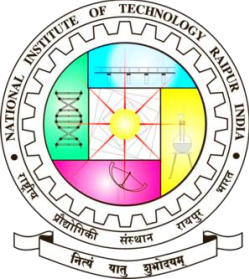
**National Institute of Technology, Raipur**

****

**Data Mining & Warehousing & AI lab Practical**

**Submitted by: Amitesh Agrawal**

**Branch : CSE**

**Sem : VI**

**Roll No. : 19115011**

Contents

[Experiment 1 2](#_Toc101874964)

[Experiment 2 10](#_Toc101874965)

[Experiment 3 13](#_Toc101874966)

[Types of Linear Regression 39](#_Toc101874967)

# Experiment 1

**Aim -** Perform an experiment on data cleaning.

**Description –**

Data cleaning is one of the important processes involved in data analysis, with it being the first step after data collection. It is a very important step in ensuring that the dataset is free of inaccurate or corrupt information. Data cleaning is the process of modifying data to ensure that it is free of irrelevances and incorrect information. Also known as data cleansing, it entails identifying incorrect, irrelevant, incomplete, and the “dirty” parts of a dataset and then replacing or cleaning the dirty parts of the data. Data cleansing is very important to companies, as lack of it may reduce marketing effectiveness, thereby reducing sales. Although the issues with the data may not be completely solved, reducing it to a minimum will have a significant effect on efficiency.

Some methods to clean data –

1. You can ignore the tuple. This is done when the class label is missing. This method is not very effective unless the tuple contains several attributes with missing values.
2. You can fill in the missing value manually. This approach is effective on small data sets with some missing values.
3. You can replace all missing attribute values with a global constant, such as a label like “Unknown” or minus infinity.

4. You can use the attribute mean to fill in the missing value. For example, customer's average income is 25000 then you can use this value to replace the missing value for income.

5. Use the most probable value to fill in the missing value.

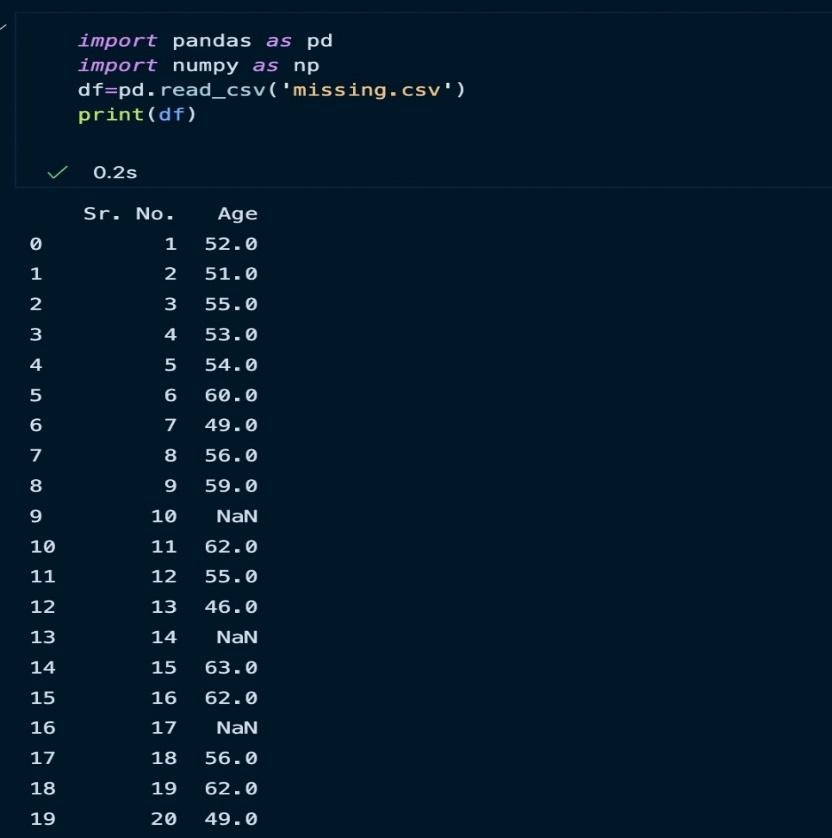
**Code:**

Python code for Cleaning Data–

1. import pandas as pd

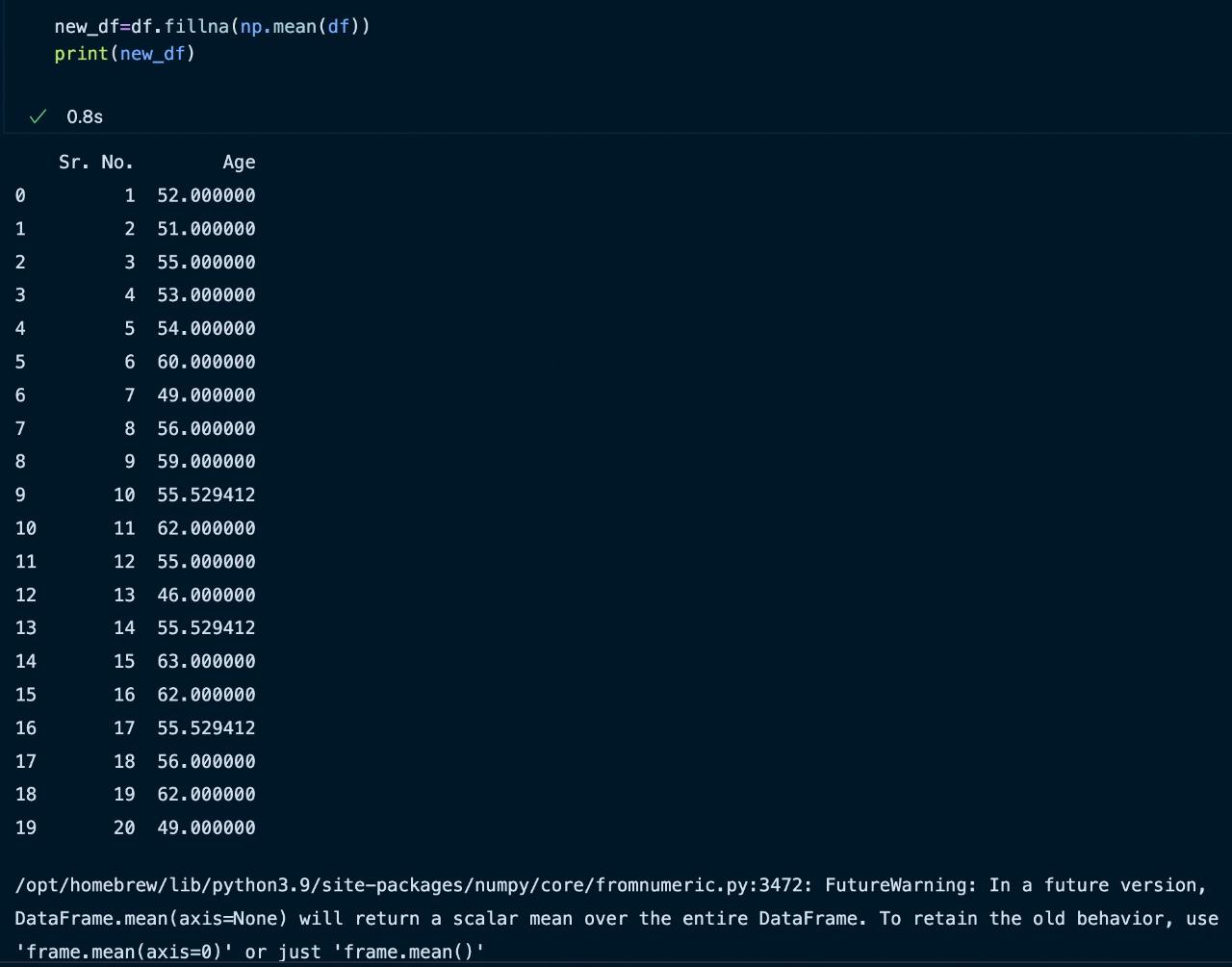
df=pd.read\_csv('missing.csv')

print(df)

Output-

1. new\_df=df.fillna(np.mean(df))

print(new\_df)

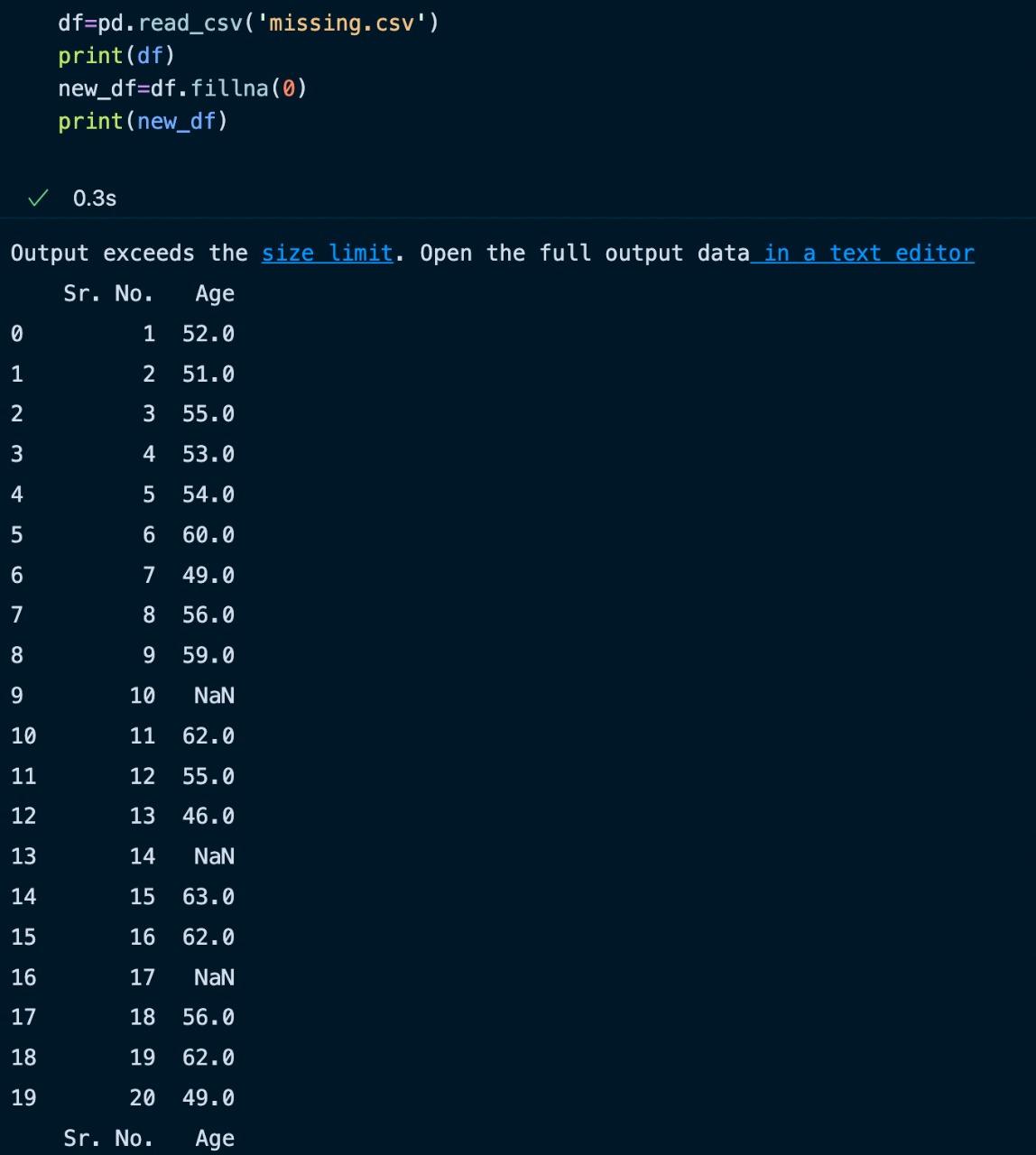
Output –

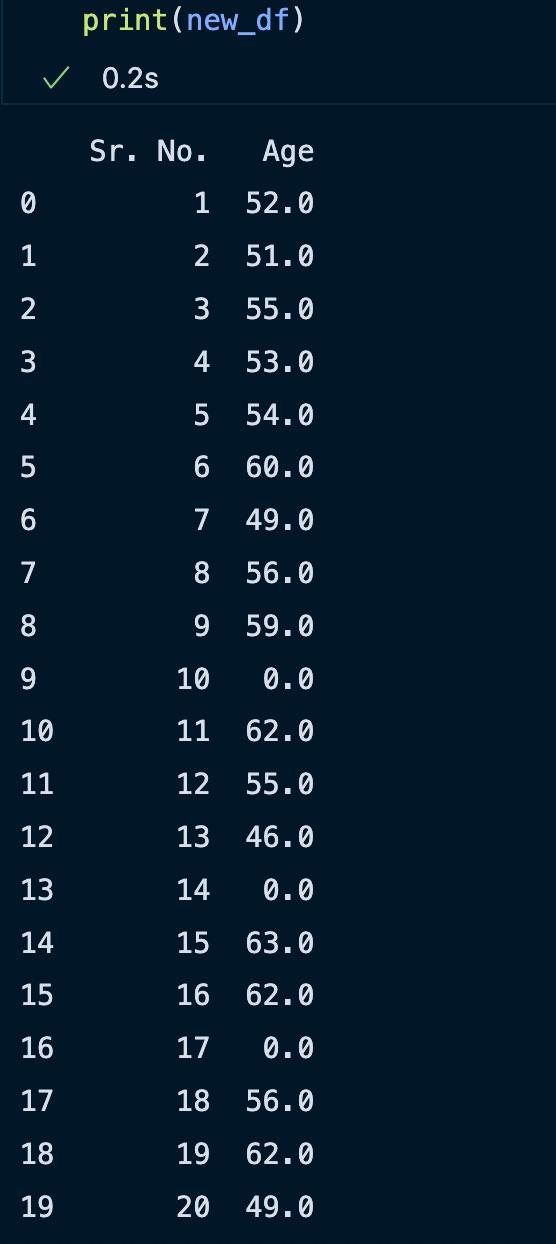
1. df=pd.read\_csv('missing.csv')

print(df)

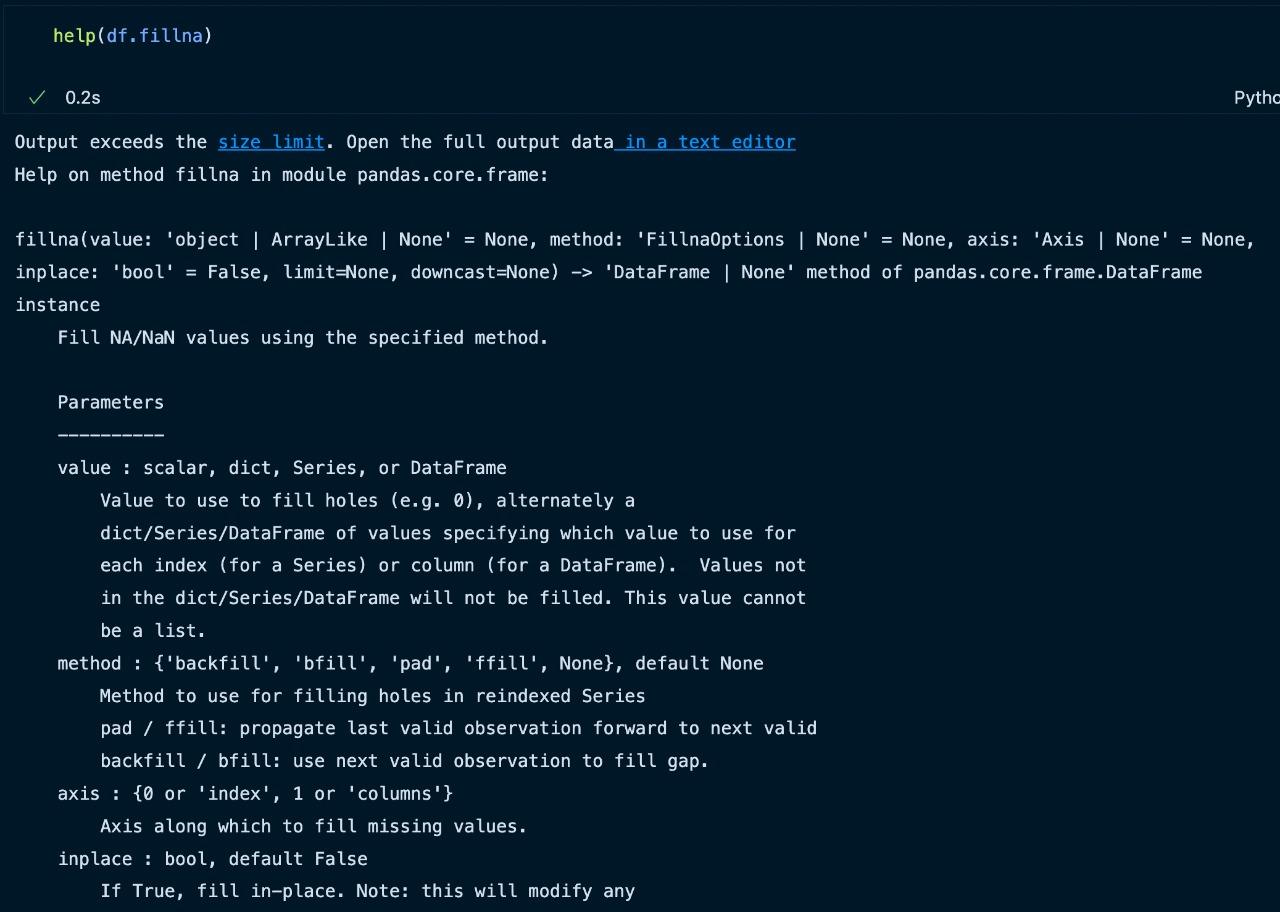
new\_df=df.fillna(0)

print(new\_df)

Output –



1. help(df.fillna)

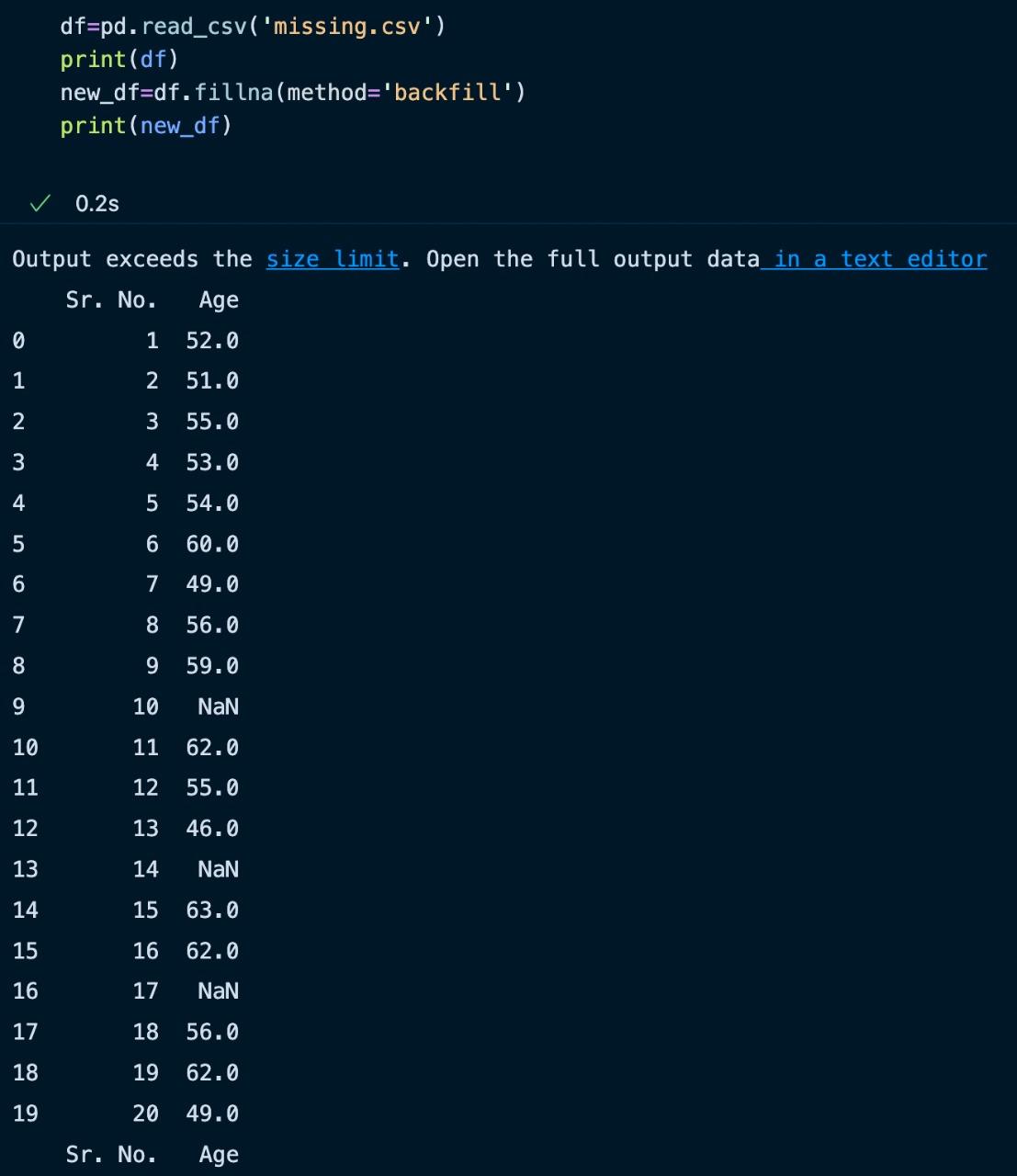
Output –

1. df=pd.read\_csv('missing.csv')

print(df)

new\_df=df.fillna(method='backfill')

print(new\_df)

Output –

# Experiment 2

Aim - Perform an experiment on binning equal width and binning equal depth.

