DATAWAREHOUSING AND DATAMINING LABORATORY

LAB MANUAL

Academic Year	:	2019 - 2020
Course Code	:	AIT102
Regulations	:	IARE - R16
Semester	:	VI
Branch	:	CSE AND IT

Prepared by
Dr. M Madhubala, Professor

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad.

INFORMATION TECHNOLOGY

1. PROGRAM OUTCOMES:

B.TECH - PROGRAM OUTCOMES (POS)	
PO-1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO-2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO-3	**Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO-4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO-5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO-6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice**.**
PO-7	**Environment and sustainability:** Understand the impact of the professional engineering solution sin societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO-8	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice **(Ethics)**.
PO-9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO-10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO-11	**Project management and finance** :Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO-12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change**.**

2. PROGRAM SPECIFIC OUTCOMES:

PROGRAM SPECIFIC OUTCOMES (PSO's)	
PSO-1	**Professional Skills:** The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity.
PSO-2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO-3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

3. ATTAINMENT OF PROGRAM OUTCOMES AND PROGRAM SPECIFIC OUTCOMES

Week No	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
WEEK-1	Matrix Operations	PO 1; PSO 1	PSO2
WEEK-2	Linear Algebra on Matrices	PO1; PO 2	PSO 1; PSO 2
WEEK-3	Understanding Data	PO 1; PO 2	PSO 1; PSO 2
WEEK-4	Correlation Matrix	PO 1; PO 2	PSO 1; PSO 2
WEEK-5	Data Preprocessing – Handling Missing Values	PO 1; PO 2	PSO 1; PSO 2
WEEK-6	Association Rule Mining- Apriori	PO 1; PO 2; PO 3; PO 4; PO 5	PSO 1; PSO 2; PSO 3
WEEK-7	Classification – Logistic Regression	PO 1; PO 2; PO 3; PO 4; PO 5	PSO 2; PSO 3
WEEK-8	Classification - Knn	PO 1; PO 2; PO 3; PO 4; PO 5	PSO 2; PSO 3
WEEK-9	Classification - Decision Trees	PO1; PO 2; PO 3; PO 4; PO 5	PSO 1; PSO 2; PSO 3
WEEK-10	Classification – Bayesian Network	PO 1; PO 2; PO 3; PO 4; PO 5	PSO 2; PSO 3
WEEK-11	Classification – Support Vector Machines (Svm)	PO 1; PO 2; PO 3; PO 4; PO 5	PSO 1; PSO 2; PSO 3
WEEK-12	Classification – Bayesian Network	PO 1; PO 2; PO 3; PO 4; PO 5	PSO 1; PSO 2; PSO 3

4. MAPPING COURSE OBJECTIVES LEADING TO THE ACHIEVEMENT OF PROGRAM OUTCOMES AND PROGRAM SPECIFIC OUTCOMES:

Course Objectives	Program Outcomes												Program Specific Outcomes		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
I	√	√	√	√	√								√	√	
II	√	√			√								√	√	
III	√	√	√	√	√								√	√	√
IV	√	√	√	√	√								√	√	√
V	√	√	√	√	√								√	√	√

5. SYLLABUS:

<table>
<tr><td colspan="10">VI Semester: IT | CSE</td></tr>
<tr><td rowspan="2">Course Code</td><td rowspan="2">Category</td><td colspan="3">Hours / Week</td><td>Credits</td><td colspan="3">Maximum Marks</td></tr>
<tr><td>L</td><td>T</td><td>P</td><td>C</td><td>CIA</td><td>SEE</td><td>Total</td></tr>
<tr><td>AIT102</td><td>Core</td><td>-</td><td>-</td><td>3</td><td>2</td><td>30</td><td>70</td><td>100</td></tr>
<tr><td>Contact Classes: Nil</td><td>Tutorial Classes: Nil</td><td colspan="4">Practical Classes: 36</td><td colspan="3">Total Classes: 36</td></tr>
<tr><td colspan="9">LIST OF EXPERIMENTS</td></tr>
<tr><td>WEEK-1</td><td colspan="8">MATRIX OPERATIONS</td></tr>
<tr><td colspan="9">Introduction to Python libraries for Data Mining : NumPy, SciPy, Pandas, Matplotlib, Scikit-Learn
Write a Python program to do the following operations:
Library: NumPy
a) Create multi-dimensional arrays and find its shape and dimension
b) Create a matrix full of zeros and ones
c) Reshape and flatten data in the array
d) Append data vertically and horizontally
e) Apply indexing and slicing on array</td></tr>
</table>

f) Use statistical functions on array - Min, Max, Mean, Median and Standard Deviation	
WEEK-2	**LINEAR ALGEBRA ON MATRICES**
Write a Python program to do the following operations: Library: NumPy a) Dot and matrix product of two arrays b) Compute the Eigen values of a matrix c) Solve a linear matrix equation such as $3 * x^0 + x^1 = 9$, $x^0 + 2 * x^1 = 8$ d) Compute the multiplicative inverse of a matrix e) Compute the rank of a matrix f) Compute the determinant of an array	
WEEK-3	**UNDERSTANDING DATA**
Write a Python program to do the following operations: Data set: brain_size.csv Library: Pandas a) Loading data from CSV file b) Compute the basic statistics of given data - shape, no. of columns, mean c) Splitting a data frame on values of categorical variables d) Visualize data using Scatter plot	
WEEK-4	**CORRELATION MATRIX**
Write a python program to load the dataset and understand the input data Dataset : Pima Indians Diabetes Dataset Library : Scipy a) Load data, describe the given data and identify missing, outlier data items b) Find correlation among all attributes c) Visualize correlation matrix	
WEEK -5	**DATA PREPROCESSING – HANDLING MISSING VALUES**
Write a python program to impute missing values with various techniques on given dataset. a) Remove rows/ attributes b) Replace with mean or mode c) Write a python program to perform transformation of data using Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.	
WEEK -6	**ASSOCIATION RULE MINING- APRIORI**
Write a python program to find rules that describe associations by using Apriori algorithm between different products given as 7500 transactions at a French retail store. Libraries: NumPy, SciPy, Matplotlib, Pandas Dataset: https://drive.google.com/file/d/1y5DYn0dGoSbC22xowBq2d4po6h1JxcTQ/view?usp=sharing a) Display top 5 rows of data b) Find the rules with min_confidence : .2, min_support= 0.0045, min_lift=3, min_length=2	
WEEK -7	**CLASSIFICATION – LOGISTIC REGRESSION**
Classification of Bank Marketing Data The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The dataset provides the bank customers' information. It includes 41,188 records and 21 fields. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y). Libraries: Pandas, NumPy, Sklearn, Seaborn Write a python program to a) Explore data and visualize each attribute b) Predict the test set results and find the accuracy of the model c) Visualize the confusion matrix d) Compute precision, recall, F-measure and support	
WEEK-8	**CLASSIFICATION - KNN**
Dataset: The data set consists of 50 samples from each of three species of Iris: Iris setosa, Iris virginica and Iris versicolor. Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Libraries: import numpy as np Write a python program to	

