**DATA MINING**

**PRACTICAL FILE**

**A picture containing logo

Description automatically generated**

**Department of Computer Science**

**Aryabhatta College**

**University of Delhi**

***Submitted to: Submitted by:***

**Dr. Sonal Linda *Name:* Sahil Gupta**

***Roll No.*: 21059570042,**

**Semester : VI**

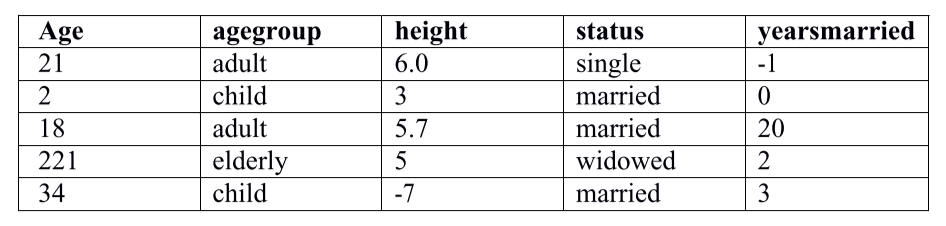
**B.Sc.(H) Computer Science**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S NO.** | **PRACTICAL** | **PAGE NO.** |
| **1.** | **Section 1: Preprocessing**  Q1. Create a file “people.txt” with the following data:   1. Read the data from the file “people.txt”. 2. Create a ruleset E that contain rules to check for the following conditions:   1. The age should be in the range 0-150.  2. The age should be greater than yearsmarried.  3. The status should be married or single or widowed.  4. If age is less than 18 the agegroup should be child, if age is between 18 and 65 the agegroup should be adult, if age is more than 65 the agegroup should be elderly.   1. Check whether ruleset E is violated by the data in the file people.txt. 2. Summarize the results obtained in part (iii) 3. Visualize the results obtained in part (iii) | 4-9 |
| **2.** | Q2. Perform the following preprocessing tasks on the dirty\_iris dataset.   1. Calculate the number and percentage of observations that are complete. 2. Replace all the special values in data with NA. 3. Define these rules in a separate text file and read them. (Use editfile function in R (package editrules). Use similar function in Python). Print the resulting constraint object.  * Species should be one of the following values: setosa, versicolor or virginica. * All measured numerical properties of an iris should be positive. * The petal length of an iris is at least 2 times its petal width. * The sepal length of an iris cannot exceed 30 cm. * The sepals of an iris are longer than its petals.  1. Determine how often each rule is broken (violatedEdits). Also summarize and plot the result. 2. Find outliers in sepal length using boxplot and boxplot.stats | 10-16 |
| **3.** | Q3. Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset. | 17-19 |
| **4.** | **Section 2: Data Mining Techniques**  Run following algorithms on 2 real datasets and use appropriate evaluation measures to compute correctness of obtained patterns:  Q4. Run Apriori algorithm to find frequent itemsets and association rules   1. Use minimum support as 50% and minimum confidence as 75% 2. Use minimum support as 60% and minimum confidence as 60 % | 20-27 |
| **5.** | Q5. Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations:  5.1 a) Training set = 75% Test set = 25%  b) Training set = 66.6% (2/3rd of total), Test set = 33.3%  5.2 Training set is chosen by   1. hold out method 2. Random subsampling 3. Cross-Validation. Compare the accuracy of the classifiers obtained.   5.3 Data is scaled to standard format. | 28 -36 |
| **6.** | Q6. Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms. | 37-60 |
| **7.** | **Section 3: Project**  Q7. Students should be promoted to take up one project on any UCI/kaggle/data.gov.in or a dataset verified by the teacher. Preprocessing steps and at least one data mining technique should be shown on the selected dataset. This will allow the students to have a practical knowledge of how to apply the various skills learnt in the subject for a single problem/project. | 61- |

**PRACTICAL 01**

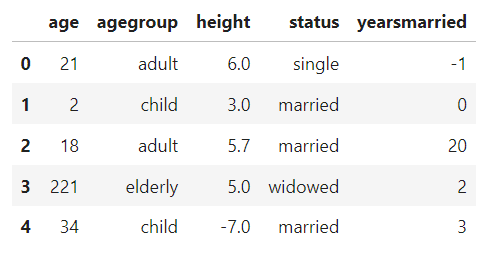
1. **Create a file “people.txt” with the following data:**

****

1. **Read the data from the file “people.txt”.**

*import* numpy *as* np  
*import* pandas *as* pd  
*import* matplotlib.pyplot *as* plt  
  
*# [i] reading data from file*path = 'people.txt'  
df = pd.read\_table(path, sep=',', header=0)  
df  
df.info()

**Output**



A screenshot of a computer code

Description automatically generated

1. **Create a ruleset E that contain rules to check for the following conditions:**
2. **The age should be in the range 0-150.**
3. **The age should be greater than yearsmarried.**
4. **The status should be married or single or widowed.**
5. **If age is less than 18 the agegroup should be child, if age is between 18 and 65 the agegroup should be adult, if age is more than 65 the agegroup should be elderly.**

*# [ii] Ruleset definition  
  
def* E(row):  
 errorlist = []  
  
 *#rule 1  
 if not* (0 <= row['age'] <= 150):  
 errorlist.append('age should be in range 0-150')  
  
 *#rule 2  
 if not*(row['age'] > row['yearsmarried']):  
 errorlist.append('age should be greater than years married')  
  
 *#rule 3  
 if* row['status'] *not in* ['single','married','widowed']:  
 errorlist.append("status must be 'single' or 'married' or 'widowed'")  
  
 *#rule 4  
 #expected\_agegroup = ''  
 if* row['age'] < 18:  
 expected\_agegroup = 'child'  
 *elif* 18 <= row['age'] < 65:  
 expected\_agegroup = 'adult'  
 *else*:  
 expected\_agegroup = 'elderly'  
  
 *if* row['agegroup'] != expected\_agegroup:  
 errorlist.append(f"expected age group:'{expected\_agegroup}', received: '{row['agegroup']}'")  
  
 *return* errorlist

1. **Check whether ruleset E is violated by the data in the file people.txt.**

*# [iii] Applying Rulset to check for rule violations*df['ERRORS'] = df.apply(E, axis=1)  
df

**Output**

**A screenshot of a computer

Description automatically generated**

1. **Summarize the results obtained in part (iii)**

*# [iv] data summarization*vio = df[df['ERRORS'].apply(*lambda* x: len(x) > 0)]  
vio

**Output**

**A screenshot of a computer

Description automatically generated**

**v) Visualize the results obtained in part (iii)**

*# [v] visualizing the summarizations  
  
# Flatten the list of errors and count occurrences*error\_counts = vio['ERRORS'].explode().value\_counts()  
print(error\_counts)  
  
*# Plotting*error\_counts.plot(kind='bar')  
plt.title('Visualization of Rule Violations')  
plt.xlabel('Rules')  
plt.ylabel('Number of Violations')  
plt.xticks(rotation=45)  
plt.tight\_layout()  
plt.show()

**Output**

**A screenshot of a graph

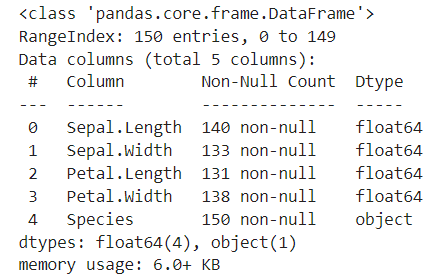
Description automatically generated**

**PRACTICAL 2**

1. **Perform the following preprocessing tasks on the dirty\_iris datasetii.**
2. **Calculate the number and percentage of observations that are complete.**

*import* pandas *as* pd  
*import* numpy *as* np  
*import* matplotlib.pyplot *as* plt  
*import* seaborn *as* sns  
  
df = pd.read\_csv('iris\_dirty.csv')  
df.info()  
  
df['Sepal.Width'].value\_counts()  
  
*# [i] Number and percentage of complete obeservations*df\_completeobs = df.dropna()  
  
total\_obs = len(df)   
notna\_obs = len(df\_completeobs)  
perc\_notna\_obs = notna\_obs/total\_obs\*100 *#BODMAS WORKS*print('Total Observations: ', total\_obs)  
print('Complete Observations: ', notna\_obs)  
print('Percentage of Complete Observations: ', perc\_notna\_obs,' %', sep='')

