MEDI-CAPS UNIVERSITY

Academic year 2023-24

Department of Computer Science and Engineering

Machine Learning Lab File

(CS3EL15)

Submitted To:

Mr. Vivek Gupta

Submitted By:

Ajinkya Namra

EN21CS301042

6-CSE- 'A'

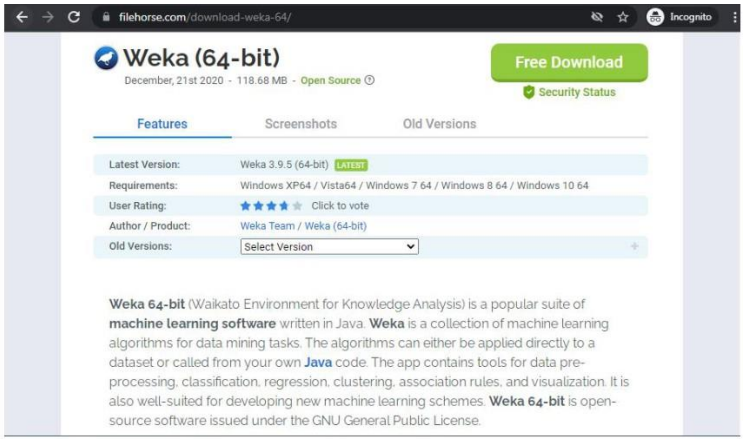
EXPERIMENT NO -1

AIM :
Install WEKA on your system and brief the below points
1. What is WEKA
2. Advantages & Disadvantages of WEKA
3. Minimum Hardware Requirement
4. Installation steps

THEORY:
Weka stands for Waikato Environment for Knowledge Analysis, it is software that is used in the data science field for data mining. It is free software. It is written in Java hence it can be run on any system supporting Java, so weka can be run on different operating systems like Windows, Linux, Mac, etc. Weka provides a collection of visualization tools that can be used for data analysis, cleaning, and predictive modeling. Weka can perform a number of tasks like data preprocessing, clustering, classification, regression, visualization, and feature selection.

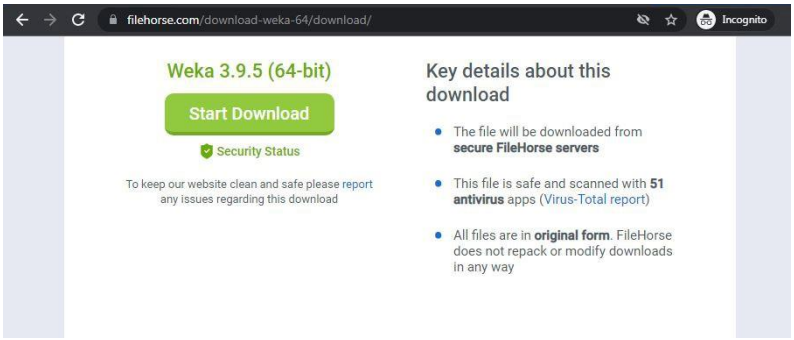
Installing Weka on Windows:
Follow the below steps to install Weka on Windows:

Step 1: Visit this website using any web browser. Click on Free Download.

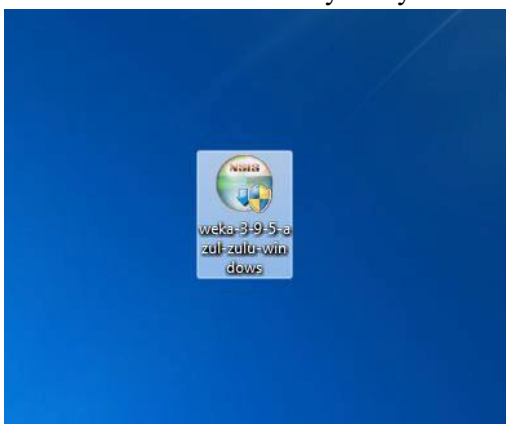


Weka download page

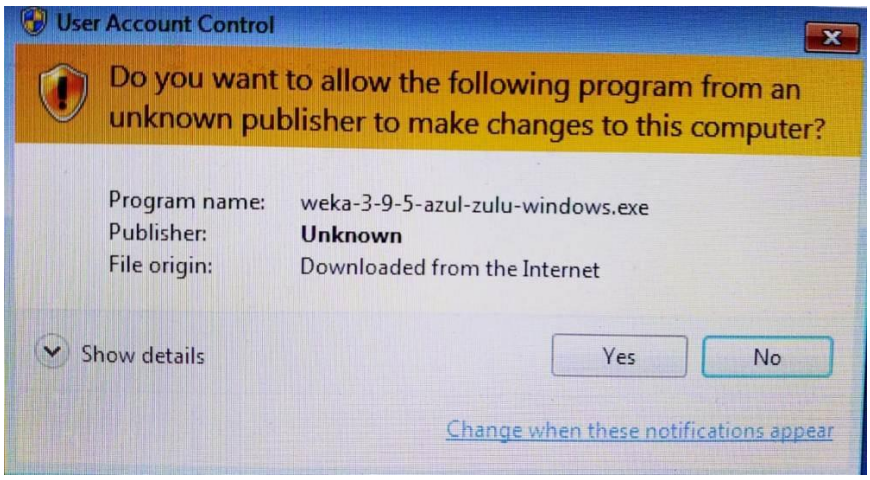
Step 2: It will redirect to a new webpage, click on Start Download. Downloading of the executable file will start shortly. It is a big 118 MB file that will take some minutes.



Step 3: Now check for the executable file in downloads in your system and run it.

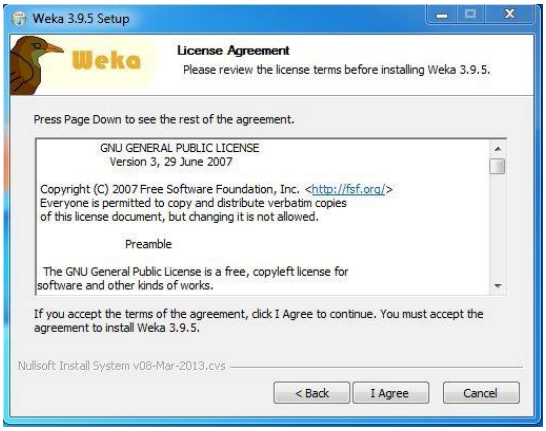


Step 4: It will prompt confirmation to make changes to your system. Click on Yes.

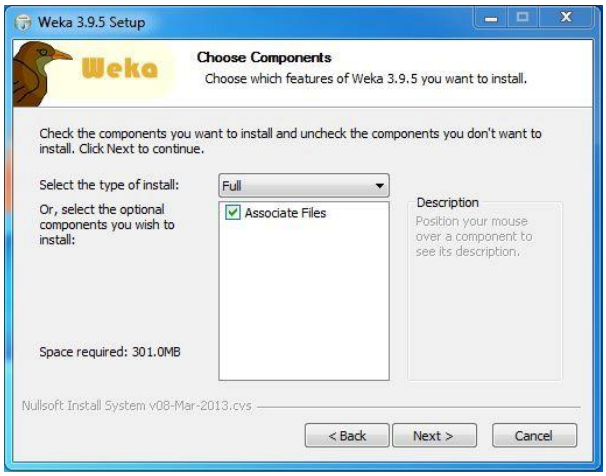


Step 5: Setup screen will appear, click on Next.

