CS3362 Foundations OF DATA Science LAB Manual

Computer Science and Engineering (Anna University)

LIST OF EXPERIMENTS

CS3362 DATA SCIENCE LABORATORY L T P C
0 0 4 2

COURSE OBJECTIVES:

- To understand the python libraries for data science
- To understand the basic Statistical and Probability measures for data science.
- To learn descriptive analytics on the benchmark data sets.
- To apply correlation and regression analytics on standard data sets.
- To present and interpret data using visualization packages in Python.

LIST OF EXPERIMENTS:

L Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels, and Pandas packages.

2. . Working with Numpy arrays.

3. Working with Pandas data frames.

4. Reading data from text files, Excel, and the web and exploring various commands for doing descriptive analytics on the Iris data set.

5. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following:

a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness, and Kurtosis.

b. Bivariate analysis: Linear and logistic regression modeling

c. Multiple Regression analysis

d. Also compare the results of the above analysis for the two data sets.

6. Apply and explore various plotting functions on UCI data sets.

a. Normal curves

b. Density and contour plots

c. Correlation and scatter plots

d. Histograms

e. Three-dimensional plotting

7. Visualizing Geographic Data with Basemap

List of Equipments:(30 Students per Batch)
Tools: Python, Numpy, Scipy, Matplotlib, Pandas, statmodels, seaborn, plotly, bokeh
Note: Example data sets like: UCI, Iris, Pima Indians Diabetes etc.

EX. N0.:1 PACKAGES FOR DATA SCIENCE IN PYTHON

DATE

AIM:

To download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels, and Pandas packages in Python.

ALGORITHM:

Python is an open-source, object-oriented, and cross-platform programming language. Compared to programming languages like C++ or Java, Python is very concise. It allows us to build a working software prototype in a very short time. It has become the most used language in the data scientist's toolbox. It is also a general-purpose language, and it is very flexible due to a variety of available packages that solve a wide spectrum of problems and necessities. To install the necessary packages, use 'pip'.

Anaconda (http://continuum.io/downloads) is a Python distribution offered by Continuum Analytics that includes nearly 200 packages, which comprises NumPy, SciPy, pandas, Jupyter, Matplotlib, Scikit-learn, and NLTK. It is a cross-platform distribution (Windows, Linux, and Mac OS X) that can be installed on machines with other existing Python distributions and versions. Its base version is free; instead, add-ons that contain advanced features are charged separately. Anaconda introduces 'conda', a binary package manager, as a command-line tool to manage your package installations. Anaconda's goal is to provide enterprise-ready Python distribution for large-scale processing, predictive analytics, and scientific computing.

1. Download Anaconda
2. Install Anaconda
3. Start Anaconda
4. Install data science packages

1. Download Anaconda

This step downloads the Anaconda Python package for the Windows platform.

Anaconda is a free and easy-to-use environment for scientific Python.

1. Visit the Anaconda homepage.
2. Click "Anaconda" from the menu and click "Download" to go to the download page.
3. Choose the download suitable for your platform (Windows, OSX, or Linux):
    - Choose Python 3.5
    - Choose the Graphical Installer

2. Install Anaconda

This step installs the Anaconda Python software on the system.

This step assumes that sufficient administrative privileges are contained to install software on the system.

1. Double click the downloaded file.
2. Follow the installation wizard.

3. Start Anaconda

Anaconda comes with a suite of graphical tools called Anaconda Navigator. Start Anaconda Navigator by opening it from the application launcher.

First, start with the Anaconda command line environment called conda.
Conda is fast, and simple, it's hard for error messages to hide, and you can quickly confirm your environment is installed and working correctly.