a) Calculate Euclidean Distance. b) Get Nearest Neighbors c) Make Predictions.
WEEK-9 **CLASSIFICATION - DECISION TREES**
Write a python program a) to build a decision tree classifier to determine the kind of flower by using given dimensions. b) training with various split measures(Gini index, Entropy and Information Gain) c)Compare the accuracy
WEEK -10 **CLUSTERING – K-MEANS**
Predicting the titanic survive groups: The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships. One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class. Libraries: Pandas, NumPy, Sklearn, Seaborn, Matplotlib Write a python program a)to perform preprocessing b)to perform clustering using k-means algorithm to cluster the records into two i.e. the ones who survived and the ones who did not.
WEEK -11 **CLASSIFICATION – BAYESIAN NETWORK**
Predicting Loan Defaulters : A bank is concerned about the potential for loans not to be repaid. If previous loan default data can be used to predict which potential customers are liable to have problems repaying loans, these "bad risk" customers can either be declined a loan or offered alternative products. Dataset: The stream named bayes_bankloan.str, which references the data file named bankloan.sav. These files are available from the Demos directory of any IBM® SPSS® Modeler installation and can be accessed from the IBM SPSS Modeler program group on the Windows Start menu. The bayes_bankloan.str file is in the streams directory. a) Build Bayesian network model using existing loan default data b) Visualize Tree Augmented Naïve Bayes model a) Predict potential future defaulters, and looks at three different Bayesian network model types (TAN, Markov, Markov-FS) to establish the better predicting model.
WEEK-12 **CLASSIFICATION – SUPPORT VECTOR MACHINES (SVM)**
A wide dataset is one with a large number of predictors, such as might be encountered in the field of bioinformatics (the application of information technology to biochemical and biological data). A medical researcher has obtained a dataset containing characteristics of a number of human cell samples extracted from patients who were believed to be at risk of developing cancer. Analysis of the original data showed that many of the characteristics differed significantly between benign and malignant samples. **Dataset**: The stream named svm_cancer.str, available in the Demos folder under the streams subfolder. The data file is cell_samples.data. The dataset consists of several hundred human cell sample records, each of which contains the values of a set of cell characteristics. a) Develop an SVM model that can use the values of these cell characteristics in samples from other patients to give an early indication of whether their samples might be benign or malignant. Hint: Refer UCI Machine Learning Repository for data set.
References:
1. https://www.dataquest.io/blog/sci-kit-learn-tutorial/ 2. https://www.ibm.com/support/knowledgecenter/en/SS3RA7_sub/modeler_tutorial_ddita/modeler_tutorial_ddita-gentopic1.html 3. https://archive.ics.uci.edu/ml/datasets.php
SOFTWARE AND HARDWARE REQUIREMENTS FOR A BATCH OF 24 STUDENTS: **HARDWARE:** Intel Desktop Systems: 24 Nos **SOFTWARE:** Application Software: Python, IBM SPSS Modeler - CLEMENTINE

6. INDEX

S. No	List of Experiments	Page No
1	WEEK-1:MATRIX OPERATIONS	
2	WEEK-2 : LINEAR ALGEBRA ON MATRICES	
3	WEEK-3 :UNDERSTANDING DATA	
4	WEEK-4 :CORRELATION MATRIX	
5	WEEK-5 :DATA PREPROCESSING – HANDLING MISSING VALUES	
6	WEEK-6 :ASSOCIATION RULE MINING - APRIORI	
7	WEEK-7 :CLASSIFICATION – LOGISTIC REGRESSION	
8	WEEK-8 :CLASSIFICATION - KNN	
9	WEEK-9 :CLASSIFICATION - DECISION TREES	
10	WEEK-10 :CLASSIFICATION – BAYESIAN NETWORK	
11	WEEK-11:CLASSIFICATION – SUPPORT VECTOR MACHINES (SVM)	
12	WEEK-12 :CLASSIFICATION – BAYESIAN NETWORK	

WEEK-1

MATRIC OPERATIONS

OBJECTIVE:

Introduction to Python libraries for Data Mining :NumPy, SciPy, Pandas, Matplotlib, Scikit-Learn
Write a Python program to do the following operations:
Library: NumPy
a) Create multi-dimensional arrays and find its shape and dimension
b) Create a matrix full of zeros and ones
c) Reshape and flatten data in the array
d) Append data vertically and horizontally
e) Apply indexing and slicing on array
f) Use statistical functions on array - Min, Max, Mean, Median and Standard Deviation

RESOURCES:

Python 3.7.0
Install : pip installer, NumPy library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

a) Create multi-dimensional arrays and find its shape and dimension

Import numpy as np

#creation of multi-dimensional array
a=np.array([[1,2,3],[2,3,4],[3,4,5]])

#shape
b=a.shape
print("shape:",a.shape)

#dimension
c=a.ndim
print("dimensions:",a.ndim)

b) Create a matrix full of zeros and ones

#matrix full of zeros
z=np.zeros((2,2))
print("zeros:",z)

#matrix full of ones
o=np.ones((2,2))
print("ones:",o)

c) Reshape and flatten data in the array

#matrix reshape

```
a=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]])
b=a.reshape(4,2,2)
print("reshape:",b)
```

#matrix flatten

```
c=a.flatten()
print("flatten:",c)
```

d) Append data vertically and horizontally

#Appending data vertically

```
x=np.array([[10,20],[80,90]])
y=np.array([[30,40],[60,70]])
v=np.vstack((x,y))
print("vertically:",v)
```