**Output**

******

A screenshot of a math test

Description automatically generated

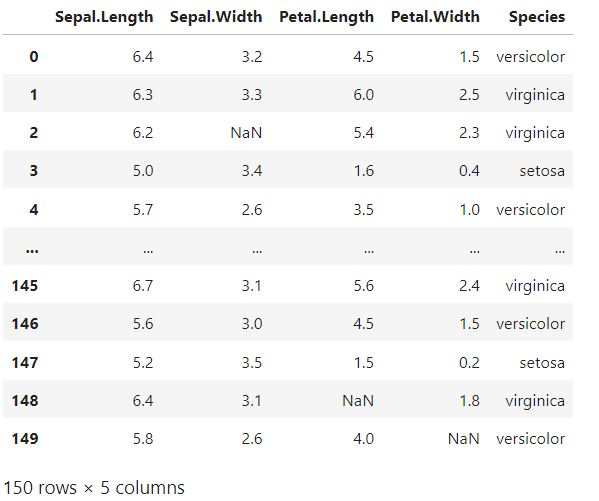
A close up of a number

Description automatically generated

**ii) Replace all the special values in data with NA.**

*# [ii] Replace all special values with NA*cols\_to\_check = ['Sepal.Length','Sepal.Width','Petal.Length','Petal.Width']  
  
df[df[cols\_to\_check] == 'inf'] = pd.NA

**Output**

****

**iii) Define these rules in a separate text file and read them. (Use editfile function in R (package editrules). Use similar function in Python). Print the resulting constraint object.**

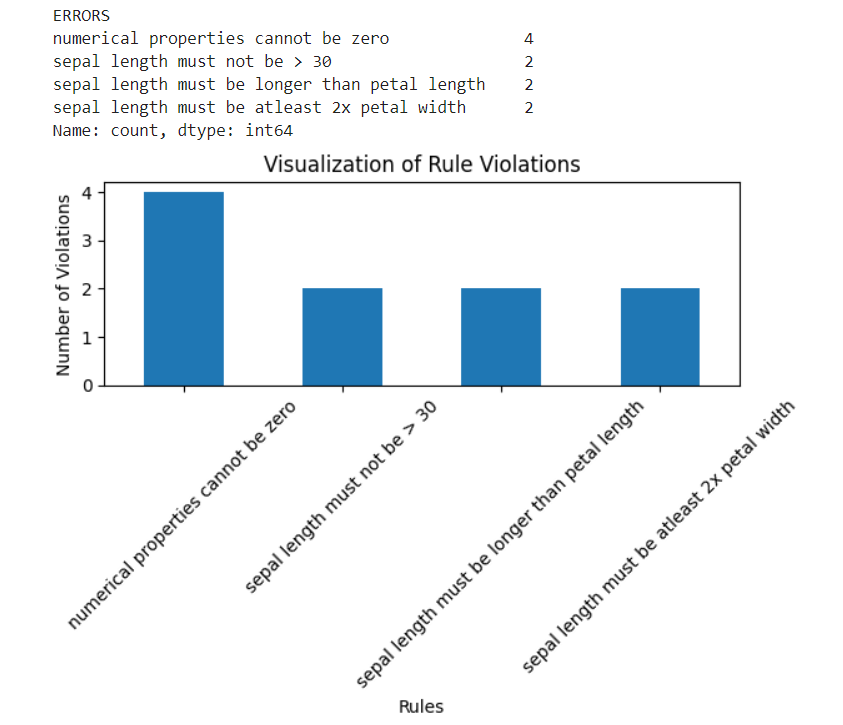
* + - **Species should be one of the following values: setosa, versicolor or virginica.**
    - **All measured numerical properties of an iris should be positive.**
    - **The petal length of an iris is at least 2 times its petal width.**
    - **The sepal length of an iris cannot exceed 30 cm.**
    - **The sepals of an iris are longer than its petals.**

*# [iii] Ruleset definition  
  
def* DirtyIrisRuleset(row):  
 errorlist = []  
  
 *#rule 1  
 if* row['Species'] *not in* ['setosa','versicolor','virginica']:  
 errorlist.append("species must be 'setosa' or 'versicolor' or 'virginica'")  
   
 *#rule 2  
 if* (row['Sepal.Length'] <= 0) *or* (row['Sepal.Width'] <= 0):  
 errorlist.append('numerical properties cannot be zero')  
 *elif* (row['Petal.Length'] <= 0) *or* (row['Petal.Length'] <= 0):  
 errorlist.append('numerical properties cannot be zero')  
 *else*:  
 *pass  
  
 #rule 3  
 if* row['Sepal.Length'] < row['Petal.Width']:  
 errorlist.append('sepal length must be atleast 2x petal width')  
   
 *#rule 4  
 if* row['Sepal.Length'] > 30:  
 errorlist.append("sepal length must not be > 30")  
  
 *#rule 5  
 if* row['Sepal.Length'] <= row['Petal.Length']:  
 errorlist.append("sepal length must be longer than petal length")  
  
 *return* errorlist

**iv) Determine how often each rule is broken (violatedEdits). Also summarize and plot the result.**

*#Importing Ruleset  
from* ruleset\_for\_dmp2 *import* DirtyIrisRuleset *as* E  
  
*# [iv] Applying Rulset and visualizing results*df['ERRORS'] = df.apply(E, axis=1)  
  
*# data summarization*vio = df[df['ERRORS'].apply(*lambda* x: len(x) > 0)]  
df = df.drop(columns='ERRORS')  
  
*# Flatten the list of errors and count occurrences*error\_counts = vio['ERRORS'].explode().value\_counts()  
print(error\_counts)  
  
*# Plotting*error\_counts.plot(kind='bar')  
plt.title('Visualization of Rule Violations')  
plt.xlabel('Rules')  
plt.ylabel('Number of Violations')  
plt.xticks(rotation=45)  
plt.tight\_layout()  
plt.show()

**Output**

****

1. **Find outliers in sepal length using boxplot and boxplot.stats**

*# [v] Outliers in sepal lenth using boxplot.stats  
  
# plotting boxplot*plt.figure()  
sns.boxplot(x=df['Sepal.Length'])  
plt.title('Boxplot of Sepal Length')  
plt.show()  
  
*# indentifying outliers using interquartile ranges*Q1 = df['Sepal.Length'].quantile(0.25)  
Q3 = df['Sepal.Length'].quantile(0.75)  
IQR = Q3 - Q1  
  
lower\_bound = Q1 - 1.5 \* IQR  
upper\_bound = Q3 + 1.5 \* IQR  
  
*# detecting and printing outliers*outliers = df[(df['Sepal.Length'] < lower\_bound) | (df['Sepal.Length'] > upper\_bound)]  
print("Detected Outliers:")  
print(outliers)

**Output**

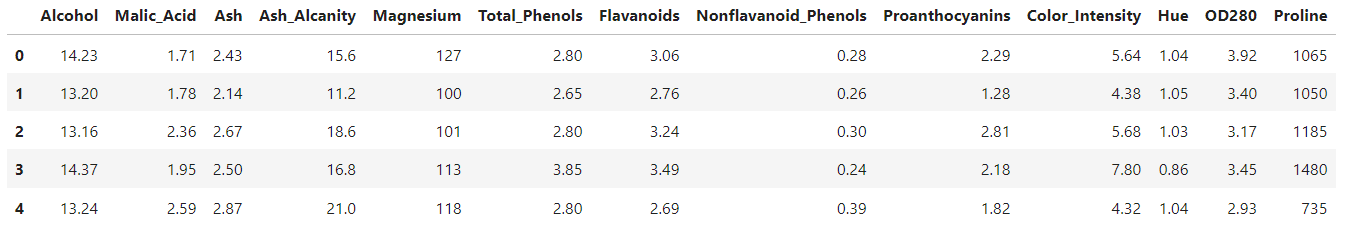
**A graph with numbers and a line

Description automatically generated**

**PRACTICAL 3**

**Load the data from wine dataset. Check whether all attributes are standardized or not (mean is 0 and standard deviation is 1). If not, standardize the attributes. Do the same with Iris dataset.**

*import* pandas *as* pd  
*from* sklearn.preprocessing *import* StandardScaler  
  
*# Load the Wine dataset*wine\_df = pd.read\_csv("Wine dataset.csv")  
wine\_df.head(5)

****

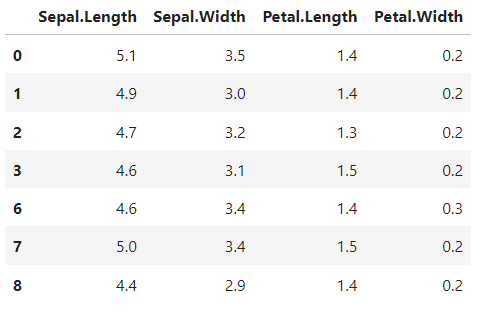
*# Check if attributes are standardized  
if* ((iris\_df.mean(axis=0) == 0).all() *and* (iris\_df.std(axis=0) == 1).all()):  
 print("All attributes in the iris dataset are already standardized")  
*else*:  
 print("All attributes in the iris dataset are not standardized")

****

*# Standardize the attributes*scaler = StandardScaler()  
iris\_df\_standardized = scaler.fit\_transform(iris\_df)  
iris\_df = pd.DataFrame(iris\_df\_standardized, columns=iris\_df.columns)  
print("Now, all attributes are standardized")

