# INDIAN INSTITUTE OF INFORMATION TECHNOLOGY BHOPAL

## Computer Science and Engineering Department


## DATA WAREHOUSE AND MINING(DWM)(CSE-322)

## PRATICAL EXAMINATION FILE

**Semester-6**

**SUBMITTED TO:**

**Dr.Yatendra Sahu**

**SUBMITTED BY**

**Labhanshu Gupta**
**22U02007**

# Assignment-1

## 1) Dataset Exploration and Attribute categorization

- **Dataset Name**:Traffic Crashes Dataset
- **Source**:Kaggle
- (https://www.kaggle.com/datasets/oktayrdeki/traffic-accidents)
- **Domain**: The dataset contains detailed information on the traffic accidents across various regions and time periods.
- **Number of Data Objects**:8077 rows (accident records)
- **Number of features**:22 attributes,including:
    - **crash_date:** The date the accident occurred.(continuous)
    - **traffic_control_device:** The type of traffic control device involved (e.g., traffic light, sign).(nominal)
    - **weather_condition:** The weather conditions at the time of the accident.(nominal)
    - **lighting_condition:** The lighting conditions at the time of the accident.(nominal)
    - **first_crash_type:** The initial type of the crash (e.g., head-on, rear-end).(nominal)
    - **trafficway_type:** The type of roadway involved in the accident (e.g., highway, local road).(nominal)
    - **alignment:** The alignment of the road where the accident occurred (e.g., straight, curved).(nominal)
    - **roadway_surface_cond:** The condition of the roadway surface (e.g., dry, wet, icy).(nominal)
    - **road_defect:** Any defects present on the road surface.(nominal)
    - **crash_type:** The overall type of the crash.(nominal)
    - **intersection_related_i**: Whether the accident was related to an intersection.(nominal)
    - **damage**: The extent of the damage caused by the accident.(continuous)
    - **prim_contributory_cause**: The primary cause contributing to the crash.(nominal)
    - **most_severe_injury:** The most severe injury sustained in the crash.(nominal)
    - **injuries_total**: The total number of injuries reported.(continuous)
    - **injuries_fatal:** The number of fatal injuries resulting from the accident. (continuous)
    - **injuries_incapacitating**: The number of incapacitating injuries. (continuous)
    - **injuries_non_incapacitating:** The number of non-incapacitating injuries. (continuous)
    - **injuries_reported_not_evident:** The number of injuries reported but not visibly evident. (continuous)
    - **crash_hour**: The hour the accident occurred. (continuous)
    - **crash_day_of_week:** The day of the week the accident occurred.(continuous)
    - **crash_month:** The month the accident occurred.(continuous)

- **Categorial (Nominal) Attributes:**
    - Traffic_control_device
    - Weather_condition
    - Lightining_condition
    - First_crash_type
    - Trafficway_type
    - Alignment
    - Roadway_surface_condion
    - Road_defect
    - Crash_type
    - Intersection_related_i
    - Prim_contributory_cause
    - Most_severe_injury

- Real-Valued(Continuous) Attributes:
    - Crash_date
    - Damage
    - Injuries_total
    - Injuries_fatal
    - Injuries_incapacitating
    - injuries_non_incapacitating
    - injuries_reported_not_evident
    - crash_hour
    - crash_day_of_week
    - crash_month

# Assignment-2

## 1)Dataset pre-processing and exploration.

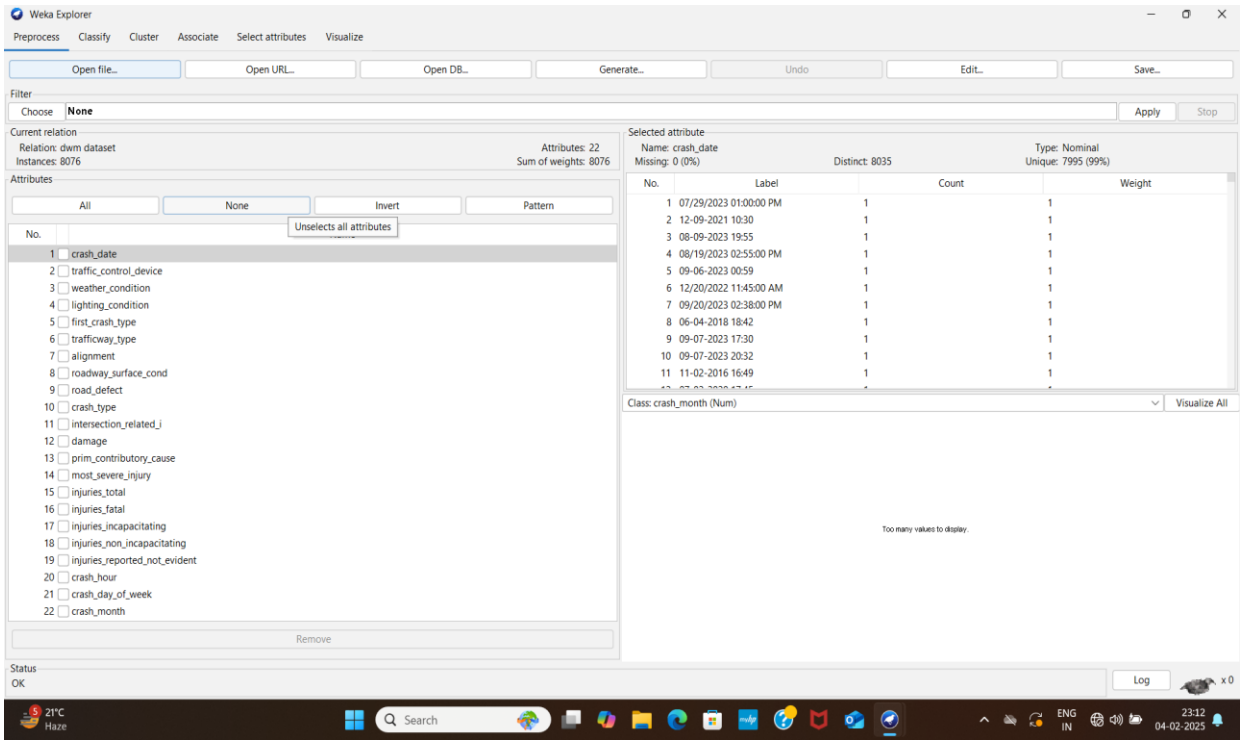
### 1. Loading the dataset

->Open Weka Explorer.

->In the preprocess bar click on “open file…”.

->select the file you want to open and press ok.

->your data set is loaded and ready to be pre processed.




**Result->** your data set file is opened in the weka explorer.

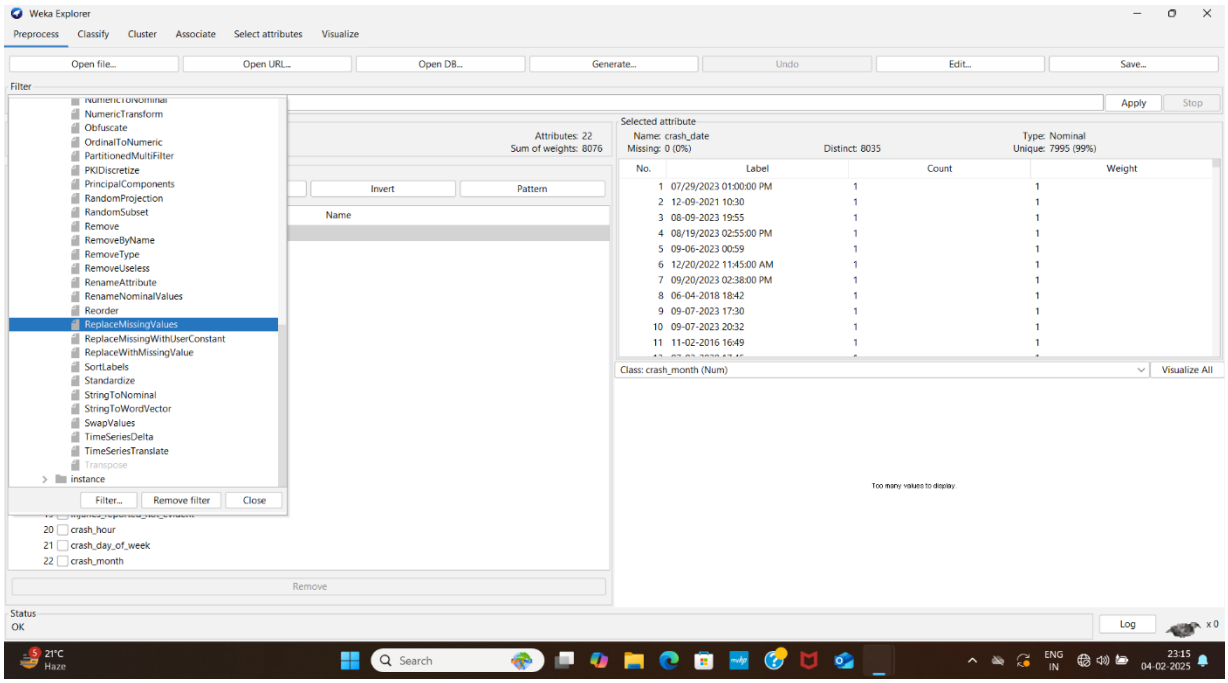
### 2. Handle missing values

All values are present so we don’t need to apply any filter for replacing missing values

But if there were we can do it from the filters option.

->Click on the choose option and then click on the filter option :Unsupervised

->attribute->ReplaceMissingValues.

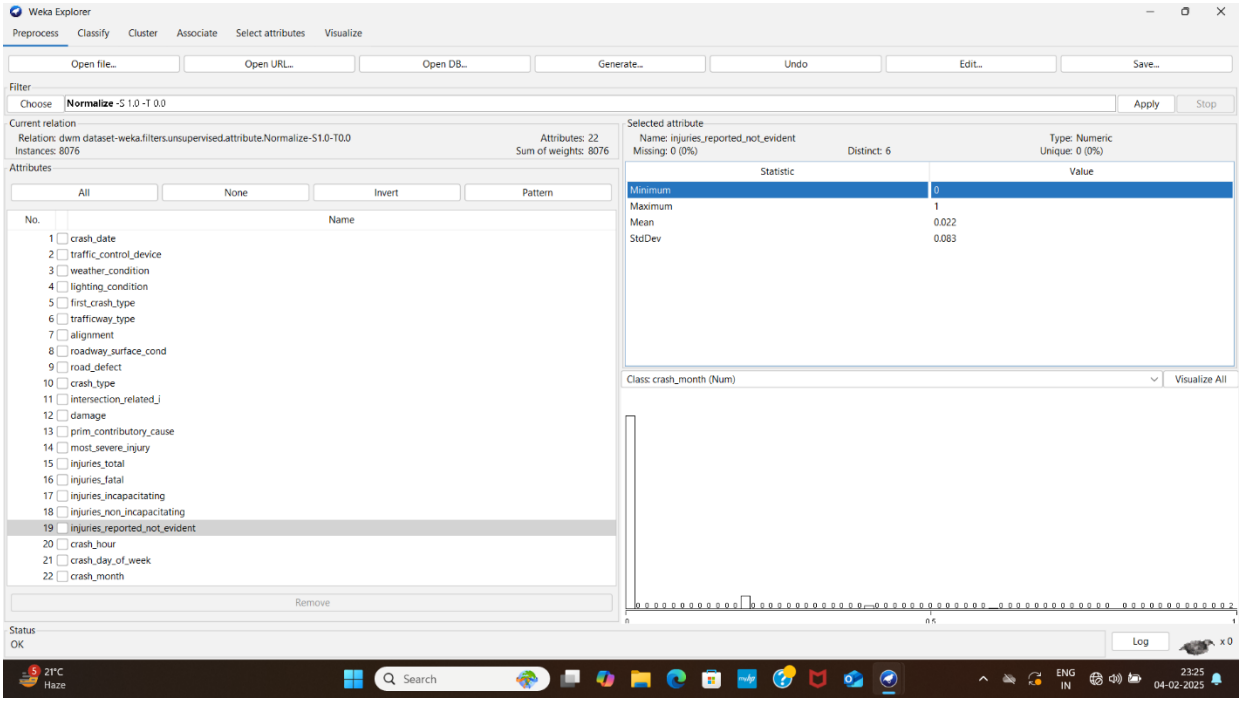



**Result->**This replaces the missing values with either the mean(for numeric attributes) or the mode(for nominal attributes).

## 3. Normalize data

->Go to the filters section again .

->Choose the filter :Unsupervised->Attribute->Normalize.



**Result->**All the numeric attributes are now scaled between 0 and 1.

3. **Convert data types.**

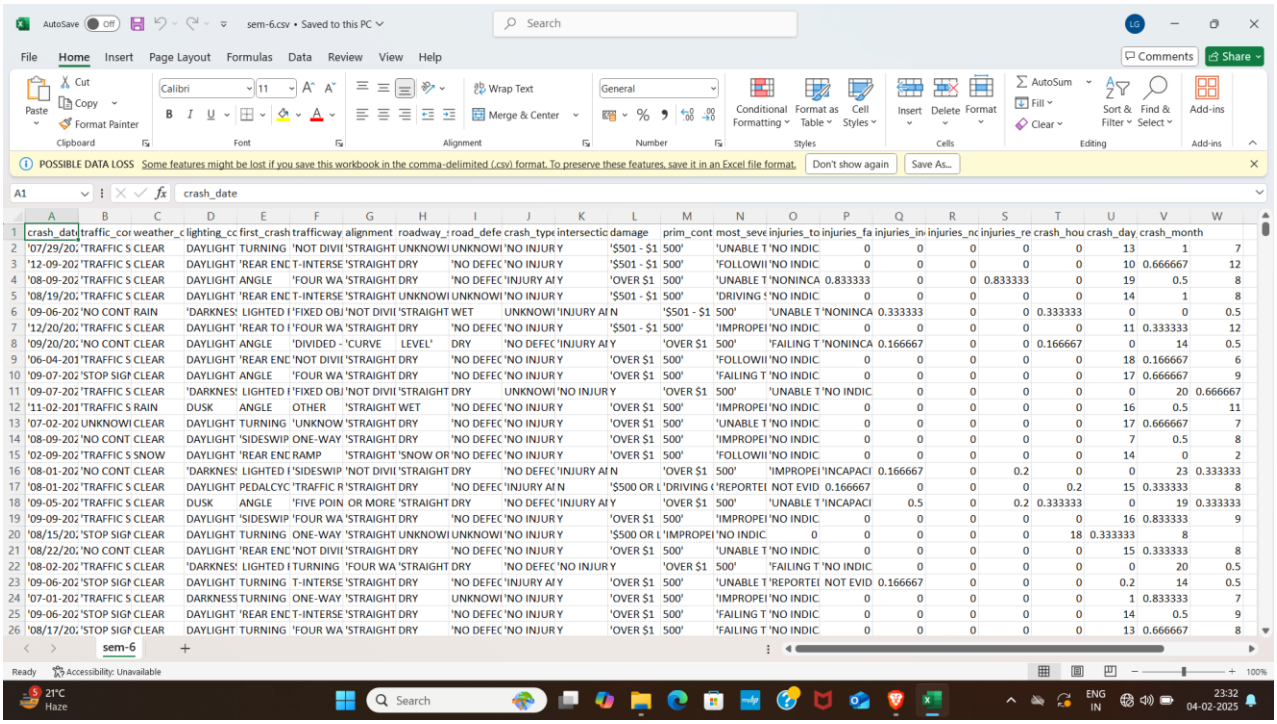
->Use the filter unsupervised->Attribute->NumericToNominal to convert any attribute from Numeric to normal.

->Use the filter unsupervised->Attribute->NominalToNumeric to convert any attribute from Numeric to normal.

**5 Save the preprocessed dataset**

**->**After performing all the operations Click on the save button at the top right corner of the Screen.

**Result->**Normalized dataset

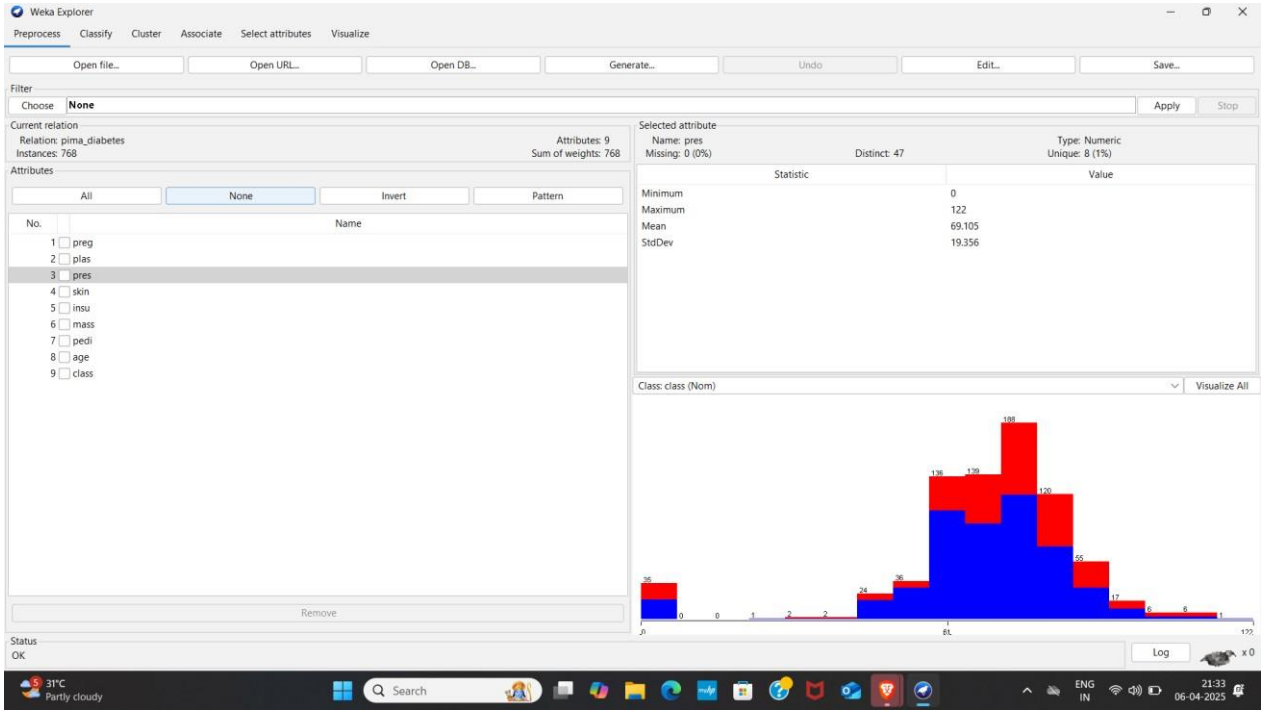


# Assignment-3

## 1)Bayesian classification for a dataset after pre processing using weka and python libraries

### 1. Loading the dataset

Diabetes dataset is chosen to perform Bayesian classification.



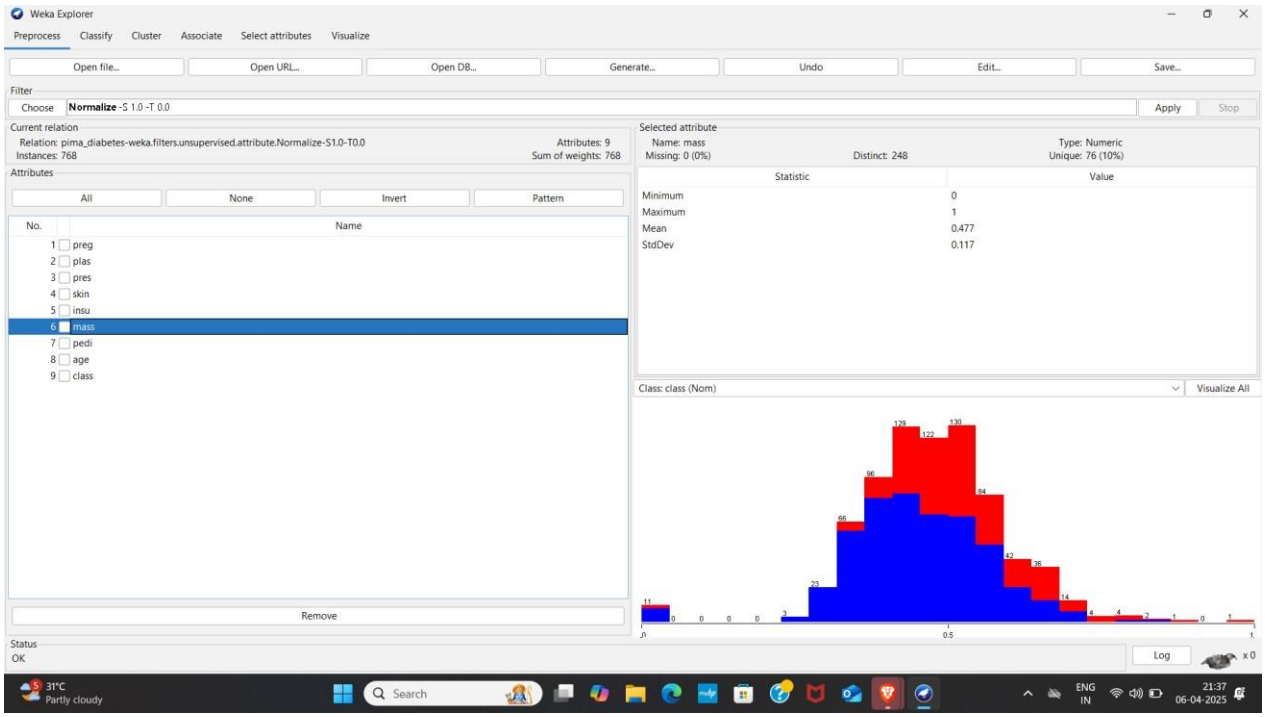

### 2.Handle missing values

All values are present so we don't need to apply any filter for replacing missing values

### 3.Normalize data

->Go to the filters section again .