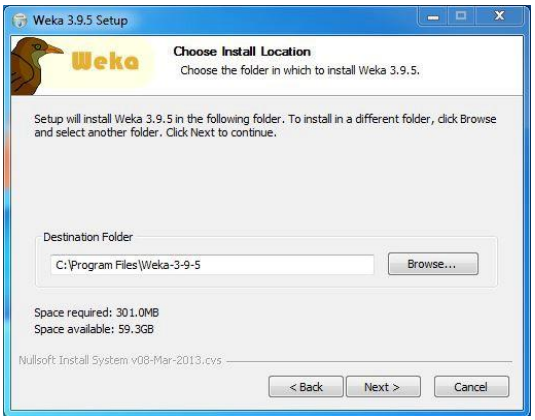
Step 6: The next screen will be of License Agreement, click on I Agree.



Step 7: Next screen is of choosing components, all components are already marked so don't change anything just click on the Install button.



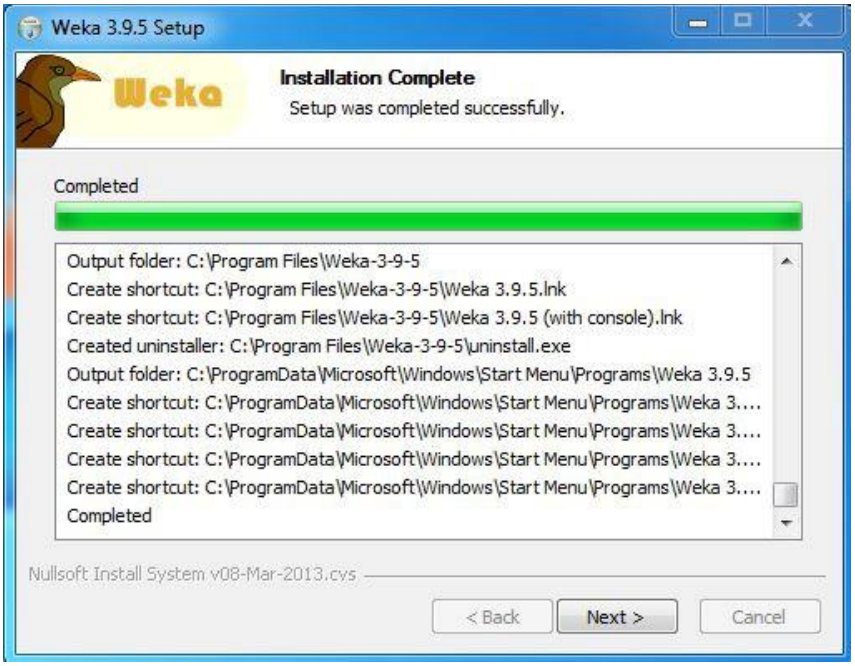
Step 8: The next screen will be of installing location so choose the drive which will have sufficient memory space for installation. It needed a memory space of 301 MB.



Step 9: Next screen will be of choosing the Start menu folder so don't do anything just click on Install Button.

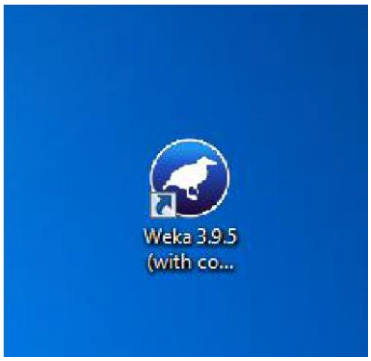
Step 10: After this installation process will start and will hardly take a minute to complete the installation.

Step 11: Click on the Next button after the installation process is complete.



Step 12: Click on Finish to finish the installation process.

Step 13: Weka is successfully installed on the system and an icon is created on the desktop.



Step 14: Run the software and see the interface.

Congratulations!! At this point, you have successfully installed Weka on your windows system.

EXPERIMENT NO- 2

AIM:

(a) Install Anaconda Distribution on the Windows or ubuntu operating system.
(b) Get familiarized with arff file format. Create an arff file on your system and save in the WEKA installed drive of your system.

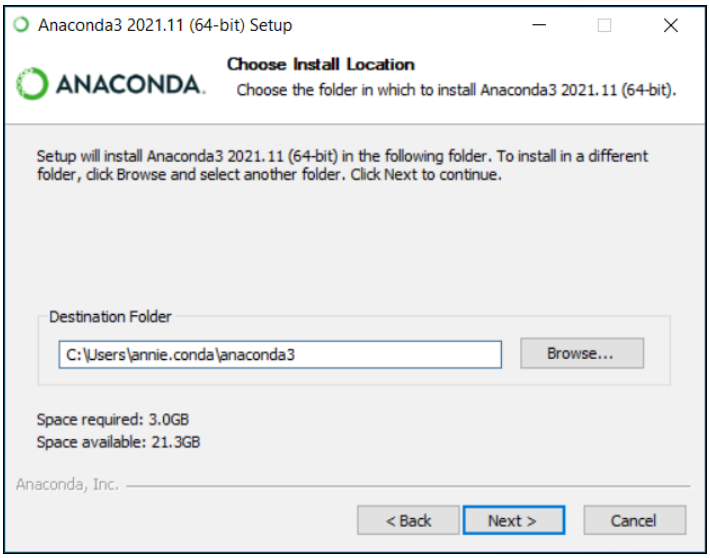
THEORY:

Installing on Windows
More of a visual learner? Watch the Installing Anaconda (Windows) video in the course linked below!

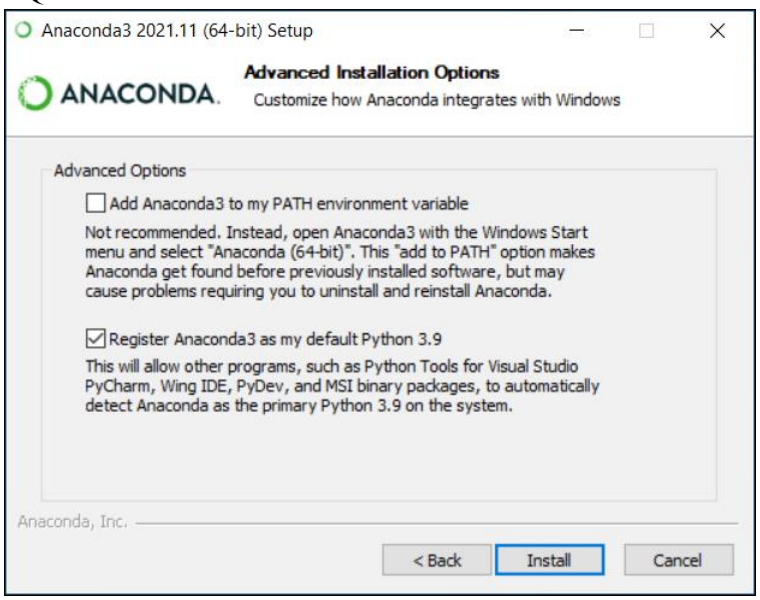
Installation

1. Download the Anaconda installer.
   (Optional) Anaconda recommends verifying the integrity of the installer after downloading it.

2. Go to your Downloads folder and double-click the installer to launch. To prevent permission errors, do not launch the installer from the Favorites folder.
3. If you encounter issues during installation, temporarily disable your anti-virus software during install, then re-enable it after the installation concludes. If you installed it for all users, uninstall Anaconda and re-install it for your user only.
4. Click Next.
5. Read the licensing terms and click I Agree.
6. It is recommended that you install for Just Me, which will install Anaconda Distribution to just the current user account. Only select an install for All Users if you need to install for all users' accounts on the computer (which requires Windows Administrator privileges).
7. Click Next.