****

*# Load the Iris dataset*iris\_df = pd.read\_csv("iris\_dirty.csv")  
iris\_df = iris\_df.drop(columns=['Species']).dropna()  
iris\_df.head(7)



*# Check if attributes are standardized  
if* ((iris\_df.mean(axis=0) == 0).all() *and* (iris\_df.std(axis=0) == 1).all()):  
 print("All attributes in the iris dataset are already standardized")  
*else*:  
 print("All attributes in the iris dataset are not standardized")



*# Standardize the attributes*scaler = StandardScaler()  
iris\_df\_standardized = scaler.fit\_transform(iris\_df)  
iris\_df = pd.DataFrame(iris\_df\_standardized, columns=iris\_df.columns)  
print("Now, all attributes are standardized")



**Section 2: Data Mining Techniques**

**Run following algorithms on 2 real datasets and use appropriate evaluation measures to compute correctness of obtained patterns:**

**Q4. Run Apriori algorithm to find frequent itemsets and association rules**

**1.1 Use minimum support as 50% and minimum confidence as 75%**

**1.2 Use minimum support as 60% and minimum confidence as 60 %**

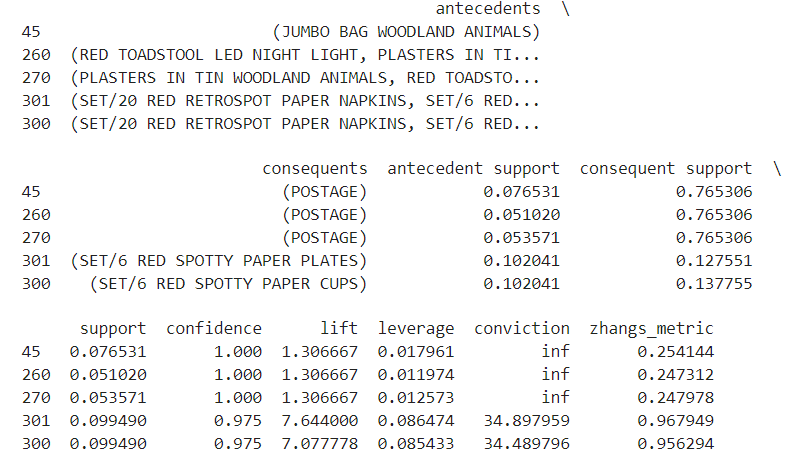
*import* numpy *as* np  
*import* pandas *as* pd  
*from* mlxtend.frequent\_patterns *import* apriori, association\_rules  
  
df = pd.read\_excel('Online Retail.xlsx')  
df

**A screenshot of a computer

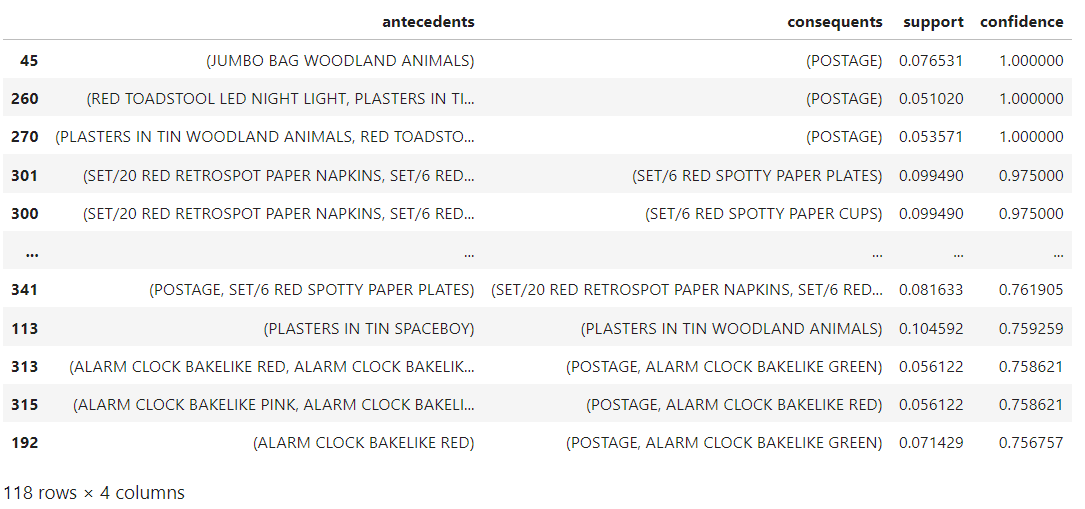
Description automatically generated**

# Load the data for the first dataset (e.g., France)

data1 = df[df.Country=="France"]  
  
*# Load the data (replace with your own dataset)*data = data1.copy() *# Make a copy of the original dataframe  
  
# Clean the data*data['Description'] = data['Description'].str.strip()  
data.dropna(axis=0, subset=['InvoiceNo'], inplace=*True*)  
data['InvoiceNo'] = data['InvoiceNo'].astype('str')  
data = data[~data['InvoiceNo'].str.contains('C')]  
  
*# Split data by region (e.g., France, UK, etc.)*basket = (data  
 .groupby(['InvoiceNo', 'Description'])['Quantity']  
 .sum().unstack().reset\_index().fillna(0)  
 .set\_index('InvoiceNo'))  
  
  
*def* hot\_encode(x):  
 *if* isinstance(x, float):  
 *return* 0 *if* x <= 0 *else* 1  
 *else*:  
 *return* x.map(hot\_encode)  
  
basket\_encoded = basket.apply(hot\_encode)  
basket\_encoded = basket\_encoded.astype(bool)  
  
*# Build the model*frq\_items = apriori(basket\_encoded, min\_support=0.05, use\_colnames=*True*)  
rules = association\_rules(frq\_items, metric="lift", min\_threshold=1)  
rules = rules.sort\_values(['confidence', 'lift'], ascending=[*False*, *False*])  
print(rules.head())

****

assc\_rules = rules.loc[:,["antecedents","consequents","support","confidence"]]  
assc\_rules

****

**Practical 4**

**4.1 Use minimum support as 50% and minimum confidence as 75%**

*#Selecting all the rules which have a minimun supprt of 50% and a minimum confidence of 75%*temp = assc\_rules[assc\_rules.support>0.05]  
rules1 = temp[assc\_rules.confidence>0.75]  
rules1

**A screenshot of a computer

Description automatically generated**

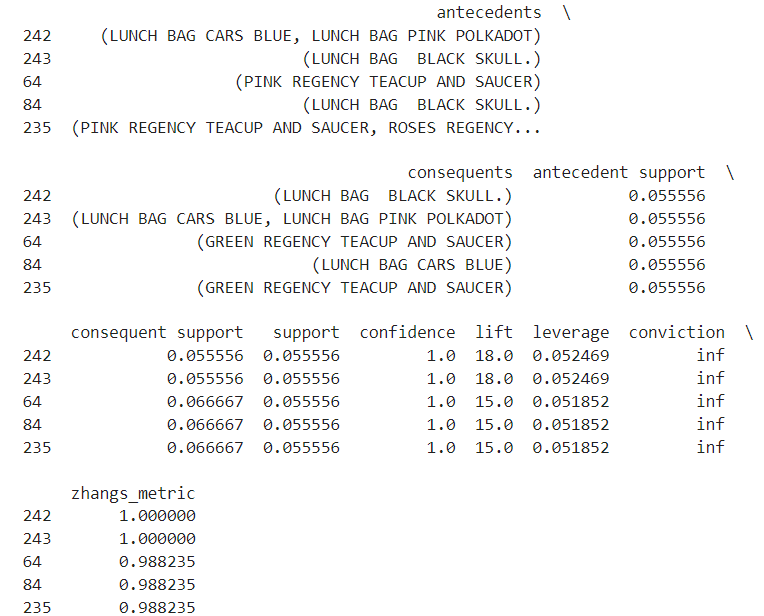
**4.2 Use minimum support as 60% and minimum confidence as 60 %**

*# Selecting all the rules which have a minimum support of 60% and a minimum confidence of 60%*temp = assc\_rules[assc\_rules.support > 0.06]  
rules2 = temp[temp.confidence > 0.6]  
rules2