1. Open a terminal (command line window).
2. Confirm conda is installed correctly, by typing:

conda -V

```
condo 4.2.9
```

3. Confirm Python is installed correctly by typing:

python -V

```
Python 3.5.2 :: Anaconda 4.2.0(x86_64)
```

4. Install packages

With pip, a package is installed. To install the < package-name > generic package, run this command:
$> pip install < package-name >

To install the <package-name> generic package, you just need to run the following command:

$> **conda install <package-name>**

To install a particular version of the package

$> **conda install <package-name>=l.11.0**

To install multiple packages at once by listing all their names:

$> **conda install <package-name-1> <package-name-2>**

To update a package that you previously installed, you can keep on using conda:

$> **conda update <package-name>**

To update all the available packages simply by using the --all argument

$> **conda update -all**

To uninstall packages using conda:

$> **conda remove <package-name>**

Installing Data Science Packages

a. NumPy

NumPy is the true analytical workhorse of the Python language. It provides the user with multidimensional arrays, along with a large set of functions to operate a multiplicity of mathematical operations on these arrays. Arrays are blocks of data arranged along multiple dimensions, which implement mathematical vectors and matrices. Characterized by optimal memory allocation, arrays are useful not just for storing data, but also for fast matrix operations (vectorization), which are indispensable when solving ad hoc data science problems.

$> **conda install numpy**

b. SciPy

SciPy completes NumPy's functionalities, offering a larger variety of scientific **algorithms for linear algebra, sparse matrices, signal and image processing**, optimization, fast Fourier transformation, and much more.

$> **conda install scipy**

c. Statsmodels

Statsmodels is a complement to SciPy's statistical functions. It features generalized linear models, discrete choice models, time series analysis, and a series of descriptive statistics as well as parametric and nonparametric tests.

$> **conda install -c conda-forge statsmodels**

d. Pandas

The pandas package deals with everything that NumPy and SciPy cannot do. Thanks to its specific data structures, namely DataFrames and Series, pandas allow us to handle complex tables of data of different types and time series. It enables the easy and smooth loading of data from a variety of sources. The data can then be sliced, diced, handled with missing elements, added, renamed, aggregated, reshaped, and finally visualized.

$> **conda install pandas**

e. Jupyter

A scientific approach requires the fast experimentation of different hypotheses in a reproducible fashion. Initially named IPython and limited to working only with the Python language, Jupyter was created to address the need for an interactive command shell for several languages (based on the shell, web browser, and application interface), featuring graphical integration, customizable commands, rich history (in the JSON format), and computational parallelism for enhanced performance .

Steps to install Jupyter using Anaconda

- Launch Anaconda Navigator
- Click on the Install Jupyter Notebook Button

RESULT

Thus, the Anaconda Navigator has been successfully downloaded and installed and the data science packages such as NumPy , SciPy, Jupyter, Statsmodels, and Pandas packages were explored in Python.

WORKING WITH NUMPY ARRAYS

EX. N0.:2.a BASIC NUMPY OPERATIONS

DATE:

AIM:

To perform basic NumPy operations in python for

(i) creating different types of NumPy arrays and displaying basic information , such as the data type, shape, size, and strides

(ii) creating an array using built-in NumPy functions

(iii) performing file operations with NumPy arrays

ALGORITHM:

Step 1: Start the program.

Step 2: Import the NumPy Library .

Step 3: Define a one-dimensional array, two-dimensional array, and three-dimensional array.

Step 4: Print the memory address , the shape, the data type, and the stride of the array.

Step 5: Then, create an array using built-in NumPy functions.

Step 6: Perform file operations with NumPy arrays.

Step 7: Display the output

Step 8: Stop the program.

PROGRAM:

(i) Creation of different types of Numpy arrays and displaying basic information

```
# Importing numpy
import numpy as np

# Defining 1D array
mylDArray = np.array([l , 8, 27, 64]

print(myl DArray)

# Defining and printing 2D array
my2DArray = np.array([[l , 2, 3, 4], [2, 4, 9, 16], [4, 8, 18, 32]])
print(my2DArray)

#Defining and printing 3D array
my3Darray = np.array([[[ 1, 2 , 3 , 4],[ 5 , 6 , 7 ,8]], [[ 1, 2, 3, 4],[ 9, 10, 11,
 12]]])
print(my3Darray)
# Print out memory address
print(my2DArray.data)
# Print the shape of array
print(my2DArray .shape)

# Print out the data type of the array
print(my2DArray .dtype)

# Print the stride of the array.
print(my2DArray .strides)
```