->Choose the filter :Unsupervised->Attribute->Normalize.




**Result->**All the numeric attributes are now scaled between 0 and 1.

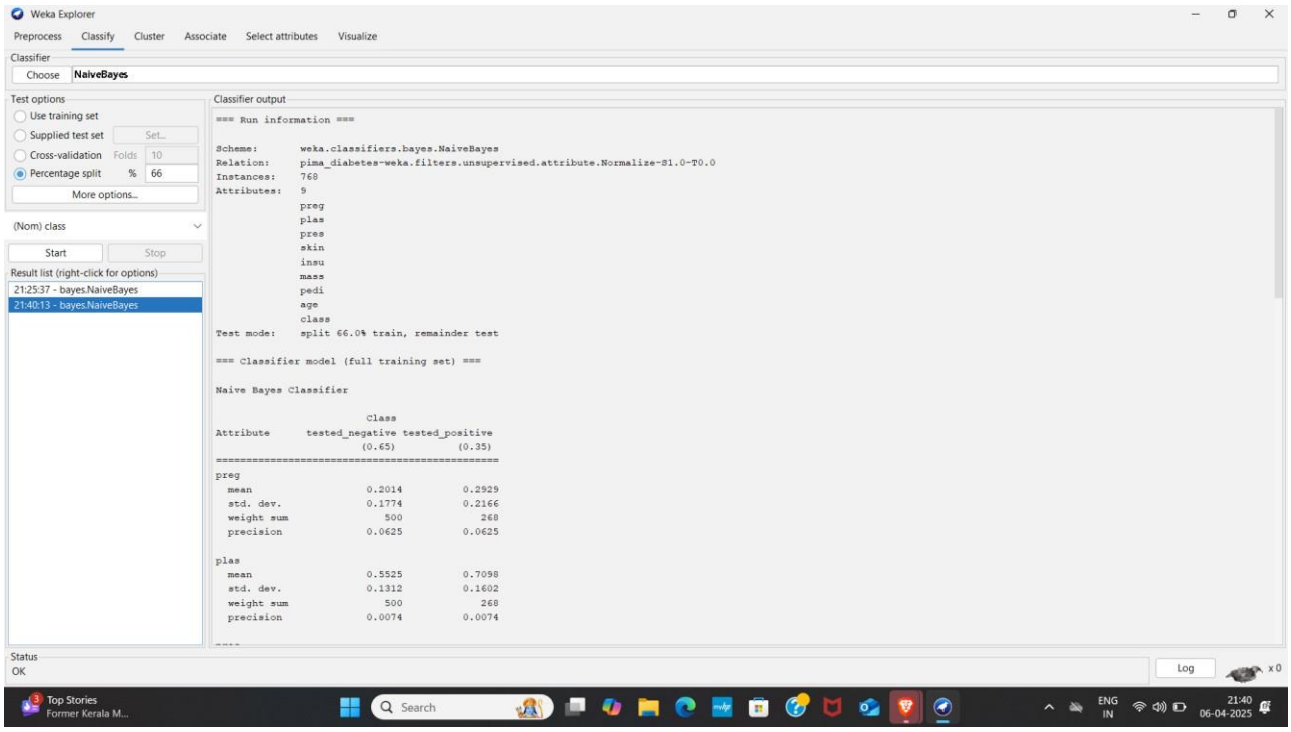
## 4)Applying Bayesian classification on the preprocessed data.

1) Click on classify in the tab above.

2)Choose the classifier->Bayes->NaiveBayes.

3)choose the values of cross-validation and percentage split

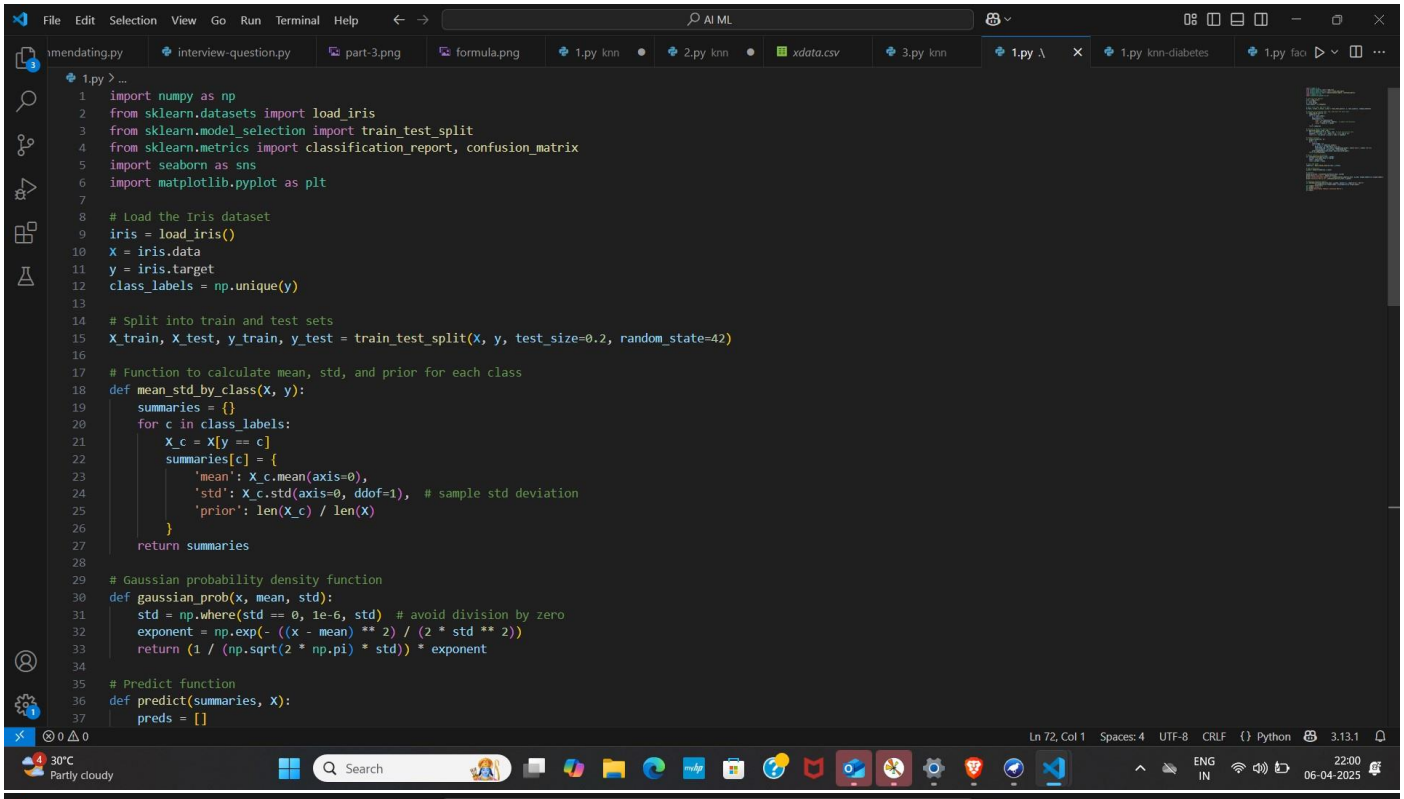
4)click on start.



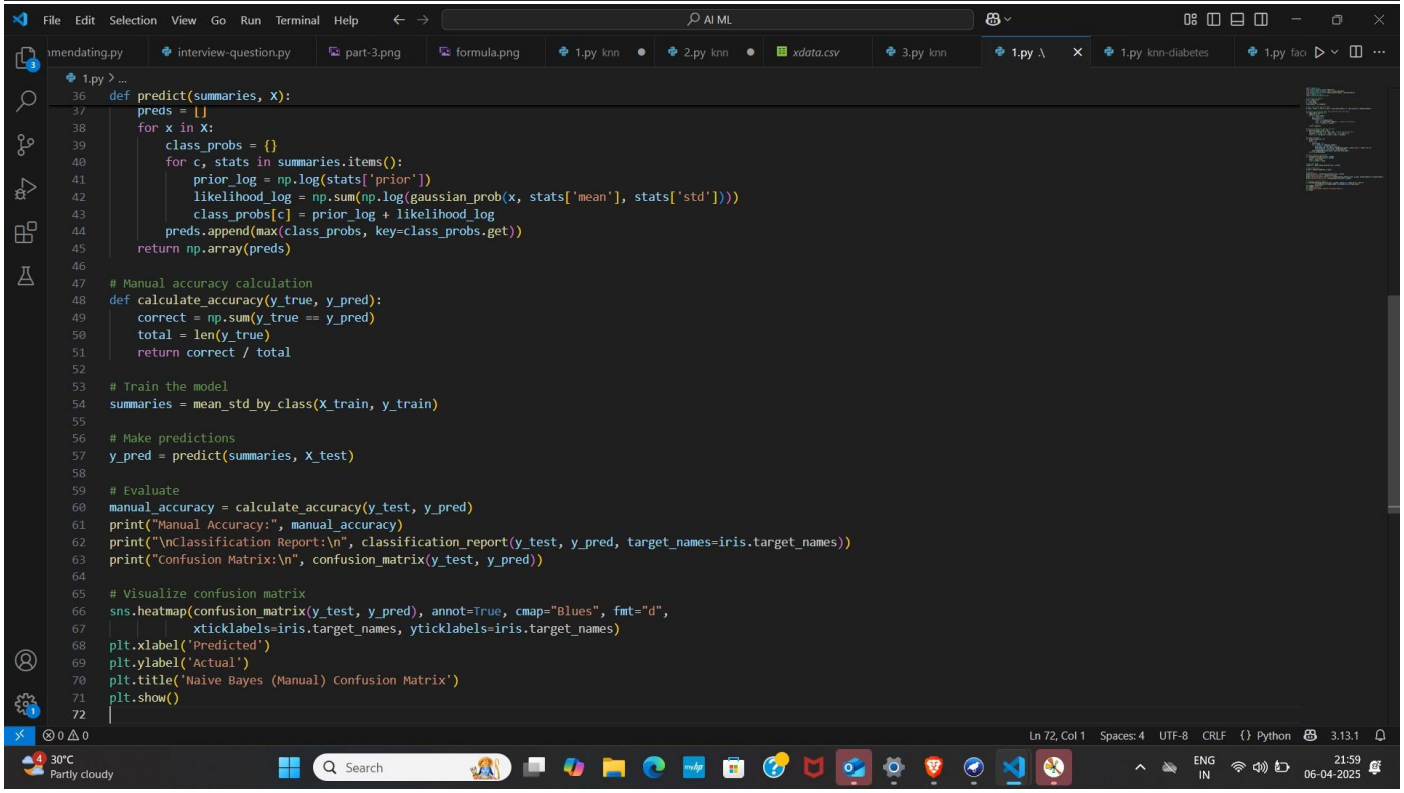

```
Time taken to build model: 0 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correctly Classified Instances         203               77.7778 %
Incorrectly Classified Instances        58               22.2222 %
Kappa statistic                          0.4776
Mean absolute error                      0.2654
Root mean squared error                  0.382
Relative absolute error                 58.8383 %
Root relative squared error             81.6152 %
Total Number of Instances              261

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.854    0.386    0.826      0.854    0.840      0.478    0.853     0.919     tested_negative
               0.614    0.146    0.662      0.614    0.638      0.478    0.853     0.760     tested_positive
Weighted Avg.  0.778    0.309    0.774      0.778    0.775      0.478    0.853     0.868

=== Confusion Matrix ===

   a   b   <-- classified as
 152  26 |   a = tested_negative
  32  51 |   b = tested_positive
```

**Result->**Naïve's baise classification is performed and accuracy of this model is 77.778 %.

## 2)Python implementatation for Bayesian classification



```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
class_labels = np.unique(y)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Function to calculate mean, std, and prior for each class
def mean_std_by_class(X, y):
    summaries = {}
    for c in class_labels:
        X_c = X[y == c]
        summaries[c] = {
            'mean': X_c.mean(axis=0),
            'std': X_c.std(axis=0, ddof=1),  # sample std deviation
            'prior': len(X_c) / len(X)
        }
    return summaries

# Gaussian probability density function
def gaussian_prob(x, mean, std):
    std = np.where(std == 0, 1e-6, std)  # avoid division by zero
    exponent = np.exp(- ((x - mean) ** 2) / (2 * std ** 2))
    return (1 / (np.sqrt(2 * np.pi) * std)) * exponent

# Predict function
def predict(summaries, X):
    preds = []
```



```python
def predict(summaries, X):
    preds = []
    for x in X:
        class_probs = {}
        for c, stats in summaries.items():
            prior_log = np.log(stats['prior'])
            likelihood_log = np.sum(np.log(gaussian_prob(x, stats['mean'], stats['std'])))
            class_probs[c] = prior_log + likelihood_log
        preds.append(max(class_probs, key=class_probs.get))
    return np.array(preds)

# Manual accuracy calculation
def calculate_accuracy(y_true, y_pred):
    correct = np.sum(y_true == y_pred)
    total = len(y_true)
    return correct / total

# Train the model
summaries = mean_std_by_class(X_train, y_train)

# Make predictions
y_pred = predict(summaries, X_test)

# Evaluate
manual_accuracy = calculate_accuracy(y_test, y_pred)
print("Manual Accuracy:", manual_accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_names))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Visualize confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt="d",
            xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Naive Bayes (Manual) Confusion Matrix')
plt.show()
```