Theory - Data binning, bucketing is a data pre-processing method used to minimize the effects of small observation errors. The original data values are divided into small intervals known as bins and then they are replaced by a general value calculated for that bin. This has a smoothening effect on the input data and may also reduce the chances of overfitting in the case of small datasets

There are 2 methods of dividing data into bins:

Equal Frequency Binning: bins have an equal frequency.

Equal Width Binning : bins have equal width with a range of each bin are defined as [min + w], [min + 2w] …. [min + nw] where w = (max – min) / (no of bins).

Program Code : (python)

# equal frequency

def equifreq(arr1, m):

a = len(arr1)

n = int(a / m)

for i in range(0, m):

arr = []

for j in range(i \* n, (i + 1) \* n):

if j >= a:

break

arr = arr + [arr1[j]]

print(arr)

# equal width

def equiwidth(arr1, m):

a = len(arr1)

w = int((max(arr1) - min(arr1)) / m)

min1 = min(arr1)

arr = []

for i in range(0, m + 1):

arr = arr + [min1 + w \* i]

arri=[]

for i in range(0, m):

temp = []

for j in arr1:

if j >= arr[i] and j <= arr[i+1]:

temp += [j]

arri += [temp]

print(arri)

# data to be binned

data = [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215]

# no of bins

m = 3

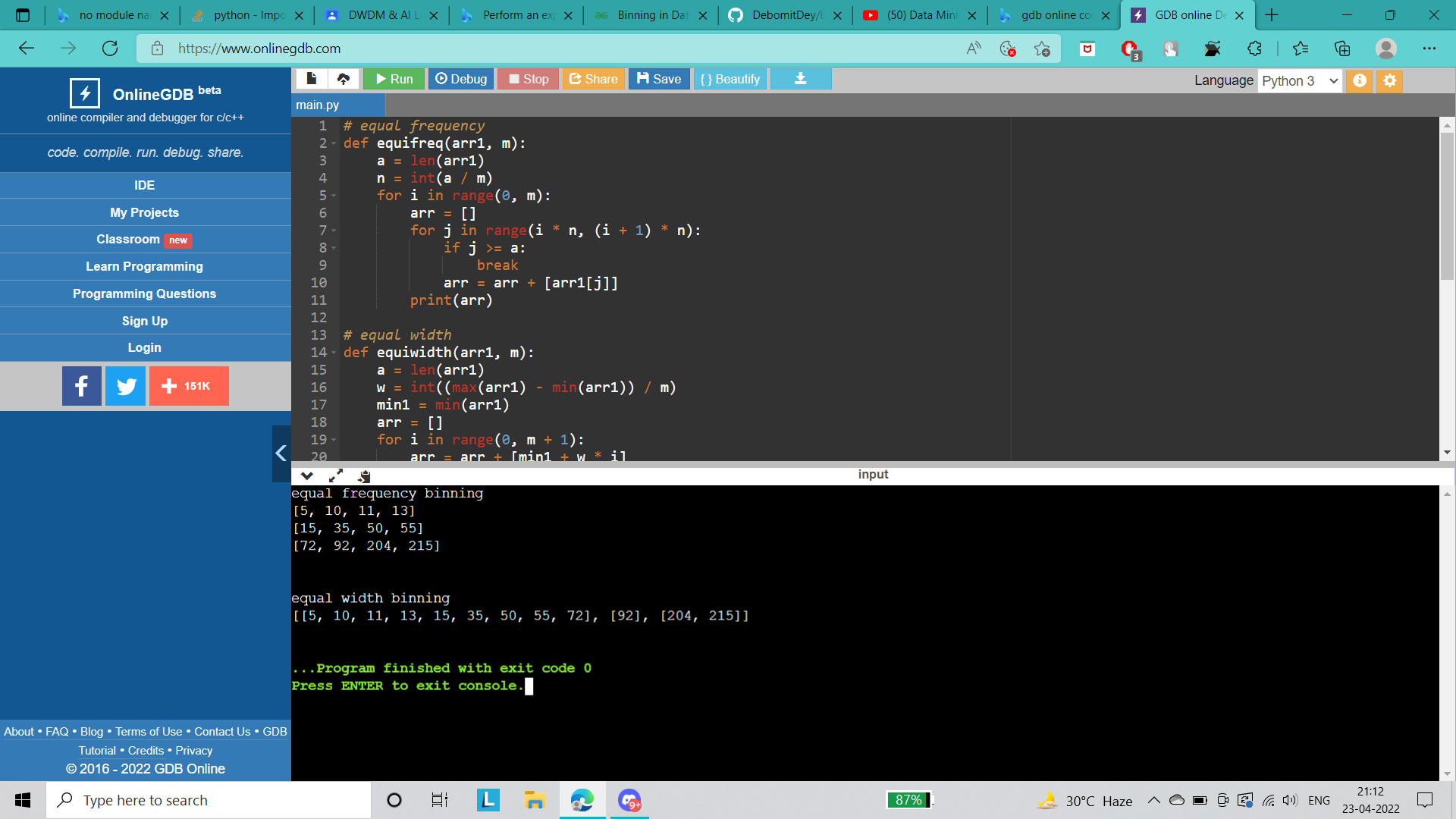
print("equal frequency binning")

equifreq(data, m)

print("\n\nequal width binning")

equiwidth(data, 3)

Output:



# Experiment 3

**Aim:** Perform an experiment on normalization.

**Description:** Normalization is used to scale the data of an attribute so that it falls in a smaller range, such as -1.0 to 1.0 or 0.0 to 1.0. It is generally useful for classification algorithms.

**Following normalization techniques are used:**

1. Decimal Scaling.

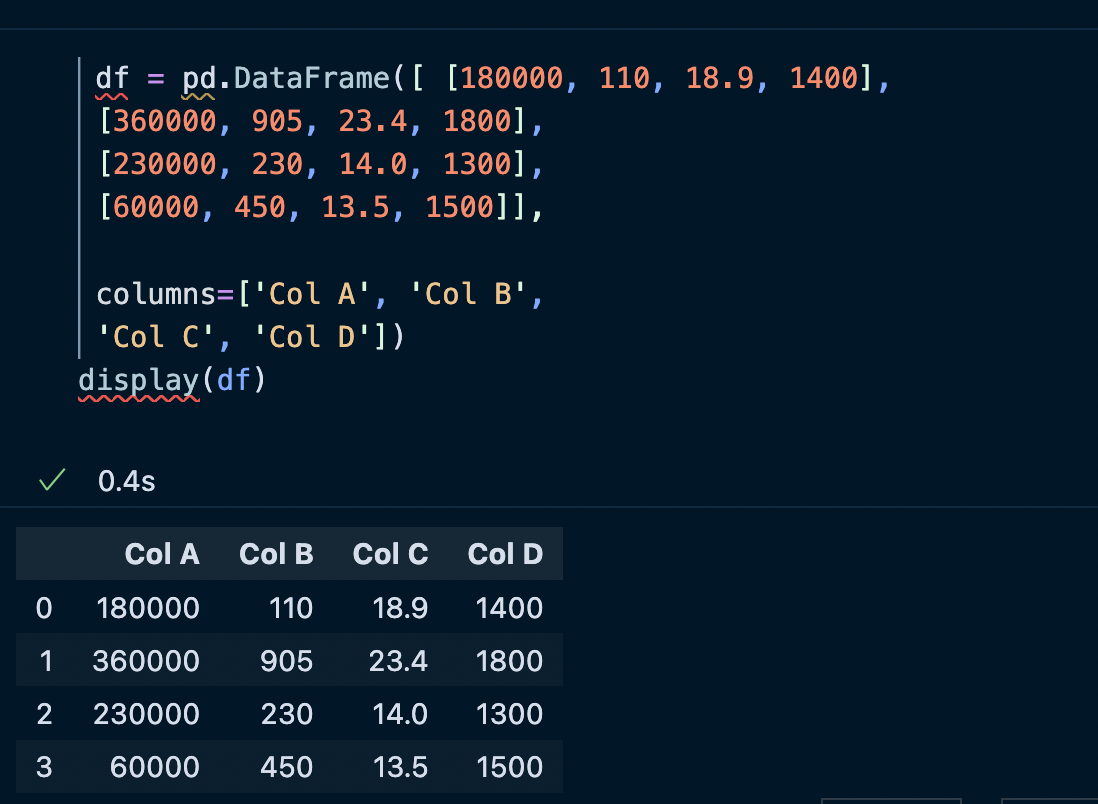
2. Min-Max Normalization.

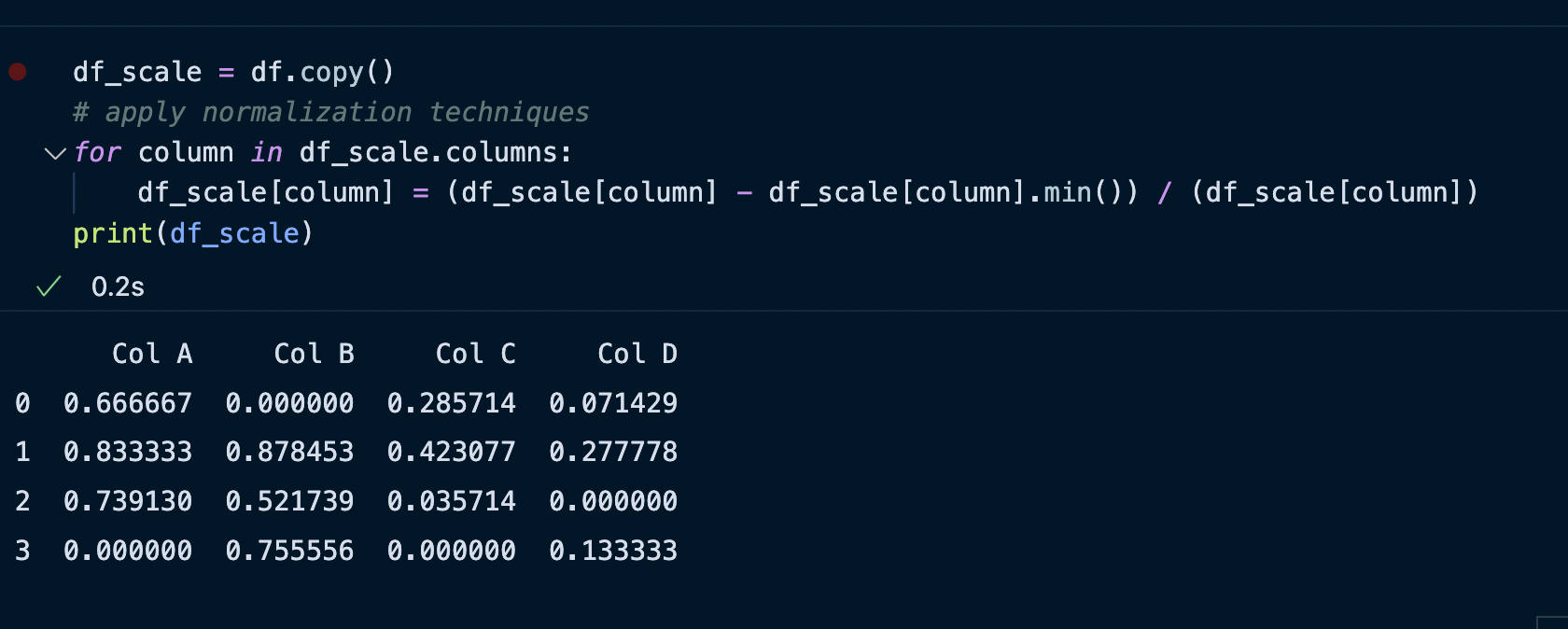
3. z-Score Normalization.



**Min-Max Scaling**

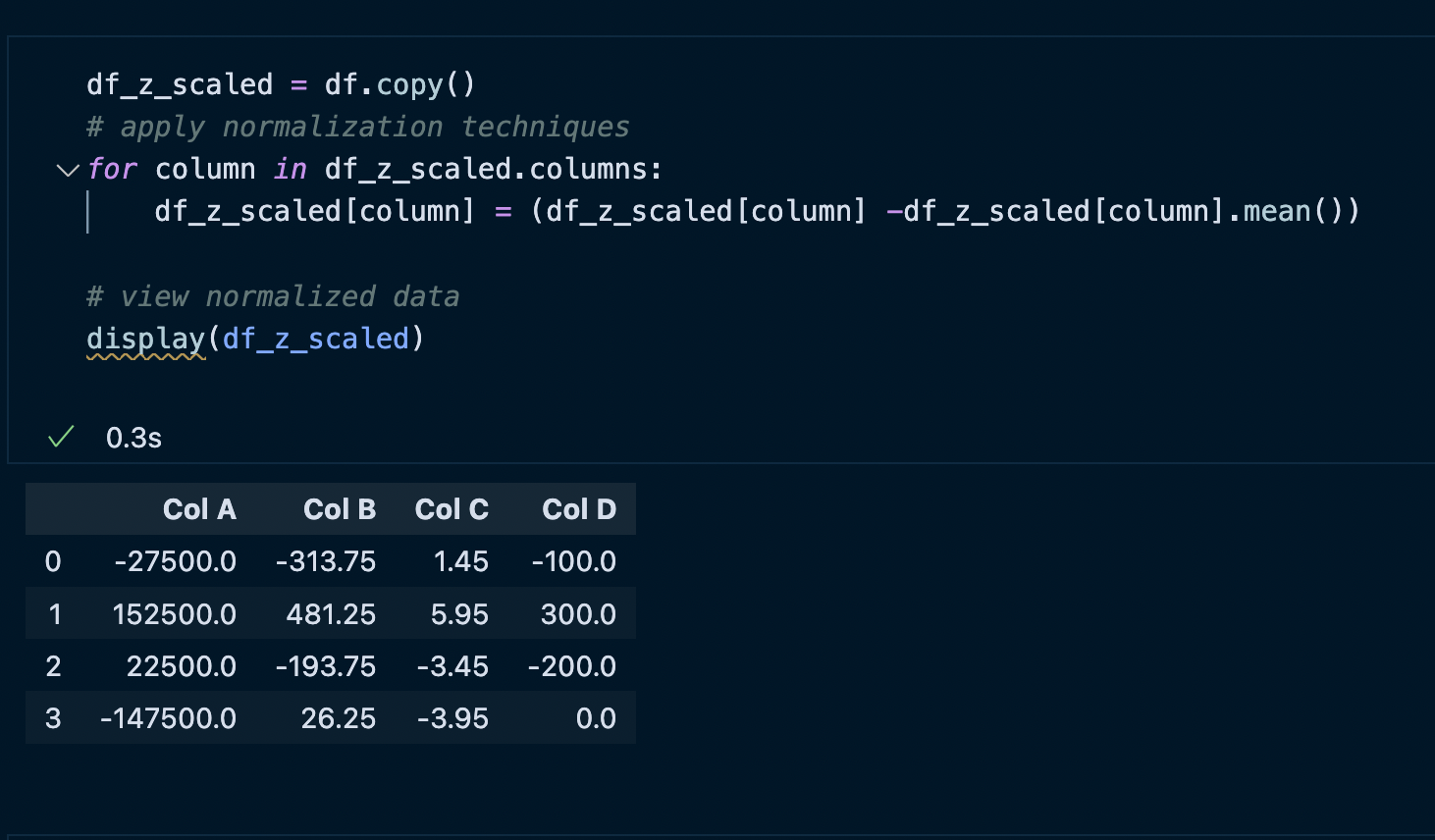
The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature and then dividing by the range. We can apply the min-max scaling in Pandas using the .min() and .max() methods.





**The z-score method :**

The z-score method (often called standardization) transforms the info into a distribution with a mean of 0 and a typical deviation of 1. Each standardized value is computed by subtracting the mean of the corresponding feature and then dividing by the quality deviation.



# Experiment 4

**Aim :** Perform an experiment on data pre-processing.

**Description :** Pre-processing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

**Following normalization techniques are used:**

1. Rescale Data

2. Binarize Data (Make Binary)

3. Standardize Data

**Rescale Data**

When our data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.