(ii) Creation of an array using built-in NumPy functions

```
# Array of ones
ones = np.ones((3 ,4))
print(ones)

# Array of zeros
zeros = np.zeros((2 ,3,4),dtype=np.int16)
print(zeros)

# Array with random values
np.random.random((2,2))
```

```
# Empty array
emptyArray = np.empty((3,2))
print( emptyArray)

# Full array
fullArray = np.full((2,2),7)
print(fullArray)
 # Array of evenly-spaced values
evenSpacedArray = np.arange( l0,25,5)
print( evenSpacedArray)

# Array of evenly-spaced values
evenSpacedArray2 = np.linspace(0,2,9)
print( evenSpacedArray2)
```

(iii) Performing file operations with NumPy arrays

```
import numpy as np

#initialize an array
arr = np.array([[[ll, 11, 9, 9], [11, 0, 2, O]], [[10, 14, 9, 14], [O, 1, 11, 11]]])

# open a binary file in write mode
file = open("arr", "wb")

# save array to the file
np.save(file, arr)

# close the file
file.close

# open the file in read binary mode
file = open("arr", "rb")

#read the file to numpy array
arr1 = np.load(file)
#close the file
print( arr1)
```

OUTPUT:

(i) Creation of different types of Numpy arrays and displaying basic information

```
[ 1  8 27 64]

[[ 1  2  3  4]

 [ 2  4  9 16]

 [ 4  8 18 32]]

[[[ 1  2  3  4]

  [ 5  6  7  8]]
 [[ 1  2  3  4]
  [ 9 10 11 12]]]

<memory at Ox000002 47AE2AOAOO >

(3, 4)

int32

(16, 4)
```

(ii) Creation of an array using built-in NumPy functions

```
[[l. 1. 1. l.]
 [l. 1. 1. l.]
 [1. 1. 1. 1.]]

[[[O 0 0 O]
  [O 0 0 O]
  [O 0 0 O]]
```

```
[[0 0 0 0J
 [0 0 0 0J
 [0 0 0 0J]]

[[O.O.]

 [O. O.]
 [O. O.]]

[[7 7]

 [7 7]]

[10 15 20]

[O.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
```

(iii) Performing ftle operations with NumPy arrays

```
[[[11 11  9  9]

  [11 0 2 0J]


 [[10 14  9 14]

  [ O   1 11 11]]]
```

RESULT:

Thus, the program to implement NumPy operations with arrays using Python has been executed and the output was verified successfully.

EX. **N0.:2.b BASIC ARITHMETIC OPERATIONS WITH NUMPY ARRAYS**

DATE

AIM:

To implement arithmetic operations with NumPy arrays using python.

ALGORITHM:

Step 1: Start the program.

Step 2: Import the NumPy Library.

Step 3: Initialize the NumPy arrays to two different variables.

Step 4: Perform the arithmetic operations on the two arrays using NumPy.

Step 5: Display the output.

Step 6: Stop the program.

PROGRAM:

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)

print ('First array:')
print (a)
print ('\n')

print ('Second array:')
b = np.array([l0,10,10])
print (b )
print ('\n')

print ('Add the two arrays:')
print (np.add(a,b))
print ('\n')

print ('Subtract the two arrays:')
print (np.subtract(a,b))
print ('\n')

print ('Multiply the two arrays:')
print (np.multiply(a,b))
print ('\n')

print ('Divide the two arrays:')
print (np.divide(a,b))
```

OUTPUT:

```
First array:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]
Second array:
[10 10 10]

Add the two arrays:
[[ 10. 11. 12.]
 [ 13. 14. 15.]
 [ 16. 17. 18.]]
Subtract the two arrays:
[[-10. -9. -8.]
 [ -7. -6. -5.]
 [ -4. -3. -2.]]