**Result-:**As the dataset was clean and well seperated we got accuracy 1.0.

```
Manual Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

## 3)Advantages and disadvantages of Bayesian classification is:-

## Advantages-:

1) Simple and Fast:
    - Easy to implement and computationally efficient.
    - Works well with large datasets and high-dimensional data.
2) Performs Well with Less Data:
    - Requires less training data compared to other classifiers.
    - Handles both binary and multiclass classification.
3) Effective for Certain Problems:
    - Surprisingly effective for text classification (e.g., spam detection, sentiment analysis).
    - Performs well when the assumption of feature independence holds.
4) Handles Missing Data:
    - Can handle missing values relatively well by simply ignoring the feature during probability calculation.
5) Scalable:
    - Scales linearly with the number of features and data points.

**Disadvantages:-**

1) Strong Independence Assumption:

- Assumes features are independent, which is rarely true in real-world data.
- Violations of this assumption can lead to poor performance.

2) Zero-Frequency Problem:
   - If a categorical variable has a category in the test data not seen in the training data, it assigns zero probability.
   - This can be mitigated with techniques like Laplace smoothing.
3) Not Ideal for Continuous Features (without modification):
   - Basic Naive Bayes works better with categorical input.
   - For continuous features, it assumes a Gaussian distribution which might not always hold true.
4) Less Flexible Compared to Other Models:
   - Doesn't learn interactions between features.
   - Other models like decision trees or random forests can model more complex relationships.
5) Can Be Outperformed by More Complex Models:
   - In practice, more sophisticated algorithms (e.g., SVM, XGBoost, neural networks) often yield better results, especially for complex datasets.

# Assignment-4

## 1)Decision tree classification for a dataset used in preprocessing using weka and python libraries.

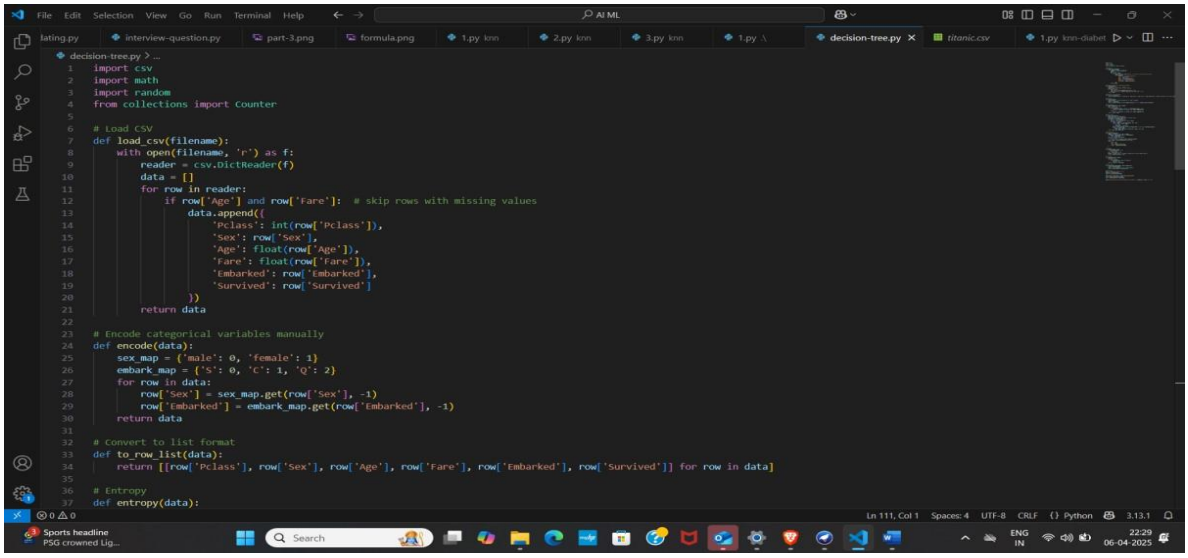
## python implementation for decision tree classification

1)We have used titanic data set to perform decision tree classification on

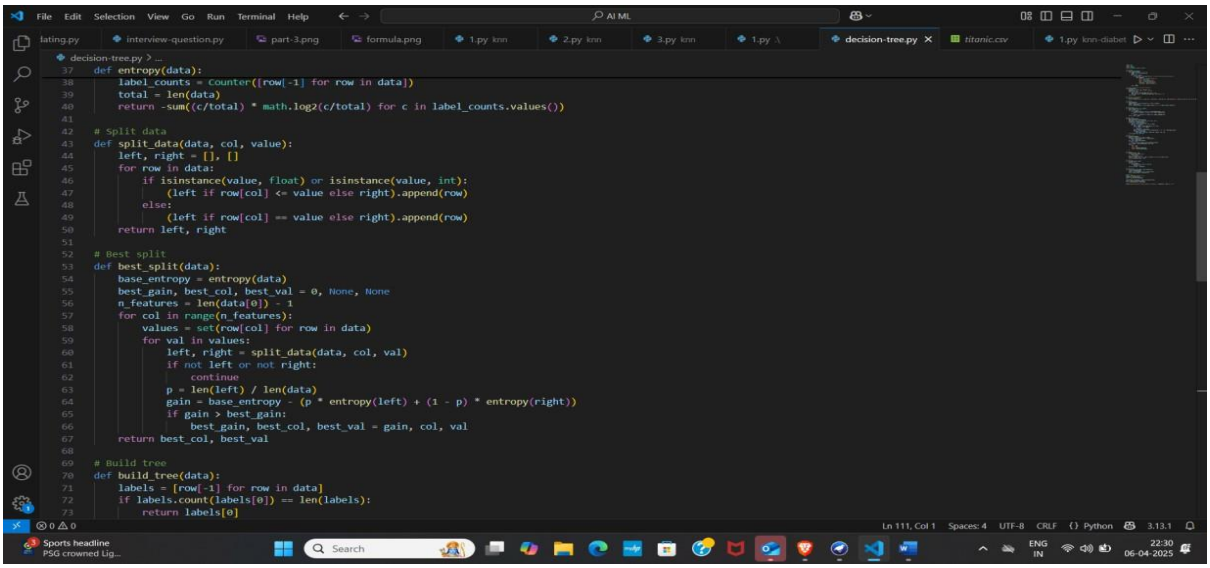
Features-> PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked.

Label->Survived .

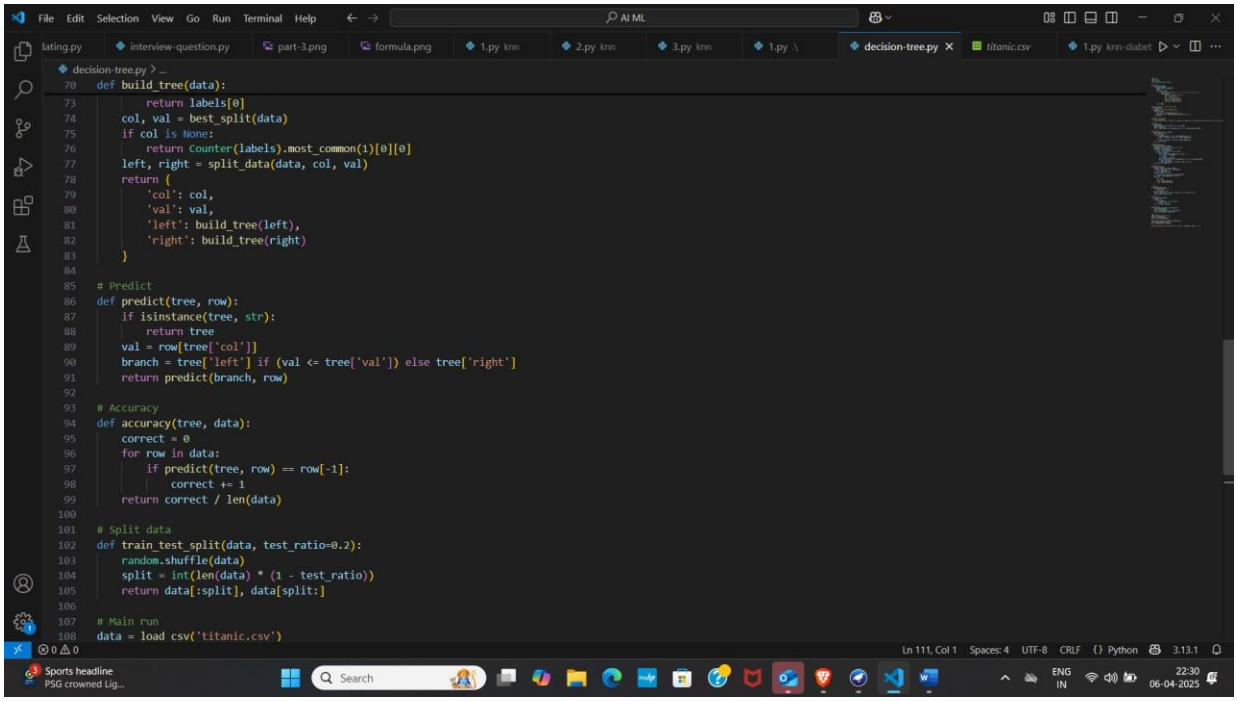
Features we have selected are-:  Pclass, Sex, Age, Fare, Embarked.



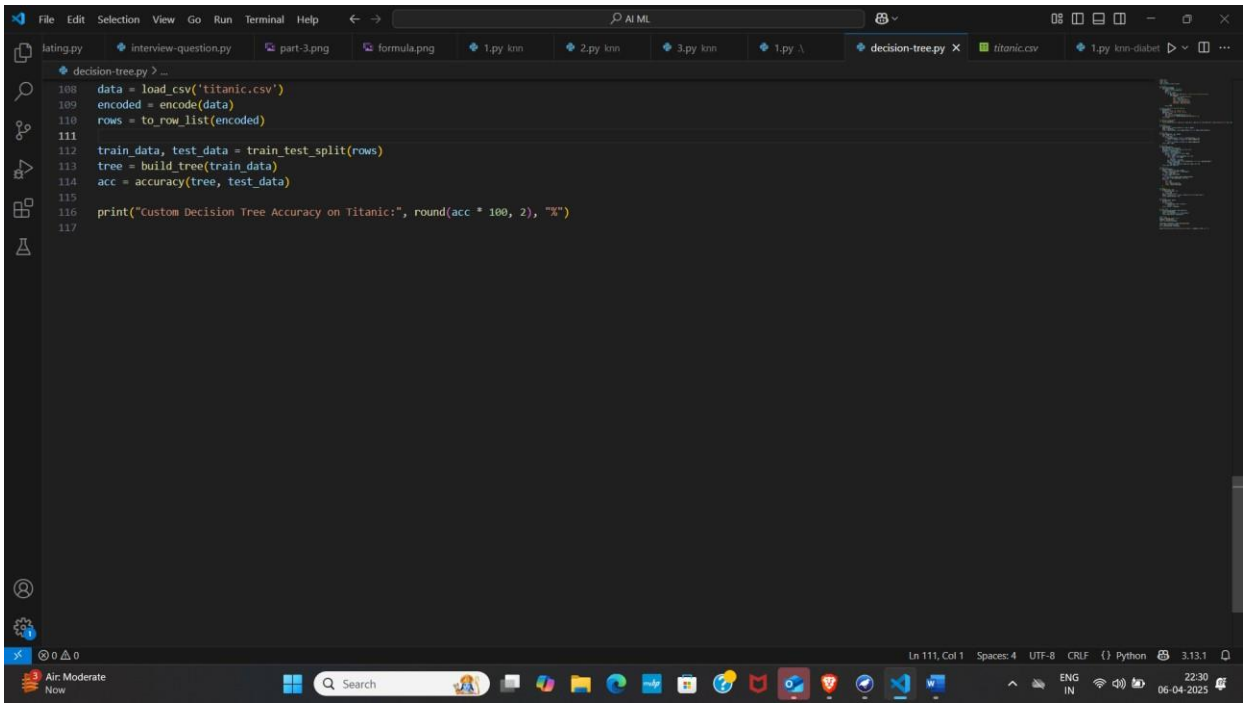
```python
import csv
import math
import random
from collections import Counter

# Load CSV
def load_csv(filename):
    with open(filename, 'r') as f:
        reader = csv.DictReader(f)
        data = []
        for row in reader:
            if row['Age'] and row['Fare']:  # skip rows with missing values
                data.append({
                    'Pclass': int(row['Pclass']),
                    'Sex': row['Sex'],
                    'Age': float(row['Age']),
                    'Fare': float(row['Fare']),
                    'Embarked': row['Embarked'],
                    'Survived': row['Survived']
                })
        return data

# Encode categorical variables manually
def encode(data):
    sex_map = {'male': 0, 'female': 1}
    embark_map = {'S': 0, 'C': 1, 'Q': 2}
    for row in data:
        row['Sex'] = sex_map.get(row['Sex'], -1)
        row['Embarked'] = embark_map.get(row['Embarked'], -1)
    return data

# Convert to list format
def to_row_list(data):
    return [[row['Pclass'], row['Sex'], row['Age'], row['Fare'], row['Embarked'], row['Survived']] for row in data]

# Entropy
def entropy(data):
```



```python
def entropy(data):
    label_counts = Counter([row[-1] for row in data])
    total = len(data)
    return -sum((c/total) * math.log2(c/total) for c in label_counts.values())

# Split data
def split_data(data, col, value):
    left, right = [], []
    for row in data:
        if isinstance(value, float) or isinstance(value, int):
            (left if row[col] <= value else right).append(row)
        else:
            (left if row[col] == value else right).append(row)
    return left, right

# Best split
def best_split(data):
    base_entropy = entropy(data)
    best_gain, best_col, best_val = 0, None, None
    n_features = len(data[0]) - 1
    for col in range(n_features):
        values = set(row[col] for row in data)
        for val in values:
            left, right = split_data(data, col, val)
            if not left or not right:
                continue
            p = len(left) / len(data)
            gain = base_entropy - (p * entropy(left) + (1 - p) * entropy(right))
            if gain > best_gain:
                best_gain, best_col, best_val = gain, col, val
    return best_col, best_val

# Build tree
def build_tree(data):
    labels = [row[-1] for row in data]
    if labels.count(labels[0]) == len(labels):
        return labels[0]
```



```python
def build_tree(data):
        return labels[0]
    col, val = best_split(data)
    if col is None:
        return Counter(labels).most_common(1)[0][0]
    left, right = split_data(data, col, val)
    return {
        'col': col,
        'val': val,
        'left': build_tree(left),
        'right': build_tree(right)
    }

# Predict
def predict(tree, row):
    if isinstance(tree, str):
        return tree
    val = row[tree['col']]
    branch = tree['left'] if (val <= tree['val']) else tree['right']
    return predict(branch, row)

# Accuracy
def accuracy(tree, data):
    correct = 0
    for row in data:
        if predict(tree, row) == row[-1]:
            correct += 1
    return correct / len(data)

# Split data
def train_test_split(data, test_ratio=0.2):
    random.shuffle(data)
    split = int(len(data) * (1 - test_ratio))
    return data[:split], data[split:]

# Main run
data = load_csv('titanic.csv')
```



```python
data = load_csv('titanic.csv')
encoded = encode(data)
rows = to_row_list(encoded)

train_data, test_data = train_test_split(rows)
tree = build_tree(train_data)
acc = accuracy(tree, test_data)

print("Custom Decision Tree Accuracy on Titanic:", round(acc * 100, 2), "%")
```

**Result-:**Accuracy we got is 77.62%.

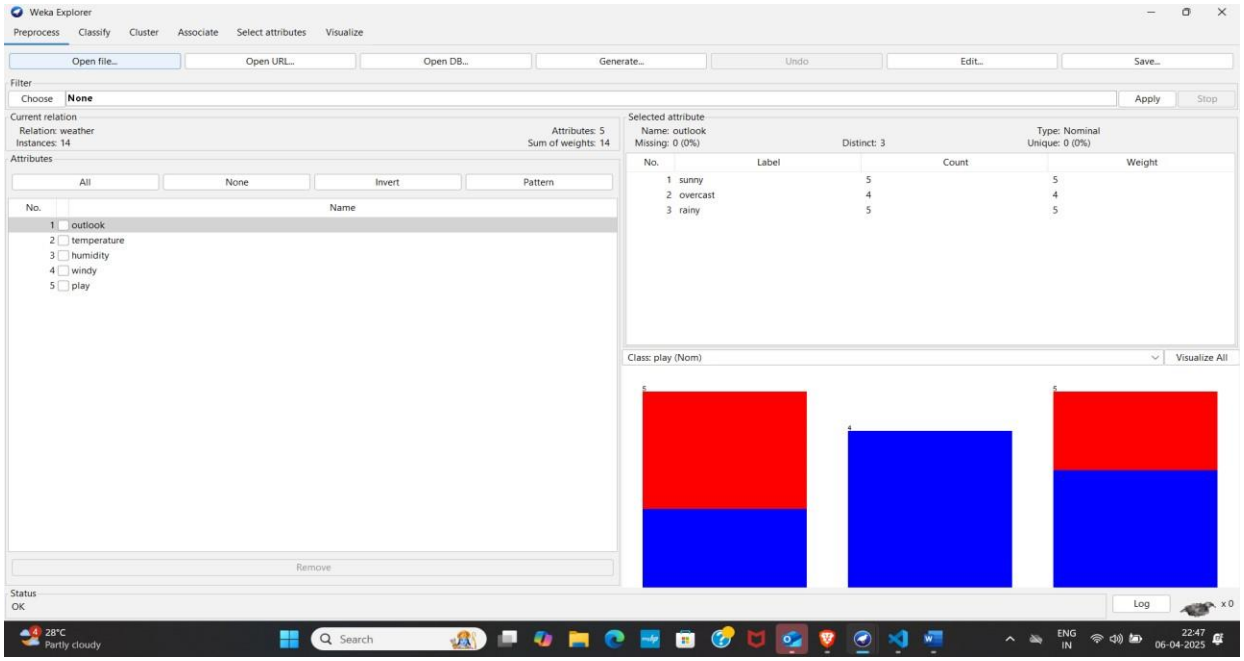
```
PS D:\AI ML> & "C:/Program Files/Python313/python.exe" "d:/AI ML/decision-tree.py"
Custom Decision Tree Accuracy on Titanic: 77.62 %
```

## 2)Decision tree classification using weka

1)loading and preprocessing the data

Features:- outlook,temperature,humidity,windy

Label:-play



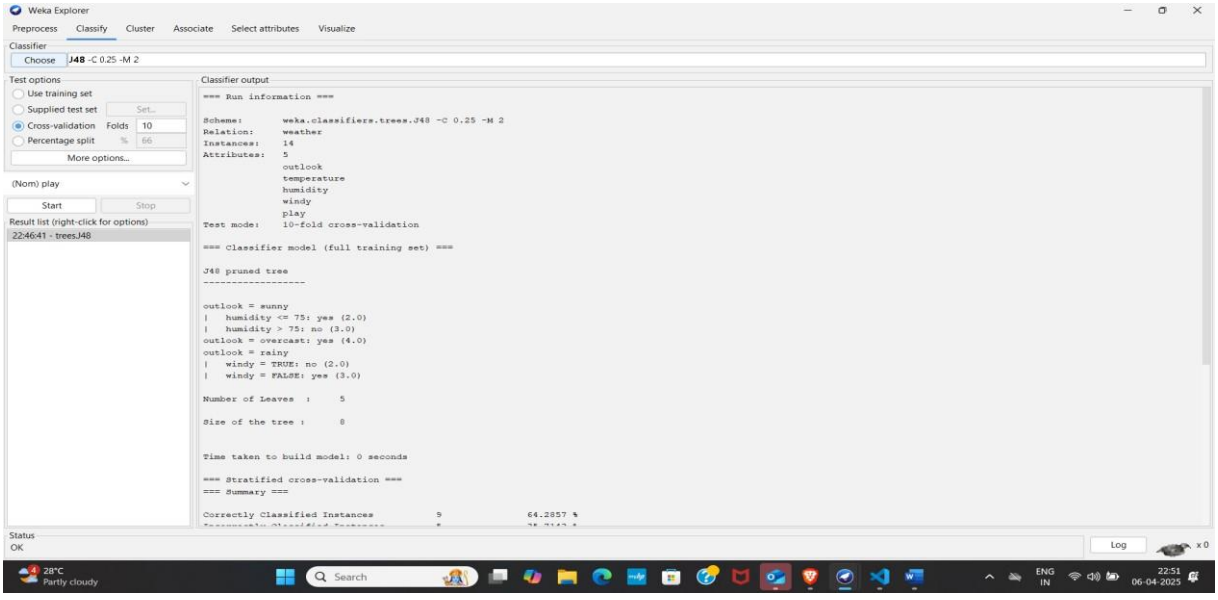

## 2) using J-48 decision tree

-> Click on classify in the tab above.

->Choose the classifier->trees->J48

->choose the values of cross-validation and percentage split

->Click on start.



```
Correctly Classified Instances           9               64.2857 %
Incorrectly Classified Instances         5               35.7143 %
Kappa statistic                          0.186
Mean absolute error                      0.2857
Root mean squared error                  0.4818
Relative absolute error                 60      %
Root relative squared error             97.6586 %
Total Number of Instances               14

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.778    0.600    0.700      0.778    0.737      0.189    0.789     0.847     yes
                 0.400    0.222    0.500      0.400    0.444      0.189    0.789     0.738     no
Weighted Avg.    0.643    0.465    0.629      0.643    0.632      0.189    0.789     0.808