**Python code to Rescale data (between 0 and 1)**

# importing libraries

import pandas

import scipy

import numpy

from sklearn.preprocessing import MinMaxScaler

# data set link

url =

"https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima

-indians-diabetes.data"

# data parameters

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

# preparating of dataframe using the data at given link and defined columns list

dataframe = pandas.read\_csv(url, names = names)

array = dataframe.values

# separate array into input and output components

X = array[:,0:8]

Y = array[:,8]

# initialising the MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0, 1))

# learning the statistical parameters for each of the data and transforming

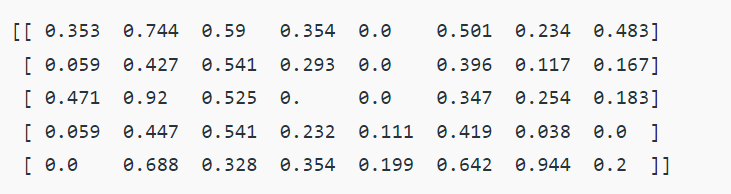
rescaledX = scaler.fit\_transform(X)

# summarize transformed data

numpy.set\_printoptions(precision=3)

print(rescaledX[0:5,:])

xOUTPUT



**Binarize Data (Make Binary)**

We can transform our data using a binary threshold. All values above

the threshold are marked 1 and all equal to or below are marked as 0.

**Code: Python code for binarization**

# import libraries

from sklearn.preprocessing import Binarizer

import pandas

import numpy

# data set link

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"

# data parameters

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

# preparating of dataframe using the data at given link and defined columns list

dataframe = pandas.read\_csv(url, names = names)

array = dataframe.values

# separate array into input and output components

X = array[:, 0:8]

Y = array[:, 8]

binarizer = Binarizer(threshold = 0.0).fit(X)

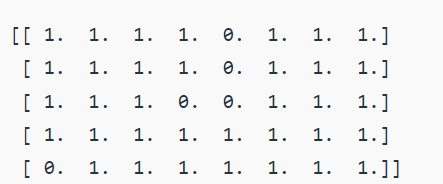
binaryX = binarizer.transform(X)

# summarize transformed data

numpy.set\_printoptions(precision = 3)

print(binaryX[0:5,:])

OUTPUT



**3. Standardize Data**

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

**Code: Python code to Standardize data (0 mean, 1 standard deviation)**

# importing libraries

from sklearn.preprocessing import StandardScaler

import pandas

import numpy

# data set link

Url =

"[https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-dia](https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data)

[betes/pima-indians-diabetes.data](https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data)"

# data parameters

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',

'class']

# preparating of dataframe using the data at given link and defined columns list

dataframe = pandas.read\_csv(url, names = names)

array = dataframe.values

# separate array into input and output components

X = array[:, 0:8]

Y = array[:, 8]

scaler = StandardScaler().fit(X)

rescaledX = scaler.transform(X)

# summarize transformed data

numpy.set\_printoptions(precision = 3)

print(rescaledX[0:5,:])

OUTPUT



# Experiment 5

**Aim-** Write a program to perform decision tree classification.

**Description-**

The decision tree is one of the most powerful and popular algorithms. Decision-tree algorithm falls under the category of supervised learning algorithms. It works for both continuous as well as categorical output variables.

**Data-set Description :**

Title : Balance Scale Weight & Distance Database

Number of Instances : 625 (49 balanced, 288 left, 288 right)

Number of Attributes : 4 (numeric) + class name = 5

**Attribute Information:**

1. **Class Name (Target variable)**: 3
   * L [balance scale tip to the left]
   * B [balance scale be balanced]
   * R [balance scale tip to the right]
2. **Left-Weight**: 5 (1, 2, 3, 4, 5)
3. **Left-Distance**: 5 (1, 2, 3, 4, 5)
4. **Right-Weight**: 5 (1, 2, 3, 4, 5)
5. **Right-Distance**: 5 (1, 2, 3, 4, 5)
6. **Missing Attribute Values**: None
7. **Class Distribution:**
   * 46.08 percent are L
   * 07.84 percent are B
   * 46.08 percent are R

**Code-**

!python get.pip.py

!pip install -U scikit-learn

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

def importdata():

balance\_data = pd.read\_csv(

'https://archive.ics.uci.edu/ml/machine-learning-'+

'databases/balance-scale/balance-scale.data',

sep= ',', header = None)

print ("Dataset Length: ", len(balance\_data))

print ("Dataset Shape: ", balance\_data.shape)

print ("Dataset: ",balance\_data.head())

return balance\_data

def splitdataset(balance\_data):

X = balance\_data.values[:, 1:5]

Y = balance\_data.values[:, 0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test

def train\_using\_gini(X\_train, X\_test, y\_train):

clf\_gini = DecisionTreeClassifier(criterion = "gini",

random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

clf\_entropy = DecisionTreeClassifier(

criterion = "entropy", random\_state = 100,

max\_depth = 3, min\_samples\_leaf = 5)

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

def prediction(X\_test, clf\_object):

y\_pred = clf\_object.predict(X\_test)

print("Predicted values:")

print(y\_pred)

return y\_pred

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

confusion\_matrix(y\_test, y\_pred))

print ("Accuracy : ",

accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ",

classification\_report(y\_test, y\_pred))

def main():

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)

print("Results Using Gini Index:")

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

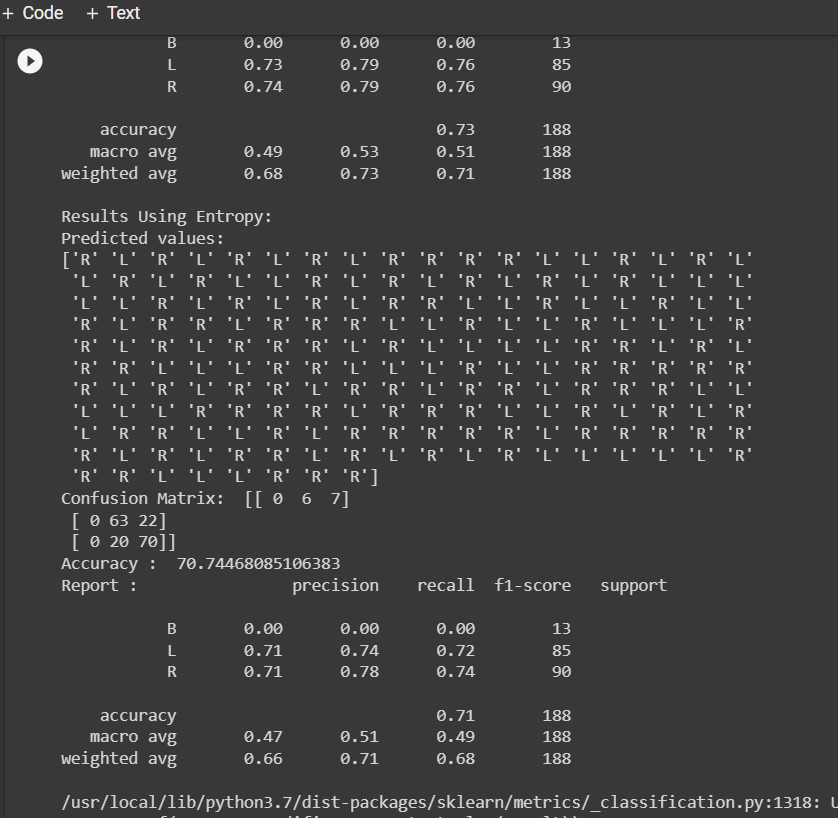
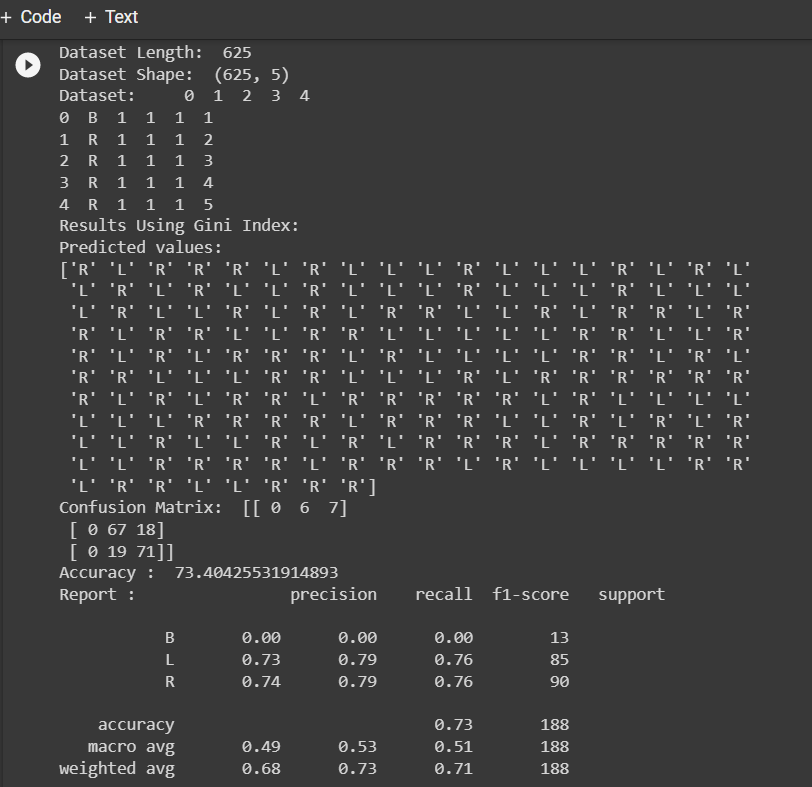
print("Results Using Entropy:")

y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

if \_\_name\_\_=="\_\_main\_\_":

main()

****

# Experiment 6

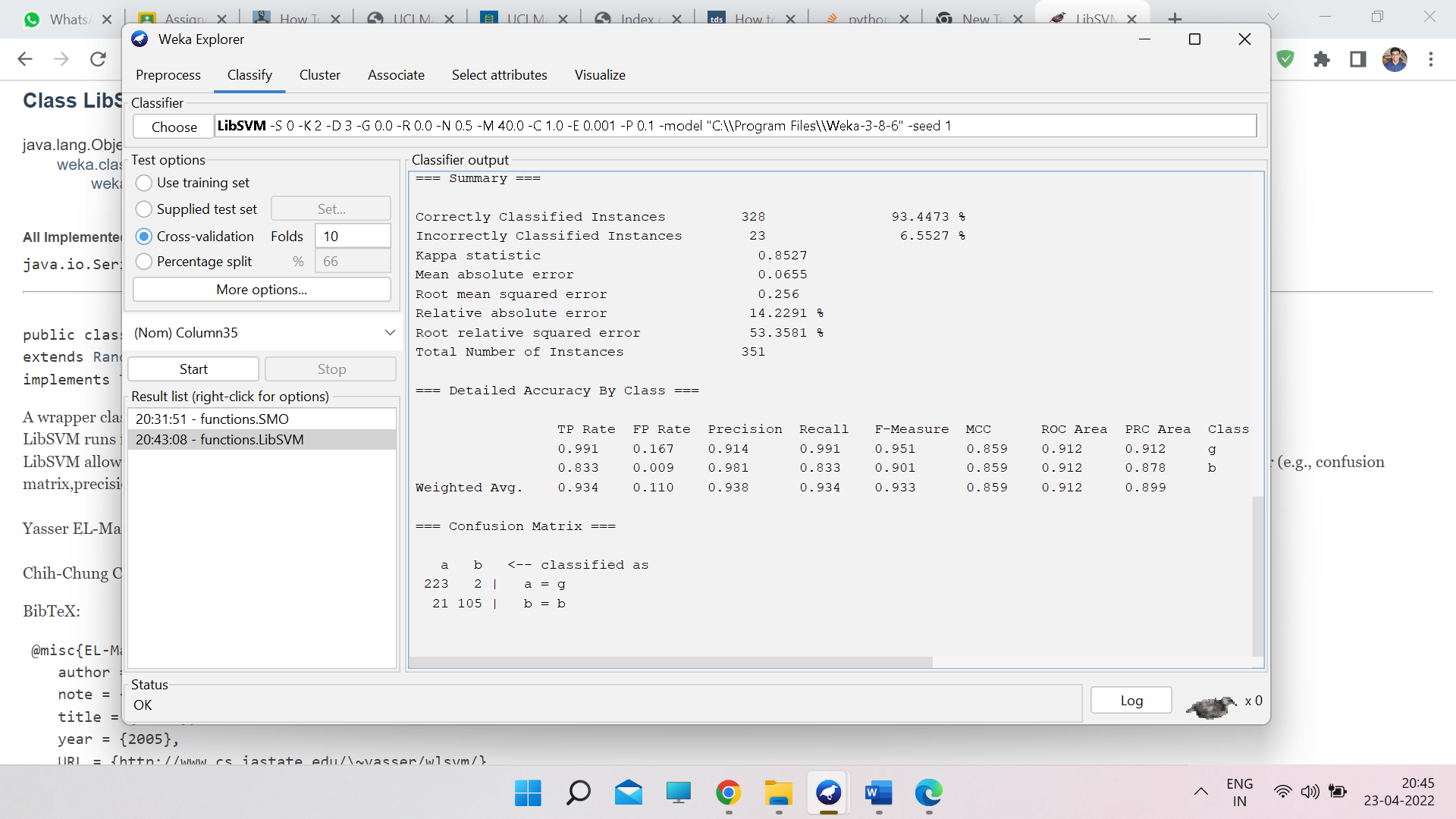
**Aim:** Write a program to perform Support Vector Machine Classification.

**Description:** The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence algorithm is termed a Support Vector Machine.

**Steps:**

1. Open the Weka GUI Chooser.
2. Click the “Explorer” button to open the Weka Explorer.
3. Load the dataset from the downloads/ionosphere.csv file.
4. Click “Classify” to open the Classify tab.
5. Select functions and select the LibSVM classifier for implementing SVM.