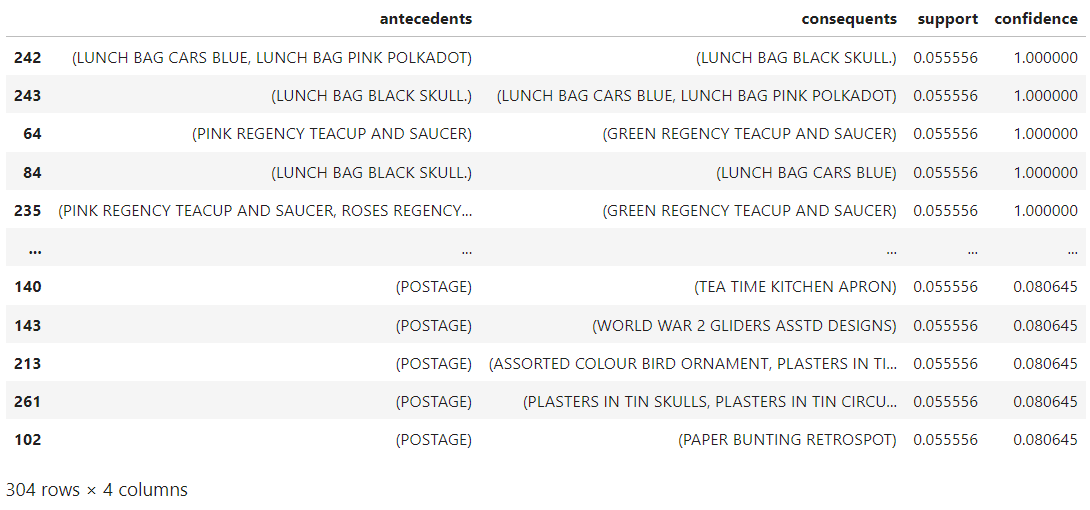
****

Load the data for the second dataset (e.g., Spain)

data2 = df[df.Country=="Spain"]  
  
*# Load the data (replace with your own dataset)*data = data2.copy()  
  
*# Clean the data*data['Description'] = data['Description'].str.strip()  
data.dropna(axis=0, subset=['InvoiceNo'], inplace=*True*)  
data['InvoiceNo'] = data['InvoiceNo'].astype('str')  
data = data[~data['InvoiceNo'].str.contains('C')]  
  
*# Split data by region (e.g., France, UK, etc.)*basket = (data  
 .groupby(['InvoiceNo', 'Description'])['Quantity']  
 .sum().unstack().reset\_index().fillna(0)  
 .set\_index('InvoiceNo'))  
  
*def* hot\_encode(x):  
 *return* x.apply(*lambda* val: 0 *if* val <= 0 *else* 1)  
  
  
basket\_encoded = basket.apply(hot\_encode, axis=1)  
  
*# Convert DataFrame to boolean type*basket\_encoded = basket\_encoded.astype(bool)  
  
*# Build the model*frq\_items = apriori(basket\_encoded, min\_support=0.05, use\_colnames=*True*)  
rules = association\_rules(frq\_items, metric="lift", min\_threshold=1)  
rules = rules.sort\_values(['confidence', 'lift'], ascending=[*False*, *False*])  
print(rules.head())

****

assc\_rules = rules.loc[:,["antecedents","consequents","support","confidence"]]  
assc\_rules

****

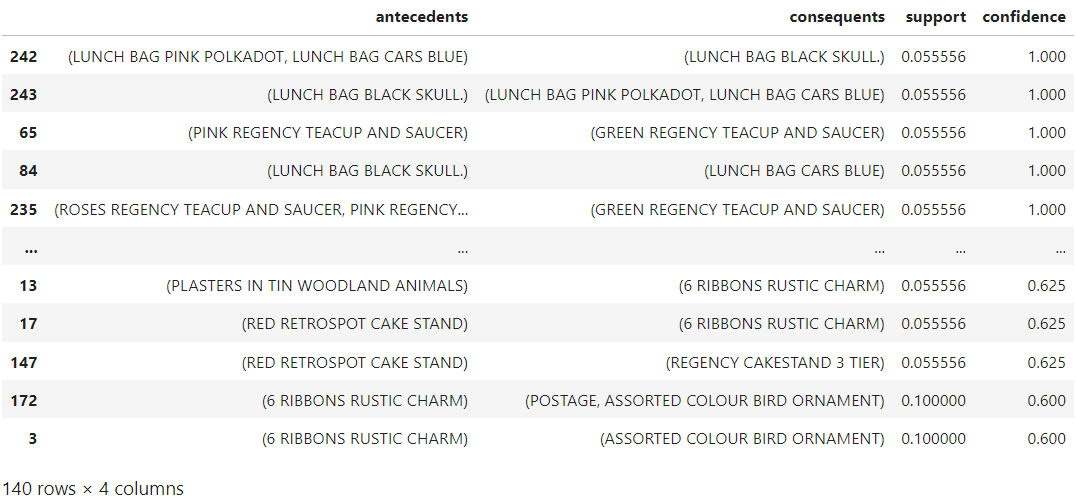
**4.1 Use minimum support as 50% and minimum confidence as 75%**

*#Selecting all the rules which have a minimun supprt of 50% and a minimum confidence of 75%*temp = assc\_rules[assc\_rules.support>0.05]  
rules2 = temp[assc\_rules.confidence>0.75]  
rules2

****

**4.2 Use minimum support as 60% and minimum confidence as 60 %**

temp = assc\_rules[assc\_rules.support>0.05]  
rules4 = temp[assc\_rules.confidence>0.60]  
rules4

****

**PRACTICAL 5**

**Q5. Use Naive bayes, K-nearest, and Decision tree classification algorithms and build classifiers. Divide the data set into training and test set. Compare the accuracy of the different classifiers under the following situations:**

**5.1 a) Training set = 75% Test set = 25%**

**b) Training set = 66.6% (2/3rd of total), Test set = 33.3%**

**5.2 Training set is chosen by**

**i) hold out method ii) Random subsampling iii) Cross-Validation. Compare the accuracy of the classifiers obtained.**

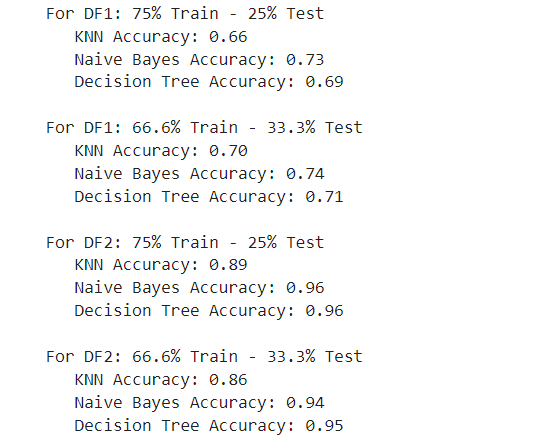
**5.3 Data is scaled to standard format.**

*import* pandas *as* pd  
*import* numpy *as* np  
  
*from* sklearn.preprocessing *import* StandardScaler  
  
*from* sklearn.neighbors *import* KNeighborsClassifier  
*from* sklearn.tree *import* DecisionTreeClassifier  
*from* sklearn.naive\_bayes *import* GaussianNB  
  
*from* sklearn.metrics *import* accuracy\_score  
*from* sklearn.model\_selection *import* KFold, cross\_val\_score,cross\_validate  
*from* sklearn.model\_selection *import* train\_test\_split  
*from* sklearn.model\_selection *import* GridSearchCV  
*from* sklearn.feature\_selection *import* SelectFromModel  
  
df1 = pd.read\_csv('diabetes.csv')  
df2 = pd.read\_csv('Thyroid\_Diff.csv')  
  
df1.info()  
  
df2.info()  
  
list\_for\_onehot = ['Thyroid Function','Physical Examination','Adenopathy','Pathology',  
 'Pathology','Risk','T','N','Stage','Response']  
  
*# binary categorical --> to --> binary numerical*df2['Gender'] = (df2['Gender']=='M').astype(int)  
df2['Smoking'] = (df2['Smoking']=='Yes').astype(int)  
df2['Hx Smoking'] = (df2['Hx Smoking']=='Yes').astype(int)  
df2['Hx Radiothreapy'] = (df2['Hx Radiothreapy']=='Yes').astype(int)  
df2['Focality'] = (df2['Focality']=='Uni-Focal').astype(int)  
df2['M'] = (df2['M']=='M1').astype(int)  
df2['Recurred'] = (df2['Recurred']=='Yes').astype(int)  
  
df2 = pd.get\_dummies(df2, columns = list\_for\_onehot)  
df2.info()  
  
tf\_map = {*False*:0, *True*:1}  
  
cols\_to\_encode = [x *for* x *in* range(8,54)]  
  
*for* col\_idx *in* cols\_to\_encode:  
 df2.iloc[:, col\_idx] = df2.iloc[:, col\_idx].map(tf\_map)  
  
df2.info()  
  
X1 = df1.loc[:,df1.columns!='Outcome']  
y1 = df1.loc[:,'Outcome']  
  
X2 = df2.loc[:,df2.columns!='Recurred']  
y2 = df2.loc[:,'Recurred']  
  
*# a) 75%-25%*X1\_train\_A, X1\_test\_A, y1\_train\_A, y1\_test\_A = train\_test\_split(X1, y1, test\_size=0.25, random\_state=42)  
X2\_train\_A, X2\_test\_A, y2\_train\_A, y2\_test\_A = train\_test\_split(X2, y2, test\_size=0.25, random\_state=42)  
  
*# b) 66.6%-33.3%*X1\_train\_B, X1\_test\_B, y1\_train\_B, y1\_test\_B = train\_test\_split(X1, y1, test\_size=0.33, random\_state=42)  
X2\_train\_B, X2\_test\_B, y2\_train\_B, y2\_test\_B = train\_test\_split(X2, y2, test\_size=0.33, random\_state=42)