#Appending data horizontally

```
h=np.hstack((x,y))
print("horizontally:",h)
```

e) Apply indexing and slicing on array

#indexing

```
a=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]])
temp = a[[0, 1, 2, 3], [1, 1, 1, 1]]
print(“indexing”,temp)
```

#slicing

```
i=a[:4,::2]
print(“slicing”,i)
```

f) Use statistical functions on array - Min, Max, Mean, Median and Standard Deviation

#min for finding minimum of an array

```
a=np.array([[1,3,-1,4],[3,-2,1,4]])
b=a.min()
print(“minimum:”,b)
```

#max for finding maximum of an array

```
C=a.max()
Print(“maximum”,c)
```

#mean

```
a=np.array([1,2,3,4,5])
d=a.mean()
print(“mean:”,d)
```

#median

```
e=np.median(a)
print("median:",e)
```

#standard deviation

```
f=a.std()
print("standard deviation",f)
```

INPUT/OUTPUT:

```
a) shape: (3, 3)
   dimensions: 2

   zeros:
   [[0. 0.]
   [0. 0.]]

   ones:
   [[1. 1.]
   [1. 1.]]

b) reshape:
   [[[1 2]
   [3 4]]
    [[2 3]
    [4 5]]
    [[3 4]
   [5 6]]
    [[4 5]
    [6 7]]]
   flatten: [1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7]

c) vertically: [[10 20]
 [80 90]
 [30 40]
 [60 70]]
   horizontally: [[10 20 30 40]
   [80 90 60 70]]

d) indexing [2 3 4 5]

   slicing [[1 3]
 [2 4]
 [3 5]
 [4 6]]

e) minimum: -2
maximum: 4
mean: 3
median: 3
standard deviation: 1.4142135623730951
```

WEEK-2

LINEAR ALGEBRA ON MATRICES

OBJECTIVE:

Write a Python program to do the following operations:
Library: NumPy

a) Dot and matrix product of two arrays
b) Compute the Eigen values of a matrix
c) Solve a linear matrix equation such as $3 * x^0 + x^1 = 9$, $x^0 + 2 * x^1 = 8$
d) Compute the multiplicative inverse of a matrix
e) Compute the rank of a matrix
f) Compute the determinant of an array

RESOURCES:

Python 3.7.0
Install : pip installer, NumPy library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

a) Dot and matrix product of two arrays

```
#dot product of two arrays
Import numpy as np
a=np.array([1,2,3])
b=np.array([2,3,4])
print("dot product of one dimension is:", np.dot(a,b))

#matrix elements multiplication
a=np.array([[1,2],[3,4]])
b=np.array([[1,2],[3,4]])
print("element multiplication of matrix;", np.multiply(a,b))

#matrix multiplication
print("matrix multiplication", np.matmul(a,b))
```

b) Compute the Eigen values of a matrix

```
#eigen values of a matrix
Import  numpy as np
a=np.array([[1,2],[3,4]])
eigvalues,eigvectors=np.linalg.eig(a)
print("eigen value:",eigvalues,"eigen vector:",eigvectors)
```

c) Solve a linear matrix equation such as $3 * x^0 + x^1 = 9$, $x^0 + 2 * x^1 = 8$

```
#linear matric equation
importnumpy as np
a=np.array([[3,1],[1,2]])
b=np.array([[9],[8]])
a_inv=np.linalg.inv(a)
e=np.matmul(a_inv,b)
```

print("linear equation:",e)

d) Compute the multiplicative inverse of a matrix

#multiplicative inverse

```
import  numpy as np
a=np.array([[3,1],[1,2]])
a_inv=np.linalg.inv(a)
print("a inverse:",a_inv)
```

e) Compute the rank of a matrix

#matric rank

```
a=np.array([[3,1],[1,2]])
b=np.linalg.matrix_rank(a)
print(“rank:”,b)
```

f) Compute the determinant of an array

```
a=np.array([[3,1],[1,2]])
b=np.linalg.det(a)
print(“determinant:”,b)
```

INPUT/OUTPUT:

```
a)
dot product of one dimension is: 20
element multiplication of matrix;
        [[ 1  4]
        [ 9 16]]
matrix multiplication
        [[ 7 10]
        [15 22]]
b)
eigen value:
[-0.37228132  5.37228132]
eigen vector:
         [[-0.82456484 -0.41597356]
         [0.56576746 -0.90937671]]
c)
linear equation:
        [[ 3.6 -1.8]
        [-1.6  4.8]]
d)
a inverse:
        [[ 0.4 -0.2]
        [-0.2  0.6]]
e)
rank: 2
f)
determinant: 5.000000000000001
```

WEEK-3

UNDERSTANDING DATA

OBJECTIVE:

Write a Python program to do the following operations:
Dataset: brain_size.csv
Library: Pandas, matplotlib
a) Loading data from CSV file
b) Compute the basic statistics of given data - shape, no. of columns, mean
c) Splitting a data frame on values of categorical variables
d) Visualize data using Scatter plot

RESOURCES:

a) Python 3.7.0
b) Install: pip installer, Pandas library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

a) Loading data from CSV file

```
#loading file csv

import pandas as pd
pd.read_csv("P:/python/newfile.csv")
```

b) Compute the basic statistics of given data - shape, no. of columns, mean

```
#shape

a=pd.read_csv("C:/Users/admin/Documents/diabetes.csv")
print('shape :',a.shape)

#no of columns
cols=len(a.axes[1])
print('no of columns:',cols)

#mean of data
m=a["Age"].mean()
print('mean of Age:',m)
```

c) Splitting a data frame on values of categorical variables

```
#adding data
a['address']=["hyderabad,ts","Warangal,ts","Adilabad,ts","medak,ts"]
#splitting dataframe
a_split=a['address'].str.split(',',1)
a['district']=a_split.str.get(0)
a['state']=a_split.str.get(1)
del(a['address'])
```

d) Visualize data using Scatter plot

```
#visualize data using scatter plot
```

importmatplotlib as plt

a.plot.scatter(x='marks',y='rollno',c='Blue')

INPUT/OUTPUT:

a)

```
student  rollno  marks
0      a1     121     98
1      a2     122     82
2      a3     123     92
3      a4     124     78
```

b)

```
shape: (4, 3)
no of colums: 3
mean: 87.5
```

c)