=== Confusion Matrix ===

 a b   <-- classified as
 7 2 | a = yes
 3 2 | b = no
```

Result-:Accuracy of decision tree classification is 64.2857 %.

**Advantages of decision tree classification-:**

1. Easy to understand – Intuitive and visual.
2. No need for feature scaling – Works on raw data.
3. Handles both categorical & numerical data.
4. Captures non-linear patterns.
5. Feature importance – Highlights key predictors.
6. Fast prediction – Quick once trained.
7. Works for classification & regression.
8. No domain knowledge needed.

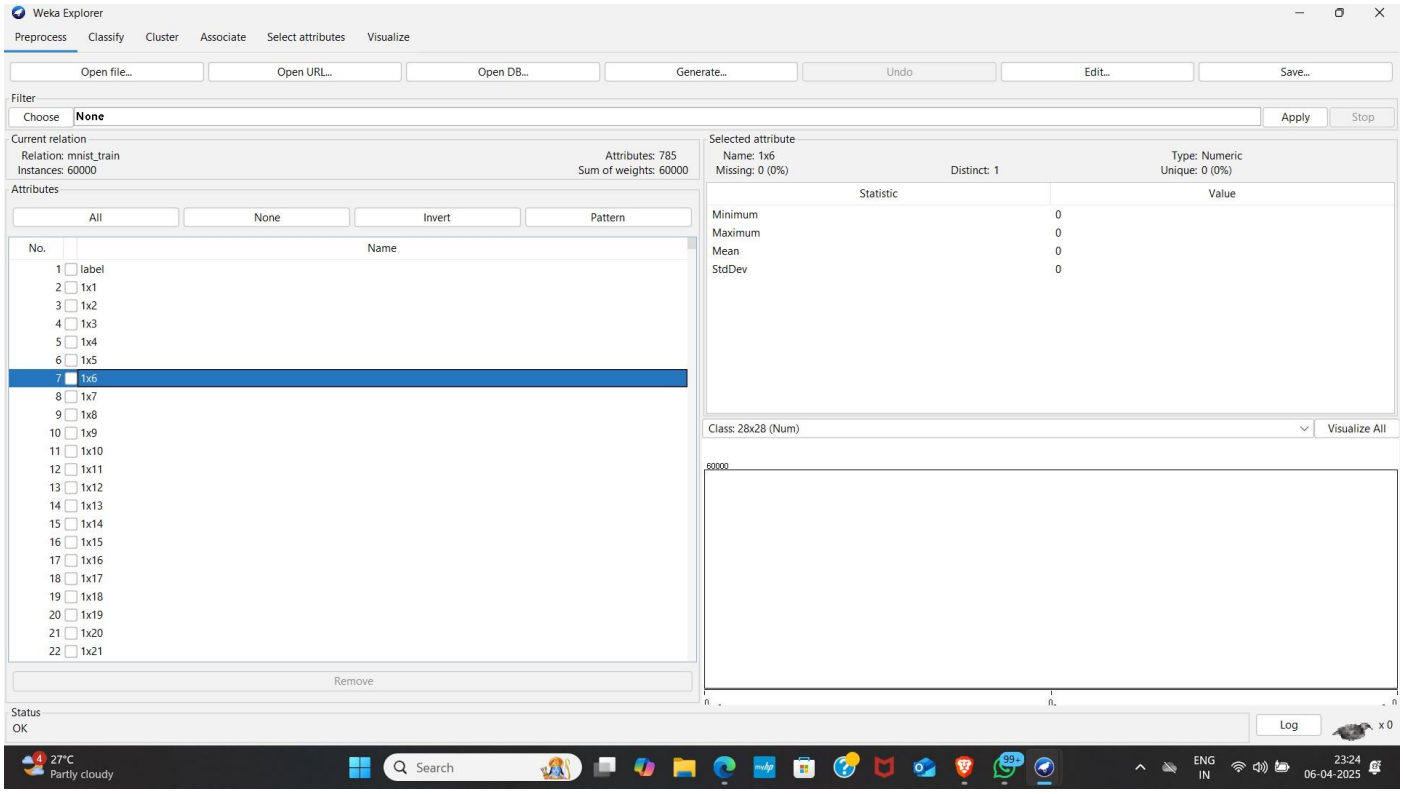
## Disadvantages of decision tree classification-:

1. Overfitting – Trees can become too complex and fit noise.
2. Unstable – Small changes in data can change the structure.
3. Biased splits – Can favor features with more levels/categories.
4. Less accurate alone – Often outperformed by ensemble methods (like Random Forest).
5. Harder with continuous variables – Needs careful handling.

# Assignment-5

## 1)K-Nearest neibours classifiers for a dataset after pre-processing using weka/python libraries.

1)Loading the mnist_train.csv dataset into weka and doing exploratory analysis on the dataset. Features->786 pixels Labels->10(0-9).




## 2)Normalize data

->Go to the filters section again .

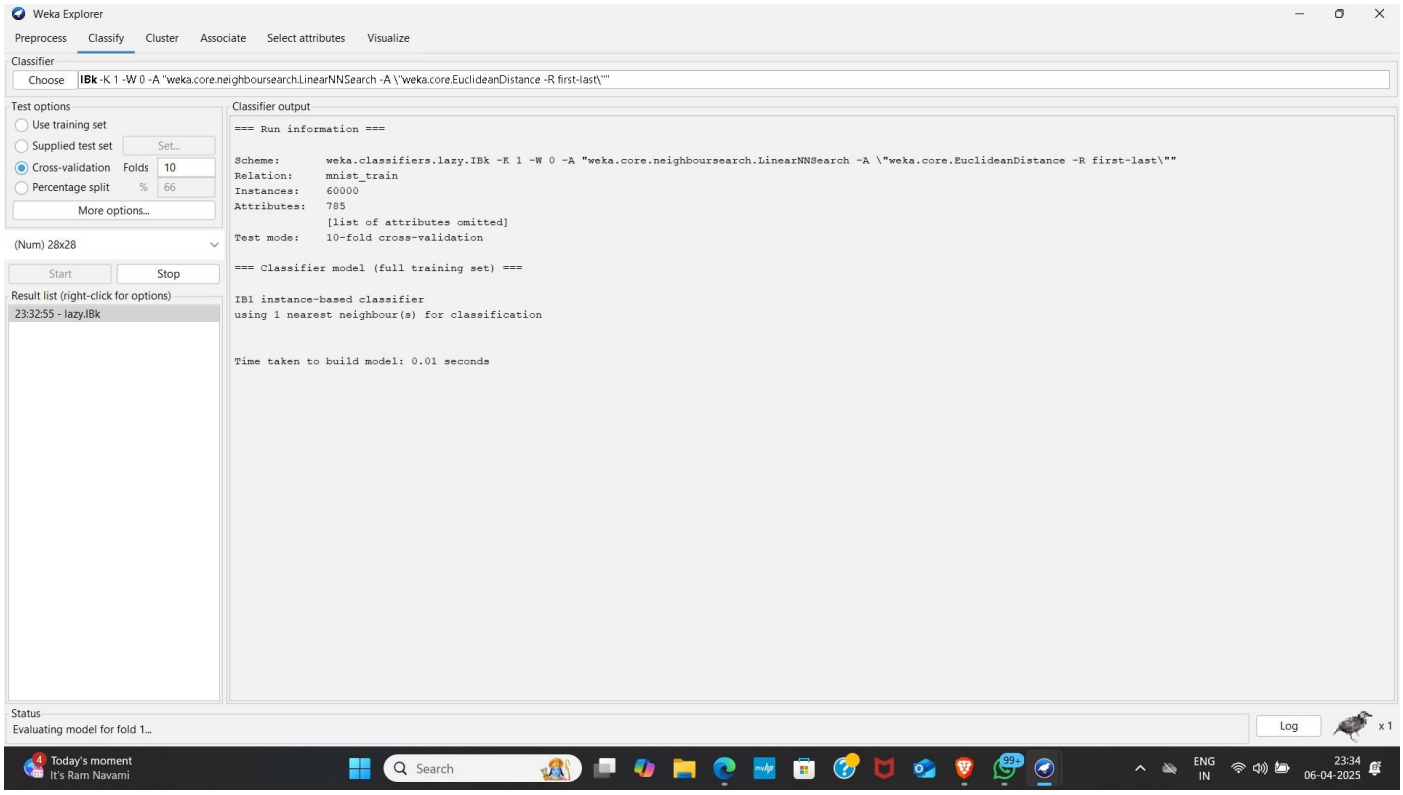
->Choose the filter :Unsupervised->Attribute->Normalize.

## 3)Appying knn-classifier

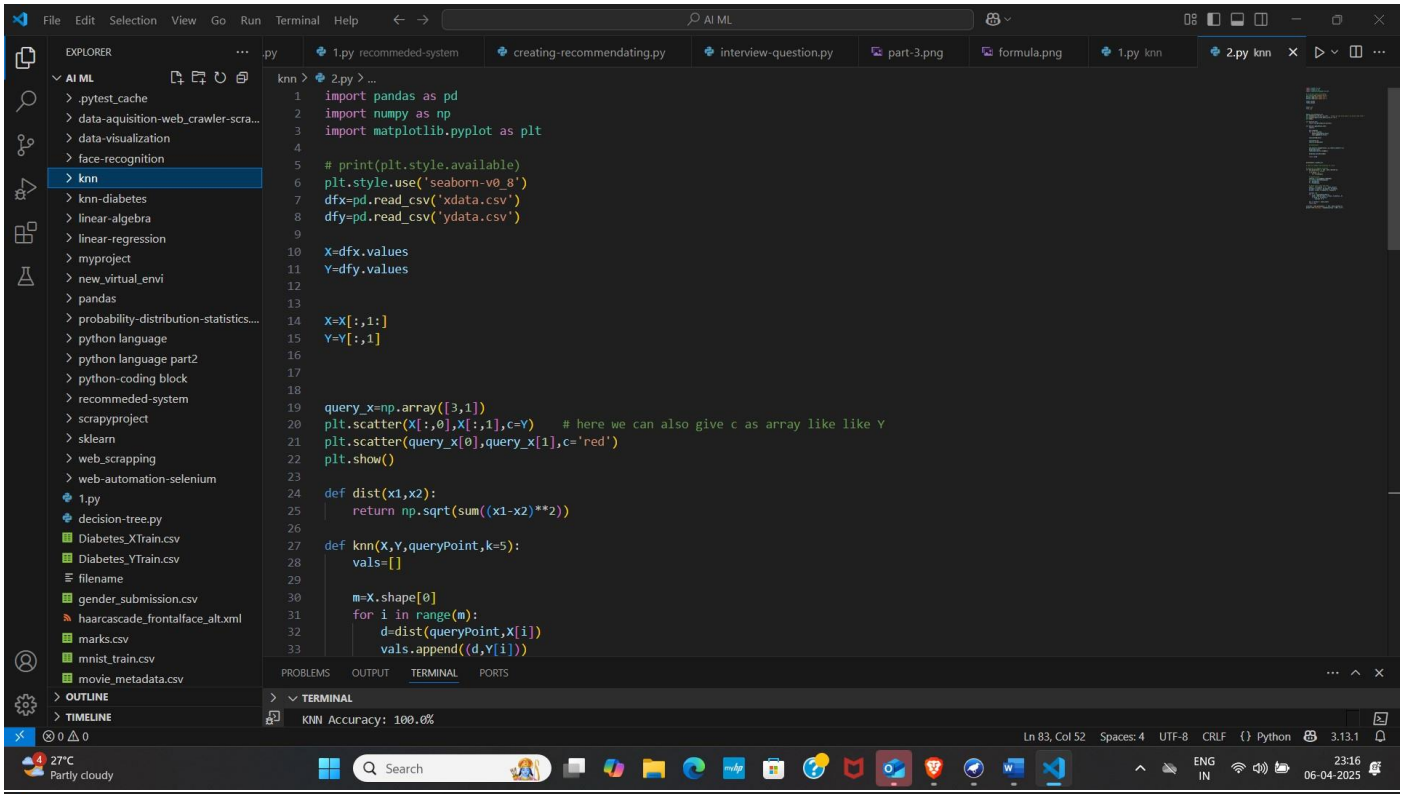
->Click on classifier in the tab.

->choose the classifier:lazy->IBK.

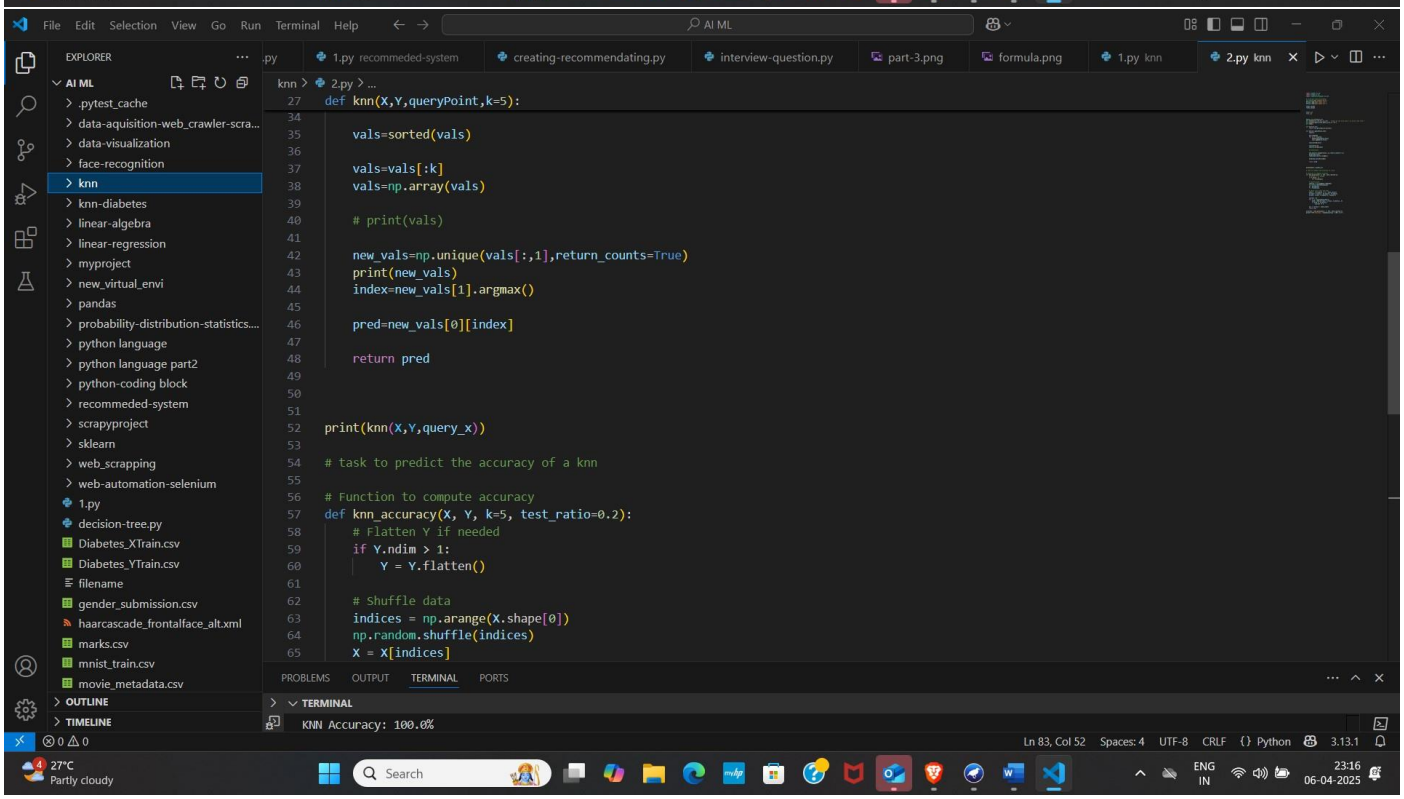
## Result-:


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Num) 28x28
Start
Stop
Result list (right-click for options)
23:32:55 - lazy.IBk
Classifier output
=== Run information ===
Scheme: weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Relation: mnist_train
Instances: 60000
Attributes: 785
[list of attributes omitted]
Test mode: 10-fold cross-validation
=== Classifier model (full training set) ===
IB1 instance-based classifier
using 1 nearest neighbour(s) for classification
Time taken to build model: 0.01 seconds
Status
Evaluating model for fold 1...
Log
x 1
Today's moment
It's Ram Navami
Search
ENG
IN
23:34
06-04-2025

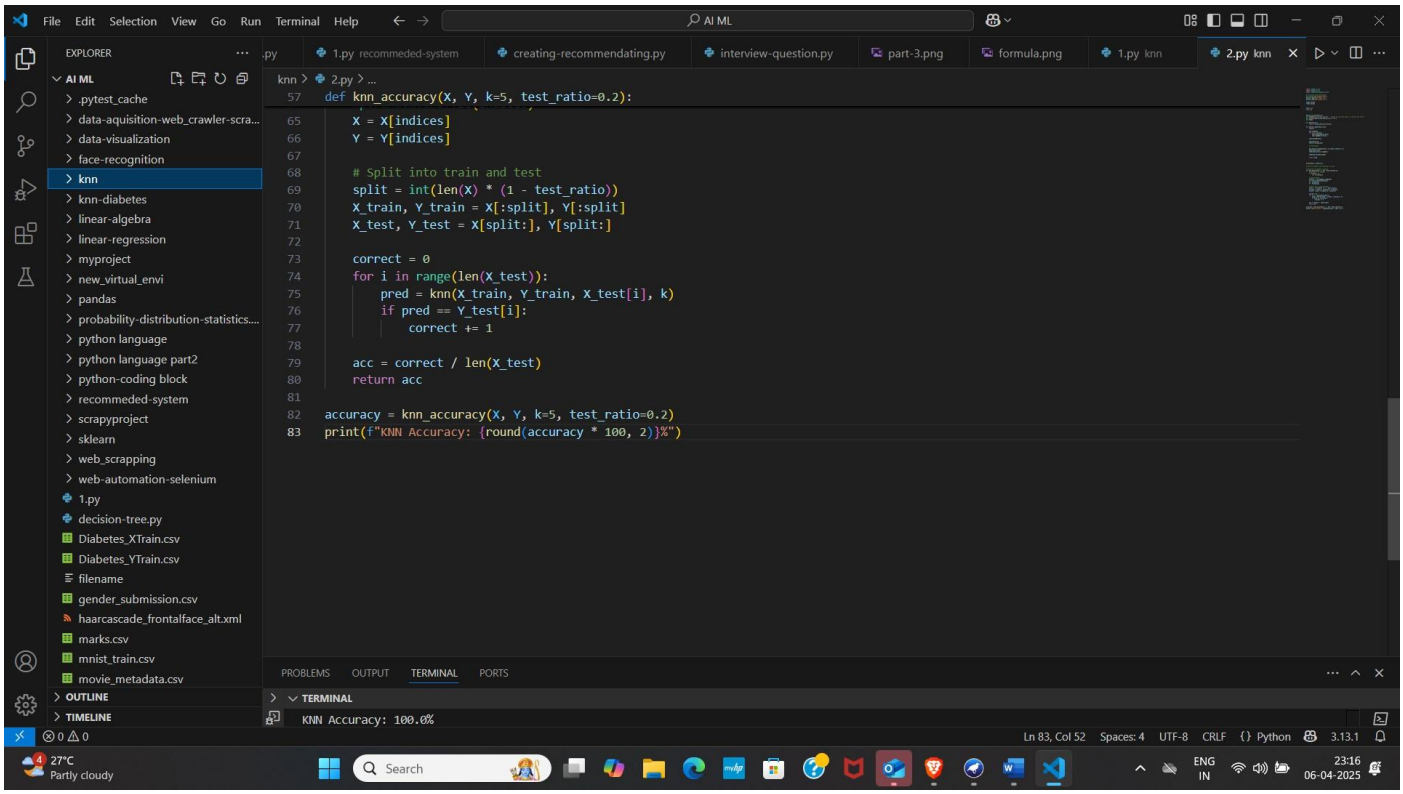
## 2)Python code for implementing k-nearest neibours.



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

# print(plt.style.available)
plt.style.use('seaborn-v0_8')
dfx=pd.read_csv('xdata.csv')
dfy=pd.read_csv('ydata.csv')

X=dfx.values
Y=dfy.values


X=X[:,1:]
Y=Y[:,1]



query_x=np.array([3,1])
plt.scatter(X[:,0],X[:,1],c=Y)    # here we can also give c as array like like Y
plt.scatter(query_x[0],query_x[1],c='red')
plt.show()

def dist(x1,x2):
    return np.sqrt(sum((x1-x2)**2))

def knn(X,Y,queryPoint,k=5):
    vals=[]

    m=X.shape[0]
    for i in range(m):
        d=dist(queryPoint,X[i])
        vals.append((d,Y[i]))
```



```
def knn(X,Y,queryPoint,k=5):

    vals=sorted(vals)

    vals=vals[:k]
    vals=np.array(vals)

    # print(vals)

    new_vals=np.unique(vals[:,1],return_counts=True)
    print(new_vals)
    index=new_vals[1].argmax()

    pred=new_vals[0][index]

    return pred



print(knn(X,Y,query_x))

# task to predict the accuracy of a knn

# Function to compute accuracy
def knn_accuracy(X, Y, k=5, test_ratio=0.2):
    # Flatten Y if needed
    if Y.ndim > 1:
        Y = Y.flatten()

    # Shuffle data
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
```


File Edit Selection View Go Run Terminal Help
AI ML
EXPLORER
AI ML
.pytest_cache
data-aquisition-web_crawler-scra...
data-visualization
face-recognition
knn
knn-diabetes
linear-algebra
linear-regression
myproject
new_virtual_envi
pandas
probability-distribution-statistics....
python language
python language part2
python-coding block
recommeded-system
scrapyproject
sklearn
web_scrapping
web-automation-selenium
1.py
decision-tree.py
Diabetes_XTrain.csv
Diabetes_YTrain.csv
filename
gender_submission.csv
haarcascade_frontalface_alt.xml
marks.csv
mnist_train.csv
movie_metadata.csv
OUTLINE
TIMELINE
1.py recommeded-system
creating-recommendating.py
interview-question.py
part-3.png
formula.png
1.py knn
2.py knn
knn > 2.py > ...
def knn_accuracy(X, Y, k=5, test_ratio=0.2):
X = X[indices]
Y = Y[indices]
# Split into train and test
split = int(len(X) * (1 - test_ratio))
X_train, Y_train = X[:split], Y[:split]
X_test, Y_test = X[split:], Y[split:]
correct = 0
for i in range(len(X_test)):
pred = knn(X_train, Y_train, X_test[i], k)
if pred == Y_test[i]:
correct += 1
acc = correct / len(X_test)
return acc
accuracy = knn_accuracy(X, Y, k=5, test_ratio=0.2)
print(f"KNN Accuracy: {round(accuracy * 100, 2)}%")
PROBLEMS OUTPUT TERMINAL PORTS
TERMINAL
KNN Accuracy: 100.0%
Ln 83, Col 52 Spaces: 4 UTF-8 CRLF {} Python 3.13.1
27°C Partly cloudy
Search
ENG IN
23:16 06-04-2025

## Plot

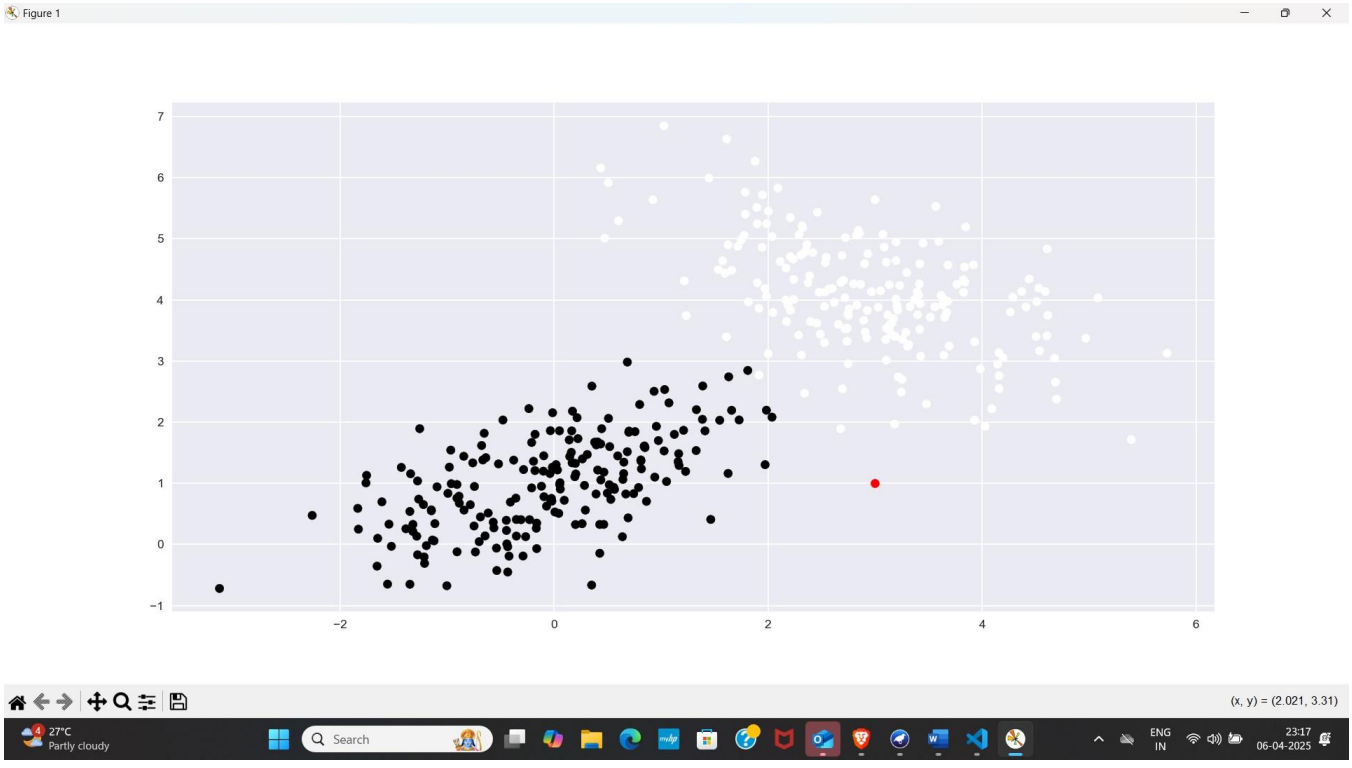


Figure 1
7
6
5
4
3
2
1
0
−1
−2
0
2
4
6
(x, y) = (2.021, 3.31)
27°C Partly cloudy
Search
ENG IN
23:17 06-04-2025

**Result**-:As the dataset is clean and well separated we got an accuracy of 100 %

```
(array([0.]), array([5]))
(array([0.]), array([5]))
KNN Accuracy: 100.0%
PS D:\AI ML>
```

**Applications of KNN Classifier (for Classification)**

1. Medical Diagnosis
   - Classifying diseases based on symptoms or test results.
   - Example: Predicting if a tumor is **benign or malignant**.