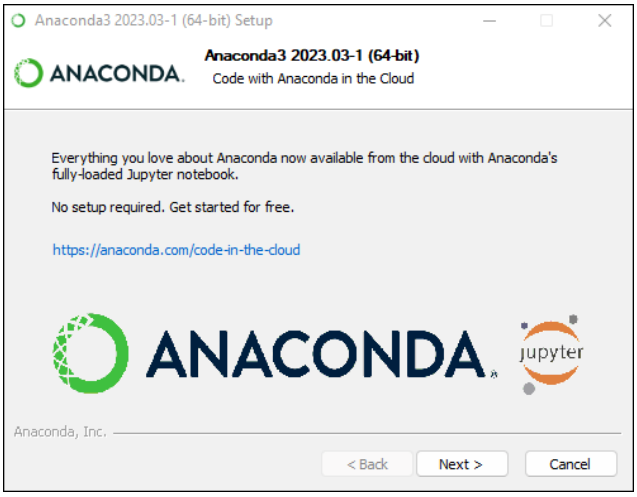
8. Select a destination folder to install Anaconda and click Next. Install Anaconda to a directory path that does not contain spaces or unicode characters. For more information on destination folders, see the FAQ.



9. Do not install as Administrator unless admin privileges are required.
10. Choose whether to add Anaconda to your PATH environment variable or register Anaconda as your default Python. We don't recommend adding Anaconda to your PATH environment variable, since this can interfere with other software. Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.
11. As of Anaconda Distribution 2022.05, the option to add Anaconda to the PATH environment variable during an All Users installation has been disabled. This was done to address a security exploit. You can still add Anaconda to the PATH environment variable during a Just Me installation.

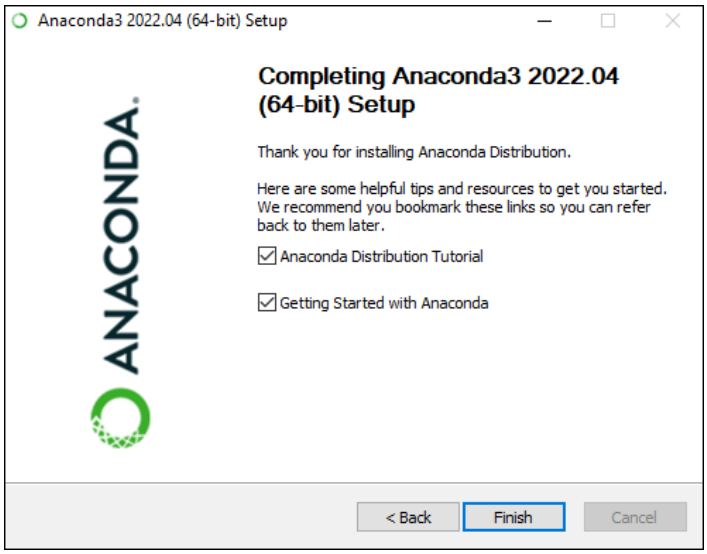
12. Click Install. If you want to watch the packages Anaconda is installing, click Show Details.
13. Click Next.

14. Optional: To learn more about Anaconda's cloud notebook service, go to https://www.anaconda.com/code-in-the-cloud .



15. Or click Continue to proceed.

16. After a successful installation you will see the “Thanks for installing Anaconda” dialog box:



17. If you wish to read more about Anaconda.org and how to get started with Anaconda, check the boxes “Anaconda Distribution Tutorial” and “Learn more about Anaconda”. Click the Finish button.

18. Verify your installation.

(b) Get familiarized with arff file format. Create an arff file on your system and save in the WEKA installed drive of your system.

THEORY:

ARFF file contains 2 sections

- Header Section
- Data Section

All the keywords in ARFF file start with @ symbol.

1. Header Section

This section contains various information related to the dataset like the name of the relation, columns, and type of columns. The header section contains 2 parts Table/relation and attribute part.

@relation :used to give the table name
@attribute: used to give a column name

Datatypes:

nominal: represented inside curly brackets (Like constants) string : data type which accepts only string value numeric: used to store numbers date: used to store date

Syntax:

```
@relation tablename
@attribute column_name type
```

example:

```
@relation "employee"
@attribute f_name string
@attribute l_name string
@attribute contact_num numeric
@attribute dept {HR,IT,MANAGEMENT,MAINTAINANCE}
@attribute DOB date dd-mm-yyyy
@attribute city string
```

Here dept column is having nominal data type so it can only accept above mentioned types of data only,

2. Data section

Data section is used to represents the data or entries for available columns. (according to the order in header section data would be inserted).

Data section starts with @data, and this section must be added after the Header section. Only a single record can be written in a single line.

@data: Used to start data section
%: % sign is used to represent the comment in file.

Syntax:

```
@data
<record1>
<record2>
.
.
<record N>
```

all the Records must be in the same format as their attributes are defined in Header section Like

example:

```
1,naman,N,1234556678,IT,02-08-2000,rjt
2,yash,M,1234556679,HR,04-05-2001,amd
3,kishan,G,1214556678,MANAGEMENT,02-11-2001,pbr
4,?,?,5234556678,IT,03-05-2000,amd
```

entire file would look like this:
emp.arff file:

```
@relation "employee"
@attribute id numeric
@attribute f_name string
@attribute l_name string
@attribute contact_num numeric
@attribute dept {HR,IT,MANAGEMENT,MAINTAINANCE}
@attribute DOB date dd-mm-yyyy
@attribute city string

@data
1,naman,N,1234556678,IT,02-08-2000,rjt
2,yash,M,1234556679,HR,04-05-2001,amd
3,kishan,G,1214556678,MANAGEMENT,02-11-2001,pbr
4,?,?,5234556678,IT,03-05-2000,amd
```

How to Create and open arff file

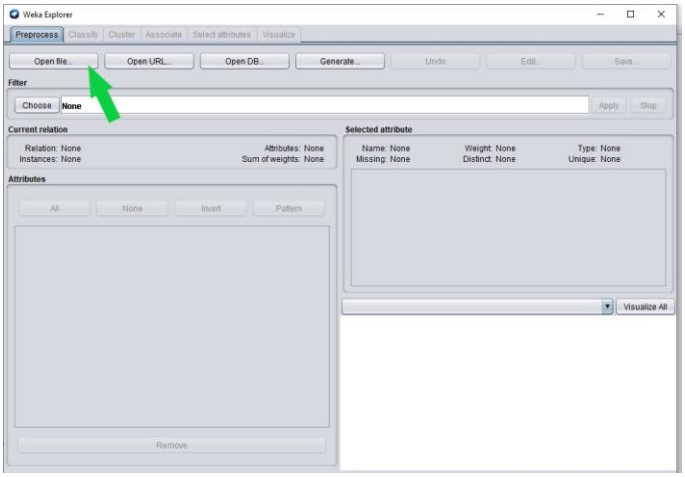
Step 1: Open any text editor and paste the above code.