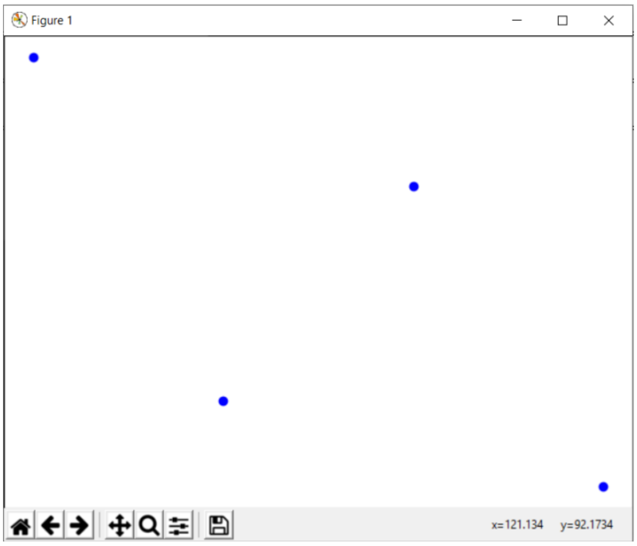
before:

```
student  rollno  marks       address
0      a1     121     98  hyderabad,ts
1      a2     122     82   Warangal,ts
2      a3     123     92   Adilabad,ts
3      a4     124     78      medak,ts
```

After:

```
student  rollno  marks   district state
0      a1     121     98  hyderabadts
1      a2     122     82   Warangal    ts
2      a3     123     92   Adilabadts
3      a4     124     78      medakts
```

d)



WEEK-4

CORRELATION MATRIX

OBJECTIVE:

Write a python program to load the dataset and understand the input data
Dataset: Pima Indians Diabetes Dataset
https://www.kaggle.com/uciml/pima-indians-diabetes-database#diabetes.csv
Library: Scipy

a) Load data, describe the given data and identify missing, outlier data items
b) Find correlation among all attributes
c) Visualize correlation matrix

RESOURCES:

a) Python 3.7.0
b) Install: pip installer, pandas, SciPy library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

a) Load data

```
import pandas as pd
importnumpy as np
importmatplotlib as plt
%matplotlib inline
```

#Reading the dataset in a dataframe using Pandas

```
df = pd.read_csv("C:/Users/admin/Documents/diabetes.csv")
```

#describe the given data

```
print(df. describe())
```

#Display first 10 rows of data

```
print(df.head(10))
```

#Missing values

In Pandas missing data is represented by two values:

None: None is a Python singleton object that is often used for missing data in Python code.

NaN :NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems

- isnull()
- notnull()
- dropna()
- fillna()
- replace()
- interpolate()

 # identify missing items

```
print(df.isnull())
```

#outlier data items

Methods

Z-score method

Modified Z-score method

IQR method

#Z-score function defined in scipy library to detect the outliers

```
importnumpy as np
defoutliers_z_score(ys):
threshold = 3
mean_y = np.mean(ys)
stdev_y = np.std(ys)
z_scores = [(y - mean_y) / stdev_y for y in ys]
returnnp.where(np.abs(z_scores) > threshold)
```

b) Find correlation among all attributes

 # importing pandas as pd

import pandas as pd

 # Making data frame from the csv file

df = pd.read_csv("nba.csv")

 # Printing the first 10 rows of the data frame for visualization

df[:10]

 # To find the correlation among columns

 # using pearson method

df.corr(method ='pearson')

using ‘kendall’ method.

df.corr(method ='kendall')

c) Visualize correlation matrix

INPUT/OUTPUT:

```
import pandas as pd
df = pd.read_csv("C:/Users/admin/Documents/diabetes.csv")

print(df. describe())
print(df.head(10))
```

```
       Pregnancies     Glucose  ...         Age     Outcome
count   768.000000  768.000000  ...  768.000000  768.000000
mean      3.845052  120.894531  ...   33.240885    0.348958
std       3.369578   31.972618  ...   11.760232    0.476951
min       0.000000    0.000000  ...   21.000000    0.000000
25%       1.000000   99.000000  ...   24.000000    0.000000
50%       3.000000  117.000000  ...   29.000000    0.000000
75%       6.000000  140.250000  ...   41.000000    1.000000
max      17.000000  199.000000  ...   81.000000    1.000000

[8 rows x 9 columns]
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            6      148             72  ...                     0.627   50        1
1            1       85             66  ...                     0.351   31        0
2            8      183             64  ...                     0.672   32        1
3            1       89             66  ...                     0.167   21        0
4            0      137             40  ...                     2.288   33        1
5            5      116             74  ...                     0.201   30        0
6            3       78             50  ...                     0.248   26        1
7           10      115              0  ...                     0.134   29        0
8            2      197             70  ...                     0.158   53        1
9            8      125             96  ...                     0.232   54        1

[10 rows x 9 columns]
```

WEEK -5

DATA PREPROCESSING – HANDLING MISSING VALUES

OBJECTIVE:

Write a python program to impute missing values with various techniques on given dataset.
a) Remove rows/ attributes
b) Replace with mean or mode
c) Write a python program to perform transformation of data using Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.

https://www.kaggle.com/uciml/pima-indians-diabetes-database#diabetes.csv
Library: Scipy

RESOURCES:

a) Python 3.7.0
b) Install: pip installer, pandas, SciPy library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

 # filling missing value using fillna()

df.fillna(0)

 # filling a missing value with previous value

df.fillna(method ='pad')

#Filling null value with the next ones

df.fillna(method ='bfill')

filling a null values using fillna()

data["Gender"].fillna("No Gender", inplace = True)

will replace **Nan** value in dataframe with value -99

data.replace(to_replace = np.nan, value = -99)

 # Remove rows/ attributes

using dropna() function to remove rows having one **Nan**

df.dropna()

using dropna() function to remove rows with all **Nan**

df.dropna(how = 'all')

using dropna() function to remove column having one **Nan**

df.dropna(axis = 1)

 # Replace with mean or mode

mean_y = np.mean(ys)

 # Perform transformation of data using Discretization (Binning)

Binning can also be used as a discretization technique. Discretization refers to the process of converting or partitioning continuous attributes, features or variables to discretized or nominal attributes/ features/ variables/ intervals.

For example, attribute values can be discretized by applying equal-width or equal-frequency binning, and then replacing each bin value by the bin mean or median, as in smoothing by bin means or smoothing by bin medians, respectively. Then the continuous values can be converted to a nominal or discretized value which is same as the value of their corresponding bin.