**2. Handwriting Recognition**

• Classifying handwritten digits or letters (like in **MNIST dataset**).

**3 Recommendation Systems**

   - Suggesting products based on user similarity (user-based KNN).
   - Example: **"People who bought this also bought..."**

4. Image Classification
   - Classifying images based on pixel similarity.
   - Simple KNN can work well on small, clean datasets.

**5. Spam Detection**

   - Classifying emails as spam or not based on content features.

**6. Music Genre Classification**

   - Classify songs into genres based on audio features.

7. Customer Segmentation

• Grouping users by behavior or preferences for targeted marketing.

# Assignment-6

**1)K-mean Clustering for a dataset after pre-processing using weka/Python libraries.**

**Iris Dataset Summary**

- **Rows:** 150 flower samples
- **Classes:** 3 types of Iris
  - *Setosa*, *Versicolor*, *Virginica*
- **Features:**
  - SepalLengthCm
  - SepalWidthCm
  - PetalLengthCm
  - PetalWidthCm
- **Label Column:** Species (used for comparison only in clustering)
- **Use Case:** Classification & Clustering (e.g., with K-Means)

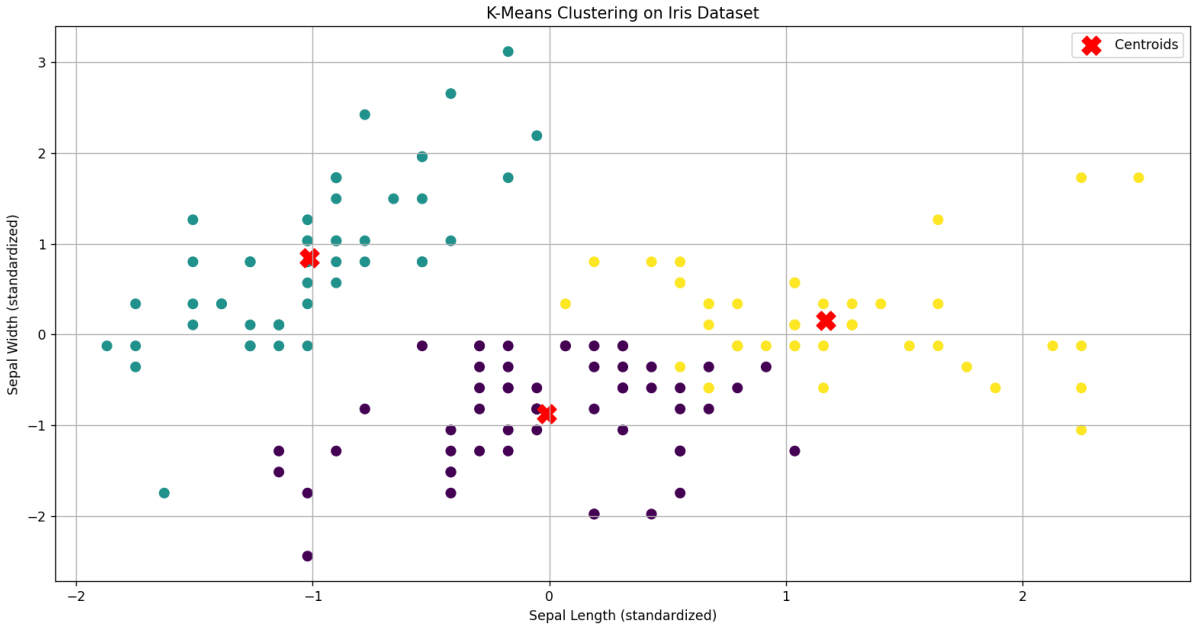
**Python code for K-mean clustering for iris data set using python**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('Iris.csv')
X = df.drop(columns=['Id', 'Species'])

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X = np.array(X_scaled)
```

```
def kmeans(X, k, max_iters=100):
    np.random.seed(42)
    n_samples, n_features = X.shape

    centroids = X[np.random.choice(n_samples, k, replace=False)]

    for _ in range(max_iters):

        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        cluster_labels = np.argmin(distances, axis=1)
        # Step 3: Recompute centroids
        new_centroids = np.array([X[cluster_labels == i].mean(axis=0) for i in range(k)])

        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids

    return cluster_labels, centroids

# Run K-Means with k=3 (for Iris dataset)
# Run K-Means
k = 3
labels, centroids = kmeans(X, k)

# Add cluster labels to the DataFrame
df['Cluster'] = labels

print(df[['SepalLengthCm', 'SepalWidthCm', 'Cluster']].head())
```

```
# Optional: visualize clusters (based on 1st 2 features)
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], color='red', marker='X', s=200, label='Centroids')
plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel('Sepal Length (standardized)')
plt.ylabel('Sepal Width (standardized)')
plt.legend()
plt.grid(True)
plt.show()
```

## Plot




**Result-:**

| | SepalLengthCm | SepalWidthCm | Cluster |
|---|---|---|---|
| 0 | 5.1 | 3.5 | 1 |
| 1 | 4.9 | 3.0 | 1 |
| 2 | 4.7 | 3.2 | 1 |
| 3 | 4.6 | 3.1 | 1 |
| 4 | 5.0 | 3.6 | 1 |

**Top Applications of K-Means Clustering**

1. **Customer Segmentation**
   - **Goal:** Group customers based on purchasing behavior, demographics, or preferences.
   - **Example:** E-commerce platforms use K-Means to segment users for targeted marketing.

2. **Market Basket Analysis / Product Recommendation**
   - **Goal:** Group products that are often bought together or group users with similar purchase patterns.
   - **Example:** Amazon or Netflix uses it to recommend products/movies.

3. **Document or Text Clustering**
   - **Goal:** Automatically group similar documents or articles.
   - **Example:** News aggregation, topic modeling, grouping tweets, etc.

4. **Image Compression / Segmentation**
   - **Goal:** Reduce colors in an image by clustering similar color pixels.
   - **Example:** Use K-Means to replace similar colors with their centroid → smaller file size with preserved visual quality.

5. **Education Analytics**
   - **Goal:** Group students based on performance, engagement, or learning styles.

# Assignment-7

## Title: Agglomerative (Bottom-Up) Clustering for a Dataset After Preprocessing using Python Libraries

**Objective:** To implement and evaluate Agglomerative (bottom-up) clustering on a preprocessed dataset using Python libraries. The experiment involves data pre-processing followed by the application of Agglomerative clustering to identify natural groupings within the dataset.
**Dataset:** new.csv

**Description:** The dataset relates to marketing campaigns of a Portuguese banking institution, consisting of demographic and contact attributes along with campaign outcomes.

**Tools used-**: Python 3.x

**Libraries used-:**

Pandas, numpy , matplotlib, seaborn, sklearn (StandardScaler, LabelEncoder, AgglomerativeClustering) , scipy (linkage, dendrogram) .

**Code**

```
# Importing libraries import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelEncoder from sklearn.cluster import AgglomerativeClustering from scipy.cluster.hierarchy import dendrogram, linkage


# Load the dataset
```


```
# Importing libraries import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelEncoder from sklearn.cluster import AgglomerativeClustering from scipy.cluster.hierarchy import dendrogram, linkage


# Load the dataset
```

df = pd.read_csv( '/mnt/data/bank-full - bank-full.csv.csv' , sep= ';' )

Tools Used:

Procedure:

# Display basic information print (df.info()) print (df.head())

# Pre-processing

# Encode categorical variables

label_encoders = {} for column in df.select_dtypes(include=[ 'object' ]).columns:

le = LabelEncoder() df[column] = le.fit_transform(df[column]) label_encoders[column] = le

# Feature scaling scaler = StandardScaler() df_scaled = scaler.fit_transform(df)

# Create Linkage matrix for dendrogram linked = linkage(df_scaled, method= 'ward' )

# Plot dendrogram

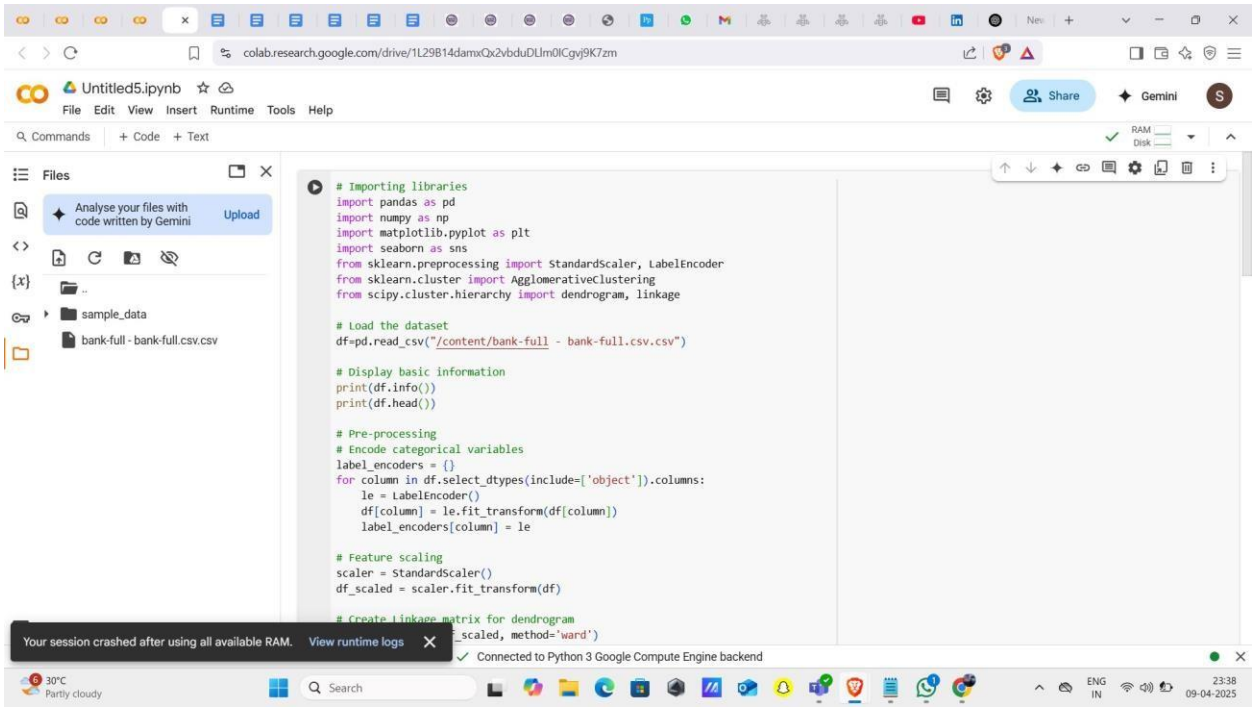
plt.figure(figsize=( 12 , 6 )) dendrogram(linked, truncate_mode= 'level' , p= 5 ) plt.title( 'Dendrogram for Hierarchical Clustering' ) plt.xlabel( 'Data points' ) plt.ylabel( 'Euclidean distance' ) plt.show()

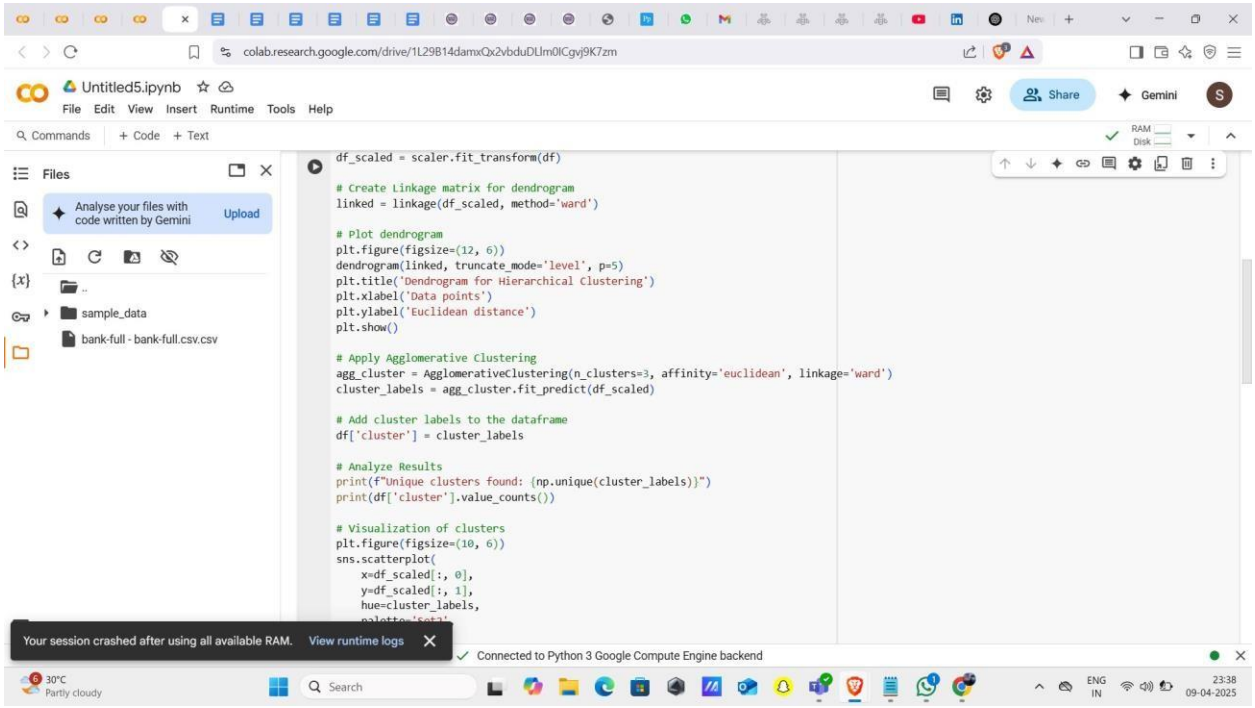
# Apply Agglomerative Clustering agg_cluster = AgglomerativeClustering(n_clusters= 3 , affinity= 'euclidean' , linkage= 'ward' ) cluster_labels = agg_cluster.fit_predict(df_scaled)

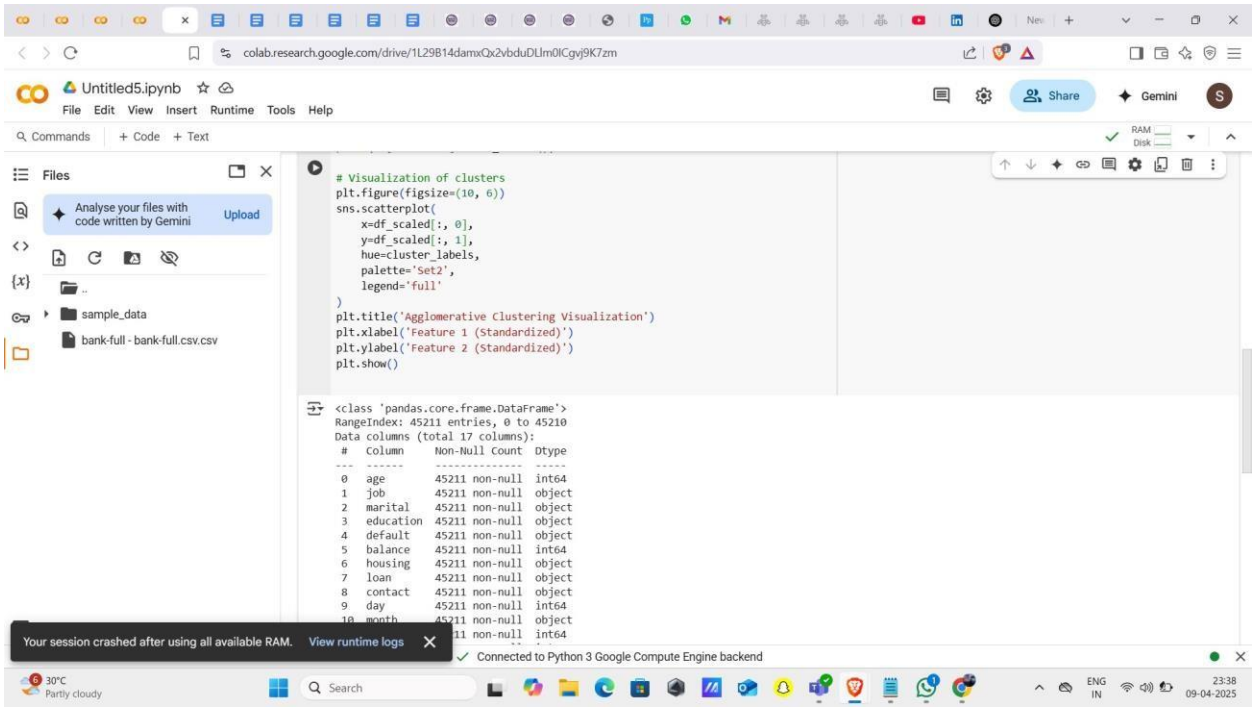
# Add cluster labels to the dataframe df[ 'cluster' ] = cluster_labels

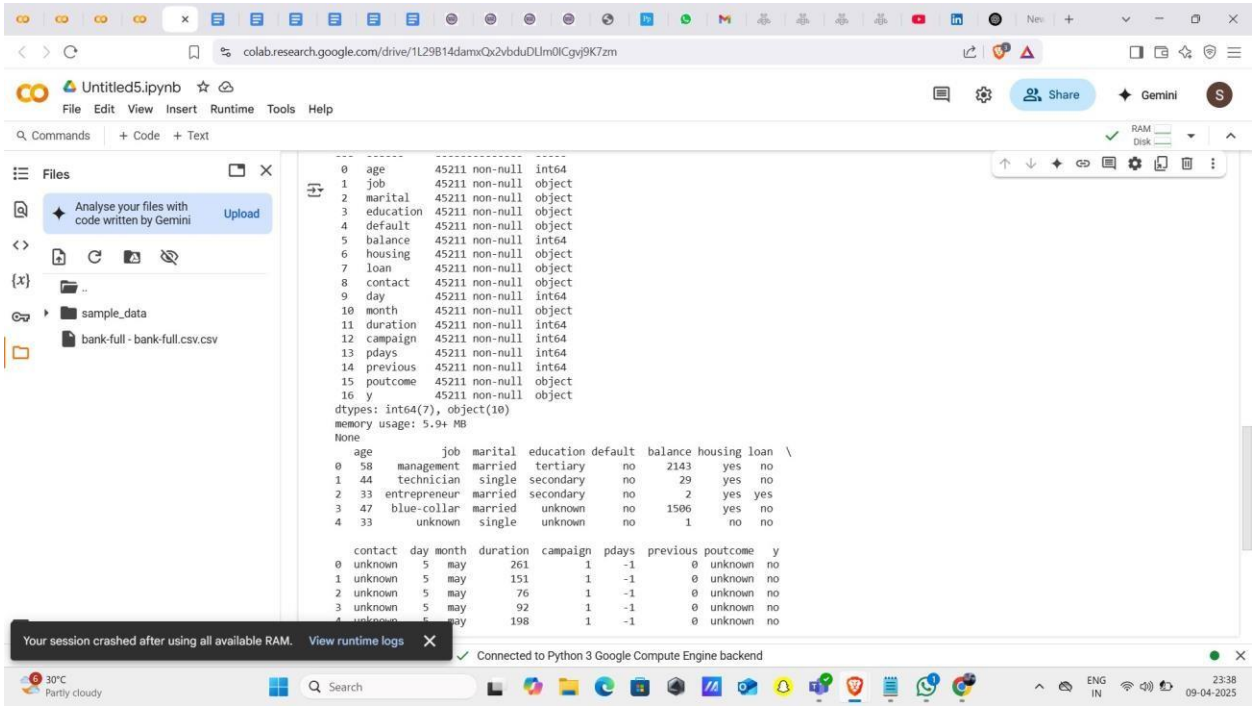
# Analyze Results print ( f "Unique clusters found: {np.unique(cluster_labels)} " ) print (df[ 'cluster' ].value_counts())