Step 2: Save the file with emp_dm.arff file extension

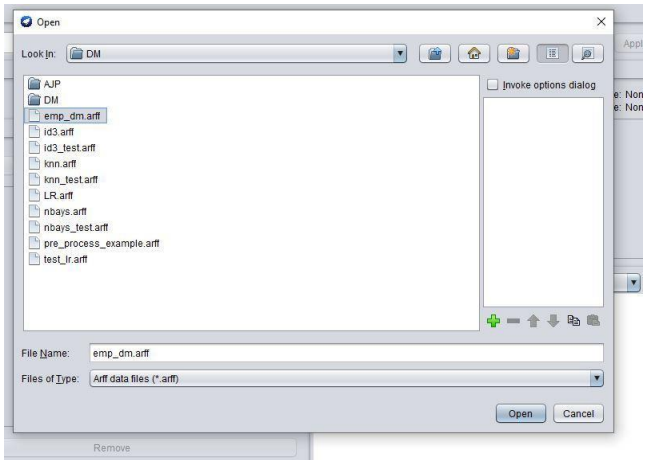
Step 3: Open weka tool

Step 4: Click on Explorer

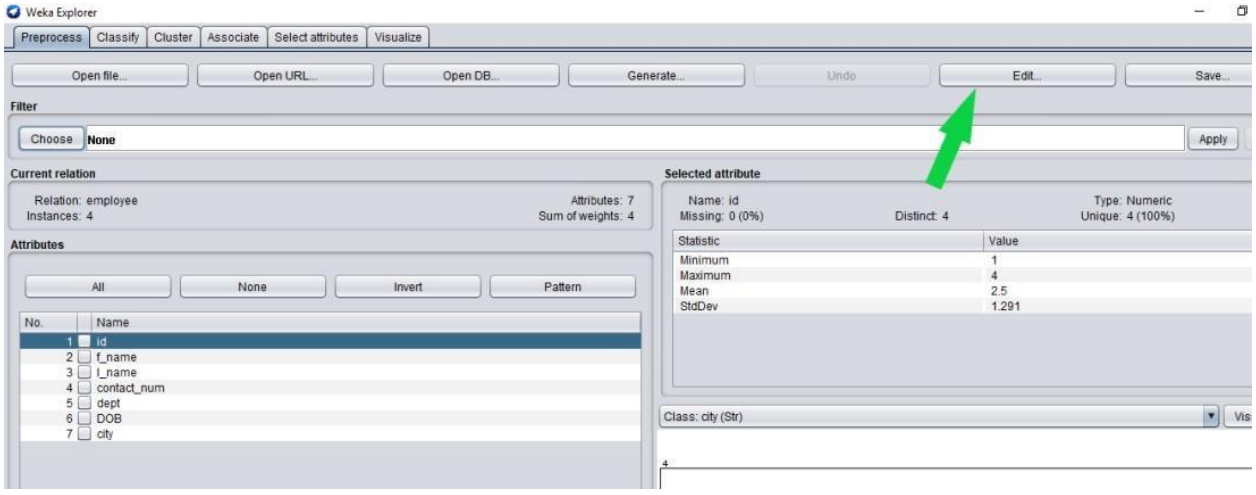
Then click on Open file



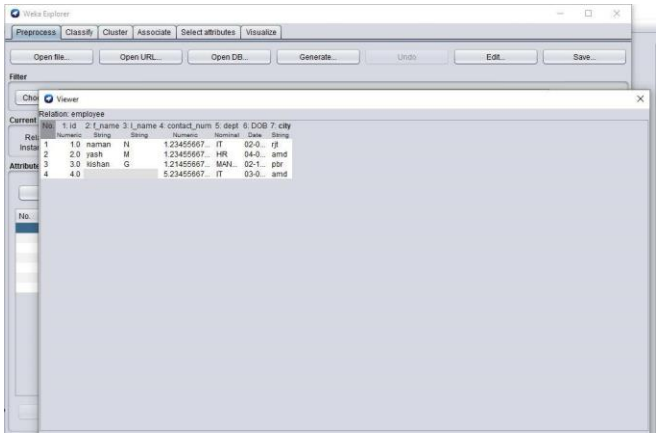
Select/Locate arff file from disk then click On Open.



Step 6: file is now Loaded now click on Edit from Preprocess Tab



Step 7: dataset would be shown like this.



EXPERIMENT No -3

AIM:

(a) Execute the Linear Regression algorithm on WEKA with the help of suitable data set.When you select your data set try to do the splitting of data set for training and testing as: i) Training 80 % and Testing 20%
ii) Training 60 % and testing 40 %

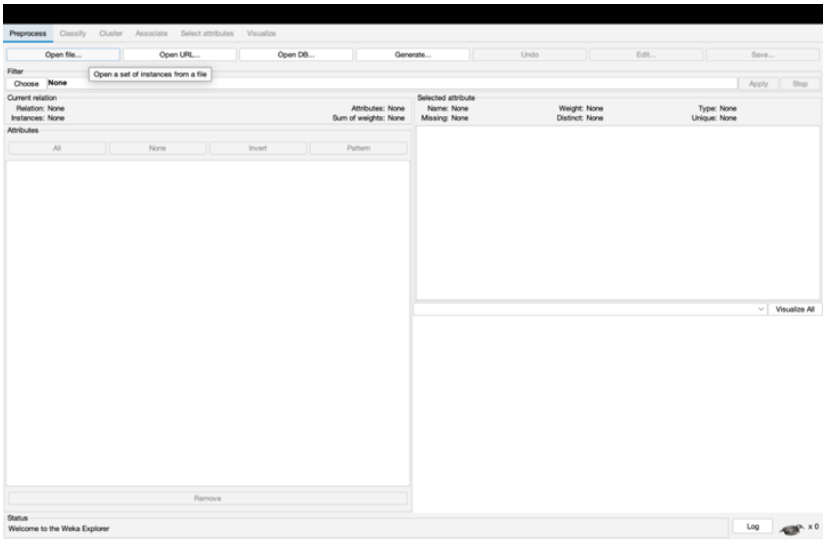
(b) Implement linear regression using python.

THEORY:

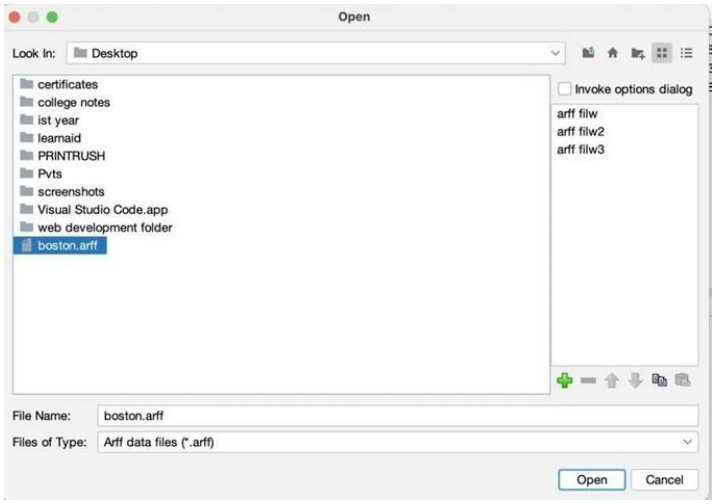
Linear regression is a fundamental supervised learning algorithm used in machine learning and statistics for modeling the relationship between a dependent variable and one or more independent variables. It's commonly used for predictive analysis and to understand the relationship between variables.

At its core, the algorithm aims to establish a linear relationship between the variables, often depicted as a straight line on a graph. Through the process of training, the algorithm determines the optimal slope and intercept of this line, minimizing the disparity between the actual and predicted values of the dependent variable. This optimization typically involves minimizing the mean squared error or mean absolute error. Once trained, the model can be employed to make predictions on new data, facilitating tasks such as forecasting, trend analysis, and understanding correlations between variables. Its simplicity and effectiveness make linear regression a widely used tool across diverse fields, serving as a cornerstone for more complex predictive modeling techniques.