5.1 Evaluation on Train-Test Split as 75-25 and 66.6-33.3

classifiers = {  
 "KNN": KNeighborsClassifier(),  
 "Naive Bayes": GaussianNB(),  
 "Decision Tree": DecisionTreeClassifier()  
}  
  
print('For DF1: 75% Train - 25% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X1\_train\_A, y1\_train\_A)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X1\_test\_A)  
 accuracy = accuracy\_score(y1\_test\_A, y\_pred)  
 print(' ', f"{name} Accuracy: {accuracy:.2f}")  
  
print('\nFor DF1: 66.6% Train - 33.3% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X1\_train\_B, y1\_train\_B)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X1\_test\_B)  
 accuracy = accuracy\_score(y1\_test\_B, y\_pred)  
 print(' ', f"{name} Accuracy: {accuracy:.2f}")  
  
print('\nFor DF2: 75% Train - 25% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X2\_train\_A, y2\_train\_A)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X2\_test\_A)  
 accuracy = accuracy\_score(y2\_test\_A, y\_pred)  
 print(' ', f"{name} Accuracy: {accuracy:.2f}")  
  
print('\nFor DF2: 66.6% Train - 33.3% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X2\_train\_B, y2\_train\_B)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X2\_test\_B)  
 accuracy = accuracy\_score(y2\_test\_B, y\_pred)  
 print(' ', f"{name} Accuracy: {accuracy:.2f}")

**Output**

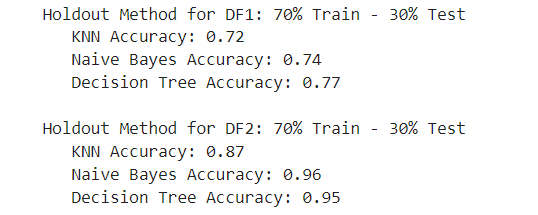


5.2 Evaluation using Holdout, Random Subsampling and 5-Fold CV

*# a) holdout (70%-30%)*X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(X1, y1, test\_size=0.3, stratify=y1, random\_state=42)  
X2\_train, X2\_test, y2\_train, y2\_test = train\_test\_split(X2, y2, test\_size=0.3, stratify=y2, random\_state=42)  
  
print('Holdout Method for DF1: 70% Train - 30% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X1\_train, y1\_train)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X1\_test)  
 accuracy = accuracy\_score(y1\_test, y\_pred)  
 print(' ',f"{name} Accuracy: {accuracy:.2f}")  
  
print('\nHoldout Method for DF2: 70% Train - 30% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X2\_train, y2\_train)

*# Evaluate the classifier* y\_pred = clf.predict(X2\_test)  
 accuracy = accuracy\_score(y2\_test, y\_pred)  
 print(' ',f"{name} Accuracy: {accuracy:.2f}")

**Output**



*# b) random subsample*X1\_train\_1, X1\_test\_1, y1\_train\_1, y1\_test\_1 = train\_test\_split(X1, y1, test\_size=0.2, random\_state=42) *# 80-20*X1\_train\_2, X1\_test\_2, y1\_train\_2, y1\_test\_2 = train\_test\_split(X1, y1, test\_size=0.33, random\_state=42) *# 66.6-33.3*X1\_train\_3, X1\_test\_3, y1\_train\_3, y1\_test\_3 = train\_test\_split(X1, y1, test\_size=0.3, random\_state=42) *# 70-30*X2\_train\_1, X2\_test\_1, y2\_train\_1, y2\_test\_1 = train\_test\_split(X2, y2, test\_size=0.2, random\_state=42) *# 80-20*X2\_train\_2, X2\_test\_2, y2\_train\_2, y2\_test\_2 = train\_test\_split(X2, y2, test\_size=0.33, random\_state=42) *# 66.6-33.3*X2\_train\_3, X2\_test\_3, y2\_train\_3, y2\_test\_3 = train\_test\_split(X2, y2, test\_size=0.3, random\_state=42) *# 70-30*print('Random Subsample for DF1:')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X1\_train\_1, y1\_train\_1)  
 y\_pred\_1 = clf.predict(X1\_test\_1)  
 acc\_1 = accuracy\_score(y1\_test\_1, y\_pred\_1)  
  
 clf.fit(X1\_train\_2, y1\_train\_2)  
 y\_pred\_2 = clf.predict(X1\_test\_2)  
 acc\_2 = accuracy\_score(y1\_test\_2, y\_pred\_2)  
  
 clf.fit(X1\_train\_3, y1\_train\_3)  
 y\_pred\_3 = clf.predict(X1\_test\_3)  
 acc\_3 = accuracy\_score(y1\_test\_3, y\_pred\_3)  
  
 accuracy = (acc\_1 + acc\_2 + acc\_3) / 3  
  
 print(' ', f"{name} Accuracy: {accuracy:.2f}")  
  
print('\nRandom Subsample for DF2:')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X2\_train\_1, y2\_train\_1)  
 y\_pred\_1 = clf.predict(X2\_test\_1)  
 acc\_1 = accuracy\_score(y2\_test\_1, y\_pred\_1)  
  
 clf.fit(X2\_train\_2, y2\_train\_2)  
 y\_pred\_2 = clf.predict(X2\_test\_2)  
 acc\_2 = accuracy\_score(y2\_test\_2, y\_pred\_2)  
  
 clf.fit(X2\_train\_3, y2\_train\_3)  
 y\_pred\_3 = clf.predict(X2\_test\_3)  
 acc\_3 = accuracy\_score(y2\_test\_3, y\_pred\_3)  
  
 accuracy = (acc\_1 + acc\_2 + acc\_3) / 3  
  
 print(' ', f"{name} Accuracy: {accuracy:.2f}")

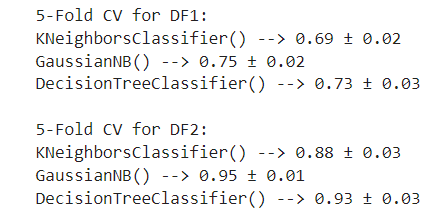
**Output**

**A screenshot of a computer code

Description automatically generated**

*# c) 5-Fold Cross-Validation*k = 5  
kf = KFold(n\_splits=k, shuffle=*True*, random\_state=42)  
  
scoring = ['accuracy']  
  
models = [KNeighborsClassifier(), GaussianNB(), DecisionTreeClassifier()]  
  
print('5-Fold CV for DF1:')  
*for* model *in* models:  
 result = list()  
 scores = cross\_validate(model, X1, y1, cv=kf, scoring=scoring)  
  
 *for* value *in* scores:  
 v = str(value)  
 mean\_score = scores[v].mean()  
 std\_score = scores[v].std()  
 *if* (v == "fit\_time" *or* v == "score\_time"):  
 *pass  
 else*:  
 print(f"{model} --> {mean\_score:.2f} ± {std\_score:.2f}")  
  
print('\n5-Fold CV for DF2:')  
*for* model *in* models:  
 result = list()  
 scores = cross\_validate(model, X2, y2, cv=kf, scoring=scoring)  
  
 *for* value *in* scores:  
 v = str(value)  
 mean\_score = scores[v].mean()  
 std\_score = scores[v].std()  
 *if* (v == "fit\_time" *or* v == "score\_time"):  
 *pass  
 else*:  
 print(f"{model} --> {mean\_score:.2f} ± {std\_score:.2f}")

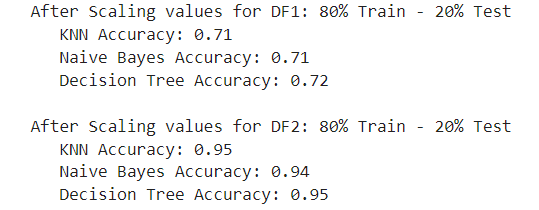
**Output**

****

5.3 Results After Scaling the Values

scaler = StandardScaler()  
  
X1 = scaler.fit\_transform(X1)  
X2 = scaler.fit\_transform(X2)  
  
X1\_train, X1\_test, y1\_train, y1\_test = train\_test\_split(X1, y1, test\_size=0.2, stratify=y1, random\_state=42)  
X2\_train, X2\_test, y2\_train, y2\_test = train\_test\_split(X2, y2, test\_size=0.2, stratify=y2, random\_state=42)  
  
print('After Scaling values for DF1: 80% Train - 20% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X1\_train, y1\_train)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X1\_test)  
 accuracy = accuracy\_score(y1\_test, y\_pred)  
 print(' ',f"{name} Accuracy: {accuracy:.2f}")  
  
print('\nAfter Scaling values for DF2: 80% Train - 20% Test')  
*for* name, clf *in* classifiers.items():  
 *# Train the classifier* clf.fit(X2\_train, y2\_train)  
  
