated 16 association rules, which means that there are 16 combinations of the attributes in the data set that satisfy the minimum support of 0.02 and have confidence of 1 with the outcome "Diagnosis=altered". These 16 rules are the rules that are considered to be the most relevant and interesting, and they provide insights into the relationship between the attributes in the data set and the diagnosis outcome.

| *lhs* | *rhs* | *support* | *confidence* | *coverage* | *lift* | *count* |
|---|---|---|---|---|---|---|
| **A** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **B** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **C** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **D** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **E** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **F** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **G** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **H** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **I** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **J** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **K** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **L** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **M** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **N** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **O** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |
| **P** | {Diagnosis=O} | 0.02 | 1 | 0.02 | 8.333333 | 2 |

From the above16 rules, we got the lhs Left-hand side:
A: Season=1,Frequency.of.Alcohol=0.6,Smoking=[-1,0)
B: Season=1,High.Fever=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
C: Season=1,Surgery=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
D: Season=1,Accident=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
E: Season=1,Disease=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
F: Season=1,Surgery=[0,1],High.Fever=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
G: Season=1,Accident=[0,1],High.Fever=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
H: Season=1,Disease=[0,1],High.Fever=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
I: Season=1,Accident=[0,1],Surgery=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
J: Season=1,Disease=[0,1],Surgery=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
K: Season=1,Disease=[0,1],Accident=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
L: Season=1,Accident=[0,1],Surgery=[0,1],High.Fever=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)
M: Season=1,Disease=[0,1],Surgery=[0,1],High.Fever=[0,1],Frequency.of.Alcohol=0.6,Smoking=[-1,0)

N: Season=1,Disease=[0,1],Accident=[0,1],High.Fever=[0,1],Frequency.of.Alcohol=0.6, Smoking=[-1,0)
O: Season=1,Disease=[0,1],Accident=[0,1],Surgery=[0,1],Frequency.of.Alcohol=0.6, Smoking=[-1,0)
P: Season=1,Disease=[0,1],Accident=[0,1],Surgery=[0,1],High.Fever=[0,1], Frequency.of.Alcohol=0.6,Smoking=[-1,0)

Lastly, subtask iii removes the redundant rules from the set of rules obtained in subtask ii based on the lift of each rule. The script first removes the redundant rules by using the is.redundant() function and then prints the final set of rules, which are the rules that have not been classified as redundant. The final set of rules is then transformed into a dataframe that includes the left-hand side (lhs), right-hand side (rhs), and quality (support and confidence) of each rule. This rule is the rule that is considered to have the highest lift and is not redundant. This means that it is most interesting and informative rule for the dataset based on the lift measure.

It is this transaction:

$$Season = 1, Frequency.of.Alcohol = 0.6, Smoking = [-1, 0) -> Diagnosis = O$$

and it has the following characteristics:

Support = 0.02

Confidence = 1

Coverage = 0.02

Lift = 8.333333

Count = 2