**Output:**

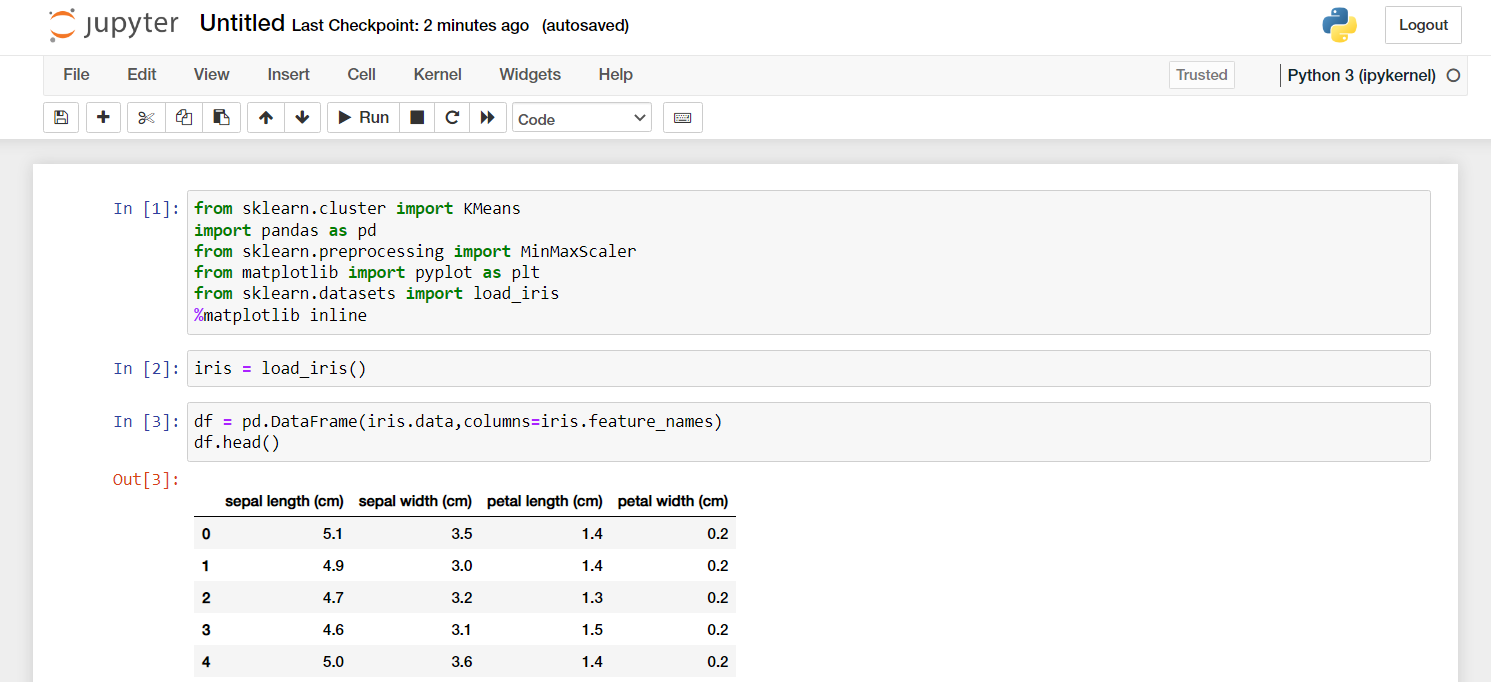


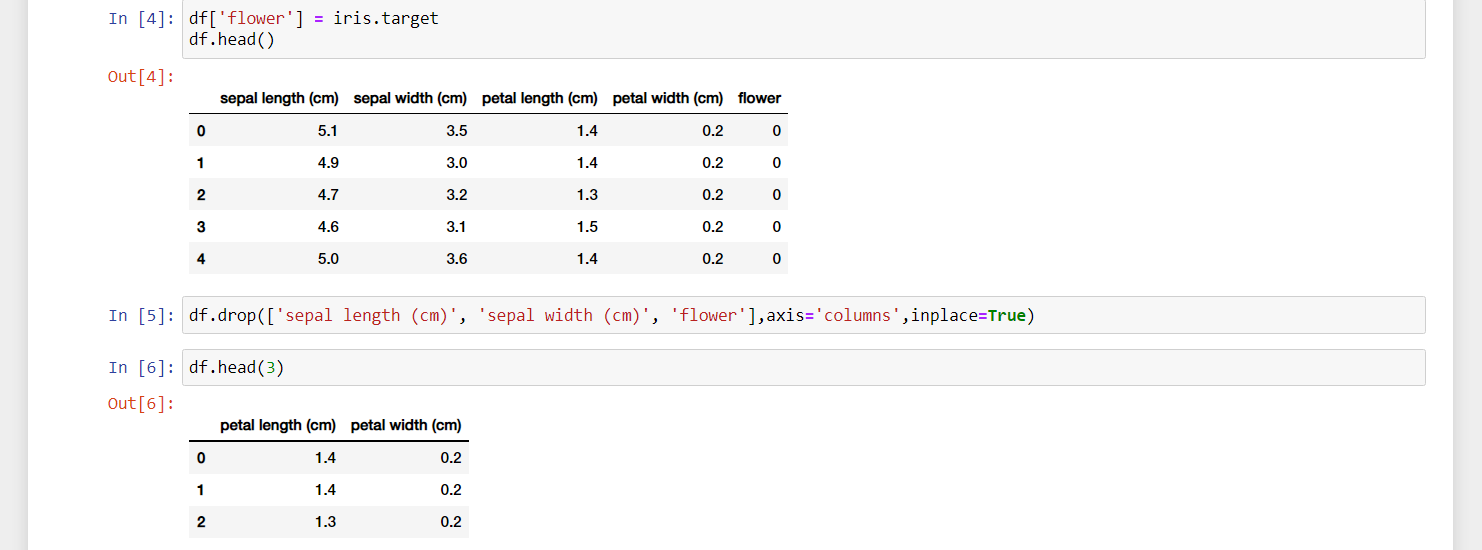
Experiment 7

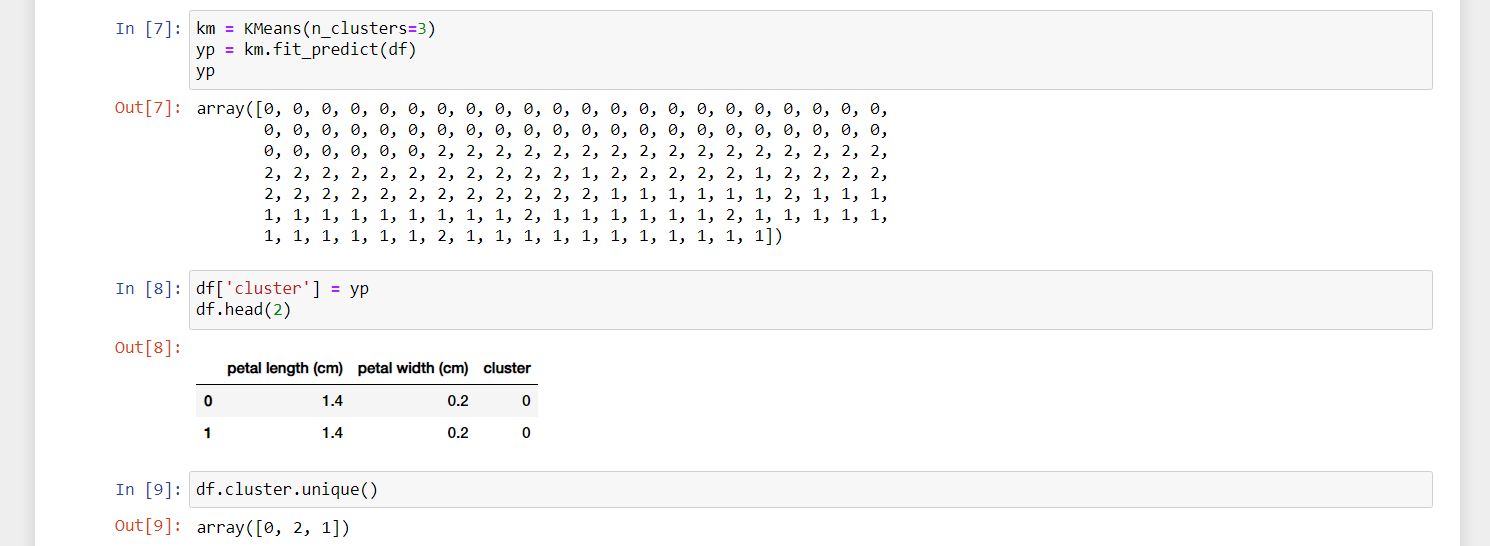
**Aim:** Write a program to perform K means clustering

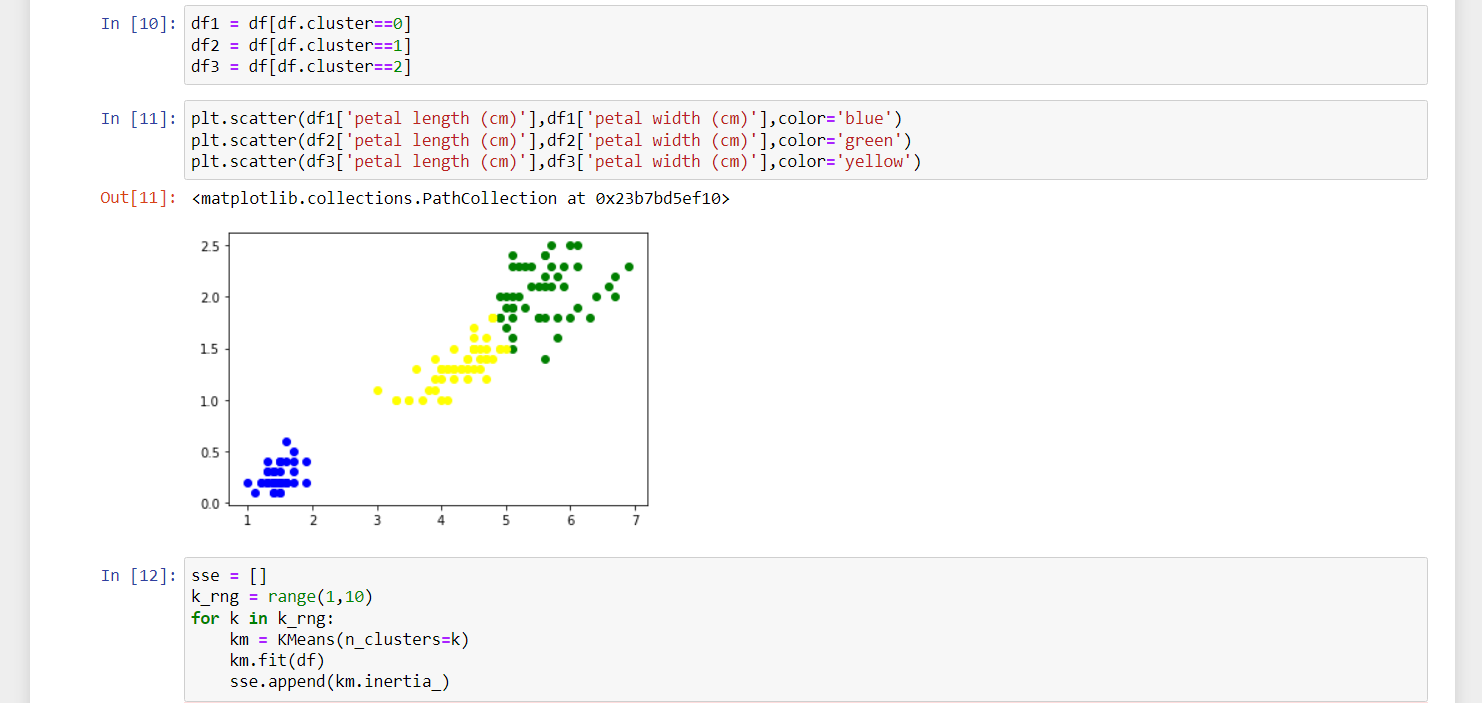
**Description:** This experiment illustrates the use of simple k-mean clustering with Jupyter Notebook. The sample data set used for this example is based on the iris data available in ARFF format. This document assumes that appropriate pre-processing has been performed. This iris dataset includes 150 instances.

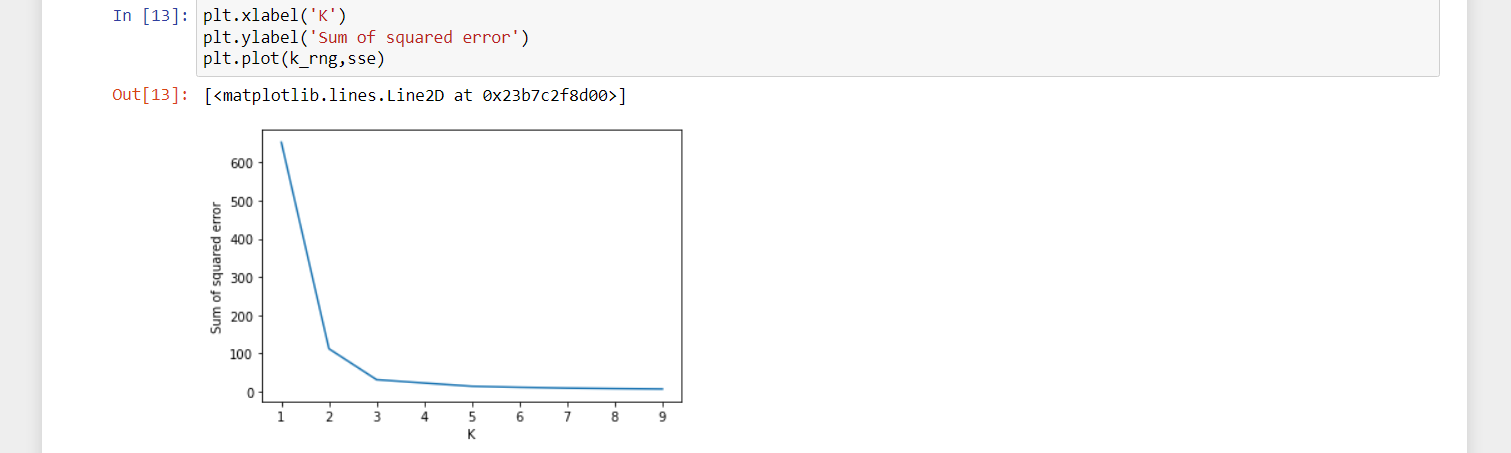
**Code and Output:**











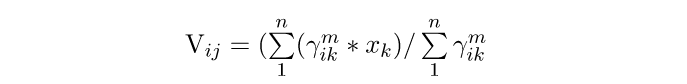
We also drew the elbow plot and figured out that the optimal k value is 5.

# Experiment 8

**Aim:** Write a program to perform Fuzzy c-means Clustering.

**Description:**

Fuzzy C-Means clustering is a soft clustering approach, where each data point is assigned a likelihood or probability score to belong to that cluster. The step-wise approach of the Fuzzy c-means clustering algorithm is:

* Fix the value of c (number of clusters), and select a value of m (generally 1.25<m<2), and initialize partition matrix U.
* Calculate cluster centers (centroid).

Here,

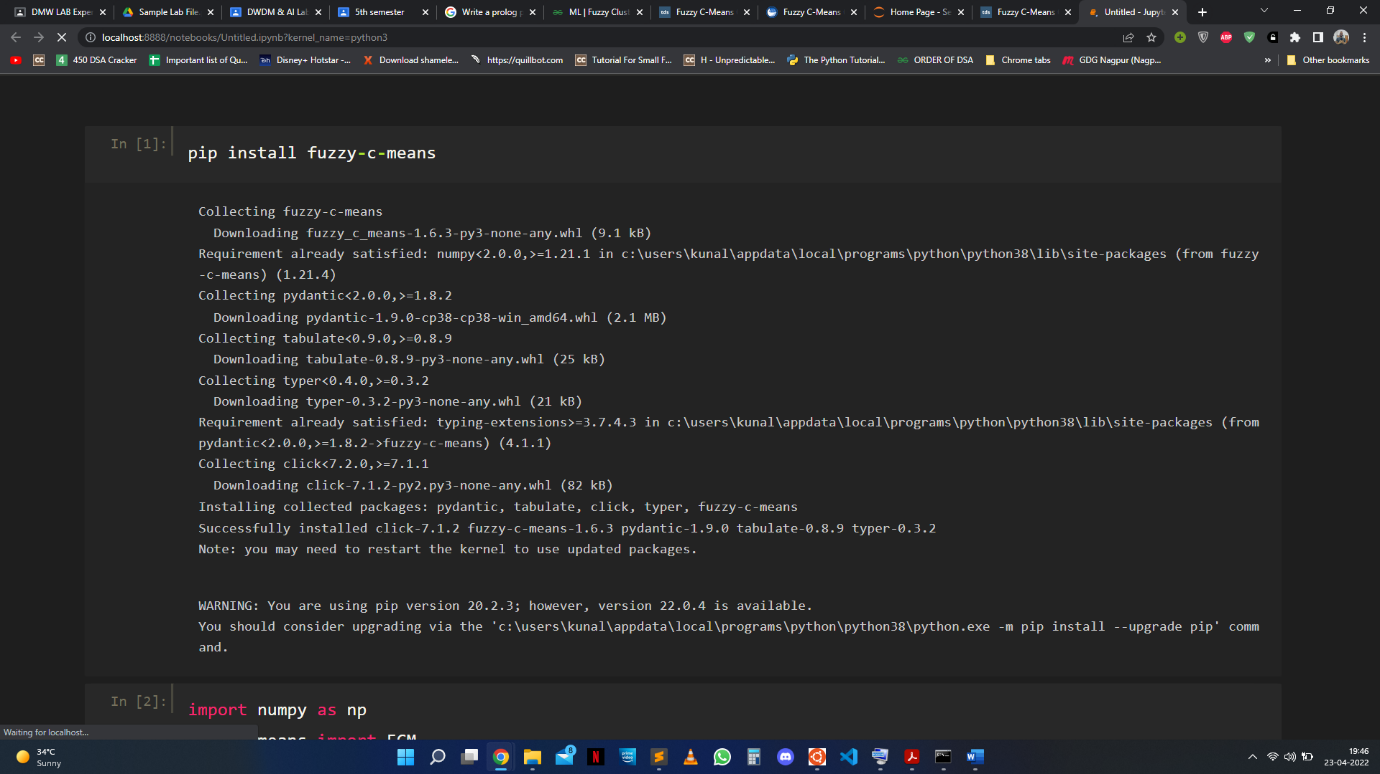
µ: Fuzzy membership value

m: fuzziness parameter

* Update Partition Matrix
* Repeat the above steps until convergence.

**Code and Output:**

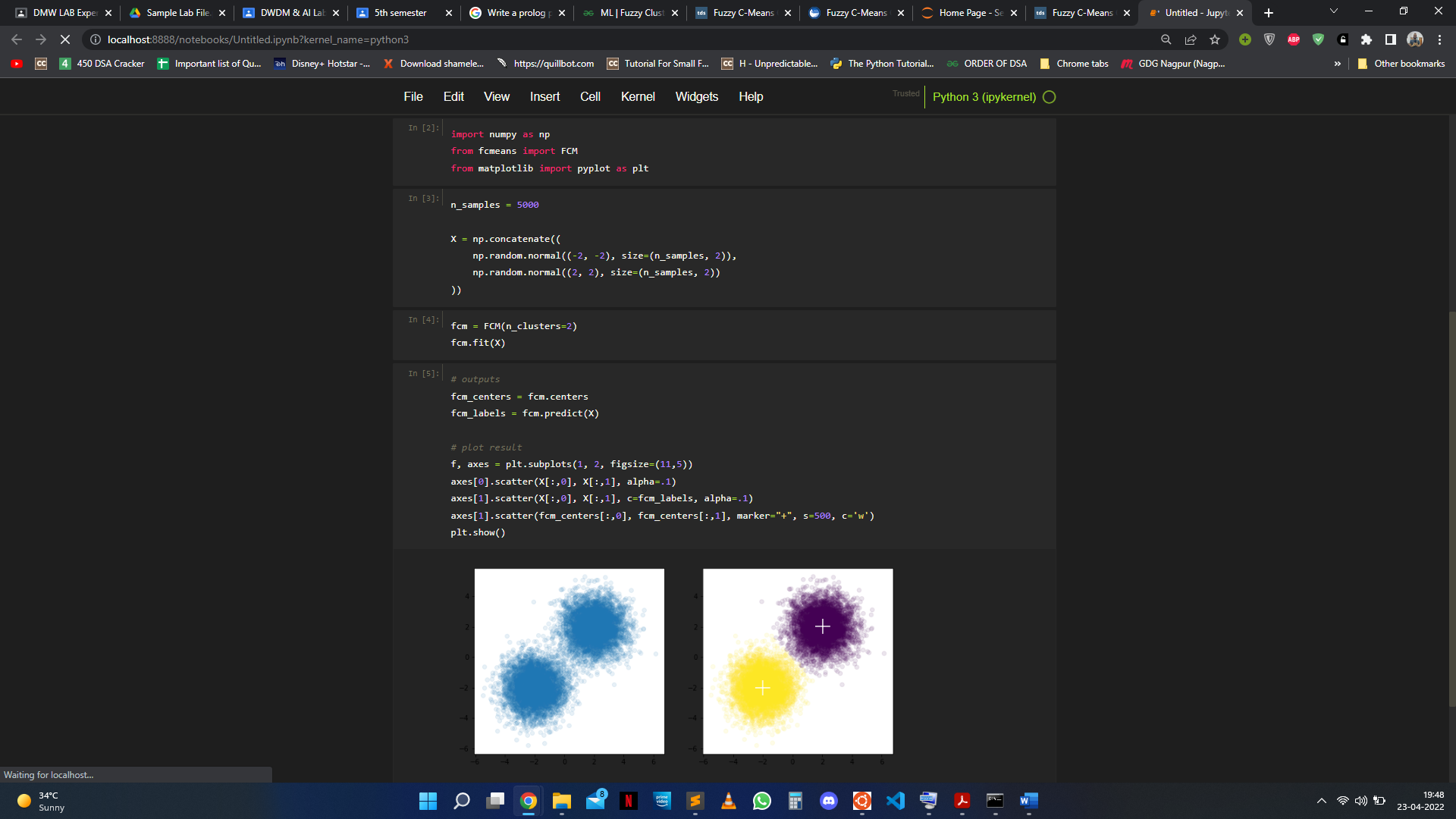
To implement the fuzzy c-means algorithm, we have an open-sourced Python package, that can be installed using PyPl:

pip install fuzzy-c-means

Fuzzy c-means is a Python module that can implement the fuzzy c-means algorithm. This module has an API similar to that of Scikit-learn.