 *# Evaluate the classifier* y\_pred = clf.predict(X2\_test)  
 accuracy = accuracy\_score(y2\_test, y\_pred)  
 print(' ',f"{name} Accuracy: {accuracy:.2f}")

**Output**



**PRACTICAL 6**

**Use Simple Kmeans, DBScan, Hierachical clustering algorithms for clustering. Compare the performance of clusters by changing the parameters involved in the algorithms.**

*import* pandas *as* pd  
*import* numpy *as* np  
*import* matplotlib.pyplot *as* plt  
*import* seaborn *as* sns  
  
*from* sklearn.cluster *import* KMeans  
*from* sklearn.cluster *import* DBSCAN  
*from* sklearn.cluster *import* AgglomerativeClustering  
*from* scipy.cluster.hierarchy *import* dendrogram, linkage, fcluster  
  
df1 = pd.read\_csv(r'Country-data.csv')  
df2 = pd.read\_csv(r'wine dataset.csv')

PREPROCESSING DATA FOR DF1

df1.info()

**A screenshot of a computer code

Description automatically generated**

df1 = df1.drop(columns='country')  
df1.info()

**A screenshot of a computer code

Description automatically generated**

PREPROCESSING DF2

df2.info()

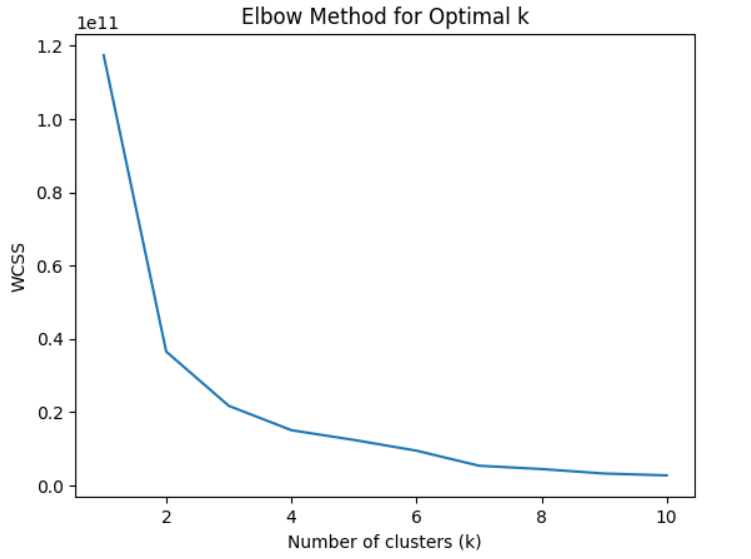
**A screenshot of a computer

Description automatically generated**

K MEANS CLUSTERING DF1

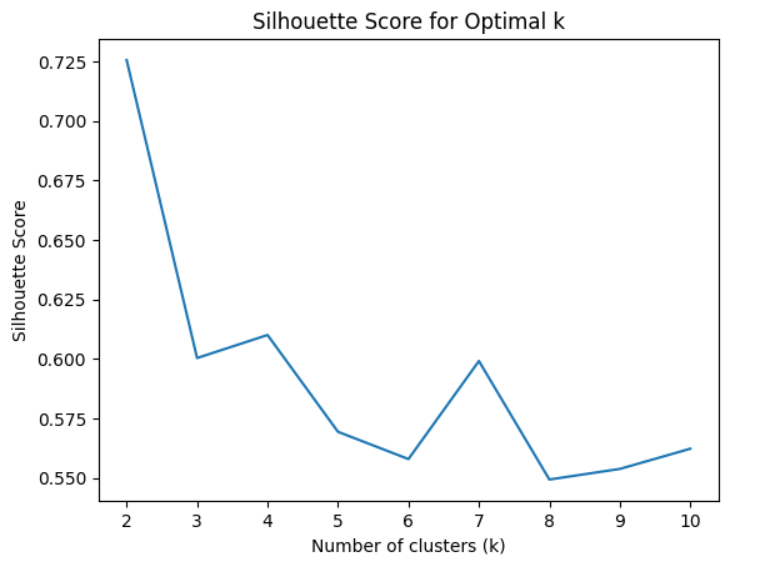
*# DETERMINING OPTIMAL K VALUE FOR DF1 using Elbow Method  
  
# Calculate WCSS for different k values*sse = {}  
*for* k *in* range(1, 11): *# Adjust the range as needed* kmeans = KMeans(n\_clusters=k, max\_iter=1000).fit(df1)  
 sse[k] = kmeans.inertia\_ *# Inertia: Sum of distances of samples to their closest cluster center  
  
# Plot the WCSS values*plt.figure()  
plt.plot(list(sse.keys()), list(sse.values()))  
plt.xlabel("Number of clusters (k)")  
plt.ylabel("WCSS")  
plt.title("Elbow Method for Optimal k")  
plt.show()

**Output**



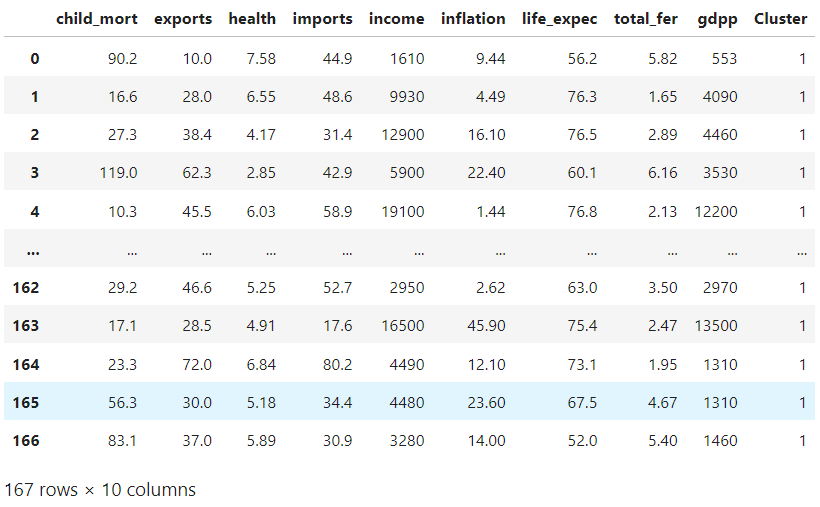
*from* sklearn.metrics *import* silhouette\_score  
  
*# Calculate Silhouette scores for different k values*silhouette\_scores = {}  
*for* k *in* range(2, 11): *# Adjust the range as needed* kmeans = KMeans(n\_clusters=k, max\_iter=1000).fit(df1)  
 labels = kmeans.labels\_  
 silhouette\_scores[k] = silhouette\_score(df1, labels)  
  
*# Plot the Silhouette scores*plt.figure()  
plt.plot(list(silhouette\_scores.keys()), list(silhouette\_scores.values()))  
plt.xlabel("Number of clusters (k)")  
plt.ylabel("Silhouette Score")  
plt.title("Silhouette Score for Optimal k")  
plt.show()

**Output**



*# Applying K-means clustering*kmeans = KMeans(n\_clusters=2, random\_state=0).fit(df1)  
  
*# Add cluster labels to the original dataframe*df1['Cluster'] = kmeans.labels\_  
  
*# Display the dataframe with cluster labels*df1

**Output**



df1['Cluster'].value\_counts()

A black text on a white background

Description automatically generated

*#Visualizing Clusters*cols = df2.columns  
num\_features = len(cols) - 1 *# Number of features (excluding the 'Cluster' column)*num\_rows = (num\_features - 1) // 4 + 1 *# Number of rows needed*num\_plots\_last\_row = (num\_features - 1) % 4 *# Number of plots in the last row  
  
# Create a figure and subplots grid*fig, axs = plt.subplots(num\_rows, 4, figsize=(20, 5 \* num\_rows))  
  
*# Flatten the subplots grid for easy indexing*axs = axs.flatten()  
  
*# Iterate through combinations of features*subplot\_idx = 0  
*for* i *in* range(num\_features):  
 *for* j *in* range(i + 1, num\_features):  
 *if* subplot\_idx >= num\_rows \* 4:  
 *break* x\_label = cols[i]  
 y\_label = cols[j]  
  
 *# Plot the scatterplot in the current subplot* sns.scatterplot(x=x\_label, y=y\_label, data=df2, hue='Cluster', ax=axs[subplot\_idx])  
 axs[subplot\_idx].set\_title(f'{x\_label} vs {y\_label}')  
  
 subplot\_idx += 1  
  
*# Hide empty subplots in the last row  
if* num\_plots\_last\_row > 0:  
 *for* i *in* range(num\_plots\_last\_row, 4):  
 axs[-i].axis('off')  
  
*# Adjust layout*plt.tight\_layout()  
  
*# Show the plot*plt.show()