```
# Visualization of clusters

plt.figure(figsize=( 10 ,  6 ))

sns.scatterplot(   x=df_scaled[:,  0 ],

y=df_scaled[:,  1 ],   hue=cluster_labels,   palette= 'Set2' ,   legend= 'full'

)

plt.title( 'Agglomerative Clustering Visualization' )   plt.xlabel( 'Feature 1 (Standardized)' )   plt.ylabel(
'Feature 2 (Standardized)' )

plt.show()
```


colab.research.google.com/drive/1L29B14damxQx2vbduDLlm0ICgvj9K7zm
Untitled5.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
# Load the dataset
df=pd.read_csv("/content/bank-full - bank-full.csv.csv")
# Display basic information
print(df.info())
print(df.head())
# Pre-processing
# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
le = LabelEncoder()
df[column] = le.fit_transform(df[column])
label_encoders[column] = le
# Feature scaling
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
# Create Linkage matrix for dendrogram
_scaled, method='ward')
Your session crashed after using all available RAM. View runtime logs
Connected to Python 3 Google Compute Engine backend
30°C Partly cloudy
Search
ENG IN
23:38 09-04-2025


colab.research.google.com/drive/1L29B14damxQx2vbduDLlm0ICgvj9K7zm
Untitled5.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv
df_scaled = scaler.fit_transform(df)
# Create Linkage matrix for dendrogram
linked = linkage(df_scaled, method='ward')
# Plot dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linked, truncate_mode='level', p=5)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Data points')
plt.ylabel('Euclidean distance')
plt.show()
# Apply Agglomerative Clustering
agg_cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
cluster_labels = agg_cluster.fit_predict(df_scaled)
# Add cluster labels to the dataframe
df['cluster'] = cluster_labels
# Analyze Results
print(f"Unique clusters found: {np.unique(cluster_labels)}")
print(df['cluster'].value_counts())
# Visualization of clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
x=df_scaled[:, 0],
y=df_scaled[:, 1],
hue=cluster_labels,
Your session crashed after using all available RAM. View runtime logs
Connected to Python 3 Google Compute Engine backend
30°C Partly cloudy
Search
ENG IN
23:38 09-04-2025


colab.research.google.com/drive/1L29B14damxQx2vbduDLm0ICgvj9K7zm
Untitled5.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv
# Visualization of clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
x=df_scaled[:, 0],
y=df_scaled[:, 1],
hue=cluster_labels,
palette='Set2',
legend='full'
)
plt.title('Agglomerative Clustering Visualization')
plt.xlabel('Feature 1 (Standardized)')
plt.ylabel('Feature 2 (Standardized)')
plt.show()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
# Column Non-Null Count Dtype
0 age 45211 non-null int64
1 job 45211 non-null object
2 marital 45211 non-null object
3 education 45211 non-null object
4 default 45211 non-null object
5 balance 45211 non-null int64
6 housing 45211 non-null object
7 loan 45211 non-null object
8 contact 45211 non-null object
9 day 45211 non-null int64
Your session crashed after using all available RAM. View runtime logs
Connected to Python 3 Google Compute Engine backend
30°C Partly cloudy
Search
ENG IN
23:38 09-04-2025


colab.research.google.com/drive/1L29B14damxQx2vbduDLm0ICgvj9K7zm
Untitled5.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv
0 age 45211 non-null int64
1 job 45211 non-null object
2 marital 45211 non-null object
3 education 45211 non-null object
4 default 45211 non-null object
5 balance 45211 non-null int64
6 housing 45211 non-null object
7 loan 45211 non-null object
8 contact 45211 non-null object
9 day 45211 non-null int64
10 month 45211 non-null object
11 duration 45211 non-null int64
12 campaign 45211 non-null int64
13 pdays 45211 non-null int64
14 previous 45211 non-null int64
15 poutcome 45211 non-null object
16 y 45211 non-null object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
None
age job marital education default balance housing loan \
0 58 management married tertiary no 2143 yes no
1 44 technician single secondary no 29 yes no
2 33 entrepreneur married secondary no 2 yes yes
3 47 blue-collar married unknown no 1506 yes no
4 33 unknown single unknown no 1 no no
contact day month duration campaign pdays previous poutcome y
0 unknown 5 may 261 1 -1 0 unknown no
1 unknown 5 may 151 1 -1 0 unknown no
2 unknown 5 may 76 1 -1 0 unknown no
3 unknown 5 may 92 1 -1 0 unknown no
Your session crashed after using all available RAM. View runtime logs
Connected to Python 3 Google Compute Engine backend
30°C Partly cloudy
Search
ENG IN
23:38 09-04-2025

# Assignment-8

## Objective

To implement and evaluate DBSCAN clustering on a pre-processed dataset using Python libraries. The experiment involves data pre-processing steps followed by the application of DBSCAN to identify clusters and noise

## Dataset Description

**Name**: bank-full.csv

**Description**: Dataset related to direct marketing campaigns of a Portuguese banking institution. It includes attributes such as age, job, marital status, education, default status, housing loan, and personal loan.

**Code**

```
# Load the dataset
df = pd.read_csv( '/mnt/data/bank-full - bank-full.csv.csv' ,  sep= ';' )

# Display basic information
print (df.info())

# Display first few rows
print (df.head())
```

Procedure

Step 1: Import Required Libraries

```
import  pandas  as  pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import DBSCAN
```

Step 2: Load and Explore Dataset

Step 3: Pre-processing the Data

Encoding categorical variables using Label Encoding.

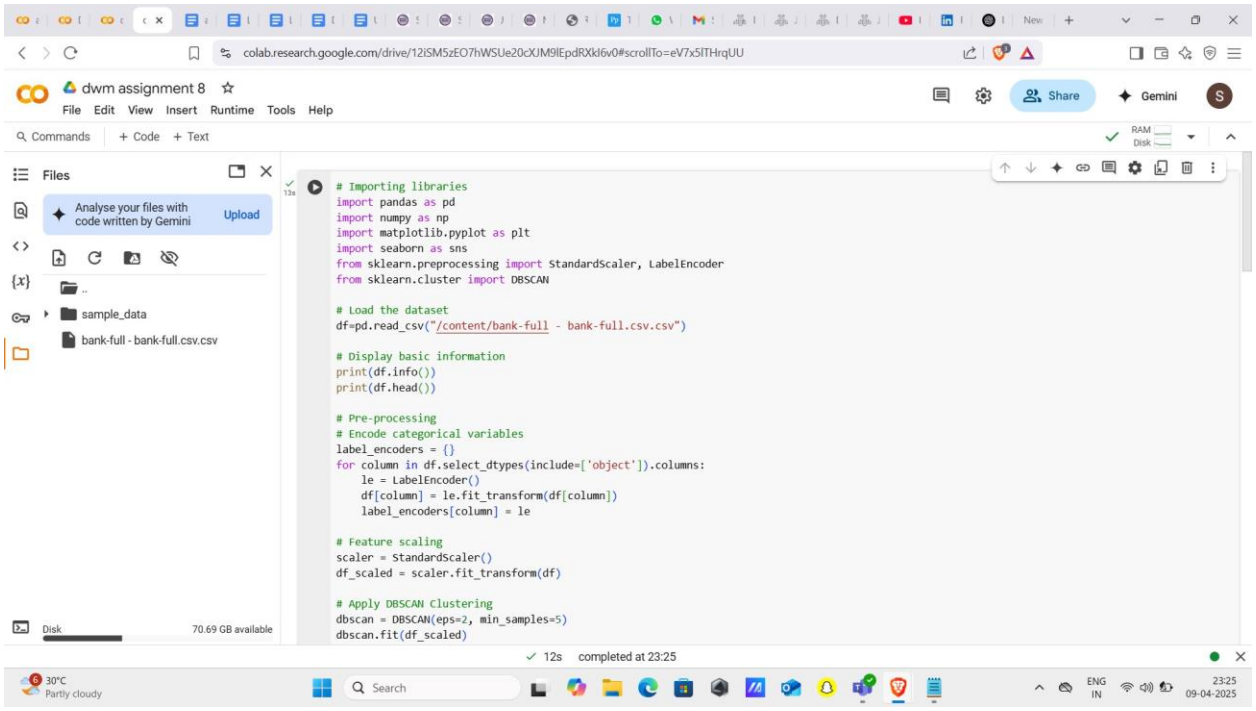
Scaling the features using StandardScaler

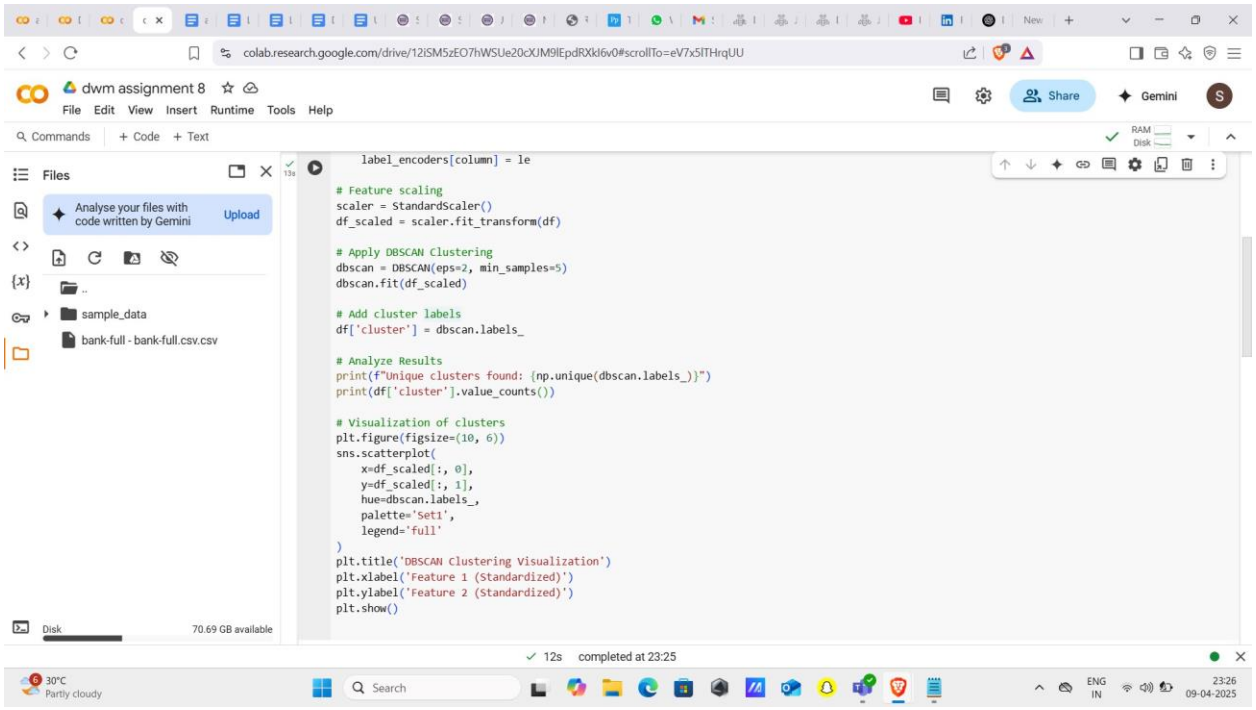
```
# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include=[ 'object' ]).columns:
le = LabelEncoder()
df[column] = le.fit_transform(df[column])
label_encoders[column] = le

# Feature scaling
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

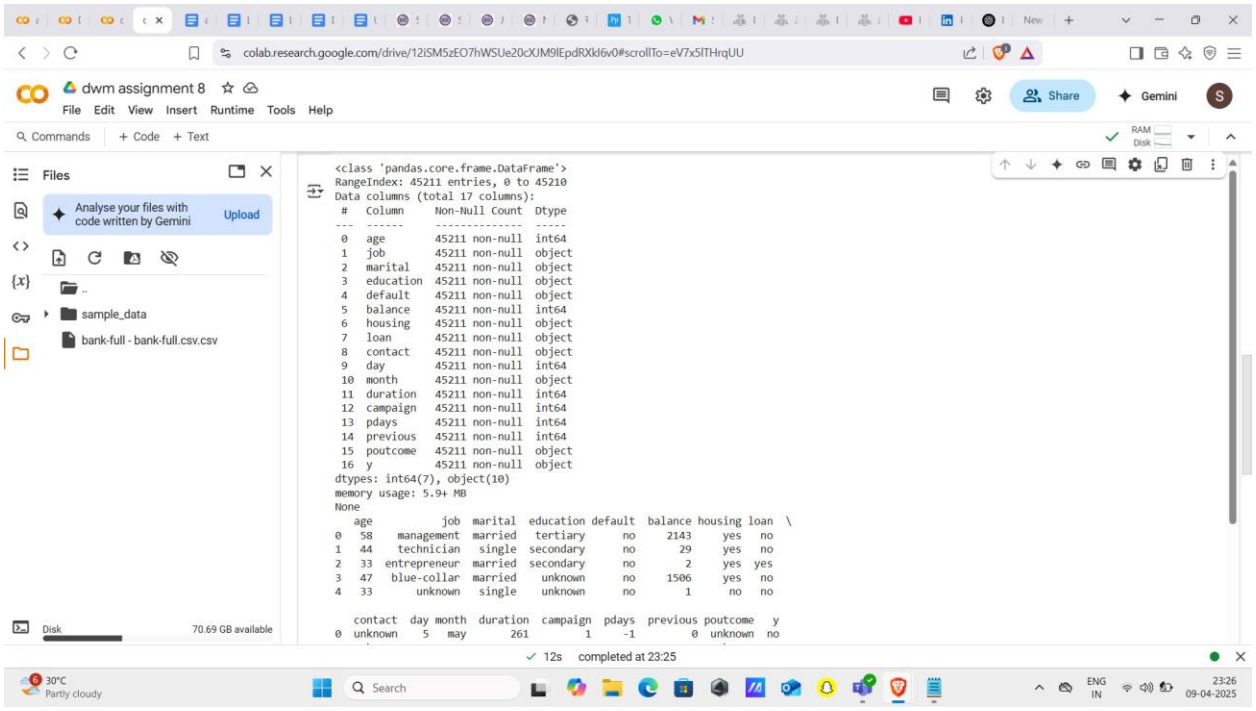
Step 4: Apply DBSCAN Clustering

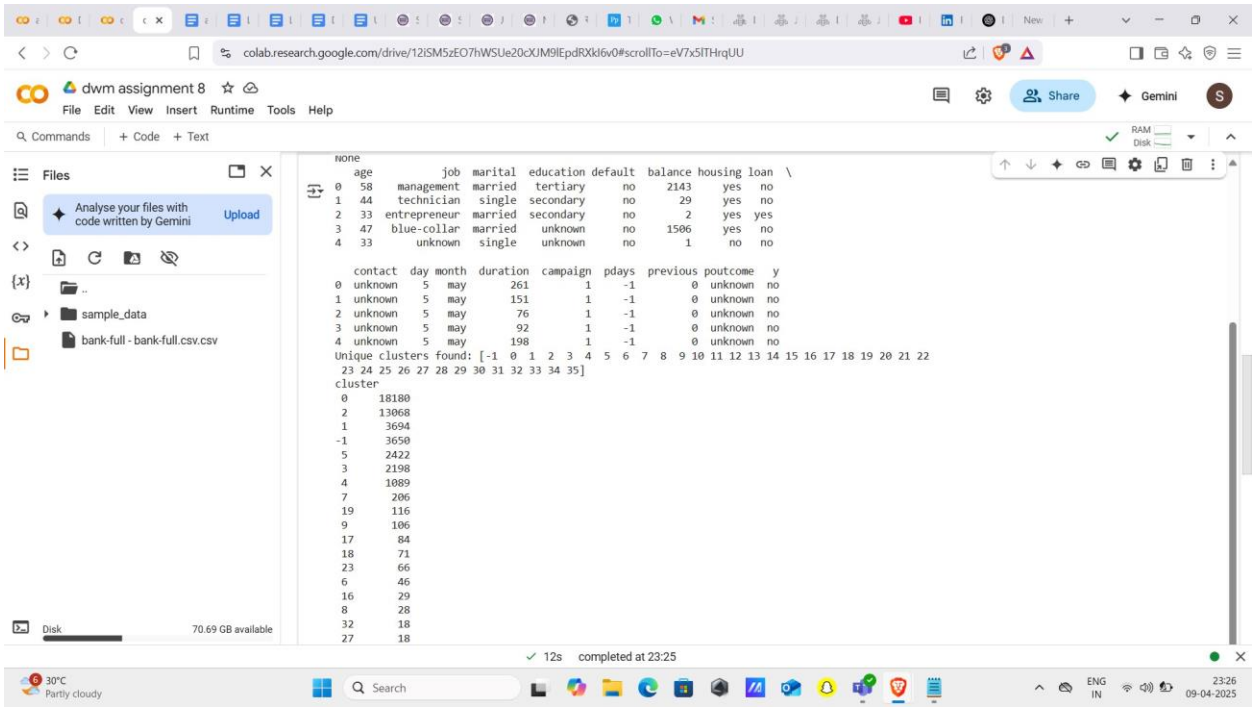
```
# Applying DBSCAN algorithm
dbscan = DBSCAN(eps= 2 , min_samples= 5 )
dbscan.fit(df_scaled)
```

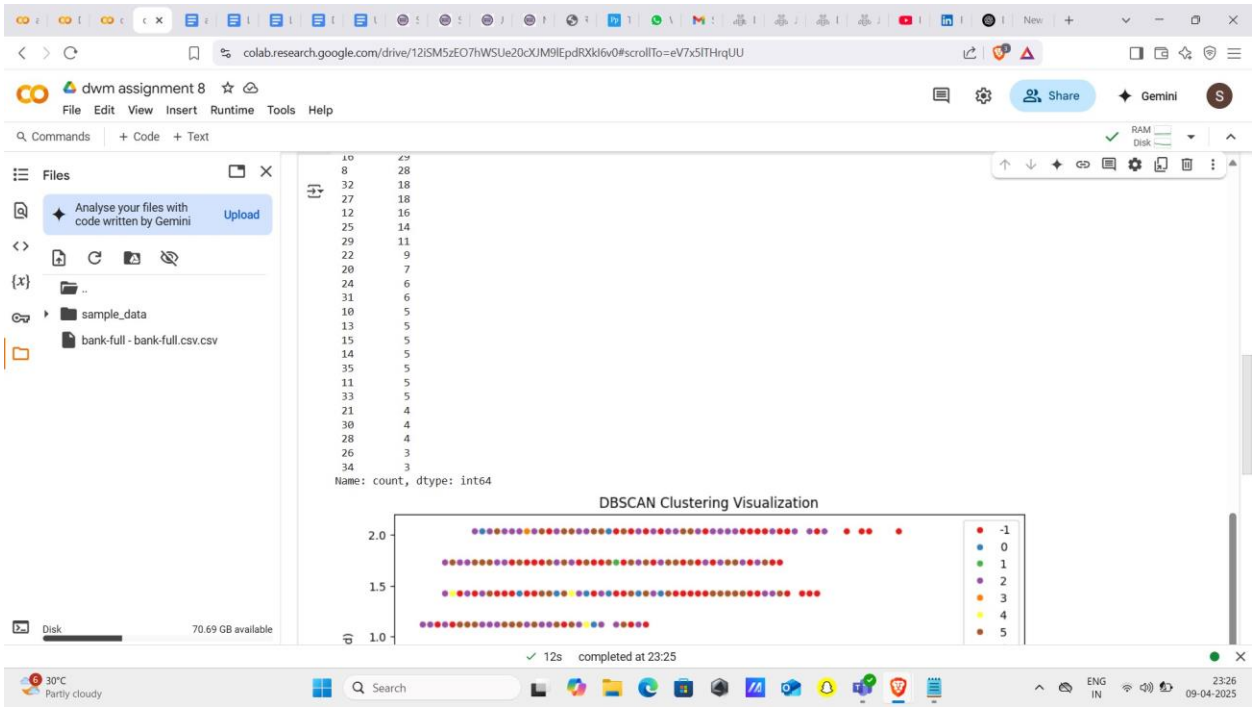
```
# Assigning cluster labels
df[ 'cluster' ] = dbscan.labels_

Step 5: Analyze and Visualize the Results
# Unique clusters
print ( f "Unique clusters found:  {np.unique(dbscan.labels_)} " )

# Count of points in each cluster
print (df[ 'cluster' ].value_counts())

# Visualization of clusters
plt.figure(figsize=( 10 ,  6 ))
sns.scatterplot(
x=df_scaled[:,  0 ],
y=df_scaled[:,  1 ],
hue=dbscan.labels_,

palette= 'Set1' ,
legend= 'full'
)
plt.title( 'DBSCAN Clustering Visualization' )
plt.xlabel( 'Feature 1 (Standardized)' )
plt.ylabel( 'Feature 2 (Standardized)' )
plt.show()
```


colab.research.google.com/drive/12iSM5zEO7hWSUe20cXJM9lEpdRXkl6v0#scrollTo=eV7x5ITHrqUU
dwm assignment 8
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv.csv
Disk 70.69 GB available
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import DBSCAN
# Load the dataset
df=pd.read_csv("/content/bank-full - bank-full.csv.csv")
# Display basic information
print(df.info())
print(df.head())
# Pre-processing
# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
le = LabelEncoder()
df[column] = le.fit_transform(df[column])
label_encoders[column] = le
# Feature scaling
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
# Apply DBSCAN Clustering
dbscan = DBSCAN(eps=2, min_samples=5)
dbscan.fit(df_scaled)
12s completed at 23:25
30°C Partly cloudy
Search
ENG IN
23:25 09-04-2025