**import** numpy **as** np

**from** fcmeans **import** FCM

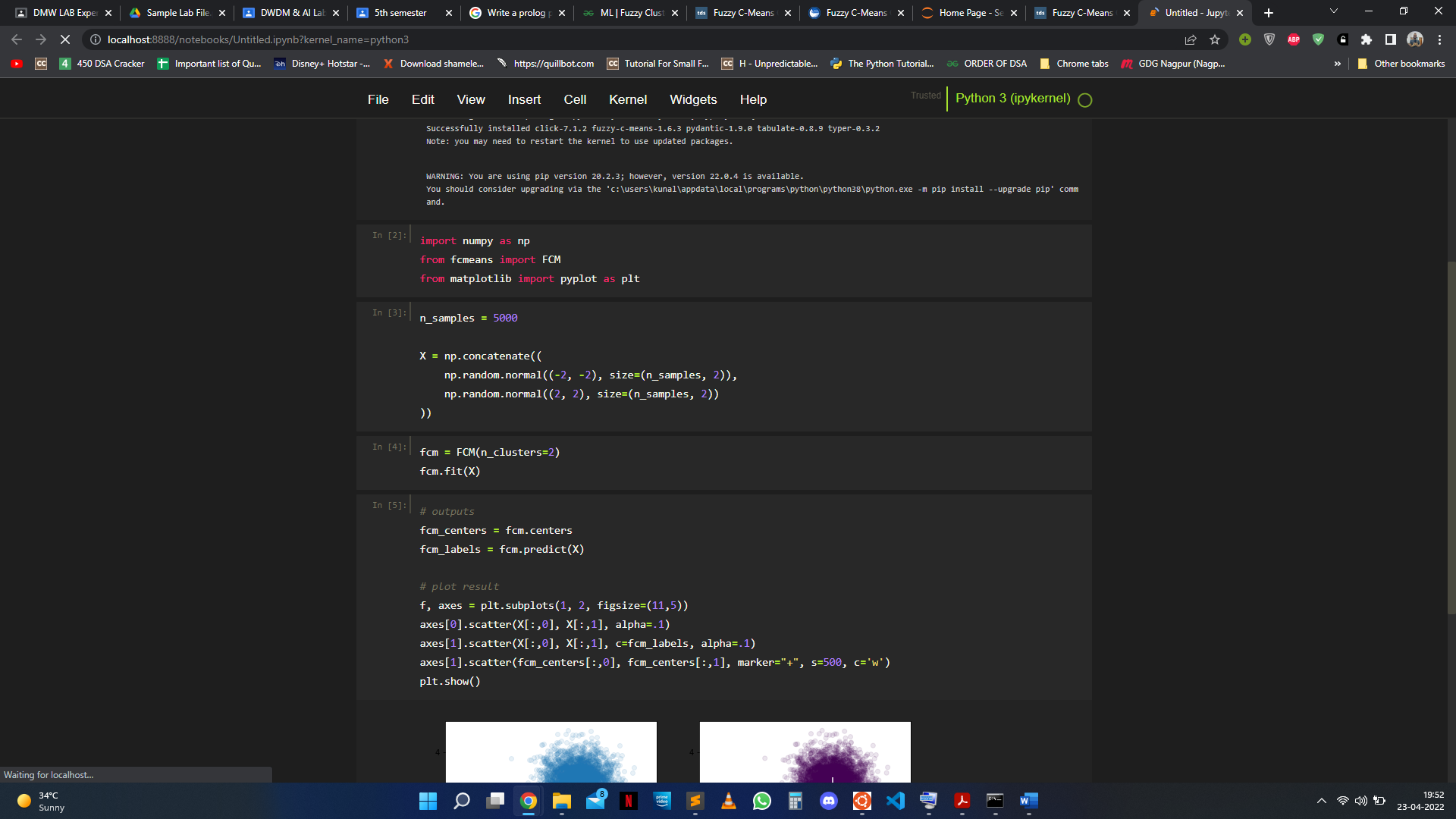
**from** matplotlib **import** pyplot **as** plt

n\_samples **=** 5000

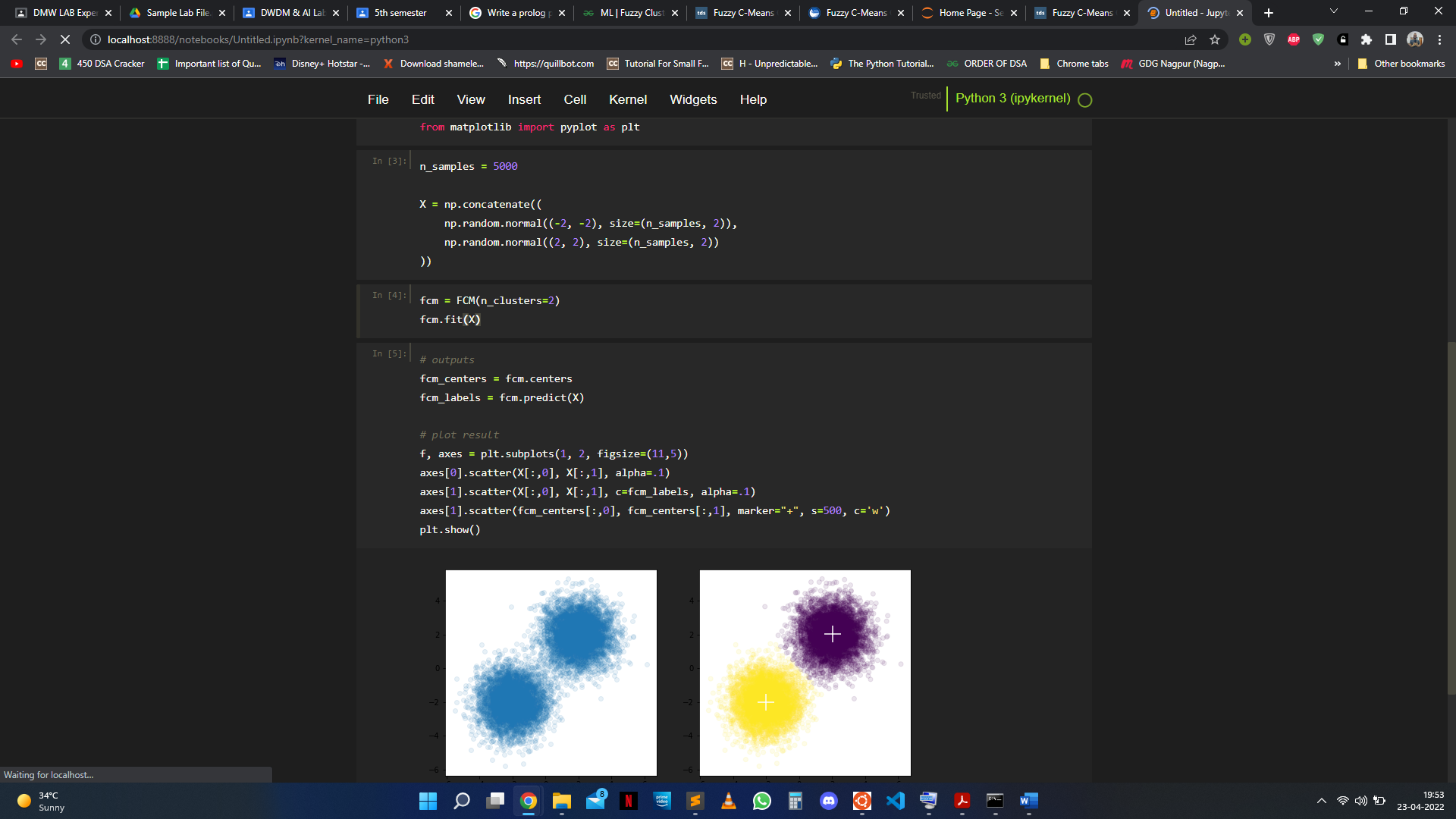
X **=** np**.**concatenate((

np**.**random**.**normal((**-**2, **-**2), size**=**(n\_samples, 2)),

np**.**random**.**normal((2, 2), size**=**(n\_samples, 2))

))

fcm **=** FCM(n\_clusters**=**2)

fcm**.**fit(X)

*# outputs*

fcm\_centers **=** fcm**.**centers

fcm\_labels **=** fcm**.**predict(X)

*# plot result*

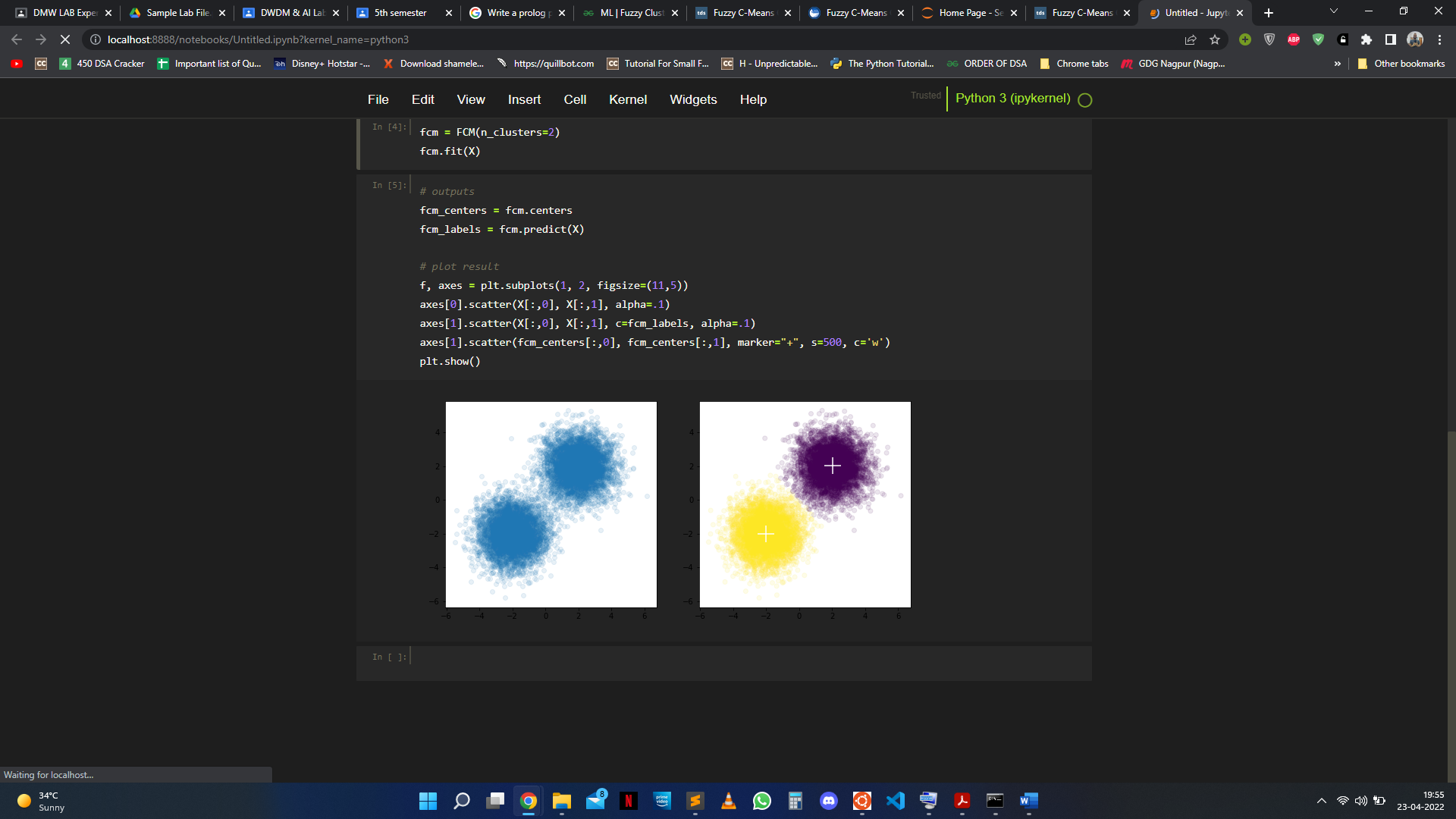
f, axes **=** plt**.**subplots(1, 2, figsize**=**(11,5))

axes[0]**.**scatter(X[:,0], X[:,1], alpha**=**.1)

axes[1]**.**scatter(X[:,0], X[:,1], c**=**fcm\_labels, alpha**=**.1)

axes[1]**.**scatter(fcm\_centers[:,0], fcm\_centers[:,1], marker**=**"+", s**=**500, c**=**'w')

plt**.**show()



# Experiment 9

**Aim:**

Write a program to perform DBSCAN clustering.

**Description:**

Density Based Spatial Clustering of Applications with Noise(DBCSAN) is a clustering algorithm which was proposed in 1996. In 2014, the algorithm was awarded the ‘Test of Time’ award at the leading Data Mining conference, KDD.

DBSCAN algorithm can be abstracted in the following steps :

Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.

* For each core point if it is not already assigned to a cluster, create a new cluster.
* Find recursively all its density connected points and assign them to the same cluster as the core point.
* A point a and b are said to be density connected if there exist a point c which has a sufficient number of points in its neighbors and both the points a and b are within the eps distance. This is a chaining process. So, if b is neighbor of c, c is neighbor of d, d is neighbor of e, which in turn is neighbor of a implies that b is neighbor of a.
* Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

Link to Dataset Used: [Credit Card Dataset](https://www.kaggle.com/datasets/arjunbhasin2013/ccdata)

**Program:**

# Step 1: Importing the required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import normalize

from sklearn.decomposition import PCA

# Step 2: Loading the data

# modify the input path as per your system path

X = pd.read\_csv('..input\_path/CC\_GENERAL.csv')

# Dropping the CUST\_ID column from the data

X = X.drop('CUST\_ID', axis = 1)

# Handling the missing values

X.fillna(method ='ffill', inplace = True)

#Step 3: Preprocessing the data

# Scaling the data to bring all the attributes to a comparable level

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Normalizing the data so that

# the data approximately follows a Gaussian distribution

X\_normalized = normalize(X\_scaled)

# Converting the numpy array into a pandas DataFrame

X\_normalized = pd.DataFrame(X\_normalized)

#Step 4: Reducing the dimensionality of the data to make it visualizable

pca = PCA(n\_components = 2)

X\_principal = pca.fit\_transform(X\_normalized)

X\_principal = pd.DataFrame(X\_principal)

X\_principal.columns = ['P1', 'P2']

Step 5: Building the clustering model

# Numpy array of all the cluster labels assigned to each data point

db\_default = DBSCAN(eps = 0.0375, min\_samples = 3).fit(X\_principal)

labels = db\_default.labels\_

# Step 6: Visualizing the clustering

# Building the label to colour mapping

colours = {}

colours[0] = 'r'

colours[1] = 'g'

colours[2] = 'b'

colours[-1] = 'k'

# Building the colour vector for each data point

cvec = [colours[label] for label in labels]

# For the construction of the legend of the plot

r = plt.scatter(X\_principal['P1'], X\_principal['P2'], color ='r');

g = plt.scatter(X\_principal['P1'], X\_principal['P2'], color ='g');

b = plt.scatter(X\_principal['P1'], X\_principal['P2'], color ='b');

k = plt.scatter(X\_principal['P1'], X\_principal['P2'], color ='k');

# Plotting P1 on the X-Axis and P2 on the Y-Axis

# according to the colour vector defined

plt.figure(figsize =(9, 9))

plt.scatter(X\_principal['P1'], X\_principal['P2'], c = cvec)

# Building the legend

plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -1'))

plt.show()

Step 7: Tuning the parameters of the model

db = DBSCAN(eps = 0.0375, min\_samples = 50).fit(X\_principal)

labels1 = db.labels\_

#Step 8: Visualizing the changes

colours1 = {}

colours1[0] = 'r'

colours1[1] = 'g'

colours1[2] = 'b'

colours1[3] = 'c'

colours1[4] = 'y'

colours1[5] = 'm'

colours1[-1] = 'k'

cvec = [colours1[label] for label in labels]

colors = ['r', 'g', 'b', 'c', 'y', 'm', 'k' ]

r = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[0])

g = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[1])

b = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[2])

c = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[3])

y = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[4])

m = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[5])

k = plt.scatter(

X\_principal['P1'], X\_principal['P2'], marker ='o', color = colors[6])

plt.figure(figsize =(9, 9))

plt.scatter(X\_principal['P1'], X\_principal['P2'], c = cvec)

plt.legend((r, g, b, c, y, m, k),

('Label 0', 'Label 1', 'Label 2', 'Label 3 'Label 4',

'Label 5', 'Label -1'),

scatterpoints = 1,

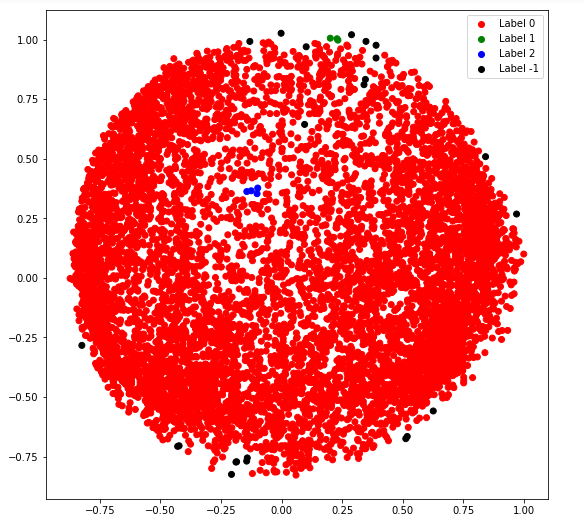
loc ='upper left',

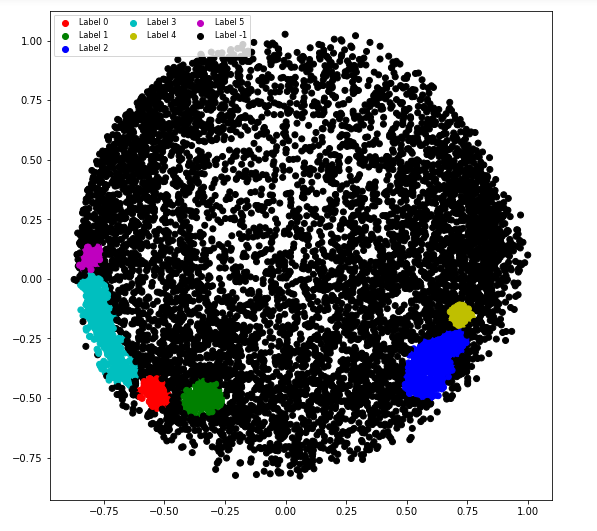
ncol = 3,

fontsize = 8)

plt.show()

**Output:**





# Experiment 10

**Aim-** Write a program for performing Linear Regression.

**Theory-**

Linear Regression is a machine learning algorithm based

on supervised learning. It performs a regression task. Regression models a

target prediction value based on independent variables.

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Mathematically, we can represent a linear regression as:

y= a0+a1x+ ε

**Here,**

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

## Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

* Simple Linear Regression:  
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
* **Multiple Linear regression:**If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

**Program-**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "m",

marker = "o", s = 30)

# predicted response vector

y\_pred = b[0] + b[1]\*x

# plotting the regression line

plt.plot(x, y\_pred, color = "g")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {} \

\nb\_1 = {}".format(b[0], b[1]))

# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output-**

**Estimated coefficients:**

**b\_0 = -0.0586206896552**

**b\_1 = 1.45747126437**

# Experiment 11

**AIM:** Perform an experiment for finding frequent item sets, confidence and support using

association rules of mining.

**THEORY:**

Apriori algorithm was given by R. Agrawal and R. Srikant in 1994 for finding frequent itemsets in a dataset for boolean association rule. Name of the algorithm is Apriori because it uses prior knowledge of frequent itemset properties. We apply an iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

To improve the efficiency of level-wise generation of frequent itemsets, an important property is used called Apriori property which helps by reducing the search space.

**Apriori Property –**

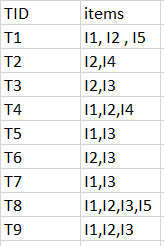
All non-empty subsets of frequent itemset must be frequent. The key concept of Apriori algorithm is its anti-monotonicity of support measure. Apriori assumes that

*All subsets of a frequent itemset must be frequent(Apriori property).*

*If an itemset is infrequent, all its supersets will be infrequent.*

**PROCEDURE:**

Consider the following dataset and we will find frequent itemsets and generate association rules for them.

****

*minimum support count is 2*

*minimum confidence is 60%*

**Step-1:** K=1

(I) Create a table containing support count of each item present in dataset – Called C1(candidate set)



(II) compare candidate set item’s support count with minimum support count(here min\_support=2 if support\_count of candidate set items is less than min\_support then remove those items). This gives us itemset L1.