Step 1:
Open Weka tool and go to explorer then go to open files

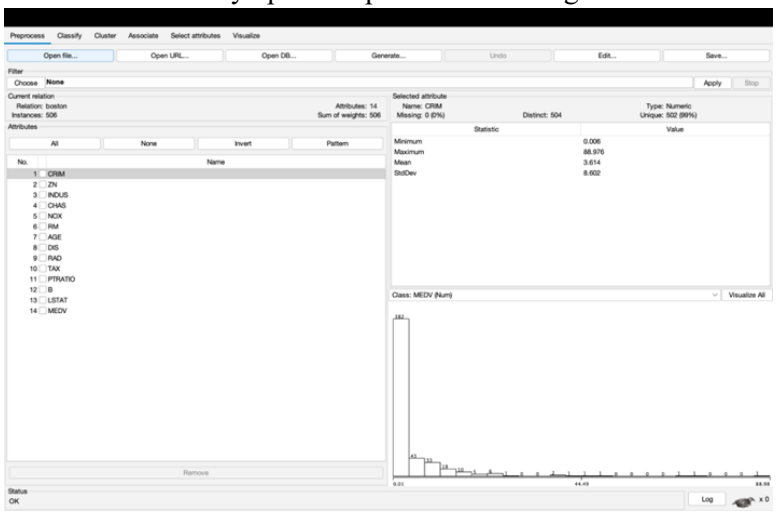


Select the arff file and click on open.



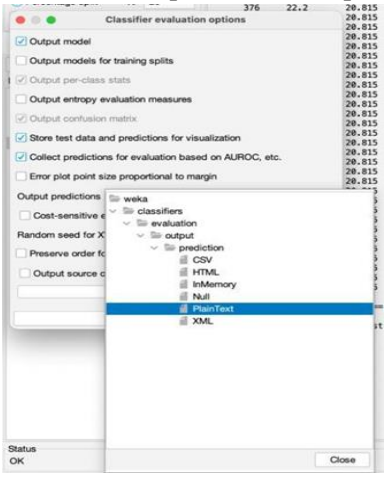
Step 3:

Your file has now opened. Go to Classify option to perform linear regression.



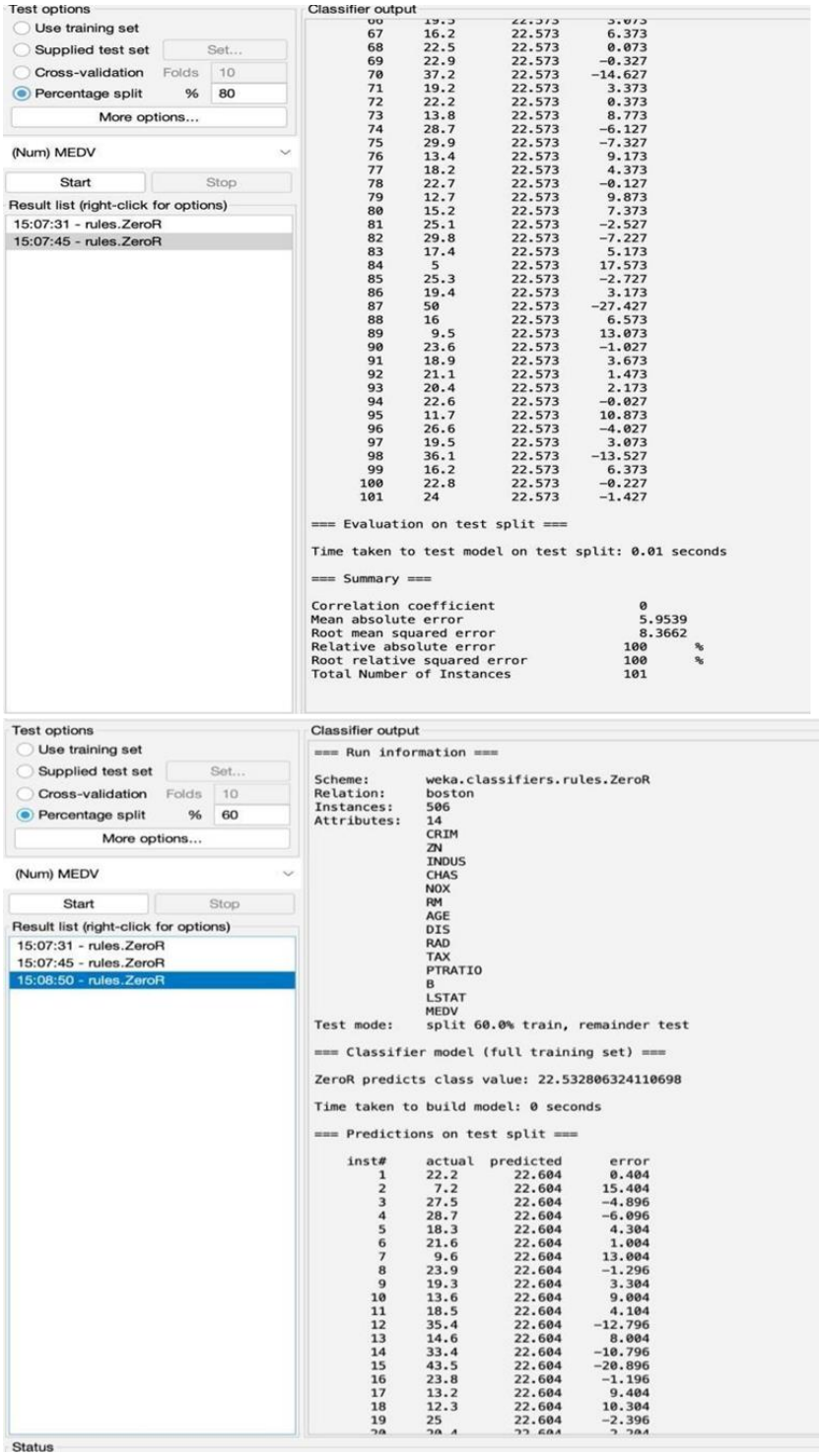
Step 4:

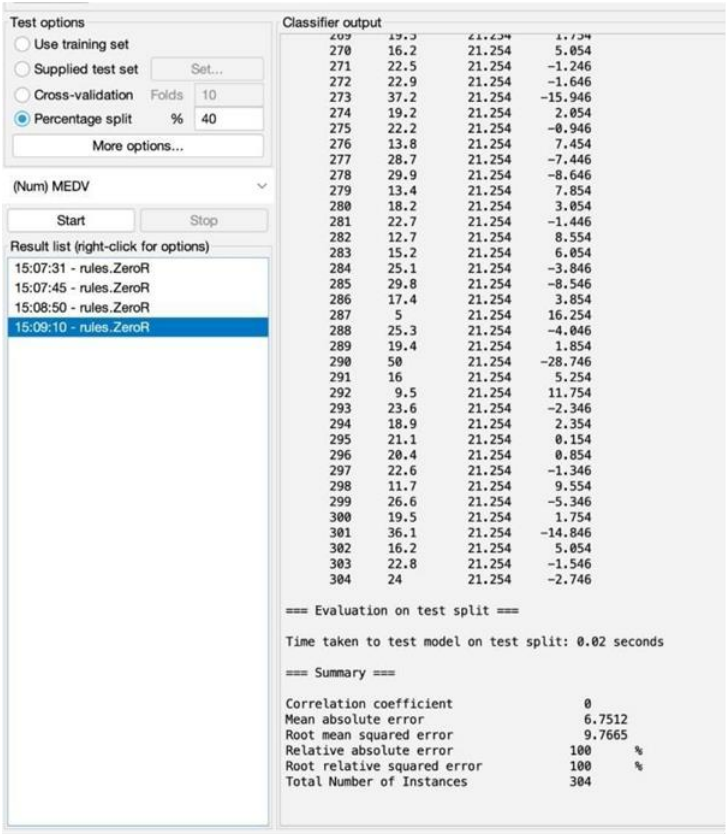
In test options go to percentage split. Go to more options then output predictions and select PlainText.