Multiply the two arrays:
[[ 0. 10. 20.]
 [ 30. 40. 50.]
 [ 60. 70. 80.]]
Divide the two arrays:
[[ 0. 0.1 0.2]
 [ 0.3 0.4 0.5]
 [ 0.6 0.7 0.8]]
```

RESULT:
Thus, the program to implement NumPy arithmetic operations with arrays using Python has been executed and the output was verified successfully.

EX. N0.:3 WORKING WITH PANDAS DATAFRAMES

DATE:

AIM:

(i) To create a dataframe from a series
(ii) To create a dataframe from a dictionary
(iii) To create a dataframe from n-dimensional arrays
(iv) To load a dataset from an external source into a pandas dataframe

ALGORITHM:

Step 1: Start the program.

Step 2: Import the NumPy and pandas packages.

Step 3: Create a dataframe for the list of elements (numbers, dictionary, and n-dimensional arrays)

Step 4: Load a dataset from an external source into a pandas dataframe

Step 5: Display the output.

Step 6: Stop the program.

PROGRAM:

(i) CREATION OF A DATAFRAME FROM A SERIES

```
import numpy as np
import pandas as pd
print("Pandas Version :", pd ._version_)
pd.set_option('display.max_columns',500)
pd.set_option('display.max_rows' ,  500)
series = pd.Series([2 , 3, 7, 11, 13, 17, 19, 23])
print( series)
series_df = pd .DataFrame( {
'A': range(l , 5),
'B': pd .Timestamp('20190526') ,
'C': pd .Series(5 , index=list(range(4)) , dtype='float64') ,
'D': np.array([3] * 4, dtype='int64'),
'E': pd .Categorical(["Depression" , "Social Anxiety" , "Bipolar Disorder" , "Eating
Disorder"]),
'F': 'Mental health',
'G': 'is challenging'
 })
print( series_df)
```

(ii) CREATION OF A DATAFRAME FROM DICTIONARY

```
import numpy as np
import pandas as pd

dict_df = [{'A': 'Apple', 'B': 'Ball'},{'A': 'Aeroplane' , 'B':'Bat', 'C': 'Cat'}]
diet_df = pd .DataFrame( diet_df)
print(diet_df)
```

(iii) CREATION OF A DATAFRAME FROM N-DIMENSIONAL ARRAYS

```
import numpy as np
import pandas as pd

sdf = {'County':['Ostfold', 'Hordaland' , 'Oslo', 'Hedmark', 'Oppland' , 'Buskemd'] ,
'ISO-Code' :[l,2,3,4,5,6],
'Area': [4180.69, 4917.94 , 454.07 , 27397.76 , 25192 .10, 14910.94],
'Administrative centre' : ["Sarpsborg", "Oslo" , "City of Oslo" , "Hamar",
"Lillehammer" , "Drammen"]}
sdf = pd.DataFrame(sdf)
print(sdf)
```

(iv) LOADING A DATASET FROM AN EXTERNAL SOURCE INTO A PANDAS DATAFRAME

```
import numpy as np
import pandas as pd
columns=['age', 'workclass', 'fnlwgt', 'education', 'education_num' , 'marital_status' , 'occupation', 'relationship', 'ethnicity' , 'gender', 'capital_gain' , 'capital_loss','hours_per_week','country_of_origin','income']
df=pd.read_csv('http ://archive.ics.uci.edu/ml/machine-leaming-databases/adult/adult.data' ,names=columns)
df.head(lO)
```

OUTPUT:

(i) Creation of a dataframe from a series

```
Pandas Version : 1.3.4
0    2
1    3
2    7
3    11
4    13
5    17
6    19
7    23
dtype: int64
A       B   C  D            E           F            G
0  1 2019-05-26  5.0  3      Depression  Mental health  is challenging
1  2 2019-05-26  5.0  3    Social Anxiety  Mental health  is challenging
2  3 2019-05-26  5.0  3  Bipolar Disorder  Mental health  is challenging
3  4 2019-05-26  5.0  3   Eating Disorder  Mental health  is challenging
```

(ii) Creation of a dataframe from a dictionary