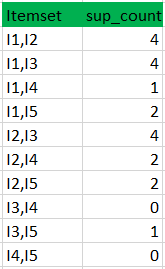
****

**Step-2: K=2**

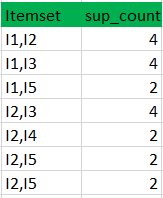
Generate candidate set C2 using L1 (this is called join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common.

Check all subsets of an itemset are frequent or not and if not frequent remove that itemset.(Example subset of{I1, I2} are {I1}, {I2} they are frequent.Check for each itemset)

Now find the support count of these itemsets by searching in the dataset.

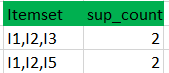


(II) compare candidate (C2) support count with minimum support count(here min\_support=2 if support\_count of candidate set item is less than min\_support then remove those items) this gives us itemset L2.

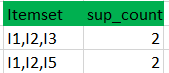
****

**Step-3:**

* Generate candidate set C3 using L2 (join step). Condition of joining Lk-1 and Lk-1 is that it should have (K-2) elements in common. So here, for L2, first element should match.So itemset generated by joining L2 is {I1, I2, I3}{I1, I2, I5}{I1, I3, i5}{I2, I3, I4}{I2, I4, I5}{I2, I3, I5}
* Check if all subsets of these itemsets are frequent or not and if not, then remove that itemset.(Here subset of {I1, I2, I3} are {I1, I2},{I2, I3},{I1, I3} which are frequent. For {I2, I3, I4}, the subset {I3, I4} is not frequent so remove it. Similarly check for every itemset)
* find the support count of these remaining itemset by searching in the dataset.



(II) Compare candidate (C3) support count with minimum support count(here min\_support=2 if support\_count of candidate set item is less than min\_support then remove those items) this gives us itemset L3.



**Step-4:**

* Generate candidate set C4 using L3 (join step). Condition of joining Lk-1 and Lk-1 (K=4) is that they should have (K-2) elements in common. So here, for L3, first 2 elements (items) should match.
* Check if all subsets of these itemsets are frequent or not (Here the itemset formed by joining L3 is {I1, I2, I3, I5} so its subset contains {I1, I3, I5}, which is not frequent). So no itemset in C4
* We stop here because no frequent itemsets are found further

Thus, we have discovered all the frequent item-sets.

Now generation of strong association rule comes into picture. For that we need to calculate the confidence of each rule.

**Confidence –**

A confidence of 60% means that 60% of the customers, who purchased milk and bread also bought butter.

So here, by taking an example of any frequent itemset, we will show the rule generation.

Itemset {I1, I2, I3} //from L3

SO rules can be

* [I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = =50%
* [I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) ==50%
* [I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) ==50%
* [I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) ==33%
* [I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = =28%
* [I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) ==33%

So if minimum confidence is 50%, then the first 3 rules can be considered as strong association rules.

**CONCLUSION:**

Apriori Algorithms can be slow. The main limitation is time required to hold a vast number of candidate sets with frequent itemsets, low minimum support or large itemsets i.e. it is not an efficient approach for a large number of datasets. For example, if there are from frequent 1- itemsets, it needs to generate more than candidates into 2-lengths which in turn they will be tested and accumulate. Furthermore, to detect frequent patterns in size 100 i.e. , it has to generate candidate itemsets that yield on costly and wasting of time of candidate generation. So, it will check for many sets from candidate itemsets, also it will scan the database many times repeatedly for finding candidate itemsets. Apriori will be very low and inefficiency when memory capacity is limited with large number of transactions.

# Experiment 12

**Objective:** Implement apriori algorithm for association rules of mining.

**Theory:** The steps for Apriori algorithm are:

• In the first iteration of the algorithm, each item is a member of the set of candidate1-itemsets, *C*1. The algorithm simply scans all the transactions to count the number of occurrences of each item. Suppose that the minimum support count required is k, that is, *min sup* = k.

• The set of frequent 1-itemsets, *L*1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support. Here, *Li* shows the database with *i* elements in each set.

• To discover the set of frequent 2-itemsets, *L*2, the algorithm uses the join *L*1 \* *L*1 to generate a candidate set of 2-itemsets*.*

• The set of frequent 2-itemsets, *L*2, is then determined, consisting of those candidate2-itemsets in *C*2 having minimum support.

• The generation of the set of the candidate3-itemsets, *C*3, is then calculated. It is based on the Apriori property that all subsets of a frequent itemset must also be frequent.

• Further, higher order candidate sets are generated and finally determine the maximum of such set that also satisfies the minimum support.

**Program:**

package lab14;

public class AssociationRule {

public static void main(String[] args) {

int totalAttrs = 3, numberTransac = 4, c1 = 0, c2 = 0;

double minsup = 0.5, sup = 0.0;

// a → b Shows the association rule between two sets.

int a[] = {1, 1, 0};

int b[] = {0, 0, 0};

int aorb[] = new int[totalAttrs];

49

int aInt = 0;

int aorbInt = 0;

int dbInt[] = new int[numberTransac];

int db[][] = {{1, 0, 1},

{0, 0, 0},

{0, 0, 0},

{1, 1, 1}};

// show data set

System.out.println("The database is: ");

for(int i = 0; i < numberTransac; i++) {

for(int j = 0; j < totalAttrs; j++) {

System.out.print(db[i][j] + " ");

}

System.out.println();

}

// show item set.

System.out.println("The relation is: ");

for(int i = 0; i < totalAttrs; i++)

System.out.print(a[i] + " ");

System.out.print("-> ");

for(int i = 0; i < totalAttrs; i++)

System.out.print(b[i] + " ");

for(int i = 0; i < totalAttrs; i++) {

aorb[i] = a[i] | b[i];

}

// convert the values to integers.

for(int i = totalAttrs - 1; i >= 0; i--)

aInt = aInt + a[i] \* (int) Math.pow(2, totalAttrs - 1 - i);

for(int i = totalAttrs - 1; i >= 0; i--)

aorbInt = aorbInt + aorb[i] \* (int) Math.pow(2, totalAttrs - 1

- i);

for(int i = 0; i < numberTransac; i++)

for(int j = totalAttrs - 1; j >= 0; j--)

dbInt[i] = dbInt[i] + db[i][j] \* (int) Math.pow(2,

50

totalAttrs - 1 - j);

// 101 & 111 => 101 i.e a.b = a means the attributes of first are

present in second.

for(int i = 0; i < numberTransac; i++)

if((aInt & dbInt[i]) == aInt)

c1 = c1 + 1;

// calculate confidence.

for(int i = 0; i < numberTransac; i++)

if((aorbInt & dbInt[i]) == aorbInt)

c2 = c2 + 1;

sup = c2 / c1;

System.out.println();

// print output.

if(sup > minsup)

System.out.println("It will be included in the itemset with

confidence: " + sup);

else

System.out.println("It won't be included in itemset with

confidence: " + sup);

}

}

**Output**

The database is:

1 0 1

0 0 0

0 0 0

1 1 1

The relation is:

1 1 0 → 0 0 0

It will be included in the itemset with confidence: 1.0

**Conclusion:**

The program shows whether the a → b present or not using the apriori algorithm and the association rules. It uses the formula:

confidence(A → B) =

# EXPERIMENT 13

**Aim:** Implement Snow-flake schema in Data warehousing.

**Theory:** The snowflake schema is a variant of the star schema. In this, the centralized fact table is connected to multiple dimensions, presented in a normalized form in multiple related tables. The snowflake structure is formed when the dimensions of a star schema are detailed and densely structured, having several levels of relationship, and the child tables have multiple parent tables. The snowflake effect influences only the dimension tables and not the fact tables.

Characteristics of Snowflake Schema:

* The major benefit is that it employs smaller disk space.
* It is easier to implement the addition of a dimension into the Schema.
* Presence of multiple tables reduces query performance.
* The primary challenge is to bear the maintenance efforts because of the larger number of lookup tables.

**Program Code:** The following DMQL program code can be used to define Snowflake schema -

DEFINE CUBE Sales [Timestamp, Product, Branch, Location]:

Dollars\_sold = sum(sales in dollars), Units\_sold = count(\*), Avg\_sales = avg(sales in dollars)

DEFINE DIMENSION Timestamp as (Time ID, day, day of week, month, quarter, year)

DEFINE DIMENSION Product as (Product ID, name, type, Supplier (Supplier ID, supplier type))

DEFINE DIMENSION Branch as (Branch ID, branch name)

DEFINE DIMENSION Location as (Location ID, City (City ID, city, state, country))

**Output:** The DMQL code generates following Snowflake Schema -

|  |
| --- |
| **SALES** |
| Time ID |
| Product ID |
| Location ID |
| Branch ID |
| Dollars\_sold |
| Units\_sold |
| Avg\_sales |
| **TIMESTAMP** |
| Time ID |
| day |
| day of week |
| month |
| quarter |
| year |
| **PRODUCT** |
| Product ID |
| name |
| type |
| Supplier ID |
| **SUPPLIER** |
| Supplier ID |
| Supplier name |
| supplier type |

|  |
| --- |
| **BRANCH** |
| Branch ID |
| branch name |
| **LOCATION** |
| Location ID |
| City name |
| City ID |
| **CITY** |
| City ID |
| city |
| state |
| country |

# Experiment 14

**Aim :** Implement Star schema in Data warehousing.

**Description :** A star schema is the elementary form of a dimensional model, in which data are organized into facts and dimensions. A fact is an event that is counted or measured, such as a sale or log in. A dimension includes reference data about the fact, such as date, item, or customer.A star schema is a relational schema where a relational schema whose design represents a multidimensional data model. The star schema is the explicit data warehouse schema. It is known as star schema because the entity-relationship diagram of this schema simulates a star, with points, diverge from a central table. The center of the schema consists of a large fact table, and the points of the star are the dimension tables.

A table in a star schema which contains facts and connected to dimensions. A fact table has two types of columns: those that include fact and those that are foreign keys to the dimension table. The primary key of the fact tables is generally a composite key that is made up of all of its foreign keys.

A dimension is an architecture usually composed of one or more hierarchies that categorize data. If a dimension has not got hierarchies and levels, it is called a flat dimension or list. The primary keys of each of the dimensions table are part of the composite primary keys of the fact table. Dimensional attributes help to define the dimensional value. They are generally descriptive, textual values. Dimensional tables are usually smaller in size than the fact table.

Fact tables store data about sales while dimension tables data about the geographic region (markets, cities), clients, products, times, channels.

**Dmql :**

DEFINE CUBE Fact [CityID, CountryID, EmpID, ProductID]:

TotalQuantity = count(UnitSold), TotalSales = sum(ProductSold).

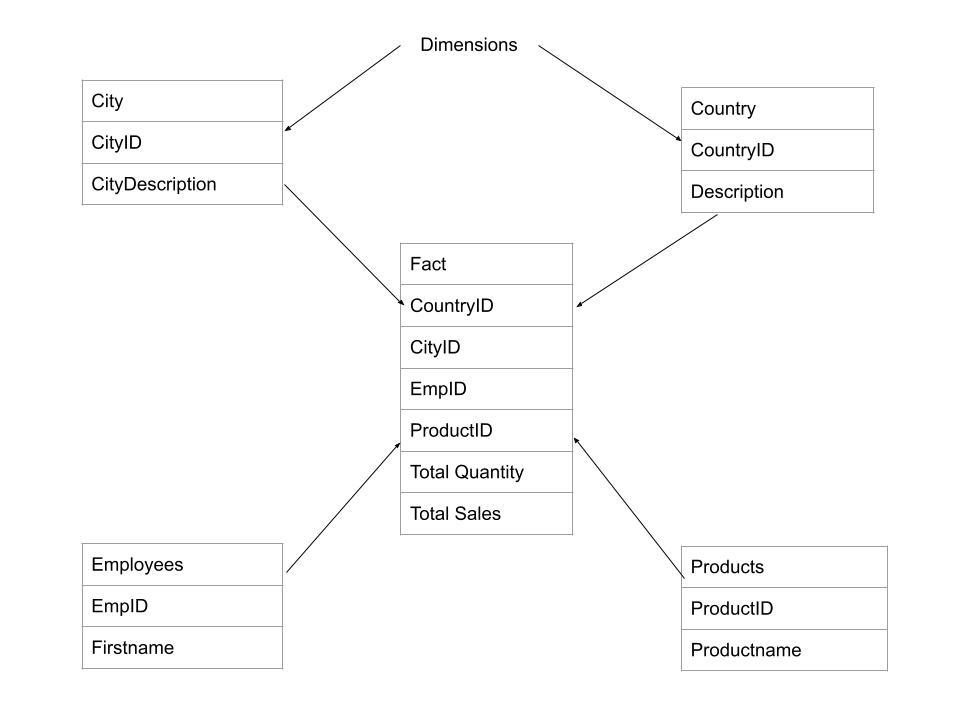
DEFINE DIMENSION City AS (CityID, CityDescription).

DEFINE DIMENSION Country AS (CountryID, Description).

DEFINE DIMENSION Employees AS (EmpID, Firstname).

DEFINE DIMENSION Products AS (ProductID, Productname).

**Output :**

****

**PRACTICAL-15**

Aim: Write a prolog program to find the rules for parent, child, male, female, son, daughter, brother, sister, uncle, aunt, ancestor given the facts about father and wife only.

Description: PROLOG stands for PROgramming in LOGic and has a very important role in artificial intelligence. It expresses any objects in the form of relations which are described using facts and rules. In this program we derive the different blood relations among the family members, and some logics like male, female and parent are given as facts while the relations between these like mother, father, brother, sister are described using the rules.

Program:

female(pam).

female(liz).

female(pat).

female(ann).

male(jim).

male(bob).

male(tom).

male(peter).

parent(pam,bob).

parent(tom,bob).

parent(tom,liz).

parent(bob,ann).

parent(bob,pat).

parent(pat,jim).

parent(bob,peter).

parent(peter,jim).

mother(X,Y):- parent(X,Y),female(X).

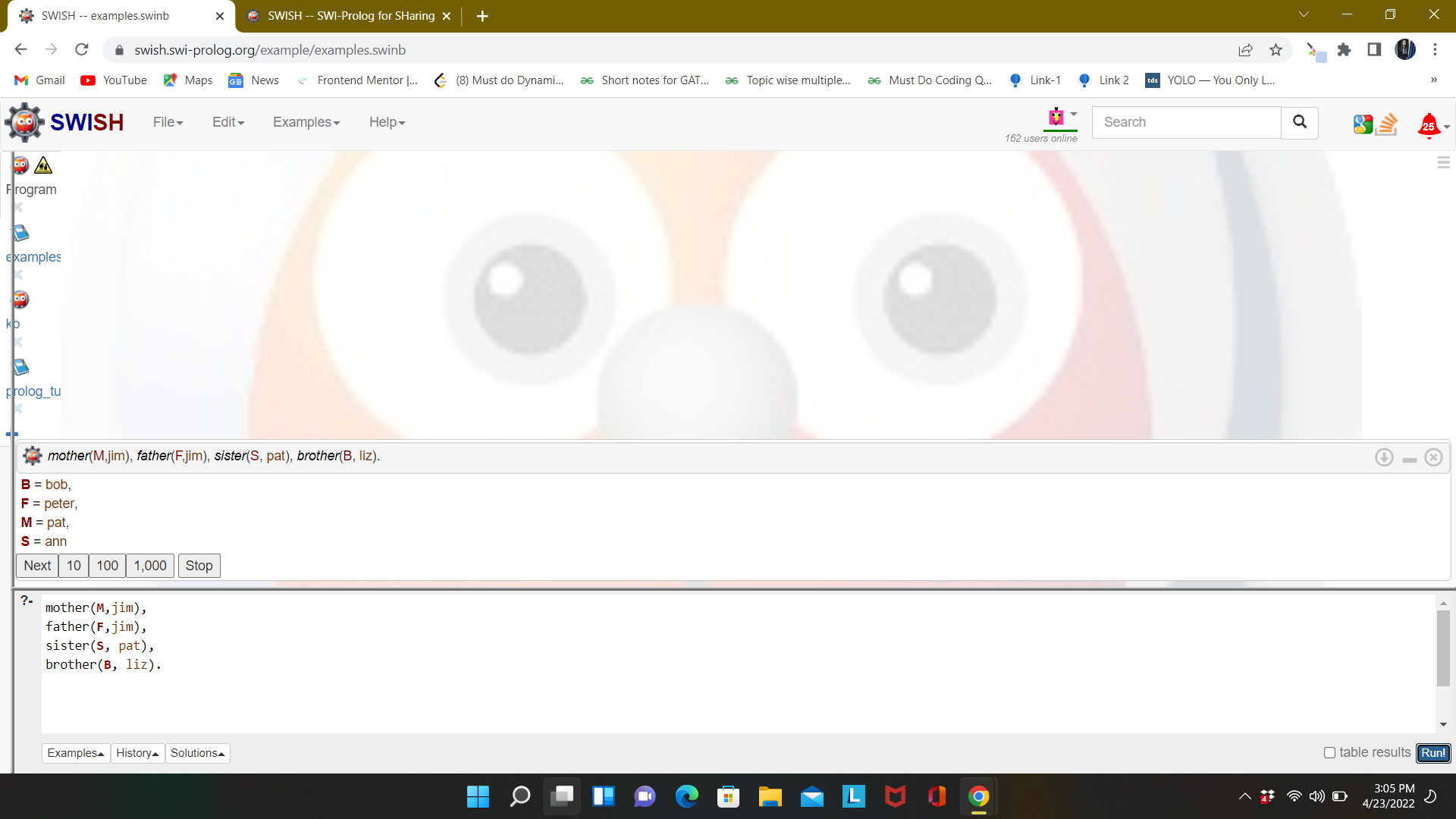
father(X,Y):- parent(X,Y),male(X).

haschild(X):- parent(X,\_).

sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.

brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.

Output:



**Experiment-16**

**Aim** - Write a prolog program to find the length of a given list.

**Theory -**

Length calculation is used to determine the length of list L. Suppose the noun name is list\_length (L, N). This takes L and N as the input counter. This will list the elements in list L and put N in their number. As was the case with our previous relationships which included lists, it is helpful to consider two situations -

* If the list is empty, the length is 0.
* If the list is empty, it means L = [Head | Tail], then its length is 1+ tail length.

**Program -**

list\_length([],0).

list\_length([\_|TAIL],N) :- list\_length(TAIL,N1), N is N1 + 1.

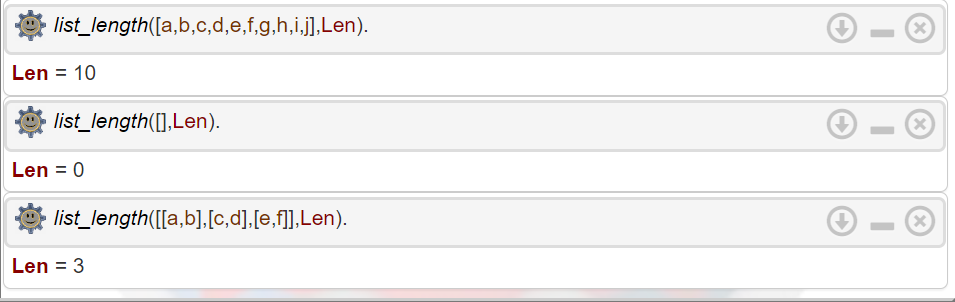
**Query -**

list\_length([a,b,c,d,e,f,g,h,i,j],Len).

list\_length([],Len).

list\_length([[a,b],[c,d],[e,f]],Len).