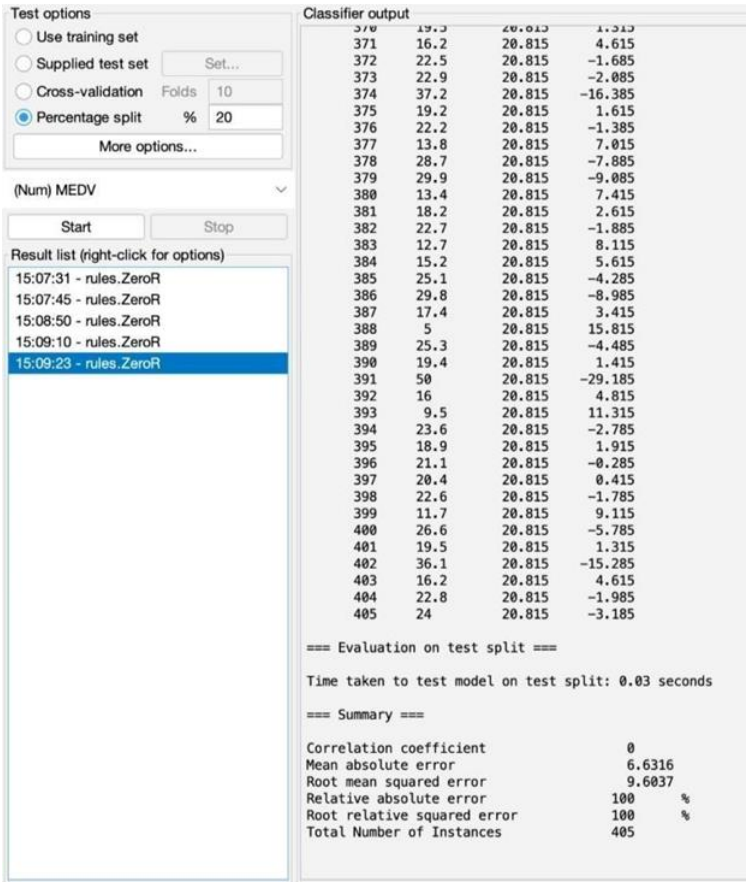


Step 5:

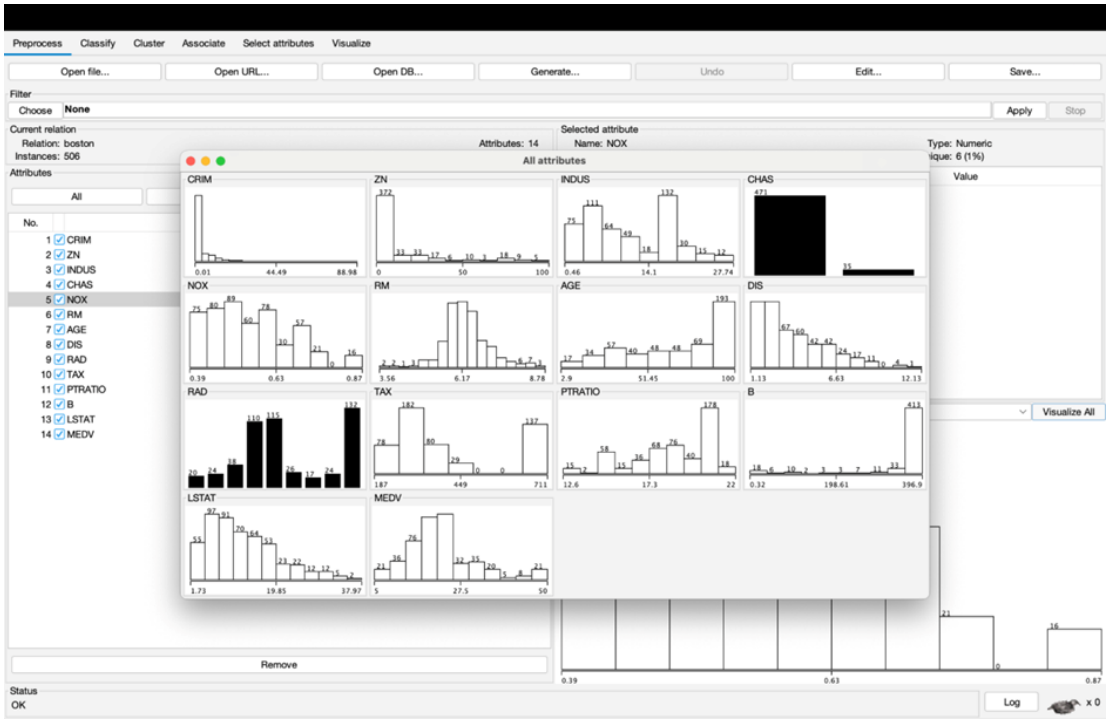
Now split the data into 80, 60, 40 and 20 percent and click on start and note the values.




Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
40
More options...
(Num) MEDV
Start
Stop
Result list (right-click for options)
15:07:31 - rules.ZeroR
15:07:45 - rules.ZeroR
15:08:50 - rules.ZeroR
15:09:10 - rules.ZeroR
Classifier output
270 16.2 21.254 5.054
271 22.5 21.254 -1.246
272 22.9 21.254 -1.646
273 37.2 21.254 -15.946
274 19.2 21.254 2.054
275 22.2 21.254 -0.946
276 13.8 21.254 7.454
277 28.7 21.254 -7.446
278 29.9 21.254 -8.646
279 13.4 21.254 7.854
280 18.2 21.254 3.054
281 22.7 21.254 -1.446
282 12.7 21.254 8.554
283 15.2 21.254 6.054
284 25.1 21.254 -3.846
285 29.8 21.254 -8.546
286 17.4 21.254 3.854
287 5 21.254 16.254
288 25.3 21.254 -4.046
289 19.4 21.254 1.854
290 50 21.254 -28.746
291 16 21.254 5.254
292 9.5 21.254 11.754
293 23.6 21.254 -2.346
294 18.9 21.254 2.354
295 21.1 21.254 0.154
296 20.4 21.254 0.854
297 22.6 21.254 -1.346
298 11.7 21.254 9.554
299 26.6 21.254 -5.346
300 19.5 21.254 1.754
301 36.1 21.254 -14.846
302 16.2 21.254 5.054
303 22.8 21.254 -1.546
304 24 21.254 -2.746
=== Evaluation on test split ===
Time taken to test model on test split: 0.02 seconds
=== Summary ===
Correlation coefficient 0
Mean absolute error 6.7512
Root mean squared error 9.7665
Relative absolute error 100 %
Root relative squared error 100 %
Total Number of Instances 304


Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
20
More options...
(Num) MEDV
Start
Stop
Result list (right-click for options)
15:07:31 - rules.ZeroR
15:07:45 - rules.ZeroR
15:08:50 - rules.ZeroR
15:09:10 - rules.ZeroR
15:09:23 - rules.ZeroR
Classifier output
371 16.2 20.815 4.615
372 22.5 20.815 -1.685
373 22.9 20.815 -2.085
374 37.2 20.815 -16.385
375 19.2 20.815 1.615
376 22.2 20.815 -1.385
377 13.8 20.815 7.015
378 28.7 20.815 -7.885
379 29.9 20.815 -9.085
380 13.4 20.815 7.415
381 18.2 20.815 2.615
382 22.7 20.815 -1.885
383 12.7 20.815 8.115
384 15.2 20.815 5.615
385 25.1 20.815 -4.285
386 29.8 20.815 -8.985
387 17.4 20.815 3.415
388 5 20.815 15.815
389 25.3 20.815 -4.485
390 19.4 20.815 1.415
391 50 20.815 -29.185
392 16 20.815 4.815
393 9.5 20.815 11.315
394 23.6 20.815 -2.785
395 18.9 20.815 1.915
396 21.1 20.815 -0.285
397 20.4 20.815 0.415
398 22.6 20.815 -1.785
399 11.7 20.815 9.115
400 26.6 20.815 -5.785
401 19.5 20.815 1.315
402 36.1 20.815 -15.285
403 16.2 20.815 4.615
404 22.8 20.815 -1.985
405 24 20.815 -3.185
=== Evaluation on test split ===
Time taken to test model on test split: 0.03 seconds
=== Summary ===
Correlation coefficient 0
Mean absolute error 6.6316
Root mean squared error 9.6037
Relative absolute error 100 %
Root relative squared error 100 %
Total Number of Instances 405

Step 6:
Now again go to preprocess and visualize all the attributes.



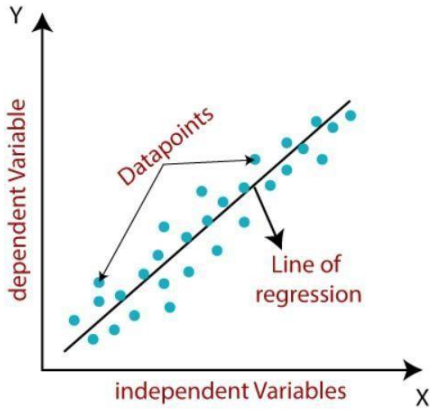
(b)IMPLEMENT LINEAR REGRESSION USING PYTHON.

Linear Regression in Machine Learning

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 as 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.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$y= a_0+a_1x+ \varepsilon$

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.

Program: -

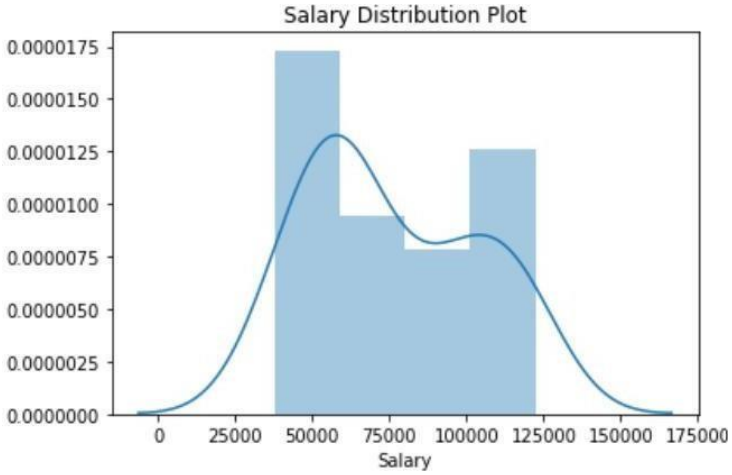
```
# Import libraries import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.model_selection import train_test_split from pandas.core.common import random_state from sklearn.linear_model import LinearRegression from sklearn.metrics import mean_squared_error


# Get dataset
df_sal = pd.read_csv('Salary_Data.csv') df_sal.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

# Describe data
df_sal.describe()

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

# Data distribution
plt.title('Salary Distribution Plot')
sns.distplot(df_sal['Salary'])
plt.show()



# Relationship between Salary and Experience
plt.scatter(df_sal['YearsExperience'], df_sal['Salary'], color = 'lightcoral')
plt.title('Salary vs Experience') plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()


```
# Splitting variables
X = df_sal.iloc[:, :1] # independent
y = df_sal.iloc[:, 1:] # dependent

# Splitting dataset into test/train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# Regressor model regressor
=          LinearRegression()
regressor.fit(X_train,
y_train)

 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
# Prediction result y_pred_test = regressor.predict(X_test) # predicted
value of y_test y_pred_train = regressor.predict(X_train) # predicted
value of y_train

# Prediction on training set plt.scatter(X_train,
y_train, color = 'lightcoral') plt.plot(X_train,
y_pred_train, color = 'firebrick') plt.title('Salary
vs Experience (Training Set)') plt.xlabel('Years
of Experience') plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False) plt.show()
```

# Prediction on test set plt.scatter(X_test, y_test, color = 'lightcoral') plt.plot(X_train, y_pred_train, color = 'firebrick') plt.title('Salary vs Experience (Test Set)') plt.xlabel('Years of Experience') plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False) plt.show()

# Regressor coefficients and intercept
print(f'Coefficient: {regressor.coef_}')
print(f'Intercept:
{regressor.intercept_}')

```
Coefficient: [[9312.57512673]]
Intercept: [26780.09915063]
```

# Calculate and print the mean squared error
mse = mean_squared_error(y_pred_test, y_test) print('Mean Squared Error:', mse)

```
Mean Squared Error: 12823412.298126562
```

EXPERIMENT NO-4

AIM: Explain each of the experiments with necessary screenshots:

(a)Execute the Logistic Regression with the help of properly identified data set. Analyse the result and identify how well the model performed on test set. Brief the steps that you have followed for analyse the data set. (b)Implement Logistic Regression with Python.

Theory:

What is Logistic Regression?

Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1.

For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 it belongs to Class 0. It's referred to as regression because it is the extension of linear regression but is mainly used for classification problems.

Key Points:

- Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Linear Regression Equation:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$

Where, y is a dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

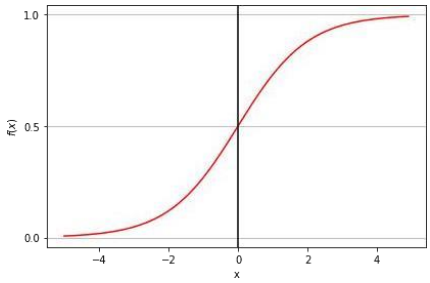
$$p = \frac{1}{1 + e^{-y}}$$

Apply Sigmoid function on linear regression:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)}}$$

Logistic Function – Sigmoid Function

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1. The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the “S” form.
- The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.



Program: -