There are basically two types of binning approaches –

Equal width (or distance) binning : The simplest binning approach is to partition the range of the variable into k equal-width intervals. The interval width is simply the range [A, B] of the variable divided by k, w = (B-A) / k

Thus, i^{th} interval range will be [A + (i-1)w, A + iw] where i = 1, 2, 3…..k

Skewed data cannot be handled well by this method.

Equal depth (or frequency) binning : In equal-frequency binning we divide the range [A, B] of the variable into intervals that contain (approximately) equal number of points; equal frequency may not be possible due to repeated values.

There are three approaches to perform smoothing –

Smoothing by bin means : In smoothing by bin means, each value in a bin is replaced by the mean value of the bin.

Smoothing by bin median : In this method each bin value is replaced by its bin median value.

Smoothing by bin boundary : In smoothing by bin boundaries, the minimum and maximum values in a given bin are identified as the bin boundaries. Each bin value is then replaced by the closest boundary value.

Example:

Sorted data for price(in dollar) : 2, 6, 7, 9, 13, 20, 21, 25, 30

```
Partition using equal frequency approach:
Bin 1 : 2, 6, 7
Bin 2 : 9, 13, 20
Bin 3 : 21, 24, 30

Smoothing by bin mean :
Bin 1 : 5, 5, 5
Bin 2 : 14, 14, 14
Bin 3 : 25, 25, 25

Smoothing by bin median :
Bin 1 : 6, 6, 6
Bin 2 : 13, 13, 13
Bin 3 : 24, 24, 24

Smoothing by bin boundary :
Bin 1 : 2, 7, 7
Bin 2 : 9, 9, 20
Bin 3 : 21, 21, 30
```

import numpy as np
import math

```
from sklearn.datasets import load_iris
from sklearn import datasets, linear_model, metrics

# load iris data set
dataset = load_iris()
a = dataset.data
b = np.zeros(150)

# take 1st column among 4 column of data set
for i in range (150):
    b[i]=a[i,1]

b=np.sort(b)  #sort the array

# create bins
bin1=np.zeros((30,5))
bin2=np.zeros((30,5))
bin3=np.zeros((30,5))

# Bin mean
for i in range (0,150,5):
    k=int(i/5)
    mean=(b[i] + b[i+1] + b[i+2] + b[i+3] + b[i+4])/5
    for j in range(5):
        bin1[k,j]=mean
print("Bin Mean: \n",bin1)

# Bin boundaries
for i in range (0,150,5):
    k=int(i/5)
    for j in range (5):
        if (b[i+j]-b[i]) < (b[i+4]-b[i+j]):
            bin2[k,j]=b[i]
        else:
            bin2[k,j]=b[i+4]
print("Bin Boundaries: \n",bin2)

# Bin median
for i in range (0,150,5):
    k=int(i/5)
    for j in range (5):
        bin3[k,j]=b[i+2]
print("Bin Median: \n",bin3)
```

OUTPUT:

Bin Mean:	Bin Boundaries:	Bin Median:
[[2.18 2.18 2.18 2.18 2.18] [2.34 2.34 2.34 2.34 2.34] [2.48 2.48 2.48 2.48 2.48] [2.52 2.52 2.52 2.52 2.52] [2.62 2.62 2.62 2.62 2.62]	[[2. 2.3 2.3 2.3 2.3] [2.3 2.3 2.3 2.4 2.4] [2.4 2.5 2.5 2.5 2.5] [2.5 2.5 2.5 2.5 2.6] [2.6 2.6 2.6 2.6 2.7]	[[2.2 2.2 2.2 2.2 2.2] [2.3 2.3 2.3 2.3 2.3] [2.5 2.5 2.5 2.5 2.5] [2.5 2.5 2.5 2.5 2.5] [2.6 2.6 2.6 2.6 2.6]

[2.7 2.7 2.7 2.7 2.7]	[2.7 2.7 2.7 2.7 2.7]	[2.7 2.7 2.7 2.7 2.7]
[2.74 2.74 2.74 2.74 2.74]	[2.7 2.7 2.7 2.8 2.8]	[2.7 2.7 2.7 2.7 2.7]
[2.8 2.8 2.8 2.8 2.8]	[2.8 2.8 2.8 2.8 2.8]	[2.8 2.8 2.8 2.8 2.8]
[2.8 2.8 2.8 2.8 2.8]	[2.8 2.8 2.8 2.8 2.8]	[2.8 2.8 2.8 2.8 2.8]
[2.86 2.86 2.86 2.86 2.86]	[2.8 2.8 2.9 2.9 2.9]	[2.9 2.9 2.9 2.9 2.9]
[2.9 2.9 2.9 2.9 2.9]	[2.9 2.9 2.9 2.9 2.9]	[2.9 2.9 2.9 2.9 2.9]
[2.96 2.96 2.96 2.96 2.96]	[2.9 2.9 3. 3. 3.]	[3. 3. 3. 3. 3.]
[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]
[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]
[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]
[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]	[3. 3. 3. 3. 3.]
[3.04 3.04 3.04 3.04 3.04]	[3. 3. 3. 3.1 3.1]	[3. 3. 3. 3. 3.]
[3.1 3.1 3.1 3.1 3.1]	[3.1 3.1 3.1 3.1 3.1]	[3.1 3.1 3.1 3.1 3.1]
[3.12 3.12 3.12 3.12 3.12]	[3.1 3.1 3.1 3.1 3.2]	[3.1 3.1 3.1 3.1 3.1]
[3.2 3.2 3.2 3.2 3.2]	[3.2 3.2 3.2 3.2 3.2]	[3.2 3.2 3.2 3.2 3.2]
[3.2 3.2 3.2 3.2 3.2]	[3.2 3.2 3.2 3.2 3.2]	[3.2 3.2 3.2 3.2 3.2]
[3.26 3.26 3.26 3.26 3.26]	[3.2 3.2 3.3 3.3 3.3]	[3.3 3.3 3.3 3.3 3.3]
[3.34 3.34 3.34 3.34 3.34]	[3.3 3.3 3.3 3.4 3.4]	[3.3 3.3 3.3 3.3 3.3]
[3.4 3.4 3.4 3.4 3.4]	[3.4 3.4 3.4 3.4 3.4]	[3.4 3.4 3.4 3.4 3.4]
[3.4 3.4 3.4 3.4 3.4]	[3.4 3.4 3.4 3.4 3.4]	[3.4 3.4 3.4 3.4 3.4]
[3.5 3.5 3.5 3.5 3.5]	[3.5 3.5 3.5 3.5 3.5]	[3.5 3.5 3.5 3.5 3.5]
[3.58 3.58 3.58 3.58 3.58]	[3.5 3.6 3.6 3.6 3.6]	[3.6 3.6 3.6 3.6 3.6]
[3.74 3.74 3.74 3.74 3.74]	[3.7 3.7 3.7 3.8 3.8]	[3.7 3.7 3.7 3.7 3.7]
[3.82 3.82 3.82 3.82 3.82]	[3.8 3.8 3.8 3.8 3.9]	[3.8 3.8 3.8 3.8 3.8]
[4.12 4.12 4.12 4.12 4.12]]	[3.9 3.9 3.9 4.4 4.4]]	[4.1 4.1 4.1 4.1 4.1]]