**Output**

A group of graphs with numbers and symbols

Description automatically generated with medium confidence

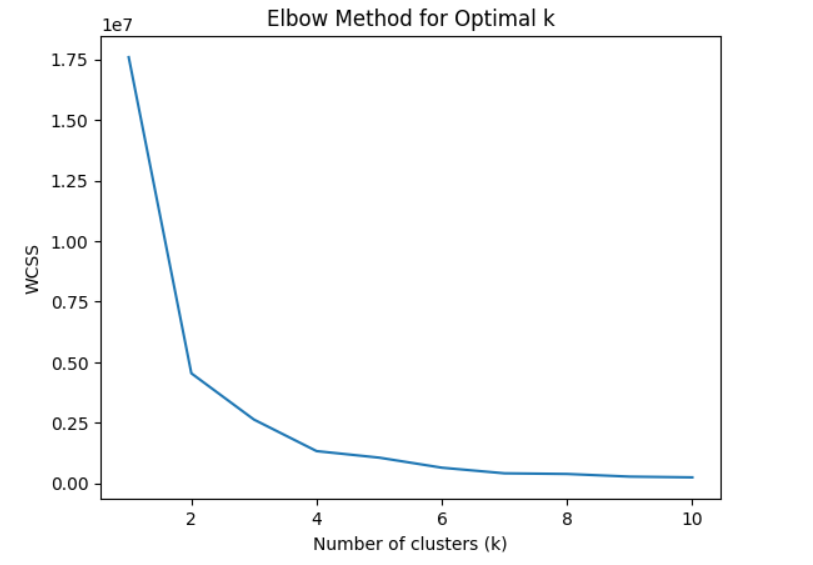
A group of graphs with numbers and dots

Description automatically generated with medium confidence

K MEANS CLUSTERING FOR DF2

*# Calculate WCSS for different k values*sse = {}  
*for* k *in* range(1, 11): *# Adjust the range as needed* kmeans = KMeans(n\_clusters=k, max\_iter=1000).fit(df2)  
 sse[k] = kmeans.inertia\_ *# Inertia: Sum of distances of samples to their closest cluster center  
  
# Plot the WCSS values*plt.figure()  
plt.plot(list(sse.keys()), list(sse.values()))  
plt.xlabel("Number of clusters (k)")  
plt.ylabel("WCSS")  
plt.title("Elbow Method for Optimal k")  
plt.show()

**Output**



*from* sklearn.metrics *import* silhouette\_score  
  
*# Calculate Silhouette scores for different k values*silhouette\_scores = {}  
*for* k *in* range(2, 11): *# Adjust the range as needed* kmeans = KMeans(n\_clusters=k, max\_iter=1000).fit(df2)  
 labels = kmeans.labels\_  
 silhouette\_scores[k] = silhouette\_score(df2, labels)  
  
*# Plot the Silhouette scores*plt.figure()  
plt.plot(list(silhouette\_scores.keys()), list(silhouette\_scores.values()))  
plt.xlabel("Number of clusters (k)")  
plt.ylabel("Silhouette Score")  
plt.title("Silhouette Score for Optimal k")  
plt.show()

**Output**

A graph with a line

Description automatically generated

*# Applying K-means clustering*kmeans = KMeans(n\_clusters=2, random\_state=0).fit(df2)  
  
*# Add cluster labels to the original dataframe*df2['Cluster'] = kmeans.labels\_  
  
*# Display the dataframe with cluster labels*df2

A table with numbers and symbols

Description automatically generated

df2['Cluster'].value\_counts()

A close up of a text

Description automatically generated

*# Visualizing Clusters*cols = df2.columns  
num\_features = len(cols) - 1 *# Number of features (excluding the 'Cluster' column)*num\_rows = (num\_features - 1) // 4 + 1 *# Number of rows needed*num\_plots\_last\_row = (num\_features - 1) % 4 *# Number of plots in the last row  
  
# Create a figure and subplots grid*fig, axs = plt.subplots(num\_rows, 4, figsize=(20, 5 \* num\_rows))  
  
*# Flatten the subplots grid for easy indexing*axs = axs.flatten()  
  
*# Iterate through combinations of features*subplot\_idx = 0  
*for* i *in* range(num\_features):  
 *for* j *in* range(i + 1, num\_features):  
 *if* subplot\_idx >= num\_rows \* 4:  
 *break* x\_label = cols[i]  
 y\_label = cols[j]  
  
 *# Plot the scatterplot in the current subplot* sns.scatterplot(x=x\_label, y=y\_label, data=df2, hue='Cluster', ax=axs[subplot\_idx])  
 axs[subplot\_idx].set\_title(f'{x\_label} vs {y\_label}')  
  
 subplot\_idx += 1  
  
*# Hide empty subplots in the last row  
if* num\_plots\_last\_row > 0:  
 *for* i *in* range(num\_plots\_last\_row, 4):  
 axs[-i].axis('off')  
  
*# Adjust layout*plt.tight\_layout()  
  
*# Show the plot*plt.show()

**Output**

A group of graphs with numbers and dots

Description automatically generated with medium confidence

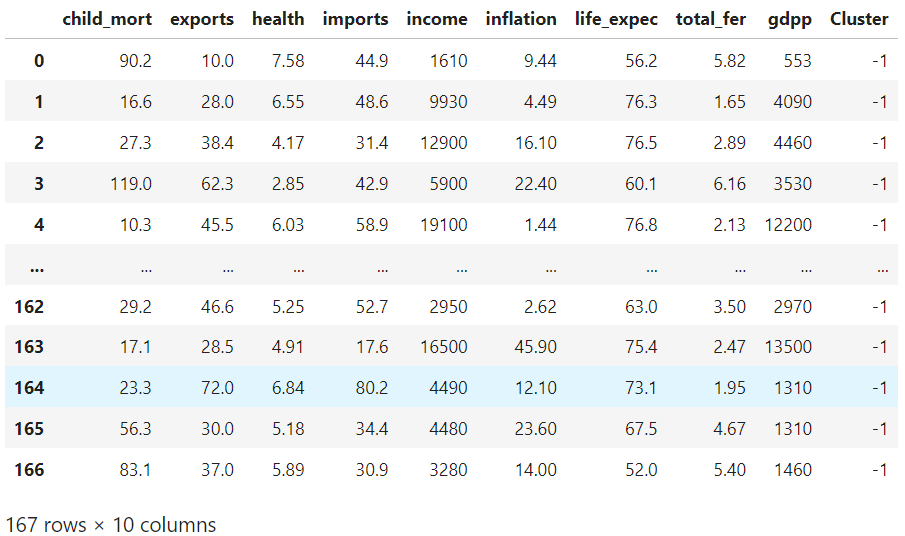
A group of graphs with numbers

Description automatically generated with medium confidence

DBSCAN for DF1

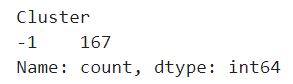
df1 = df1.drop(columns='Cluster')  
  
*# Apply DBSCAN clustering*clustering = DBSCAN(eps=15, min\_samples=10).fit(df1)  
  
*# Add the cluster labels to the original dataframe*df1['Cluster'] = clustering.labels\_  
  
*# Visualize the distribution of clusters*df1

**Output**



df1['Cluster'].value\_counts()

**Output**

****

*# Visualizing Clusters*cols = df1.columns  
num\_features = len(cols) - 1 *# Number of features (excluding the 'Cluster' column)*num\_rows = (num\_features - 1) // 4 + 1 *# Number of rows needed*num\_plots\_last\_row = (num\_features - 1) % 4 *# Number of plots in the last row  
  
# Create a figure and subplots grid*fig, axs = plt.subplots(num\_rows, 4, figsize=(20, 5 \* num\_rows))  
  
*# Flatten the subplots grid for easy indexing*axs = axs.flatten()  
  
*# Iterate through combinations of features*subplot\_idx = 0  
*for* i *in* range(num\_features):  
 *for* j *in* range(i + 1, num\_features):  
 *if* subplot\_idx >= num\_rows \* 4:  
 *break* x\_label = cols[i]  
 y\_label = cols[j]  
  
 *# Plot the scatterplot in the current subplot* sns.scatterplot(x=x\_label, y=y\_label, data=df1, hue='Cluster', ax=axs[subplot\_idx])  
 axs[subplot\_idx].set\_title(f'{x\_label} vs {y\_label}')  
  
 subplot\_idx += 1  
  
*# Hide empty subplots in the last row  
if* num\_plots\_last\_row > 0:  
 *for* i *in* range(num\_plots\_last\_row, 4):  
 axs[-i].axis('off')  
  
*# Adjust layout*plt.tight\_layout()  
  
*# Show the plot*plt.show()