```
#importing libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns import matplotlib.pyplot as plt

df= pd.read_csv("iris.csv") #importing dataset and making dataframe df.head()
#showing top 5 data entry
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
df.describe() #describes are data
```

	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

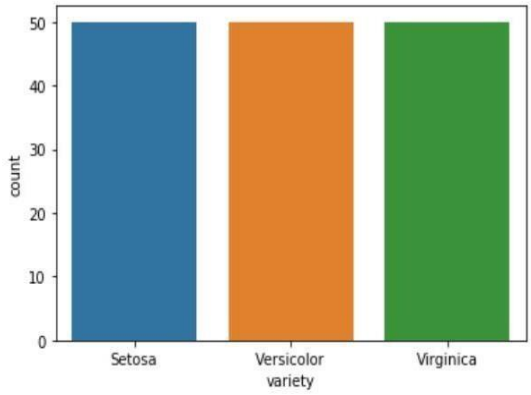
df.info() #gives information about the columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal.length    150 non-null float64
sepal.width     150 non-null float64
petal.length    150 non-null float64
petal.width     150 non-null float64
variety         150 non-null object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

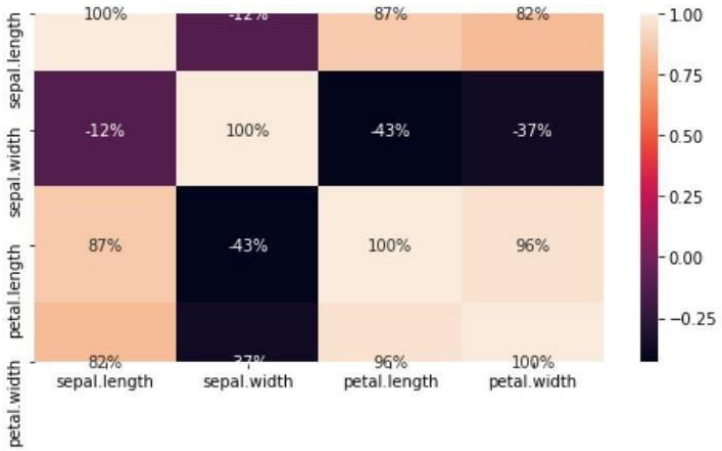
df.shape #tells us about no. of rows and column [rows , columns] (150, 5)

print(df["variety"].value_counts()) sns.countplot(df["variety"])

```
Versicolor    50
Setosa        50
Virginica     50
Name: variety, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x2cc816498c8>
```

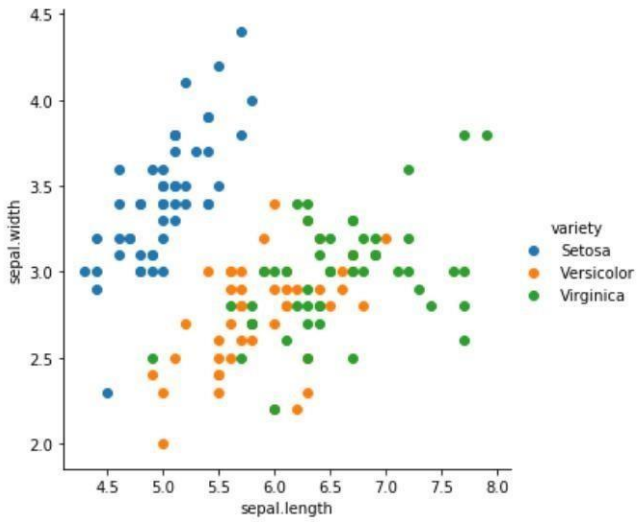


plt.figure(figsize=(8,4))
sns.heatmap(df.corr(),annot=True,fmt=".0%") #draws heatmap with input as the correlation matrix calculted by(df.corr()) plt.show()



# We'll use seaborn's FacetGrid to color the scatterplot by species
sns.FacetGrid(df, hue="variety", height=5).map(plt.scatter, "sepal.length", "sepal.width").add_legend()

```
<seaborn.axisgrid.FacetGrid at 0x2ccfad12088>
```



from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm from sklearn.model_selection import train_test_split #to split the dataset for training and testing from sklearn import metrics #for checking the model accuracy

X=df.iloc[:,0:4]
Y=df["variety"]
X.head()

	sepal.length	sepal.width	petal.length	petal.width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25,random_state=0)# in this our main data is split into train and test
the attribute test_size=0.3 splits the data into 70% and 30% ratio. train=70% and test=30% print("Train Shape",X_train.shape) print("Test Shape",X_test.shape)

```
Train Shape (112, 4)
Test Shape (38, 4)
```

```
log = LogisticRegression()
log.fit(X_train,Y_train)
prediction=log.predict(X_test)
    print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction,Y_test))
```

```
The accuracy of the Logistic Regression is 0.868421052631579
```

AJINKYA NAM RA EN21CS301062

import pandas as pd #useful for loading the dataset import numpy as np #to perform array

from google.colab import files uploaded
= files.upload()

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving DigitalAd dataset.csv to DigitalAd
```

```
dataset.csv
```
dataset = pd.read_csv("/content/DigitalAd_dataset.csv")

print(dataset.shape) print(dataset.head(10))

```
(400, 3)
   Age  Salary  Status
0    18   82000       0
1    29   80000       0
2    47   25000       1
3    45   26000       1
4    46   28000       1
5    48   29000       1
6    45   22000       1
7    47   49000       1
8    48   41000       1
9    45   22000       1
```

```
10
```
X=dataset.iloc[:,:-1].values

```
[    39,  73000],
[    59, 130000],
[    37,  80000],
[    46,  32000],
[    46,  74000],
[    42,  53000],
[    41,  87000],
[    58,  23000],
[    42,  64000],
[    48,  33000],
[    44, 139000],
[    49,  28000],
[    57,  33000],
[    56,  60000],
[    49,  39000],
[    39,  71000],
[    47,  34000],
[    48,  35000],
[    48,  33000],
[    47,  23000],
[    45,  45000],
[    60,  42000],
[    39,  59000],
[    46,  41000],
[    51,  23000],
[    50,  20000],
[    36,  33000],
[    49,  36000],
[    19,  19000],
[    35,  20000],
[    26,  43000],
[    27,  57000],
[    19,  76000],
[    27,  58000],
[    27,  84000],
[    32, 150000],
[    25,  33000],
[    35,  65000],
[    26,  80000],
[    26,  52000],
[    20,  86000],
[    32,        18000]])
```

```
Y = dataset.iloc     [: ,   -1 ].values
Y
```

```
array([0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 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, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
```

```
        1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
        0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
        1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,       1, 1, 1, 1, 1, 1, 0, 1, 1,
    1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,       0, 0, 0, 0])

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state


from sklearn.preprocessing import StandardScaler sc =
StandardScaler()
X_train = sc.fit_transform(X_train) X_test
= sc.transform(X_test)


from sklearn.linear_model import LogisticRegression model =
LogisticRegression(random_state = 0) model.fit(X_train,
y_train)

        ▾       LogisticRegression
        LogisticRegression(random_state=0)


age = int(input("Enter New Customer Age: ")) sal =
int(input("Enter New Customer Salary: ")) newCust
= [[age,sal]] result =
model.predict(sc.transform(newCust)) print(result)
if result == 1:
  print("Customer will
Buy") else:
print("Customer won't
Buy")

        Enter New Customer Age: 18
        Enter New Customer Salary: 25000
        [0]
        Customer won't Buy


y_pred = model.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[0 0]
[0 1]
[0 1]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[0 0]
[0 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[1 1]
[0 1]
[0 0]
[0 1]
[0 0]
[0 1]
[0 0]
[0 0]
[1 1]
[1 1]
[0 0]
[1 1]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
```

```
from sklearn.metrics import confusion_matrix , accuracy_score
cm = confusion_matrix (y_test , y_pred )

print ("Confusion Matrix: " )
print (cm)

print ("Accuracy of the Model: {0}%" . format(accuracy_score (y_test , y_pred )*100))
```

```
Confusion Matrix:
[[61  0]
[20 19]]
Accuracy of the Model: 80.0%
```