```
           A       B      C

   0   Apple      Ball   NaN
   1   Aeroplane  Bat  Cat
```

(iii) Creation of a dataframe from n-dimensional array

```
County  ISO-Code    Area        Administrative centre

0          Ostfold      1   4180.69          Sarpsborg

1        Hordaland     2   4917.94             Oslo

2            Oslo        3    454.07        City of Oslo

3          Hedmark      4  27397.76            Hamar

4          Oppland     5   25192.10        Lillehamme

5          Buskerud    6   14910.94          Drammen
```

RESULT:

Thus, the programs for creating and loading pandas dataframes using Python have been implemented and the output was verified successfully.

READING DATA FROM TEXT FILES, EXCEL AND THE WEB

EX.N0.4.

DATE:

Aim:

To Reading data from text files, Excel and the web using pandas package.

ALGORITHM:

STEP 1: Start the program

STEP 2: To read data from csv file using pandas
package. **STEP 3:** To read data from excel file using
pandas package. **STEP 4:** To read data from html file
using pandas package. **STEP 5:** Display the output.

STEP 6: Stop the program.

PROGRAM:

DATA INPUT AND OUTPUT

This notebook is the reference code for getting input and output, pandas can read a variety of file types using its pd.read_ methods. Let's take a look at the most common data types:

```
import numpy
as np import
pandas as pd
```

CSV

CSV INPUT:

```
df =
pd.read_csv('exam
ple') df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

CSV OUTPUT:

```
df.to_csv('example',index=False)
```

EXCEL

Pandas can read and write excel files, keep in mind, this only imports data. Not formulas or images, having images or macros may cause this read_excel method to crash.

EXCEL INPUT :

```
pd.read_excel('Excel_Sample.xlsx'  ,sheetname='Sheet 1')
```

```
    a   b   c   d
0   0   1   2   3
1   4   5   6   7
2   8   9  10  11
3  12  13  14  15
```

EXCEL OUTPUT :

```
df.to_excel('Excel_Sample.xlsx' ,sheet_name='Sheet 1')
```

HTML

You may need to install htmllib5, lxml, and Beautifu1Soup4. In your terminal/command prompt run:

```
pip install lxml
pip install
html5lib== 1.1
pip install
Beautifu1Soup
4
```

Then restart Jupyter Notebook. (or use conda

install) Pandas can read table tabs off of html.

For example:

HTML INPUT

Pandas read_html function will read tables off of a webpage and return a list of DataFrame objects:

url = https ://www.fdic.gov/resources/resolutions/bank-

failures/failed-bank-list df = pd.read_html(url)

df[O]

match = "Metcalf Bank"

df_list = pd.read_html(url,

match=match) df_list[O]

HTML OUTPUT:

	Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date	Loss Share Type	Agreement Terminated	Termination Date
0	First Cornerstone Bank	King of Prussia	PA	35312	First-Citizens Bank & Trust Company	May 6, 2016	July 12, 2016	none	NaN	NaN
	Trust Company Bank	Memphis	TN	9956	The Bank of Fayette County	Apnl 29, 2016	August 4, 2016	none	NaN	NaN
2	North Milwaukee State Bank	Milwaukee	WI	20364	First-Citizens Bank & Trust Company	March 11, 2016	June 16, 2016	none	NaN	NaN
3	Hometown National Bank	Longview	WA	35156	Twin City Bank	October 2, 2015	April 13, 2016	none	NaN	NaN
4	The Bank of Georgia	Peachtree City	GA	35259	Fidelity Bank	October 2, 2015	April 13, 2016	none	NaN	NaN
5	Premier Bank	Denver	CO	34112	United Fidelity Bank, fsb	July 10, 2015	July 12, 2016	none	NaN	NaN
6	Edgebrook Bank	Chicago	IL	57772	Republic Bank of Chicago	May B, 2015	July 12, 2016	none	NaN	NaN
7	Doral BankEn Espanol	San Juan	PR	32102	Banco Popular de Puerto Ricr1	February 27, -?O1<;	May 13, 2015	none	NaN	NaN

RESULT:

Exploring commands for read data from csv file, excel file and html are successfully executed.

EX NO 4(a). EXPLORING VARIOUS COMMANDS FOR DOING DESCRIPTIVE
DATE: **ANALYTICS ON THE IRIS DATA SET.**

AIM:

To explore various commands for doing descriptive analytics on the Iris data set.

ALGORITHM:

STEP 1: Start the program

STEP 2: To understand idea behind Descriptive Statistics.