**Output**

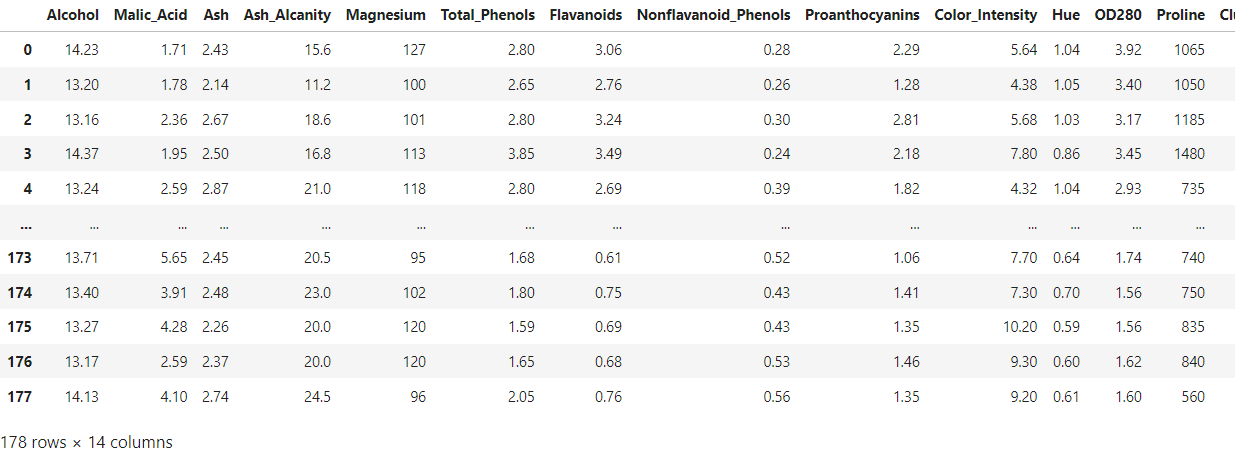
**A screenshot of a graph

Description automatically generated**

DBSCAN FOR DF2

df2 = df2.drop(columns='Cluster')  
  
*# Apply DBSCAN clustering*clustering = DBSCAN(eps=5, min\_samples=2).fit(df2)  
  
*# Add the cluster labels to the original dataframe*df2['Cluster'] = clustering.labels\_  
  
*# Visualize the distribution of clusters*df2

**Output**



df2['Cluster'].value\_counts()

**Output**

A screenshot of a math test

Description automatically generated

cols = df2.columns  
num\_features = len(cols) - 1 *# Number of features (excluding the 'Cluster' column)*num\_rows = (num\_features - 1) // 4 + 1 *# Number of rows needed*num\_plots\_last\_row = (num\_features - 1) % 4 *# Number of plots in the last row  
  
# Create a figure and subplots grid*fig, axs = plt.subplots(num\_rows, 4, figsize=(20, 5 \* num\_rows))  
  
*# Flatten the subplots grid for easy indexing*axs = axs.flatten()  
  
*# Iterate through combinations of features*subplot\_idx = 0  
*for* i *in* range(num\_features):  
 *for* j *in* range(i + 1, num\_features):  
 *if* subplot\_idx >= num\_rows \* 4:  
 *break* x\_label = cols[i]  
 y\_label = cols[j]  
  
 *# Plot the scatterplot in the current subplot* sns.scatterplot(x=x\_label, y=y\_label, data=df2, hue='Cluster', ax=axs[subplot\_idx])  
 axs[subplot\_idx].set\_title(f'{x\_label} vs {y\_label}')  
  
 subplot\_idx += 1  
  
*# Hide empty subplots in the last row  
if* num\_plots\_last\_row > 0:  
 *for* i *in* range(num\_plots\_last\_row, 4):  
 axs[-i].axis('off')  
  
*# Adjust layout*plt.tight\_layout()  
  
*# Show the plot*plt.show()

**Output**

A group of graphs with dots

Description automatically generated with medium confidence

A screenshot of a graph

Description automatically generated

HIERARCHIAL CLUSTERING for DF1

df1 = df1.drop(columns='Cluster')  
  
*# Perform hierarchical clustering on all features*linked = linkage(df1, method='ward', metric='euclidean')  
  
*# Create a DataFrame from the linked matrix*df\_linked = pd.DataFrame(linked, columns=['c1', 'c2', 'distance', 'size'])  
df\_linked[['c1', 'c2', 'size']] = df\_linked[['c1', 'c2', 'size']].astype('int')  
  
*# Visualize the dendrogram*plt.figure(figsize=(10, 7))  
dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=*True*)  
plt.title('Hierarchical Clustering Dendrogram')  
plt.xlabel('Features')  
plt.ylabel('Distance')  
plt.show()  
  
*# Optionally, you can cut the dendrogram to create clusters*num\_clusters = 2 *# Adjust based on your analysis of the dendrogram*clusters = fcluster(linked, num\_clusters, criterion='maxclust')  
df1['Cluster'] = clusters

**Output**

A diagram of a clustering structure

Description automatically generated

df1['Cluster'].value\_counts()

A black text on a white background

Description automatically generated

*# Define the number of plots you want per row*plots\_per\_row = 4  
  
*# Define the number of features (excluding the ‘Cluster’ column)*num\_features = len(df1.columns) – 1  
  
*# Calculate the number of rows needed*num\_rows = (num\_features – 1) // plots\_per\_row + 1  
  
*# Calculate the number of plots in the last row*num\_plots\_last\_row = (num\_features – 1) % plots\_per\_row  
  
*# Create a figure and subplots grid*fig, axs = plt.subplots(num\_rows, plots\_per\_row, figsize=(20, 5 \* num\_rows))  
  
*# Flatten the subplots grid for easy indexing*axs = axs.flatten()  
  
*# Iterate through combinations of features*subplot\_idx = 0  
*for* I *in* range(num\_features):  
 *for* j *in* range(I + 1, num\_features):  
 *if* subplot\_idx >= num\_rows \* plots\_per\_row:  
 *break* x\_label = df1.columns[i]  
 y\_label = df1.columns[j]  
  
 *# Plot the scatterplot in the current subplot* sns.scatterplot(x=x\_label, y=y\_label, data=df1, hue=’Cluster’, ax=axs[subplot\_idx])  
 axs[subplot\_idx].set\_title(f’{x\_label} vs {y\_label}’)  
  
 subplot\_idx += 1  
  
*# Hide empty subplots in the last row  
if* num\_plots\_last\_row > 0:  
 *for* I *in* range(num\_plots\_last\_row, plots\_per\_row):  
 axs[-i].axis(‘off’)  
  
*# Adjust layout*plt.tight\_layout()  
  
*# Show the plot*plt.show()

**Output**

A collage of graphs

Description automatically generated

HIERARCHIAL CLUSTERING for DF2

df2 = df2.drop(columns='Cluster')  
  
*# Perform hierarchical clustering on all features*linked = linkage(df2, method='ward', metric='euclidean')  
  
*# Create a DataFrame from the linked matrix*df\_linked = pd.DataFrame(linked, columns=['c1', 'c2', 'distance', 'size'])  
df\_linked[['c1', 'c2', 'size']] = df\_linked[['c1', 'c2', 'size']].astype('int')  
  
*# Visualize the dendrogram*plt.figure(figsize=(10, 7))  
dendrogram(linked, orientation='top', distance\_sort='descending', show\_leaf\_counts=*True*)  
plt.title('Hierarchical Clustering Dendrogram')  
plt.xlabel('Features')  
plt.ylabel('Distance')  
plt.show()  
  
*# Optionally, you can cut the dendrogram to create clusters*num\_clusters = 2 *# Adjust based on your analysis of the dendrogram*clusters = fcluster(linked, num\_clusters, criterion='maxclust')  
df2['Cluster'] = clusters

**Output**

A diagram of a clustering diagram

Description automatically generated

df2['Cluster'].value\_counts()

**Output**

A black text on a white background

Description automatically generated

cols = df2.columns  
  
*# Define the number of plots you want per row*plots\_per\_row = 4  
  
*# Calculate the number of features (excluding the 'Cluster' column)*num\_features = len(cols) - 1  
  
*# Calculate the number of rows needed*num\_rows = (num\_features - 1) // plots\_per\_row + 1  
  
*# Create a figure and subplots grid*fig, axs = plt.subplots(num\_rows, plots\_per\_row, figsize=(20, 5 \* num\_rows))  
  
*# Flatten the subplots grid for easy indexing*axs = axs.flatten()  
  
*# Iterate through combinations of features*subplot\_idx = 0  
*for* i *in* range(num\_features):  
 *for* j *in* range(i + 1, num\_features):  
 *if* subplot\_idx >= num\_rows \* plots\_per\_row:  
 *break* x\_label = cols[i]  
 y\_label = cols[j]  
  
 *# Plot the scatterplot in the current subplot* sns.scatterplot(x=x\_label, y=y\_label, data=df2, hue='Cluster', ax=axs[subplot\_idx])  
 axs[subplot\_idx].set\_title(f'{x\_label} vs {y\_label}')  
  
 subplot\_idx += 1  
  
*# Hide empty subplots in the last row  
if* num\_features % plots\_per\_row != 0:  
 *for* i *in* range(num\_features % plots\_per\_row, plots\_per\_row):  
 axs[-i].set\_visible(*False*)  
  
*# Adjust layout*plt.tight\_layout()  
  
*# Show the plot*plt.show()

**Output**