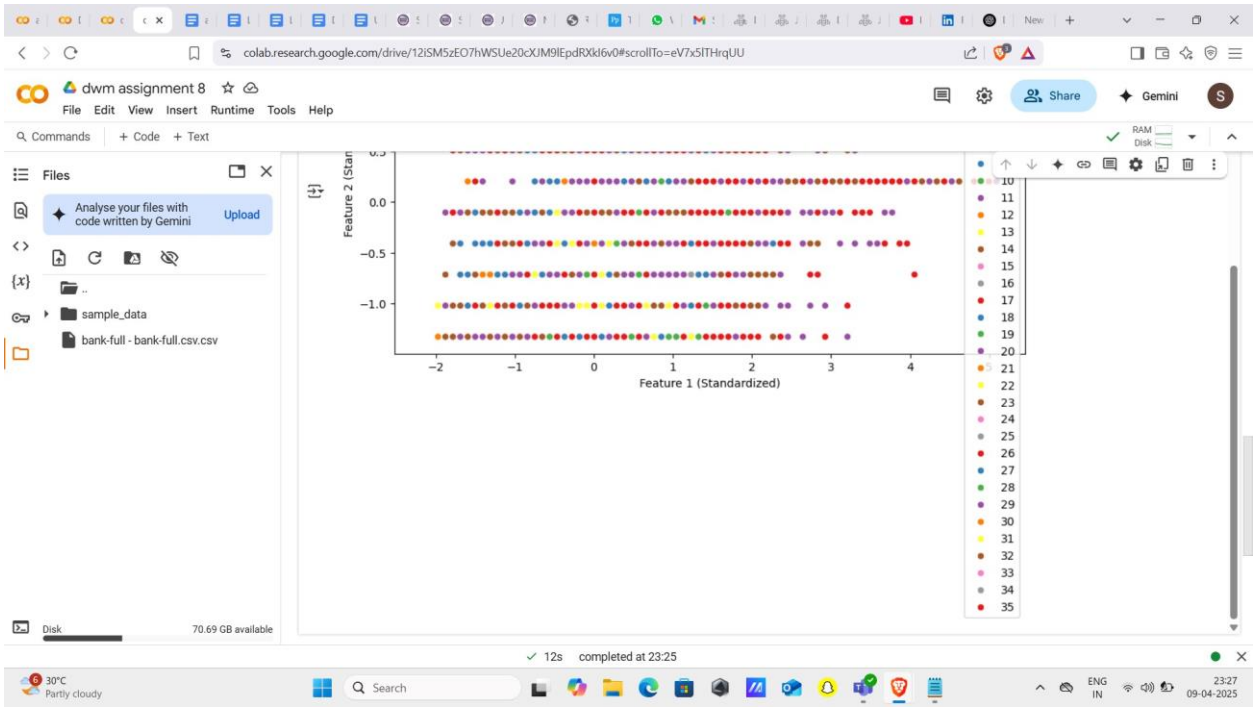

colab.research.google.com/drive/12iSM5zEO7hWSUe20cXJM9lEpdRXkl6v0#scrollTo=eV7x5ITHrqUU
dwm assignment 8
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv.csv
Disk 70.69 GB available
label_encoders[column] = le
# Feature scaling
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
# Apply DBSCAN Clustering
dbscan = DBSCAN(eps=2, min_samples=5)
dbscan.fit(df_scaled)
# Add cluster labels
df['cluster'] = dbscan.labels_
# Analyze Results
print(f"Unique clusters found: {np.unique(dbscan.labels_)}")
print(df['cluster'].value_counts())
# Visualization of clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
x=df_scaled[:, 0],
y=df_scaled[:, 1],
hue=dbscan.labels_,
palette='Set1',
legend='full'
)
plt.title('DBSCAN Clustering Visualization')
plt.xlabel('Feature 1 (Standardized)')
plt.ylabel('Feature 2 (Standardized)')
plt.show()
12s completed at 23:25
30°C Partly cloudy
Search
ENG IN
23:26 09-04-2025


colab.research.google.com/drive/12iSM5zEO7hWSUe20cXJM9lEpdRXkl6v0#scrollTo=eV7x5lTHrqUU
dwm assignment 8
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv.csv
Disk 70.69 GB available
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
# Column Non-Null Count Dtype
0 age 45211 non-null int64
1 job 45211 non-null object
2 marital 45211 non-null object
3 education 45211 non-null object
4 default 45211 non-null object
5 balance 45211 non-null int64
6 housing 45211 non-null object
7 loan 45211 non-null object
8 contact 45211 non-null object
9 day 45211 non-null int64
10 month 45211 non-null object
11 duration 45211 non-null int64
12 campaign 45211 non-null int64
13 pdays 45211 non-null int64
14 previous 45211 non-null int64
15 poutcome 45211 non-null object
16 y 45211 non-null object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
None
age job marital education default balance housing loan \
0 58 management married tertiary no 2143 yes no
1 44 technician single secondary no 29 yes no
2 33 entrepreneur married secondary no 2 yes yes
3 47 blue-collar married unknown no 1506 yes no
4 33 unknown single unknown no 1 no no
contact day month duration campaign pdays previous poutcome y
0 unknown 5 may 261 1 -1 0 unknown no
12s completed at 23:25
30°C Partly cloudy
Search
ENG IN
23:26 09-04-2025


colab.research.google.com/drive/12iSM5zEO7hWSUe20cXJM9lEpdRXkl6v0#scrollTo=eV7x5lTHrqUU
dwm assignment 8
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv.csv
Disk 70.69 GB available
None
age job marital education default balance housing loan \
0 58 management married tertiary no 2143 yes no
1 44 technician single secondary no 29 yes no
2 33 entrepreneur married secondary no 2 yes yes
3 47 blue-collar married unknown no 1506 yes no
4 33 unknown single unknown no 1 no no
contact day month duration campaign pdays previous poutcome y
0 unknown 5 may 261 1 -1 0 unknown no
1 unknown 5 may 151 1 -1 0 unknown no
2 unknown 5 may 76 1 -1 0 unknown no
3 unknown 5 may 92 1 -1 0 unknown no
4 unknown 5 may 198 1 -1 0 unknown no
Unique clusters found: [-1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35]
cluster
0 18180
2 13068
1 3694
-1 3650
5 2422
3 2198
4 1089
7 206
19 116
9 106
17 84
18 71
23 66
6 46
16 29
8 28
32 18
27 18
12s completed at 23:25
30°C Partly cloudy
Search
ENG IN
23:26 09-04-2025


colab.research.google.com/drive/12iSM5zEO7hWSUe20cXJM9lEpdRXkl6v0#scrollTo=eV7x5ITHrqUU
dwm assignment 8
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv.csv
Disk 70.69 GB available
8 28
32 18
27 18
12 16
25 14
29 11
22 9
20 7
24 6
31 6
10 5
13 5
15 5
14 5
35 5
11 5
33 5
21 4
30 4
28 4
26 3
34 3
Name: count, dtype: int64
DBSCAN Clustering Visualization
2.0
1.5
1.0
-1
0
1
2
3
4
5
12s completed at 23:25
30°C Partly cloudy
Search
ENG IN
23:26 09-04-2025


colab.research.google.com/drive/12iSM5zEO7hWSUe20cXJM9lEpdRXkl6v0#scrollTo=eV7x5ITHrqUU
dwm assignment 8
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
Files
Analyse your files with code written by Gemini
Upload
sample_data
bank-full - bank-full.csv.csv
Disk 70.69 GB available
Name: count, dtype: int64
DBSCAN Clustering Visualization
Feature 2 (Standardized)
Feature 1 (Standardized)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2
3
4
-1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
12s completed at 23:25
30°C Partly cloudy
Search
ENG IN
23:27 09-04-2025




## Observations

DBSCAN divided the data into clusters and identified noise points.

Cluster -1 corresponds to noise or outliers.

DBSCAN does not require specifying the number of clusters beforehand.

Clustering results are sensitive to the parameters eps (neighborhood size) and min_samples

# Assignment-9

**1)Apriori algorithm for association rule mining using Python libraries.**

**Dataset Overview**

- Format: CSV (Comma-Separated Values)
- Rows: 1,4965 (based on the visible data)
- Columns: 11 (labeled 0 through 10)
- Data Type: Categorical (item names as strings)
- Purpose: Likely used for market basket analysis, such as identifying purchasing patterns, frequent itemsets, or association rules (e.g., "if a customer buys X, they are likely to buy Y").

**Python code for Apriori algorithm for association rule mining.**

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Step 1: Load the dataset
try:
   data = pd.read_csv('basket.csv', header=None)
   print(f"Dataset loaded successfully. Shape: {data.shape}")
except FileNotFoundError:
   print("Error: 'basket.csv' not found. Please ensure the file is in the correct directory.")
   exit(1)

# Step 2: Preprocess the data
# Convert the dataset into a list of transactions
transactions = []
for i in range(len(data)):
   # Extract non-null items from each row
   transaction = [item for item in data.iloc[i] if pd.notna(item)]
   transactions.append(transaction)

# Get all unique items
all_items = sorted(set(item for transaction in transactions for item in transaction))
print(f"Number of unique items: {len(all_items)}")
print(f"Number of transactions: {len(transactions)}")

# Create a one-hot encoded DataFrame
one_hot_data = pd.DataFrame(0, index=range(len(transactions)), columns=all_items)
for i, transaction in enumerate(transactions):
   for item in transaction:
      one_hot_data.at[i, item] = 1
```

```
# Print a sample of the one-hot encoded data
print("\nSample of one-hot encoded data (first 5 rows):")
print(one_hot_data.head())

# Step 3: Apply the Apriori algorithm
# Lower the minimum support to 1% (adjust as needed)
min_support = 0.01  # Changed from 0.75 to 0.01
frequent_itemsets = apriori(one_hot_data, min_support=min_support, use_colnames=True)

# Check if frequent itemsets were found
if frequent_itemsets.empty:
    print(f"\nNo frequent itemsets found with min_support={min_support}. Try lowering the support
threshold.")
else:
    print("\nFrequent Itemsets:")
    print(frequent_itemsets)

# Step 4: Generate association rules (only if frequent itemsets exist)
min_confidence = 0.1
if not frequent_itemsets.empty:
    rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=min_confidence)

    # Step 5: Evaluate and display the results
    print("\nAssociation Rules:")
    if rules.empty:
        print(f"No association rules found with min_confidence={min_confidence}.")
    else:
        print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])

        # Save results to CSV files
        frequent_itemsets.to_csv('frequent_itemsets.csv', index=False)
        rules.to_csv('association_rules.csv', index=False)

        # Basic evaluation metrics
        print("\nNumber of frequent itemsets:", len(frequent_itemsets))
        print("Number of association rules:", len(rules))
        print("\nTop 5 rules sorted by lift:")
        print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].sort_values(by='lift',
ascending=False).head())
else:
    print("\nSkipping association rules generation due to empty frequent itemsets.")
```

Result-:

->With 75 % support and 85% confidence we can’t get any rule

```
Dataset loaded successfully. Shape: (14964, 11)
Number of unique items: 178
Number of transactions: 14964

Sample of one-hot encoded data (first 5 rows):
   0  1  10  2  3  4  5  6  7  8  9  ...  tropical fruit  turkey  vinegar  waffles  whipped/sour cream  whisky  white bread  white wine  whole milk  yogurt  zwieback
0  1  1   1  1  1  1  1  1  1  1  1  ...              0       0        0        0                   0       0            0           0       0         0
1  0  0   0  0  0  0  0  0  0  0  0  ...              0       0        0        0                   0       0            0           0       1         0
2  0  0   0  0  0  0  0  0  0  0  0  ...              0       0        0        0                   0       0            0           0       1         1
3  0  0   0  0  0  0  0  0  0  0  0  ...              0       0        0        0                   0       0            0           0       0         0
4  0  0   0  0  0  0  0  0  0  0  0  ...              0       0        0        0                   0       0            0           0       0         0

[5 rows x 178 columns]
C:\Users\labhg\AppData\Roaming\Python\Python313\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperform
ance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

No frequent itemsets found with min_support=0.75. Try lowering the support threshold.

Skipping association rules generation due to empty frequent itemsets.
PS D:\AI ML>
```

->Result on lowering the support and confidence we get.

```
C:\Users\labhg\AppData\Roaming\Python\Python313\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool types result in worse computationalperform
ance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(

Frequent Itemsets:
     support                      itemsets
0   0.021385                    (UHT-milk)
1   0.033948                       (beef)
2   0.021786                    (berries)
3   0.016573                   (beverages)
4   0.045309                (bottled beer)
..       ...                         ...
64  0.010559  (other vegetables, rolls/buns)
65  0.014836  (other vegetables, whole milk)
66  0.013967        (rolls/buns, whole milk)
67  0.011628              (soda, whole milk)
68  0.011160            (yogurt, whole milk)

[69 rows x 2 columns]

Association Rules:
         antecedents   consequents   support  confidence      lift
0  (other vegetables)  (whole milk)  0.014836    0.121511  0.769482
1        (rolls/buns)  (whole milk)  0.013967    0.126974  0.804082
2              (soda)  (whole milk)  0.011628    0.119752  0.758346
3            (yogurt)  (whole milk)  0.011160    0.129961  0.822995
```

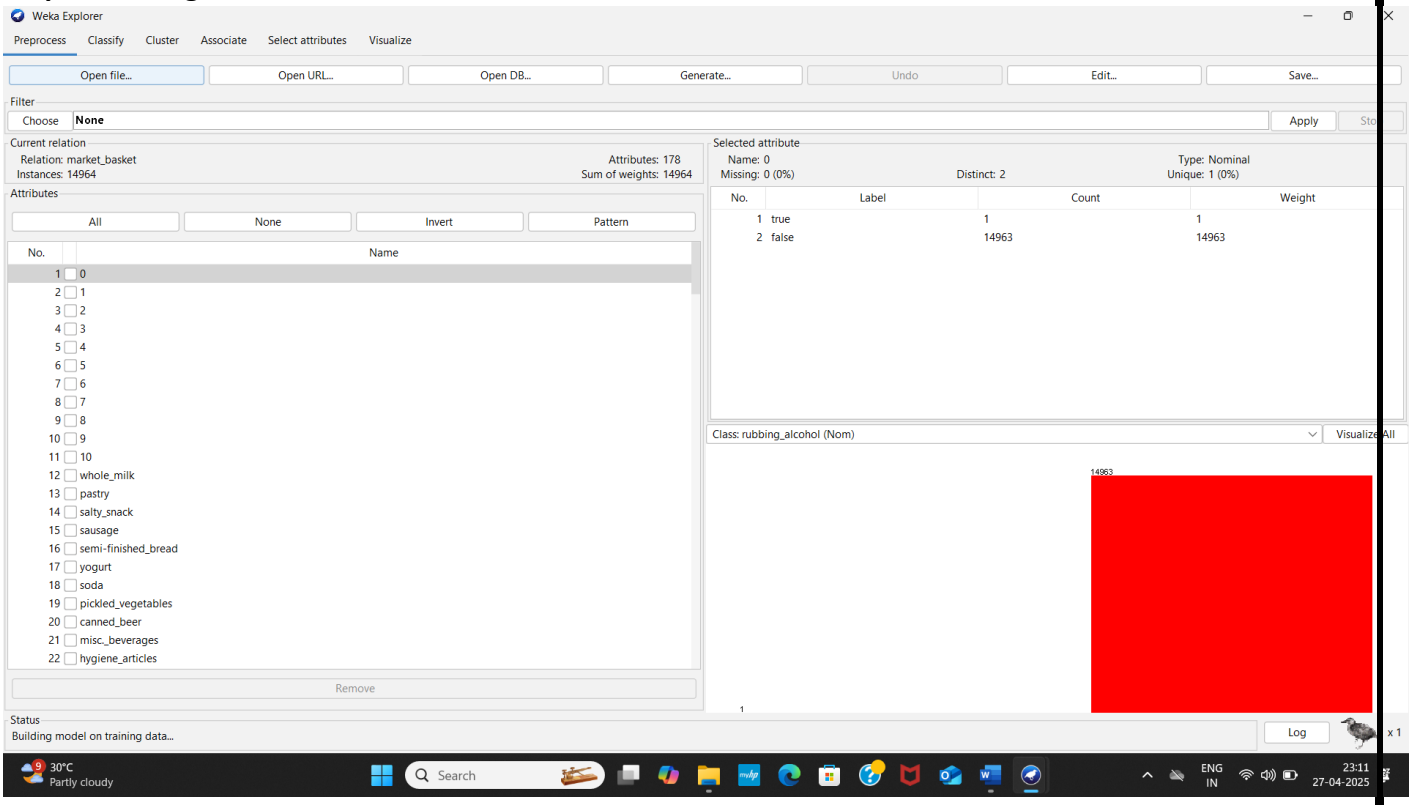
**2) Apriori algorithm for association rule mining using weka.**

The basket.csv dataset is in a transactional format (items listed in columns per transaction), but Weka's Apriori algorithm requires data in a specific format, typically **ARFF (Attribute-Relation File Format)** with binary attributes (1 for item presence, 0 for absence) or a sparse format.so we need to preprocess it.

**Python code for converting basket.csv to basket.arff.**

```
import pandas as pd
from collections import defaultdict

# Load the dataset
df = pd.read_csv('basket.csv', header=None)

# Get all unique items
items = df.values.flatten()
unique_items = pd.Series(items).dropna().unique()

# Create a binary matrix (1 if item is present, 0 otherwise)
binary_data = pd.DataFrame(0, index=df.index, columns=unique_items)
for idx, row in df.iterrows():
    for item in row.dropna():
        binary_data.loc[idx, item] = 1

# Save to CSV (intermediate step)
binary_data.to_csv('basket_binary.csv', index=False)

# Convert to ARFF
arff_content = f"""@RELATION market_basket

"""
for item in unique_items:
    arff_content += f"@ATTRIBUTE {item.replace(' ', '_')} {{true, false}}\n"

arff_content += "\n@DATA\n"
for _, row in binary_data.iterrows():
    row_values = ['true' if val == 1 else 'false' for val in row]
    arff_content += ','.join(row_values) + '\n'