STEP 3: Load the packages we will need and also the 'iris' dataset.

STEP 4: load_iris() loads in an object containing the iris dataset, which Istored in 'iris_obj' .

STEP 5: Basic statistics: count, mean, median, min, max

STEP 6: Display the output.

STEP 7: Stop the program.

PROGRAM:

```
import pandas as pd

from pandas import DataFrame

from skleam.datasets import load_iris

#skleam.datasetsincludes common example datasets

# A function to load in the iris dataset

iris_obj = load_iris()

# Dataset preview

iris_obj.data

iris = DataFrame(iris_obj.data, columns=iris_obj.feature_names,index=pd.Index([i for i in
range(iris_obj.data.shape[O]) ])).join(DataFrame(iris_obj .target, columns=pd.Index([" species"]),
index=pd.Index([i for i in range(iris_obj.target.shape[O])])))

iris # prints iris data

Commands

iris_obj.feature_names

iris.count()

iris.mean()

iris.median()

iris.var()

 iris.std()

 iris.max()

iris.min()

iris.describe()
```

OUTPUT:

```
                                          m)  petal width (cm)     species
count     150.000000     150.
```

RESULT:

Exploring various commands for doing descriptive analytics on the Iris data set successfully executed.

EX.NO 5. **USE THE DIABETES DATA SET FROM UCI AND PIMA INDIANS**
DATE: **DIABETES DATA SET FOR PERFORMING THE FOLLOWING:**

A) UNIVARIATE ANALYSIS: FREQUENCY, MEAN, MEDIAN, MODE, VARIANCE, STANDARD DEVIATION, SKEWNESS AND KURTOSIS.

AIM:

To explore various commands for doing Univariate analytics on the UCI AND PIMA INDIANS DIABETES data set.

ALGORITHM:

STEP 1: Start the program

STEP 2: To download the UCI AND PIMA INDIANS DIABETES data set using Kaggle.

STEP 3: To read data from UCI AND PIMA INDIANS DIABETES data set.

STEP 4: To find the mean, median, mode, variance, standard deviation, skewness and kurtosis in the given excel data set package.

STEP 5: Display the output.

STEP 6: Stop the program.

PROGRAM:

```
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set_style('darkgrid')
        %matplotlib inline
        from matplotlib.ticker import FormatStrFormatter
        import warnings
        warnings .filterwarnings('ignore')

        df = pd.read_csv('C:/Users/kirub/Documents/Learning/Untitled    Folder/diabetes.csv')
        df.head()
        df.shape
        df.dtypes
        df['Outcome']=df['Outcome']  .astype('bool')
        df.dtypes['Outcome']
        df.info()
        df.describe().T

        # Frequency# finding the unique count
        dfl  = df['Outcome'].value_counts()

        # displaying dfl
        print(dfl)
        #mean
        df.mean()
        #median
        df.median()
            #mode
         df.mode()
      #Variance
   df.var()
      #standard deviation df.std()
       #
      #kurtosis
        df.kurtosis(   axis=O,skipna=True)
        df['Outcome'].kurtosis(axis=O,skipna=True)
      #skewness
        # skewness along the index axis
        df.skew(axis = 0, skipna = True)

   # skip the na values
   # find skewness in each row
   df.skew(axis = 1, skipna = True)
```

```
#Pregnancy  variable
preg_proportion  = np.array( df['Pregnancies']. value_counts())
preg_month  = np.array( df['Pregnancies']. value_counts().index)
preg_proportion_perc  =
np.array(np.round(preg_proportion/ sum(preg_proportion  ),3)*100,dtype=int)

preg =
pd.DataFrame(  {'month':preg_month,'count_of_preg_prop' :preg_proportion,'percentage_pro
portion':preg_proportion_perc})
preg.set_index(['month']  ,inplace=True)
preg.head( 10)

sns.countplot(data=df['Outcome'])

sns.distplot(df['Pregnancies'])

sns.boxplot( data=df['Pregnancies '])
```