 # Perform transformation of data using normalization (MinMaxScaler or MaxAbsScaler) on given dataset.

In preprocessing, standardization of data is one of the transformation task. Standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively.

The motivation to use this scaling include robustness to very small standard deviations of features and preserving zero entries in sparse data.

Example to scale a toy data matrix to the [0, 1] range:

```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]

scaler = MinMaxScaler()

print(scaler.fit(data))

MinMaxScaler()

print("data:\n",scaler.data_max_)

print("Transformed data:\n",scaler.transform(data))
```

OUTPUT

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

data:

```
 [ 1. 18.]
Transformed data:
 [[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
```

WEEK - 6

ASSOCIATION RULE MINING – APRIORI

Write a python program to find rules that describe associations by using Apriori algorithm between different products given as 7500 transactions at a French retail store.
a) Display top 5 rows of data
b) Find the rules with min_confidence : .2, min_support= 0.0045, min_lift=3, min_length=2

Libraries: NumPy, SciPy, Matplotlib, Pandas
Dataset: https://drive.google.com/file/d/1y5DYn0dGoSbC22xowBq2d4po6h1JxcTQ/view?usp=sharing

RESOURCES:

c) Python 3.7.0
d) Install: pip installer, pandas, SciPy library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

Install Anaconda

Open spyder IDE:

Spyder is an Integrated Development Environment (IDE) for scientific computing, written in and for the Python programming language. It comes with an Editor to write code, a Console to evaluate it and view the results at any time, a Variable Explorer to examine the variables defined during evaluation, and several other facilities

Steps in Apriori:

1. Set a minimum value for support and confidence. This means that we are only interested in finding rules for the items that have certain default existence (e.g. support) and have a minimum value for co-occurrence with other items (e.g. confidence).

2. Extract all the subsets having higher value of support than minimum threshold.

3. Select all the rules from the subsets with confidence value higher than minimum threshold.

4. Order the rules by descending order of Lift.

Example:

```
from apyori import apriori

transactions = [

   ['beer', 'nuts'],

   ['beer', 'cheese'],

]
```

#CASE1:

```
results = list(apriori(transactions))

association_results = list(results)

print(results[0])

#CASE2: min support=.5,minconfidence=.8
```

```
results = list(apriori(transactions,min_support=0.5, min_confidence=0.8))
association_results = list(results)
print(len(results))
print(association_results)
```

OUTPUT:

5

RelationRecord(items=frozenset({'beer'}), support=1.0, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'beer'}), confidence=1.0, lift=1.0)])

Case 2:

3

[RelationRecord(items=frozenset({'beer'}), support=1.0, ordered_statistics=[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'beer'}), confidence=1.0, lift=1.0)]),

RelationRecord(items=frozenset({'cheese', 'beer'}), support=0.5, ordered_statistics=[OrderedStatistic(items_base=frozenset({'cheese'}), items_add=frozenset({'beer'}), confidence=1.0, lift=1.0)]),

RelationRecord(items=frozenset({'nuts', 'beer'}), support=0.5, ordered_statistics=[OrderedStatistic(items_base=frozenset({'nuts'}), items_add=frozenset({'beer'}), confidence=1.0, lift=1.0)])]

Three major measures to validate Association Rules:

- Support
- Confidence
- Lift

Suppose a record of 1 thousand customer transactions. Consider two items e.g. burgers and ketchup. Out of one thousand transactions, 100 contain ketchup while 150 contain a burger. Out of 150 transactions where a burger is purchased, 50 transactions contain ketchup as well. Using this data, Find the support, confidence, and lift.

Support:

Support(B) = (Transactions containing (B))/(Total Transactions)

For instance if out of 1000 transactions, 100 transactions contain Ketchup then the support for item Ketchup can be calculated as:

Support(Ketchup) = (Transactions containingKetchup)/(Total Transactions)

Support(Ketchup) = 100/1000 = 10%

Confidence

Confidence refers to the likelihood that an item B is also bought if item A is bought. It can be calculated by finding the number of transactions where A and B are bought together, divided by total number of transactions where A is bought.

Confidence(A→B) = (Transactions containing both (A and B))/(Transactions containing A)

A total of 50 transactions where Burger and Ketchup were bought together. While in 150 transactions, burgers are bought. Then we can find likelihood of buying ketchup when a burger is bought can be represented as confidence of Burger -> Ketchup and can be mathematically written as:

Confidence (Burger→Ketchup) = (Transactions containing both (Burger and Ketchup))/(Transactions containing A)

Confidence(Burger→Ketchup) = 50/150 = 33.3%

Lift

Lift (A -> B) refers to the increase in the ratio of sale of B when A is sold. Lift(A –> B) can be calculated by dividing Confidence(A -> B) divided by Support(B). Mathematically it can be represented as:

Lift (A→B) = (Confidence (A→B))/(Support (B))

In Burger and Ketchup problem, the Lift (Burger -> Ketchup) can be calculated as:

Lift (Burger → Ketchup) = (Confidence (Burger → Ketchup))/(Support (Ketchup))

Lift(Burger → Ketchup) = 33.3/10 = 3.33

a) Display top 5 rows of data

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori
store_data = pd.read_csv("D:/datasets/store_data.csv")
print(store_data.head())
print('Structure of store data\n',str(store_data))
```

OUTPUT:

```
shrimp    almonds     avocado    vegetables mix green grapes  \
0        burgers  meatballs       eggs             NaN         NaN
1        chutney       NaN        NaN              NaN         NaN
2         turkey    avocado        NaN              NaN         NaN
3   mineral water       milk  energy bar  whole wheat rice    green tea
4  low fat yogurt       NaN        NaN              NaN         NaN

  whole weat flour yams cottage cheese energy drink tomato juice  \
0             NaN  NaN           NaN          NaN          NaN
1             NaN  NaN           NaN          NaN          NaN
2             NaN  NaN           NaN          NaN          NaN
3             NaN  NaN           NaN          NaN          NaN
4             NaN  NaN           NaN          NaN          NaN

  low fat yogurt green tea honey salad mineral water salmon antioxydant juice  \
0           NaN       NaN   NaN   NaN          NaN    NaN              NaN
1           NaN       NaN   NaN   NaN          NaN    NaN              NaN
2           NaN       NaN   NaN   NaN          NaN    NaN              NaN
```

```
3          NaN      NaN   NaN   NaN         NaN    NaN            NaN
4          NaN      NaN   NaN   NaN         NaN    NaN            NaN

  frozen smoothie spinach  olive oil
0            NaN     NaN       NaN
1            NaN     NaN       NaN
2            NaN     NaN       NaN
3            NaN     NaN       NaN
4            NaN     NaN       NaN
Structure of store data
              shrimp          almonds     avocado    vegetables mix  \
0          burgers        meatballs        eggs             NaN
1          chutney              NaN         NaN             NaN
2           turkey          avocado         NaN             NaN
3     mineral water              milk   energy bar  whole wheat rice
4    low fat yogurt              NaN         NaN             NaN
...             ...               ...         ...              ...
7495         butter         light mayo  fresh bread              NaN
7496        burgers  frozen vegetables         eggs     french fries
7497        chicken               NaN         NaN             NaN
7498       escalope          green tea         NaN             NaN
7499           eggs    frozen smoothie  yogurt cake    low fat yogurt

     green grapes whole weat flour yams cottage cheese energy drink  \
0            NaN              NaN  NaN            NaN         NaN
1            NaN              NaN  NaN            NaN         NaN
2            NaN              NaN  NaN            NaN         NaN
3      green tea              NaN  NaN            NaN         NaN
4            NaN              NaN  NaN            NaN         NaN
...          ...              ...  ...            ...         ...
7495         NaN              NaN  NaN            NaN         NaN
7496   magazines        green tea  NaN            NaN         NaN
7497         NaN              NaN  NaN            NaN         NaN
7498         NaN              NaN  NaN            NaN         NaN
7499         NaN              NaN  NaN            NaN         NaN

     tomato juice low fat yogurt green tea honey salad mineral water salmon  \
0            NaN           NaN       NaN   NaN   NaN           NaN    NaN
1            NaN           NaN       NaN   NaN   NaN           NaN    NaN
2            NaN           NaN       NaN   NaN   NaN           NaN    NaN
```

```
3        NaN       NaN     NaN   NaN   NaN       NaN    NaN
4        NaN       NaN     NaN   NaN   NaN       NaN    NaN
...      ...       ...     ...   ...   ...       ...    ...
7495     NaN       NaN     NaN   NaN   NaN       NaN    NaN
7496     NaN       NaN     NaN   NaN   NaN       NaN    NaN
7497     NaN       NaN     NaN   NaN   NaN       NaN    NaN
7498     NaN       NaN     NaN   NaN   NaN       NaN    NaN
7499     NaN       NaN     NaN   NaN   NaN       NaN    NaN

     antioxydant juice frozen smoothie spinach  olive oil
0               NaN           NaN     NaN       NaN
1               NaN           NaN     NaN       NaN
2               NaN           NaN     NaN       NaN
3               NaN           NaN     NaN       NaN
4               NaN           NaN     NaN       NaN
...             ...           ...     ...       ...
7495            NaN           NaN     NaN       NaN
7496            NaN           NaN     NaN       NaN
7497            NaN           NaN     NaN       NaN
7498            NaN           NaN     NaN       NaN
7499            NaN           NaN     NaN       NaN

[7500 rows x 20 columns]
```

c) Find the rules with min_confidence : .2, min_support= 0.0045, min_lift=3, min_length=2

Let's suppose that we want rules for only those items that are purchased at least 5 times a day, or 7 x 5 = 35 times in one week, since our dataset is for a one-week time period.

The support for those items can be calculated as 35/7500 = 0.0045.

The minimum confidence for the rules is 20% or 0.2.

Similarly, the value for lift as 3 and finally min_length is 2 since at least two products should exist in every rule.

#Converting data frame to list

```
records = []
for i in range(0, 7500):
    records.append([str(store_data.values[i,j]) for j in range(0, 20)])
```

#Generating association rules using apriori()

#association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)

```
association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=5)
association_results = list(association_rules)
print(len(association_results))
```

```
print(association_results[0])
for item in association_rules:
 # first index of the inner list
 # Contains base item and add item
 pair = item[0]
 items = [x for x in pair]
 print("Rule: " + items[0] + " -> " + items[1])
 #second index of the inner list
 print("Support: " + str(item[1]))
 #third index of the list located at 0th
 #of the third index of the inner list
 print("Confidence: " + str(item[2][0][2]))
 print("Lift: " + str(item[2][0][3]))
 print("=====================================")
```

OUTPUT:

#association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)

RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004533333333333334, ordered_statistics=[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'chicken'}), confidence=0.2905982905982906, lift=4.843304843304844)])

#association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=5)

No of Rules: 48

RelationRecord(items=frozenset({'chicken', 'light cream'}), support=0.004533333333333334, ordered_statistics=[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'chicken'}), confidence=0.2905982905982906, lift=4.843304843304844)])

Rule: light cream -> chicken Support: 0.004532728969470737 Confidence: 0.29059829059829057 Lift: 4.84395061728395

Rule: mushroom cream sauce -> escalope Support: 0.005732568990801126 Confidence: 0.3006993006993007 Lift: 3.790832696715049

Rule: escalope -> pasta Support: 0.005865884548726837 Confidence: 0.3728813559322034 Lift: 4.700811850163794

Rule: ground beef -> herb & pepper Support: 0.015997866951073192 Confidence: 0.3234501347708895 Lift: 3.2919938411349285

WEEK - 7

CLASSIFICATION – LOGISTIC REGRESSION

Classification of Bank Marketing Data

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The dataset provides the bank customers' information. It includes 41,188 records and 21 fields. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y).
Write a python program to
a) Explore data and visualize each attribute
b) Predict the test set results and find the accuracy of the model
c) Visualize the confusion matrix
d) Compute precision, recall, F-measure and support

RESOURCES:

e) Python 3.7.0
f) Install: pip installer, pandas, SciPy, NumPy, Sklearn, Seaborn library

PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

PROGRAM LOGIC:

a) Explore data and visualize each attribute