# Write to ARFF file
with open('basket.arff', 'w') as f:
    f.write(arff_content)

print("ARFF file 'basket.arff' created.")
```

**Step-2 Loading the dataset into weka**




**Step-3 Switch to the Associate Tab**:

- In Weka Explorer, click the “Associate” tab.

**Select Apriori**:

- Click the “Choose” button and select weka.associations.Apriori from the list.

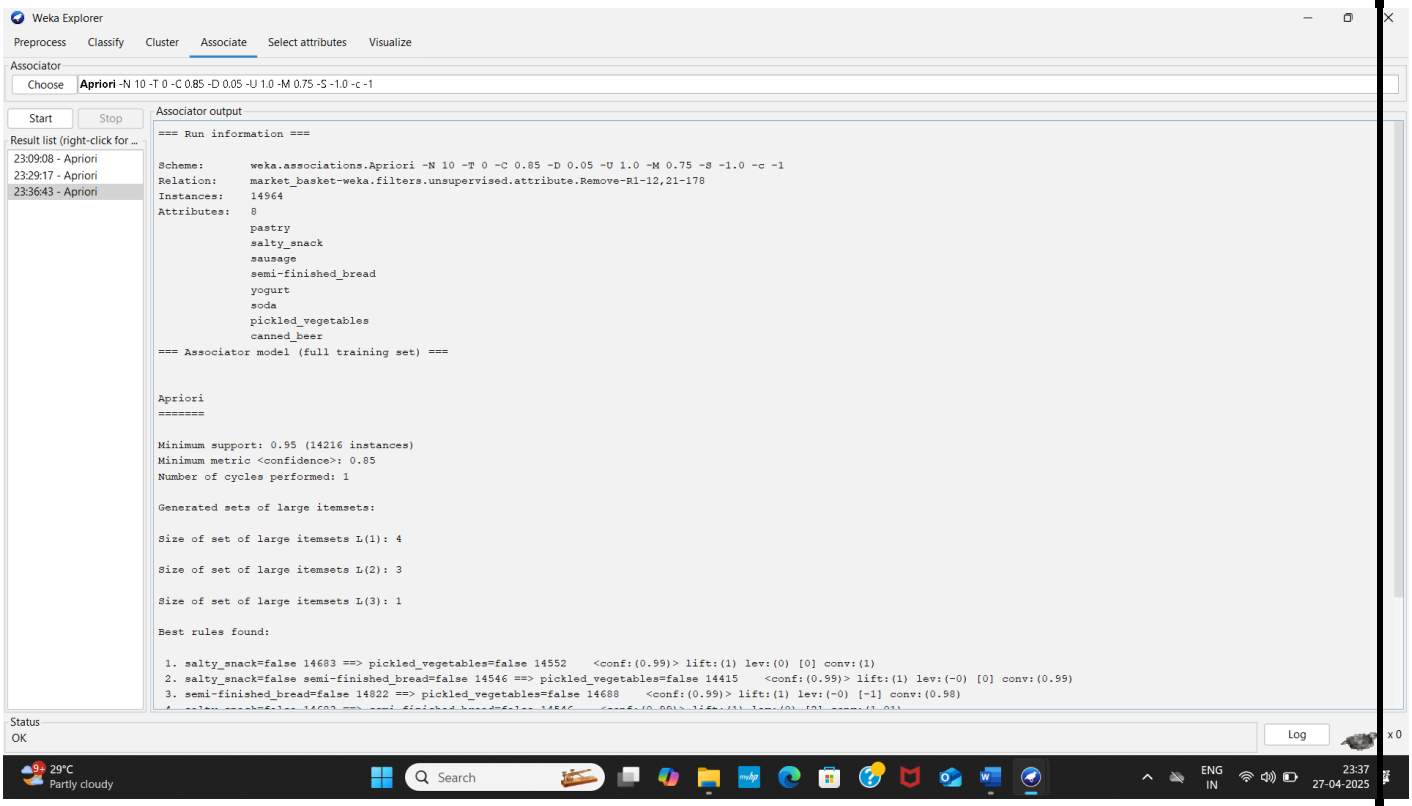
**Configure Apriori Parameters**:

- Click the “Apriori” text to open the configuration window.
- Key parameters to adjust:
    - **minSupport**: Minimum support threshold (e.g., 0.05 means an itemset must appear in 5% of transactions, or ~75 transactions). Start with a low value (e.g., 0.01–0.1) to capture frequent itemsets.
    - **minMetric**: Minimum confidence for rules (e.g., 0.9 means 90% confidence that the rule holds). Common values are 0.5–0.9.
    - **numRules**: Maximum number of rules to generate (e.g., 10–100). Set to 10 for initial testing.
    - **metricType**: Choose “Confidence” (default) for standard association rules.

**Step-4 Run the algorithm.**

Click “Start” to execute Apriori.

**Result-:**




```
1. salty_snack=false 14683 ==> pickled_vegetables=false 14552    <conf:(0.99)> lift:(1) lev:(0) [0] conv:(1)
2. salty_snack=false semi-finished_bread=false 14546 ==> pickled_vegetables=false 14415    <conf:(0.99)> lift:(1) lev:(-0) [0] conv:(0.99)
3. semi-finished_bread=false 14822 ==> pickled_vegetables=false 14688    <conf:(0.99)> lift:(1) lev:(-0) [-1] conv:(0.98)
4. salty_snack=false 14683 ==> semi-finished_bread=false 14546    <conf:(0.99)> lift:(1) lev:(0) [2] conv:(1.01)
5. salty_snack=false pickled_vegetables=false 14552 ==> semi-finished_bread=false 14415    <conf:(0.99)> lift:(1) lev:(0) [1] conv:(1)
6. pickled_vegetables=false 14830 ==> semi-finished_bread=false 14688    <conf:(0.99)> lift:(1) lev:(-0) [-1] conv:(0.98)
7. salty_snack=false 14683 ==> semi-finished_bread=false pickled_vegetables=false 14415    <conf:(0.98)> lift:(1) lev:(0) [2] conv:(1.01)
8. semi-finished_bread=false pickled_vegetables=false 14688 ==> salty_snack=false 14415    <conf:(0.98)> lift:(1) lev:(0) [2] conv:(1.01)
9. semi-finished_bread=false 14822 ==> salty_snack=false 14546    <conf:(0.98)> lift:(1) lev:(0) [2] conv:(1)
10. pickled_vegetables=false 14830 ==> salty_snack=false 14552    <conf:(0.98)> lift:(1) lev:(0) [0] conv:(1)
```

**Applications of Apriori Algorithm-:**

1. **Market Basket Analysis** – Finds frequently bought item pairs (e.g., bread & butter).
2. **Recommendation Systems** – Suggests related products (e.g., Amazon's "Frequently bought together").
3. **Healthcare** – Identifies disease-symptom or drug-treatment patterns.
4. **Fraud Detection** – Detects unusual transaction patterns.
5. **Web Usage Mining** – Analyzes user clickstreams for better website design.
6. **Inventory Management** – Optimizes stock based on purchase trends.

Used in retail, e-commerce, healthcare, banking, and more for **pattern discovery**.

# Assignment-10

**1)FP-Growth algorithm for association rule mining using Python libraries.**

**Dataset Overview**

- Format: CSV (Comma-Separated Values)
- Rows: 1,4965 (based on the visible data)
- Columns: 11 (labeled 0 through 10)
- Data Type: Categorical (item names as strings)
- Purpose: Likely used for market basket analysis, such as identifying purchasing patterns, frequent itemsets, or association rules (e.g., "if a customer buys X, they are likely to buy Y").

**Python code for Fp-Growth algorithm for association rule mining.**

```
import pandas as pd
from mlxtend.frequent_patterns import fpgrowth, association_rules

# Step 1: Load the dataset
df = pd.read_csv('basket.csv', header=None)

# Step 2: Preprocess the dataset
# Convert the dataset into a binary (one-hot encoded) format
# Get all unique items
items = df.values.flatten()
unique_items = pd.Series(items).dropna().unique()

# Create a binary matrix (1 if item is present, 0 otherwise)
binary_data = pd.DataFrame(0, index=df.index, columns=unique_items)
for idx, row in df.iterrows():
    for item in row.dropna():
        binary_data.loc[idx, item] = 1

# Step 3: Apply FP-Growth algorithm to find frequent itemsets
# Minimum support threshold (e.g., 0.05 means itemset appears in 5% of transactions)
min_support = 0.01
frequent_itemsets = fpgrowth(binary_data, min_support=min_support, use_colnames=True)

# Step 4: Generate association rules
# Minimum confidence threshold (e.g., 0.5 means 50% confidence)
min_confidence = 0.75
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=min_confidence)

# Step 5: Sort rules by lift and display top 10
```

```
rules = rules.sort_values(by='lift', ascending=False)
print("Frequent Itemsets:")
print(frequent_itemsets)
print("\nAssociation Rules (Top 10 by Lift):")
print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']].head(10))

# Step 6: Save results to CSV (optional)
frequent_itemsets.to_csv('frequent_itemsets.csv', index=False)
rules.to_csv('association_rules.csv', index=False)
print("\nResults saved to 'frequent_itemsets.csv' and 'association_rules.csv'.")
```

**Result-:**

We do not get association rules for 75 % support and 85% confidence in this data set.

->Result on lowering the support and confidence we get.

```
C:\Users\labhg\AppData\Roaming\Python\Python313\site-packages\mlxtend\frequent_patterns\fpcommon.py:161: DeprecationWarning: DataFrames with non-bool
types result in worse computationalperformance and their support might be discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
Frequent Itemsets:
     support                    itemsets
0   0.157912                 (whole milk)
1   0.051724                    (pastry)
2   0.018778                (salty snack)
3   0.085873                    (yogurt)
4   0.060345                   (sausage)
..       ...                         ...
64  0.011160          (yogurt, whole milk)
65  0.011628            (whole milk, soda)
66  0.013967       (rolls/buns, whole milk)
67  0.010559  (rolls/buns, other vegetables)
68  0.014836  (whole milk, other vegetables)

[69 rows x 2 columns]

Association Rules (Top 10 by Lift):
2   0.018778                (salty snack)
3   0.085873                    (yogurt)
4   0.060345                   (sausage)
2   0.018778                (salty snack)
2   0.018778                (salty snack)
3   0.085873                    (yogurt)
4   0.060345                   (sausage)
..       ...                         ...
64  0.011160          (yogurt, whole milk)
65  0.011628            (whole milk, soda)
66  0.013967       (rolls/buns, whole milk)
67  0.010559  (rolls/buns, other vegetables)
68  0.014836  (whole milk, other vegetables)

[69 rows x 2 columns]
```

association_rules.csv

```
antecedents,consequents,antecedent support,consequent support,support,confidence,lift,representativity,leverage,conviction,zhangs_metric,jaccard,certainty,kulczynski
frozenset({'yogurt'}),frozenset({'whole milk'}),0.0858727612937717 2,0.1579123229083 1328,0.011160117615610799,0.1299610894941634,0.8229952362211855,1.0,-0.0024002495948397
frozenset({'rolls/buns'}),frozenset({'whole milk'}),0.1099973269179 3639,0.1579123229083 1328,0.013966853782411 12,0.1269744835965978,0.8040821720437958,1.0,-0.0034030796249
frozenset({'other vegetables'}),frozenset({'whole milk'}),0.1220930232558 1395,0.1579123229083 1328,0.014835605453087 41,0.1215106732348 1117,0.7694818934768153,1.0,-0.004444
frozenset({'soda'}),frozenset({'whole milk'}),0.097099705960973,0.1579123229083 1328,0.011627906976744186,0.11975223675154852,0.7583463693398951,1.0,-0.0037053331452672547
```

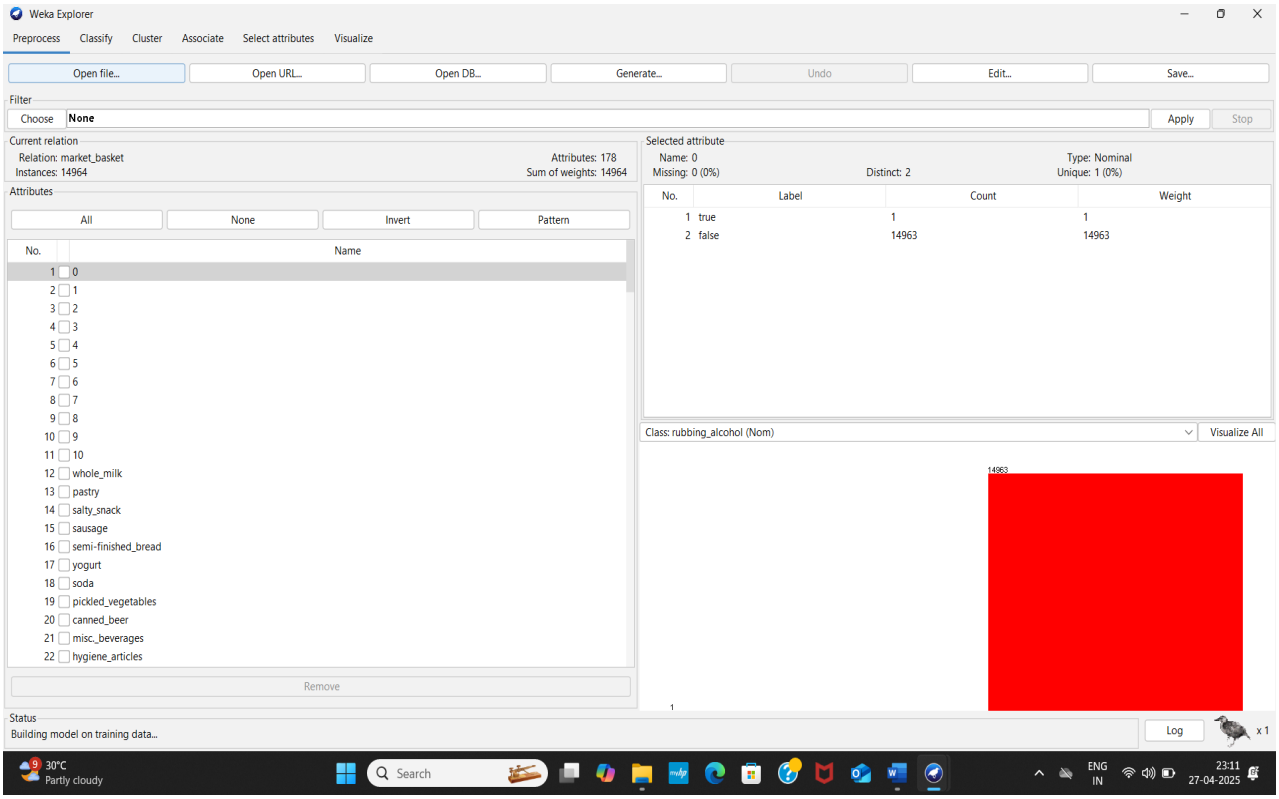
**2) Fp growth algorithm for association rule mining using weka.**

The basket.csv dataset is in a transactional format (items listed in columns per transaction), but Weka's Apriori algorithm requires data in a specific format, typically **ARFF (Attribute-Relation File Format)** with binary attributes (1 for item presence, 0 for absence) or a sparse format.so we need to preprocess it.

**Python code for converting basket.csv to basket.arff.**

```
import pandas as pd
from collections import defaultdict

# Load the dataset
df = pd.read_csv('basket.csv', header=None)

# Get all unique items
items = df.values.flatten()
unique_items = pd.Series(items).dropna().unique()

# Create a binary matrix (1 if item is present, 0 otherwise)
binary_data = pd.DataFrame(0, index=df.index, columns=unique_items)
for idx, row in df.iterrows():
    for item in row.dropna():
        binary_data.loc[idx, item] = 1

# Save to CSV (intermediate step)
binary_data.to_csv('basket_binary.csv', index=False)

# Convert to ARFF
arff_content = f"""@RELATION market_basket

"""
for item in unique_items:
    arff_content += f"@ATTRIBUTE {item.replace(' ', '_')} {{true, false}}\n"

arff_content += "\n@DATA\n"
for _, row in binary_data.iterrows():
    row_values = ['true' if val == 1 else 'false' for val in row]
    arff_content += ','.join(row_values) + '\n'

# Write to ARFF file
with open('basket.arff', 'w') as f:
    f.write(arff_content)

print("ARFF file 'basket.arff' created.")
```

**Step-2 Loading the dataset into weka**



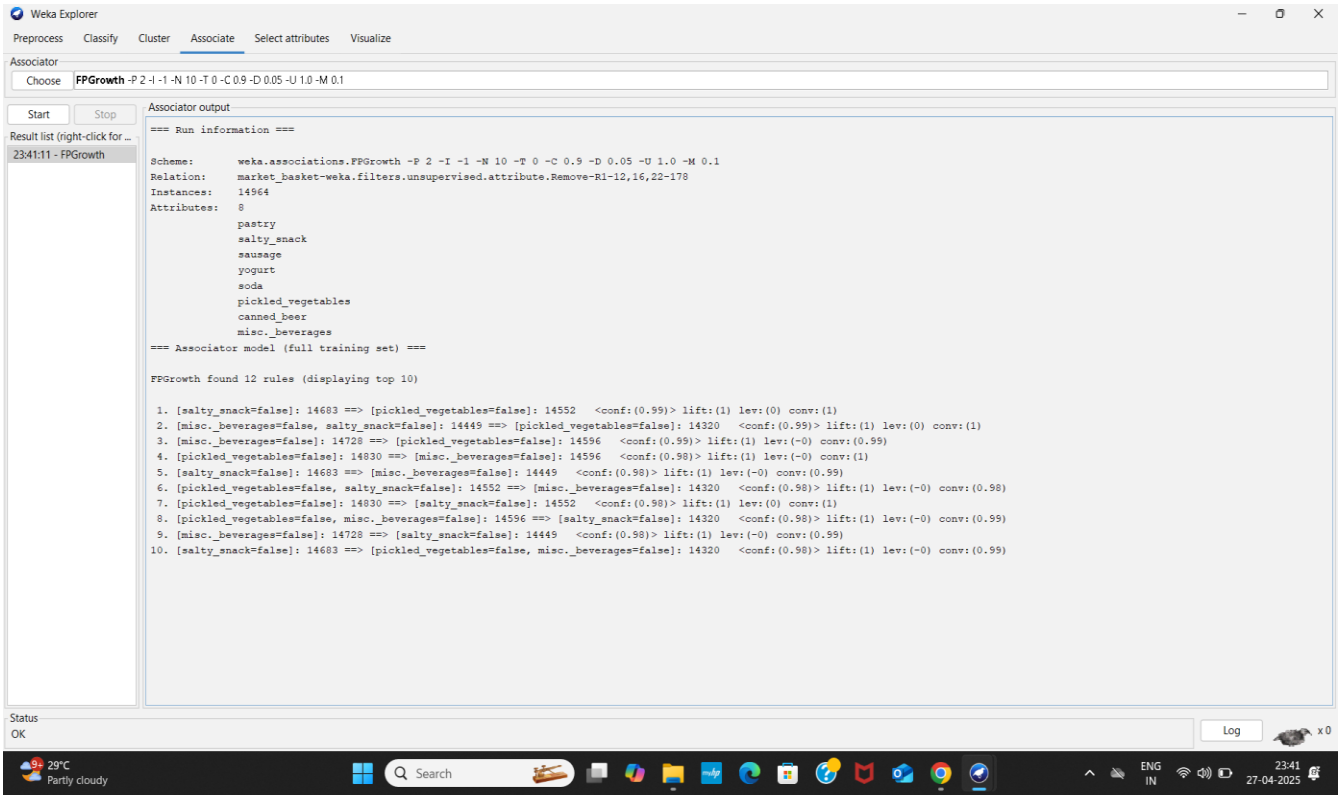

**Step 2: Run the FP-Growth Algorithm**

1. **Switch to Associate Tab**:
    - Click the “Associate” tab in Weka Explorer.
2. **Select FP-Growth**:
    - Click “Choose” and select weka.associations.FPGrowth.
    - If FP-Growth is disabled, ensure all attributes are binary and no class attribute is set. Apply NumericToBinary or NominalToBinary filters if needed.
3. **Configure Parameters**:
    - Click the “FPGrowth” text to open the settings window.
    - Key parameters:
        - **lowerBoundMinSupport**: Minimum support =10%
        - **minMetric**: Minimum confidence for rules (e.g., 0.5 for 50% confidence)
        - **numRules**: Maximum number of rules (e.g., 10–100

**Step-3 Run the algorithm.**

Click “Start” to execute Fp growth.

**Result-:**




**Applications of Fp growth Algorithm-:**

1. **Market Basket Analysis**: Finds item purchase patterns (e.g., whole milk => yogurt) for product placement, promotions, and cross-selling in retail/e-commerce.
2. **Recommendation Systems**: Suggests products based on user behavior (e.g., recommending rolls/buns with sausage).
3. **Customer Segmentation**: Groups customers by purchasing patterns for targeted marketing.
4. **Inventory Management**: Optimizes stock for frequently co-purchased items.
5. **Fraud Detection**: Identifies unusual transaction patterns in banking or insurance.
6. **Web Usage Mining**: Analyzes clickstream data for website optimization.
7. **Bioinformatics**: Discovers frequent patterns in genomic or protein sequences.
8. **Text Mining**: Finds co-occurring words or phrases for topic modeling or sentiment analysis.