OUTPUT:

```
"
            coun            n          st    lll1n   25%    59    75X     x
   agt     1025 0                  9072290   29.0    .0   SG.0   61 0   77 0
   Hlt     1   5.0    0.695610   0-160373    0.0    0.0   10     1D     10
   cp      1or D      9J2   9    1.029   1   0 0    0 0   10     2 0    JO
 UISU!ps   1   5.0   131.61 707  17.5 67  6   9  .0   0.0  130.0  140.   00 0
   chol    1or o    2..! 000000  515925 0    1260   211 0  240 0  275 0  5   0
   bs      1    .O    .1 l9 68    .3565  7    0.0    0.0   0.                1 0
 restecg   1025.0    0 529756    0.527878    0 0    0 0   1 0    1 0    2 0
 lhlilch   1or o    1-1'31-11 6  2300572     7  0   132 0  157 0  166 0  O"  0
   txng    1025.0    0.336585    0 472m      0.0    0.0   0.0    1 0    10
 oldpea    1or o      0  1512    1  1ro-3    0 0    0 0   0 8    1 8     ?
  slope       .0     .385366    o. 1nss      0.0    1.0   1.0     0      0
   ca         0      0     1-1   1 0   7 s   0 0    0 0   0 0    1 0     0
   tnal    1DT D     2.323902    0620660     0 0    2.0   2 0    30     JO
```

RESULT:

Exploring various commands for doing univariate analytics on the UCI AND PIMA INDIANS DIABETES was successfully executed.

EX.NO:5. **B) BIVARIATE ANALYSIS: LINEAR AND LOGISTIC REGRESSION**
DATE: **MODELING**

AIM:

To explore the Linear and Logistic Regression model on the USA HOUSING AND UCI AND PIMA INDIANS DIABETES data set.

ALGORITHM:

STEP 1: Start the program

STEP 2: To download the any kind of data set like housing dataset using kaggle.

STEP 3: To read data from downloaded data set.

STEP 4: To find the linear and logistic regression model using the given data set.

STEP 5: Display the output.

STEP 6: Stop the program.

PROGRAM:

BIVARIATE ANALYSIS GENERAL PROGRAM

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('darkgrid')

%matplotlib inline

from matplotlib.ticker import FormatStrFormatter

import warnings

warnings .filterwarnings('ignore')
df = pd.read_csv('C:/Users/diabetes.csv')

df.head()

df.shape

df.dtypes

df['Outcome']=df['Outcome'] .astype('bool')
fig,axes = plt.subplots(nrows=3,ncols=2,dpi= l20,figsize = (8,6))
plotOO=sns .countplot('Pregnancies' ,data=df,ax=axes [0][0],color='green')

axes[O] [0].set_title('Count',fontdict= {'fontsize':8})

axes[O] [0].set_xlabel('Month of Preg.',fontdict= {'fontsize' :7})

axes[O] [0].set_ylabel('Count',fontdict={ 'fontsize':7})

    plt.tight_layout()

plotO 1=sns.countplot('Pregnancies' ,data=df,hue='Outcome' ,ax=axes [0][1])

axes[O][l].set_title('Diab.  VS Non-Diab.',fontdict={  'fontsize':8})

axes[O][ 1].set_xlabel('Month of Preg.',fontdict= {'fontsize' :7})

axes[O] [l].set_ylabel('Count',fontdict={  'fontsize':7})

plotO1.axes.legend(loc= 1)

plt.setp( axes[O] [1].get_legend() .get_texts(), fontsize='6')

plt.setp( axes[O] [1].get_legend() .get_title(), fontsize='6')