```
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
import seaborn as sns
from pandas.plotting import scatter_matrix
from sklearn.linear_model import LogisticRegression
#Reading dataset
bank=pd.read_csv("D:/datasets/bank-additional-full.csv", index_col=0)
 # index_col will remove the index column from the csv file
# Assign outcome as 0 if income <=50K and as 1 if income >50K
bank['y'] = [0 if x == 'no' else 1 for x in bank['y']]

# Assign X as a DataFrame of features from bank dataset and y as a Series of the outcome variable
# axis : {0 or 'index', 1 or 'columns'}, default 0
# Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').
X = bank.drop('y', 1) # 1 represents column, dropping y column  for doing classification
y = bank.y
X.describe()
```

duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	job_admin.	...	month_oct	month_sep	day_of_week_fri	day_of_week_mon	day_of_week_thu	day_of_week_tue	day_of_week_wed	poutcome_failure	poutcome_nonexistent	poutcome_success
count	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	...	41188	41188	41188	41188	41188	41188	41188	41188	41188

mean	std	min	25%	50%	75%	max
258.28501	259.279249	0	102	180	319	4918
2.567593	2.770014	1	1	2	3	56
962.47545	186.91091	0	999	999	999	999
0.172963	0.494901	0	0	0	0	7
0.081886	1.57096	-3.4	-1.8	1.1	1.4	1.4
93.575664	0.57884	92.201	93.075	93.749	93.994	94.767
-40.5026	4.628198	-50.8	-42.7	-41.8	-36.4	-26.9
3.621291	1.734447	0.634	1.344	4.857	4.961	5.045
5167.0359	72.251528	4963.6	5099.1	5191	5228.1	5228.1
0.253035	0.434756	0	0	0	1	1
...	...	...	...	...	...	...
0.017432	0.130877	0	0	0	0	1
0.013839	0.116824	0	0	0	0	1
0.190031	0.39233	0	0	0	0	1
0.206711	0.404951	0	0	0	0	1
0.209357	0.406855	0	0	0	0	1
0.196416	0.397292	0	0	0	0	1
0.197485	0.398106	0	0	0	0	1
0.103234	0.304268	0	0	0	0	1
0.863431	0.343396	0	1	1	1	1

```
y.describe()
count    41188.0
mean         1.0
std          0.0
min          1.0
25%          1.0
50%          1.0
75%          1.0
max          1.0
Name: y, dtype: float64
```

X.head()

age	**job**	**marital**	**education**	**default**	**housing**	**loan**	**contact**	**month**	**day_of_week**	**duration**	**campaign**	**pdays**	**previous**	**poutcome**	**emp.var.rate**	**cons.price.idx**	**cons.conf.idx**	**euribor3m**	**nr.employed**	**y**
56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
56	services	married	high.school	no	no	yes	teleph													

```
y.head()
age
56    0
57    0
37    0
40    0
56    0
Name: y, dtype: int64
```

```
#Count of unique values(y/n)

bank['y'].value_counts()
```

OUTPUT:

4640 people opened term deposit account and 36548 have not opened the term deposit account

```
0    36548
1     4640
Name: y, dtype: int64
```

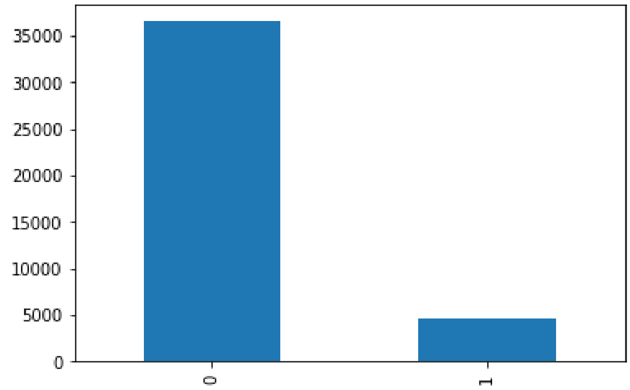
```
# Decide which categorical variables you want to use in model

for col_name in X.columns:
    if X[col_name].dtypes == 'object':# in pandas it is object
        unique_cat = len(X[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} unique categories".format(col_name=col_name,
unique_cat=unique_cat))
        print(X[col_name].value_counts())
        print()
```

OUTPUT:

```
Feature 'job' has 12 unique categories
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64

Feature 'marital' has 4 unique categories
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64

Feature 'education' has 8 unique categories
university.degree      12168
high.school             9515
```

```
basic.9y               6045
professional.course    5243
basic.4y               4176
basic.6y               2292
unknown                1731
illiterate               18
Name: education, dtype: int64

Feature 'default' has 3 unique categories
no         32588
unknown     8597
yes            3
Name: default, dtype: int64

Feature 'housing' has 3 unique categories
yes        21576
no         18622
unknown      990
Name: housing, dtype: int64

Feature 'loan' has 3 unique categories
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64

Feature 'contact' has 2 unique categories
cellular     26144
telephone    15044
Name: contact, dtype: int64

Feature 'month' has 10 unique categories
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182

Name: month, dtype: int64

Feature 'day_of_week' has 5 unique categories
thu    8623
mon    8514
wed    8134
tue    8090
fri    7827
Name: day_of_week, dtype: int64

Feature 'poutcome' has 3 unique categories
nonexistent    35563
failure         4252
success         1373
Name: poutcome, dtype: int64
```

Visualizations
#visualization of Predictor variable (y)
print(y.value_counts().plot.bar())



b) Predict the test set results and find the accuracy of the model

#Create an Logistic classifier and train it on 70% of the data set.

```
clf = LogisticRegression()

clf
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)

clf.fit(X, y)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

c) Visualize the confusion matrix

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

d) Compute precision, recall, F-measure and support

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

**** https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8