**Output -**



**Experiment-17**

**Aim** - Write a prolog program to find the last element of a given list.

**Theory -**

Prolog is a [logic programming](https://en.wikipedia.org/wiki/Logic_programming) language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations.

X is a member of list X.

Removes the first element of the list recursively.

Returns the last element of the list.

**Program -**

lastElement(X,[X]).

lastElement(X,[Y|Z]):- lastElement(X,Z).

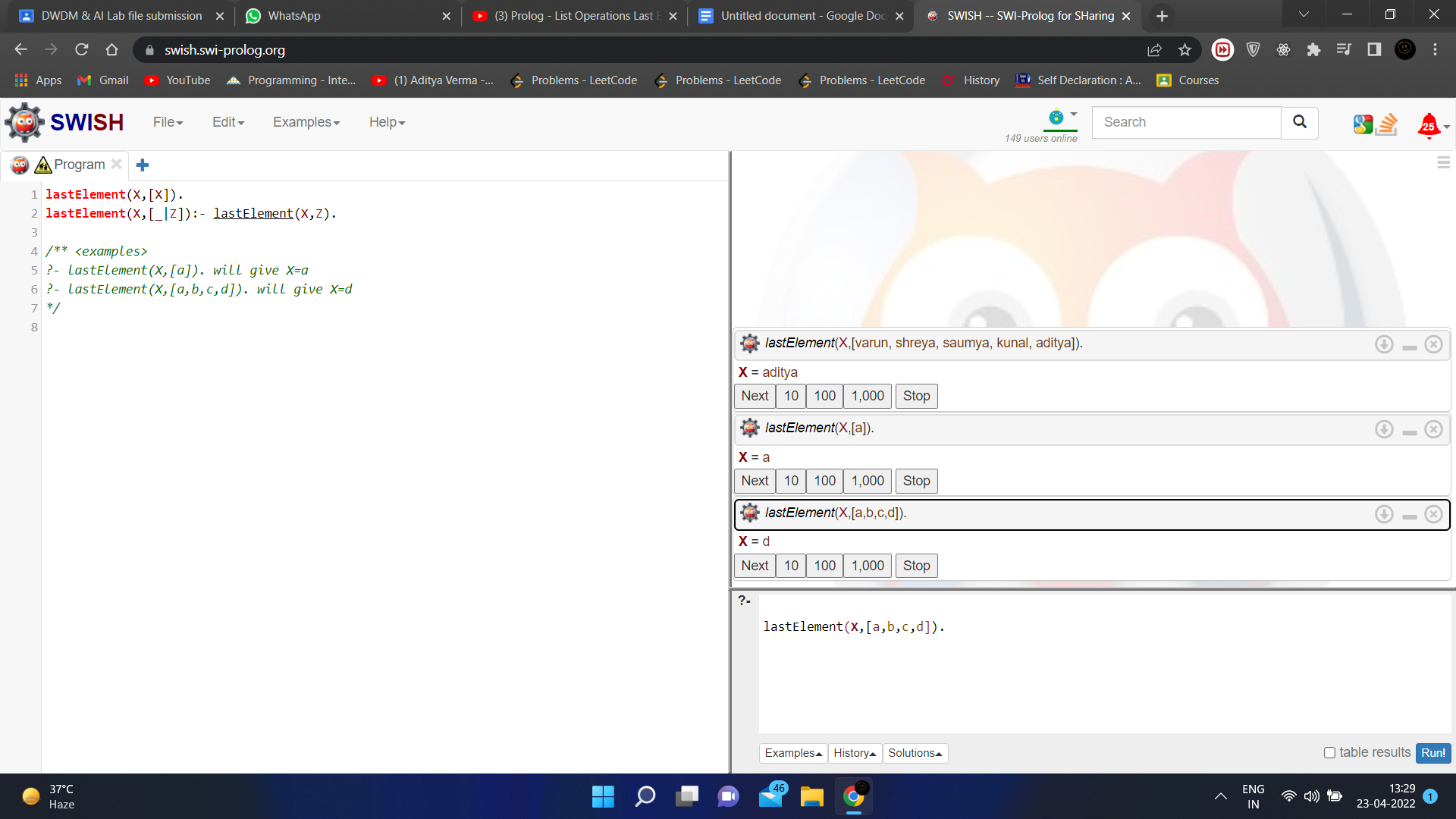
**Query -**

lastElement(X,[a,b,c,d]).

lastElement(X,[a]).

lastElement(X,[varun, shreya, saumya, kunal, aditya]).

**Output -**



**Experiment– 18**

**Aim:**

Write a prolog program to delete the first occurrence and also all occurrences of a particular

element in a given list.

**Description:**

For deleting single occurrence:

Suppose we have a list L and an element X, we have to delete X from L. So there are three cases −

* If X is the only element, then after deleting it, it will return empty list.
* If X is head of L, the resultant list will be the Tail part.
* If X is present in the Tail part, then delete from there recursively.

For deleting all occurrences:

When the “input” list is empty, then the output list is “empty”.

delete\_all(\_Head, [], []).

When the heads of the “input” and “output” lists don’t match

the element being deleted:

delete\_all(Item, [Head|Tail], [Head|New\_Tail]) :-

Item \= Head, delete\_all(Item, Tail, New\_Tail).

When the heads of the “input” list matches the element being

deleted:

delete\_all(Item, [Item|Tail], New\_Tail) :-

delete\_all(Item, Tail, New\_Tail).

**Program 1:**

(Program to delete first occurrence of an element)

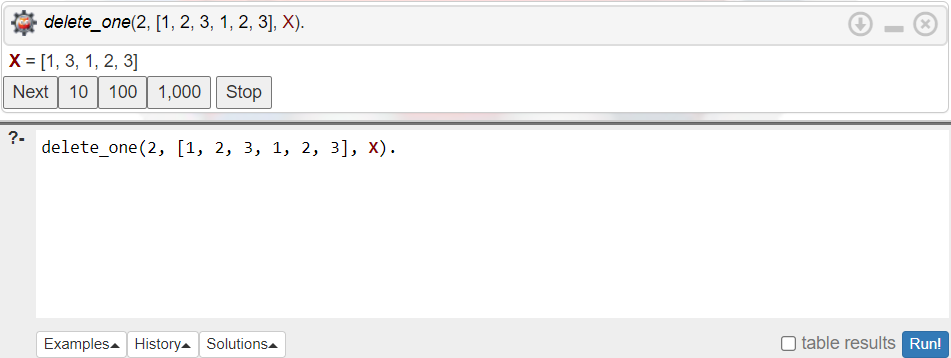
delete\_one(\_, [], []).

delete\_one(Term, [Term|Tail], Tail).

delete\_one(Term, [Head|Tail], [Head|Result]) :-

delete\_one(Term, Tail, Result).

**Output 1:**

****

**Program 2:**

(Program to delete all occurrences of an element)

delete\_all(\_Head, [], []).

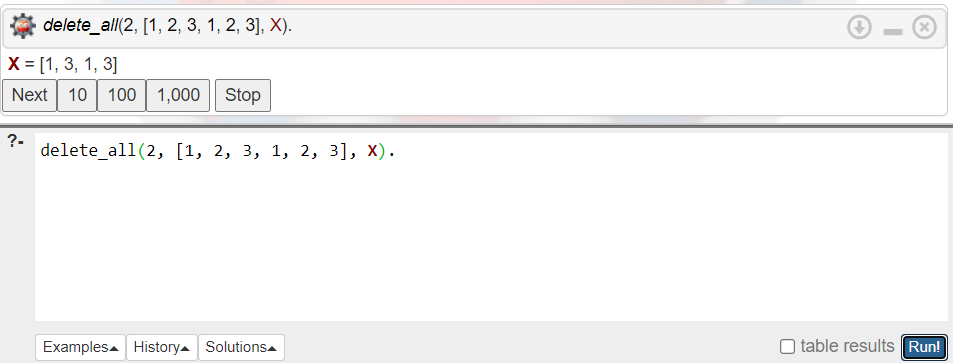
delete\_all(Item, [Head|Tail], [Head|New\_Tail]) :-

Item \= Head, delete\_all(Item, Tail, New\_Tail).

delete\_all(Item, [Item|Tail], New\_Tail) :-

delete\_all(Item, Tail, New\_Tail).

**Output 2:**

****

**Experiment-19**

**Aim:** Write a prolog program to find the union of two given sets represented as lists.

**Description:**

(Y is list1, Z is list2 and W is new list)

• For each element of list1 which is already present in list2 :

union([X|Y],Z,W) :- member(X,Z), union(Y,Z,W).

• For each element of list1 which is not present in list2:

union([X|Y],Z,[X|W]) :- \+ member(X,Z), union(Y,Z,W).

• Copy all the values of list2 into the new list.

union([],Z,Z).

**Program:**

union([X|Y],Z,W) :-

list\_member(X,Z), union(Y,Z,W).

union([X|Y],Z,[X|W]) :-

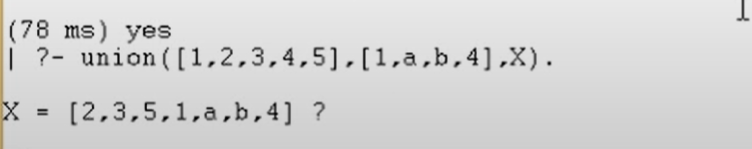
\+ list\_member(X,Z), union(Y,Z,W).

union([],Z,Z). list\_member(X,[X|\_]).

list\_member(X,[\_|TAIL])

list\_member(X,TAIL).

**Output:**



**Expermient: 20**

**Aim:** Write a prolog program to reverse a given list of values.

**Description:**

The first parameter in the reverse/3 predicate is the list. The second parameter is an empty list. The third parameter is the reverse list.

The reverse/3 recursively pushes the elements from the beginning of the first list to the front of the second list. This reverses the order of the elements.

Our base condition is reverse([], Y, R). At this point, we have pushed all elements from the input list to Y. We set R to Y and backtrack.

**Code:**

reverse([], Y, R) :-

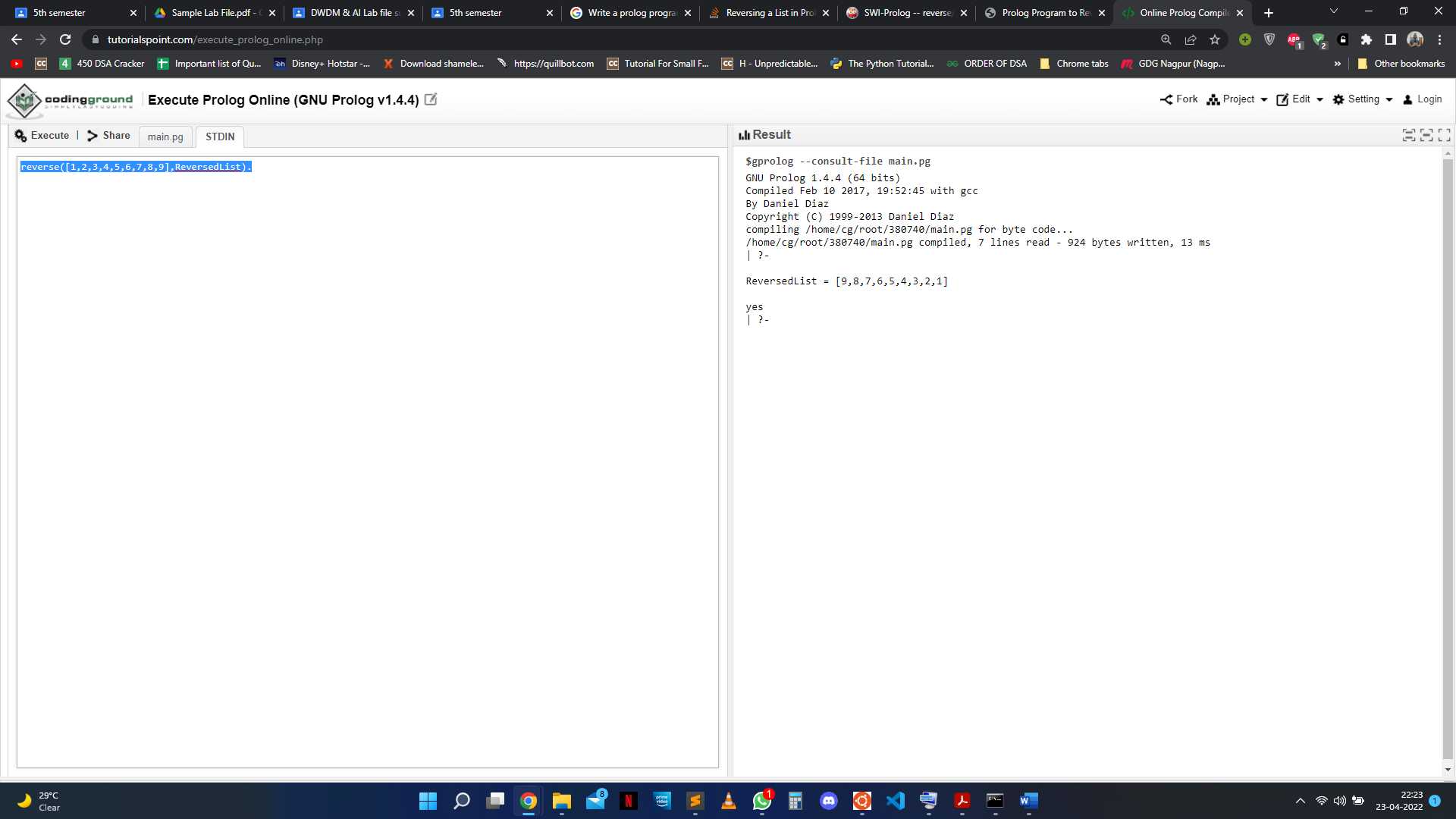
R = Y.

reverse([H|T] , Y, R) :-

reverse(T, [H|Y], R).

**STDIN:**

reverse([1,2,3,4,5,6,7,8,9],[],ReversedList).

**Output:**

**EXPERIMENT – 21**

**AIM –**

Write a prolog program given the knowledge base, If x is on the top of y, y supports x. If x

is above y and they are touching each other, x is on top of y. A cup is above a book. The cup

is touching that book. Convert the following into wffs, clausal form; Is it possible to deduce

that ‘The book supports the cup’.

**Description –**

In this prolog program, we have to find whether it is possible or not to deduce that ‘The book supports the cup’ with the help of the given knowledge base.

A cup is above a book.

above(cup,book).

The cup is touching that book.

touch(cup,book).

If x is on the top of y, y supports x. If x is above y and they are

touching each other, x is on top of y.

support(X,Y) :-above(X,Y), touch(X,Y).

Prolog Code-

:- initialization(main).

above(cup,book).

touch(cup,book).

support(Y,X) :- above(X,Y), touch(X,Y).

Input –

support(book, cup).

Output –



PROGRAM-22

Aim: Write a prolog program given the knowledge base, If Town x is connected to Town y by

highway z and bikes are allowed on z, you can get to y from x by bike. If Town x is

connected to y by z then y is also connected to x by z. If you can get to town q from p and

also to town r from town q, you can get to town r from town p.

a. Town A is connected to Town B by Road1.

b. Town B is connected to Town C by Road2.

c. Town A is connected to Town C by Road3.

d. Town D is connected to Town E by Road4.

e. Town D is connected to Town B by Road5.

f. Bikes are allowed on roads 3, 4, 5.

Bikes are only either allowed on Road1 or on Road2 every day. Convert the following

into WFF’s, clausal form and deduce that ‘One can get to town B from town D’.

Description: In this program, we express the given information as facts and then derive rules for conditions of connection, reachability and roads as given in the aim. After deriving the facts and the rules we check whether or not town B can be reached through town D; if the result comes out to be true, then it is reachable, else it is not.

Program:

connect(A,B,1).

connect(B,C,2).

connect(A,C,3).

connect(D,E,4).

connect(D,B,5).

connect(P,R,\_):- connect(P,Q,\_), connect(Q,R,\_).

bikeallowed(3).

bikeallowed(4).

bikeallowed(5).

bikeallowed(1) :- not(bikeallowed(2)).

bikeallowed(2) :- not(bikeallowed(1)).

cango(X,Y,Z) :- (connect(X,Y,Z) ; connect(Y,X,Z)), bikeallowed(Z).

Output:



**Experiment-23**

**Aim :** Solve the classical Monkey Banana problem of AI using prolog.

**Description :** The monkey can climb up on the block if both the monkey and the block are on the floor and the block is in the centre. As a result, the monkey's vertical position will be altered. The monkey can get the bananas while he is on the block and the block is in the centre.

We have some predicates that will move from one state to another state, by performing action :

* When the block is in the middle, and the monkey is on top of the block, and the monkey does not have the banana (i.e. has no state), then using the grasp action, it will change from has not state to state.
* From the floor, it can move to the top of the block (i.e. on top state), by performing the action climb.
* The push or drag operation moves the block from one place to another.
* Monkey can move from one place to another using walk or move clauses.

Another predicate will be canget(). Here we pass a state, so this will perform a move predicate from one state to another using different actions, then perform canget() on state 2. When we have reached the state ‘has>’, this indicates ‘has banana’. We will stop the execution.

**Program :**

on(floor,monkey).

on(floor,chair).

in(room,monkey).

in(room,chair).

in(room,banana).

at(ceiling,banana).

strong(monkey).

grasp(monkey).

climb(monkey,chair).

push(monkey,chair):-

strong(monkey).

under(banana,chair):-

push(monkey,chair).

canreach(banana,monkey):-

at(floor,banana);

at(ceiling,banana),

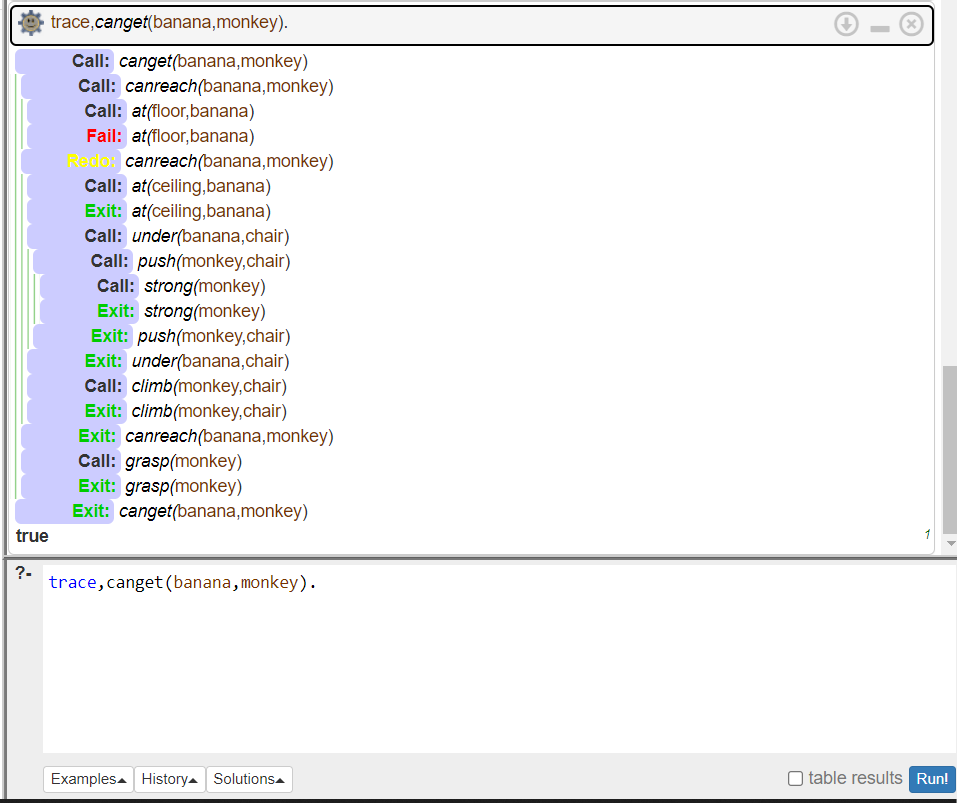
under(banana,chair),

climb(monkey,chair).

canget(banana,monkey):-

canreach(banana,monkey),grasp(monkey).