plt.tight_layout()
```

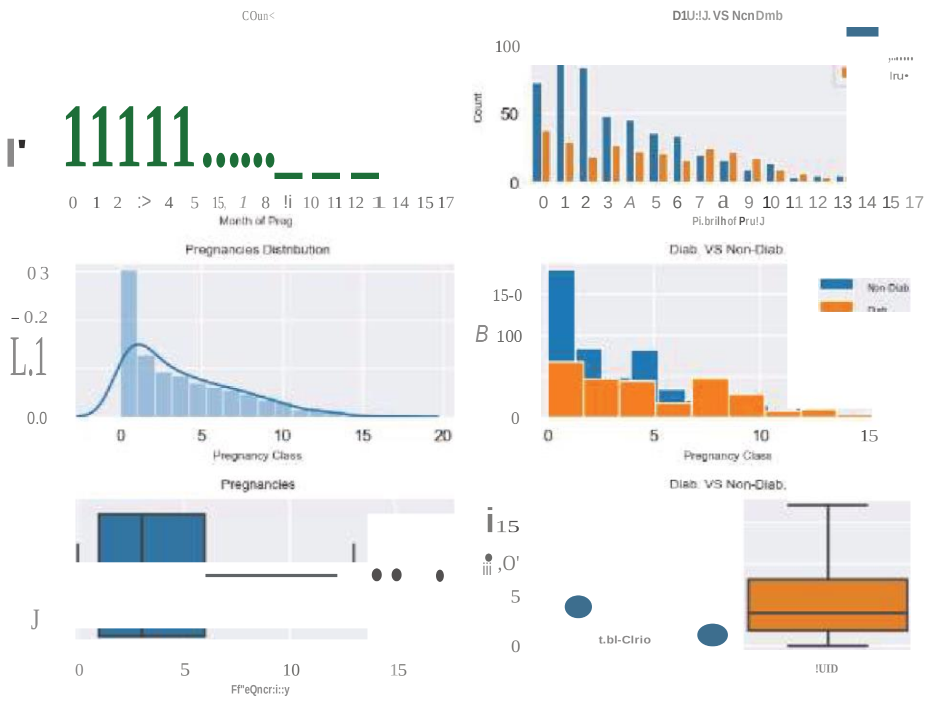
```
plot 10 = sns.distplot( df['Pregnancies '],ax=axes[ 1][0])
axes[ 1][0].set_title('Pregnancies  Distribution' ,fontdict= {'fontsize': 8})
axes[l] [0].set_xlabel('Pregnancy   Class',fontdict={  'fontsize':7})
axes[l] [0].set_ylabel('Freq/Dist',fontdict={   'fontsize':7})
plt.tight_layout()

plot 11 =  df[df['Outcome']==False]  ['Pregnancies'] .plot.hist(ax=axes[  l][l],label='Non-
Diab.')
plot l 1_2=df[df['Outcome']==True] ['Pregnancies'] .plot.hist(ax=axes[  l][l],label='Diab. ')
axes[l][l] .set_title('Diab.   VS  Non-Diab.',fontdict={  'fontsize':8})
axes[l] [l].set_xlabel('Pregnancy   Class',fontdict={  'fontsize':7})
axes[l] [l].set_ylabel('Freq/Dist',fontdict={    'fontsize':7})
plot 11.axes.legend(loc= 1)
plt.setp(axes[l][ l].get_legend().get_texts(),  fontsize='6') # for legend text
plt.setp(axes[l][ l].get_legend().get_title(),  fontsize='6') # for legend title
plt.tight_layout()

plot20  =  sns.boxplot(df['Pregnancies'] ,ax=axes [2][0],orient='v')
axes[2][0] .set_title('Pregnancies' ,fontdict={ 'fontsize':8})
axes[2] [0].set_xlabel('Pregnancy' ,fontdict={ 'fontsize':7})
axes[2][0].set_ylabel('Five  Point  Summary',fontdict={ 'fontsize':7})
plt.tight_layout()

plot2 l  =  sns.boxplot(x='Outcome' ,y='Pregnancies' ,data=df,ax=axes [2][1])
axes[2][1].set_title('Diab.  VS  Non-Diab.',fontdict={  'fontsize':8})
axes[2] [l].set_xlabel('Pregnancy'  ,fontdict={  'fontsize':7})
axes[2][1].set_ylabel('Five   Point  Summary',fontdict={ 'fontsize' :7})
plt.xticks( ticks=[O, 1],labels=['Non-Diab. ','Diab.'] ,fontsize=7)
plt.tight_layout()
plt.show()
```

OUTPUT:


Month of Preg
Count
Diab. VS Non-Diab
Pregnancies Distribution
Pregnancy Class
Non-Diab
Diab
Pregnancies
Diab. VS Non-Diab.