**Output :**

****

**Experiment- 24**

**Objective:** Define a LISP function to compute sum of squares.

**Theory:** SUM OF SQUARE

Sum of square means by adding the squares of two numbers.

Example:

Sum of Square of x ,y= ( x \* x ) + ( y \* y )

Now in prefix expression it should be written as,

( + ( \* x x ) ( \* y y ) )

**Code:**

|  |
| --- |
| >>(defun sumsqr(x y)  (+(\* x x)(\* y y)))  SUMSQR  >>(sumsqr 2 3); |

**Output:**

13

Conclusion:

We defined a LISP function to compute sum of square and test it on various values.

**Experiment-25**

**Aim:** Define a LISP function to compute the difference of squares. (if x > y return x2– y2, otherwise return y2 – x2).

**Description: If:** The if construct has various forms. In the simplest form, it is followed by a test clause, a test action, and some other consequent action(s). If the test clause evaluates to true, then the test action is executed otherwise, the consequent clause is evaluated.

**Source Code:**

(defun diff(x y)

(if (> x y)

(setq result (- (\* x x) (\* y y)))

(setq result (- (\* y y) (\* x x)))

)

(format t "difference is : ~d" result)

)

(diff 8 6)

**Output:**



Experiment – 26

**Objective:** Define a LISP function to compute the factorial of a given number.

**Theory:** The factorial of any non-negative number n, denoted by **n!**, is the product of all positive integers less than or equal to n.

For example 3! = 3\*2\*1.

The value of 0! is 1, according to the conversion for an empty product.

**Source Code:**

(defun fact(x)

(setq result 1)

(loop

(setq result (\* result x))

(setq x (- x 1))

(when (= x 1)(return))

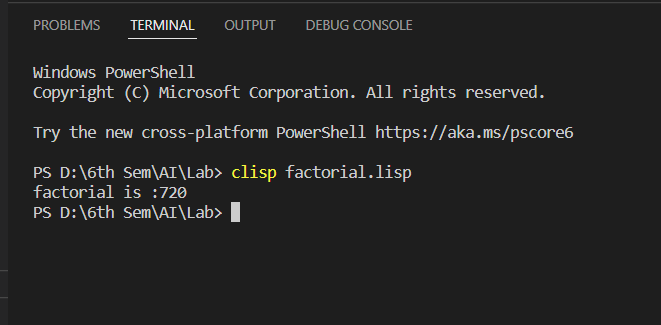
)

(format t "factorial is :~d " result)

)

(fact 6)

**Output:**



**Experiment-27**

**Aim** - Define a LISP function to reverse the number entered as parameter in function call.

**Theory -**

**Program -**

(defun read-number () (format t "Enter a number: ~%") (read))

(defun reverse-string (the-string)

(if (eq (length the-string) 0)

""

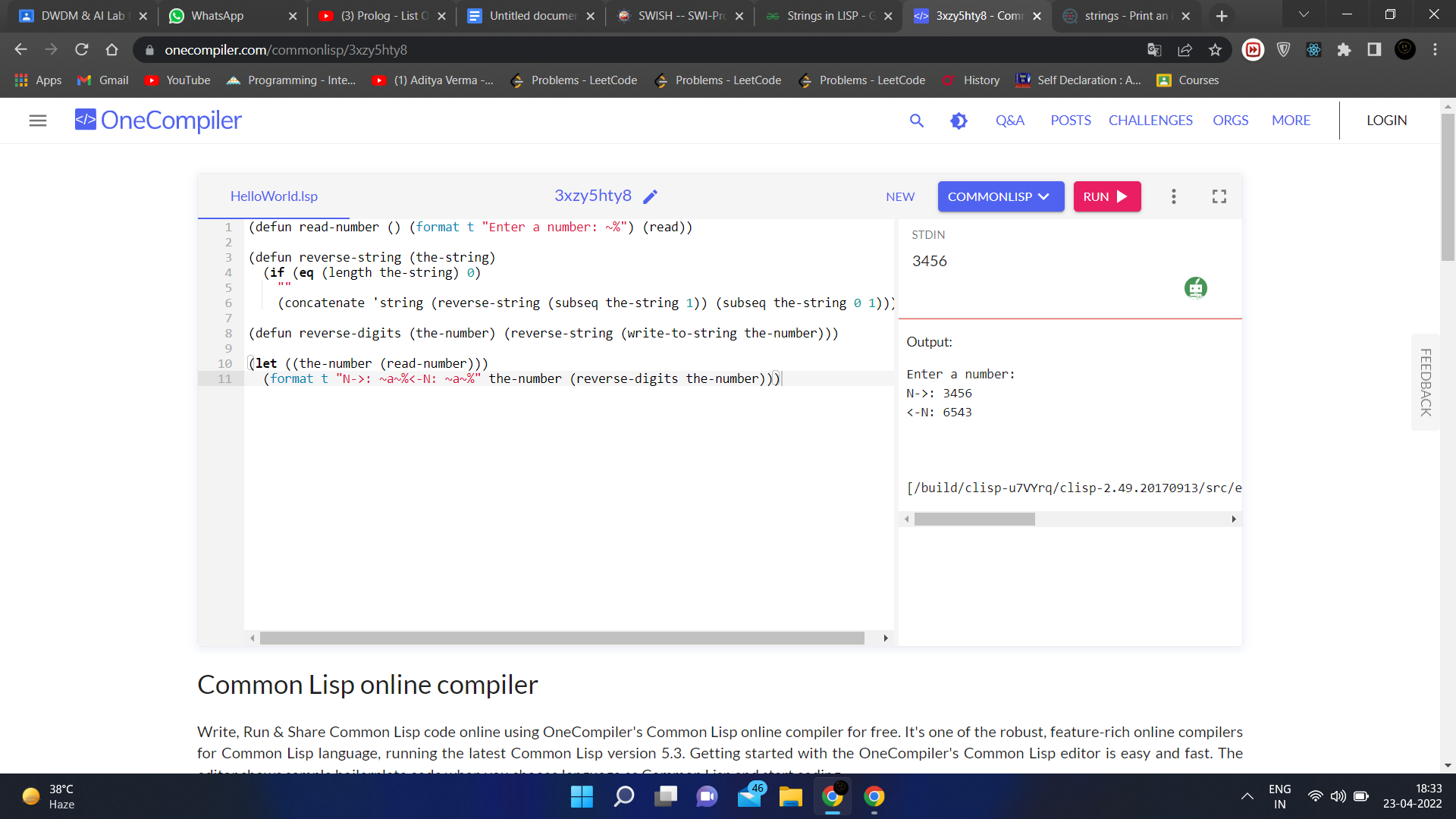
(concatenate 'string (reverse-string (subseq the-string 1)) (subseq the-string 0 1))))

(defun reverse-digits (the-number) (reverse-string (write-to-string the-number)))

(let ((the-number (read-number)))

(format t "N->: ~a~%<-N: ~a~%" the-number (reverse-digits the-number)))

**Output -**



**Experiment-28**

**Aim:** Write a LISP program containing two functions: one to read input values and one to display them.

**Description:** In LISP, we can read input from STDIN using ‘read’ and output values into STDOUT using ‘write’. This has been demonstrated in the following program which calculates the area of a circle by taking radius value from STDIN.

**Program:**

; the function AreaOfCircle

; calculates area of a circle

; when the radius is input from keyboard

(defun AreaOfCircle()

(terpri)

(princ "Enter Radius: ")

(setq radius (read))

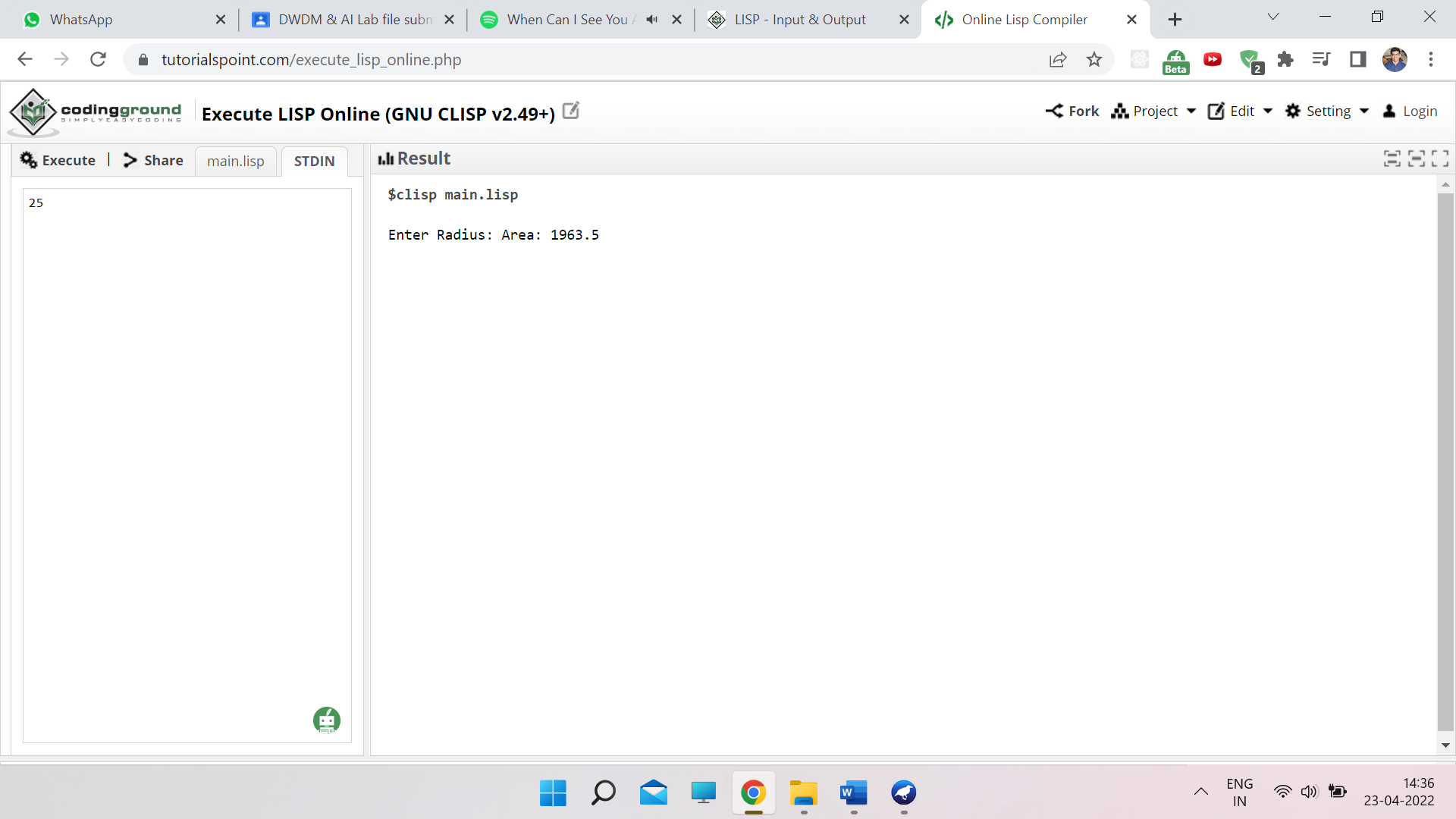
(setq area (\* 3.1416 radius radius))

(princ "Area: ")

(write area))

(AreaOfCircle)

**Output:**



**Experiment -29**

**Aim:** Write a LISP program to compute factorial of a given number using recursion.

**Theory:** The factorial of a non-negative integer n (denoted by n!), is the product of all positive integers less than or equal to n.

Given by: n! =

The program code employs recursion to calculate the factorial, by using conditional statements to check the value of n. The function is recalled when n is greater than 0.

**Program Code:** The LISP Code will be as follows-

(defun factorial(n)

(if (= n 0) 1

(\* n (factorial(- n 1)))

)

)

(write-line "Factorial of 5 is: ")

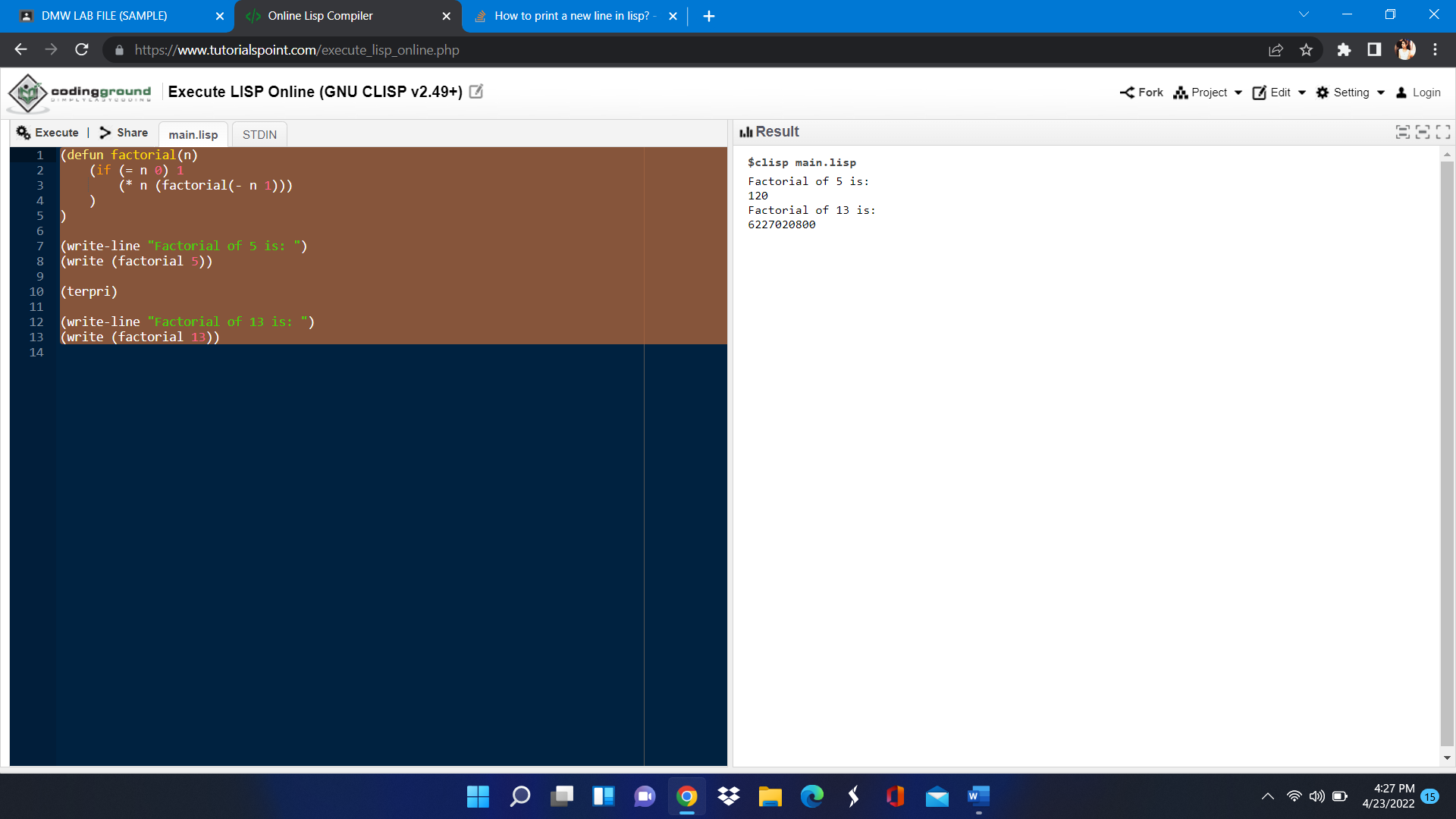
(write (factorial 5))

(terpri)

(write-line "Factorial of 13 is: ")

(write (factorial 13))

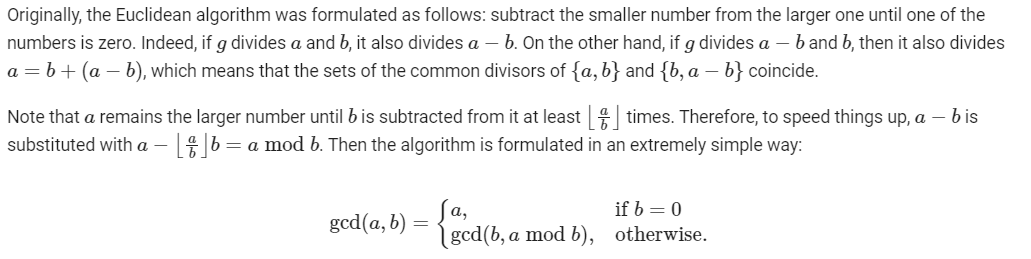
**Output:**



**Experiment-30**

**Aim**-Write a LISP Program using recursion to perform GCD of two numbers entered by user.

**Description**—Euclidean algorithm for computing the greatest common divisor



**Program**-

;;; Common Lisp implementation of GCD

;;; -------------------------------- The gcd function in LISP.

(defun gcd (a b)

(if (= b 0)

a

(gcd b (mod a b))))

;;; --------------------------------------------- Calling gcd

(defun gcdprint(a b)

(let (

(f (list "The gcd of " a " and " b " is " (gcd a b)))

)

(mapcar 'princ f)

(terpri)

)

)

;;; --------------------------------------------- Top-level

(defun run()

(princ "Greatest Common Divisor")

(terpri)

(loop

(princ "Enter two integers (0 0 to end): " )

(setq n1 (read)) ; A global. This is non-functional usage.

(setq n2 (read)) ; There is no need to use assignment in Lisp.

(cond ((= n1 0) (return))

(T (gcdprint n1 n2))

)

)

)

(run )

**Input**-

25

61

10

20

100

100

0

**Output**-

