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. 2.3 2.3 2.3 2.3]	[[2.2 2.2 2.2 2.2 2.2]
[2.34 2.34 2.34 2.34 2.34]	[2.3 2.3 2.3 2.4 2.4]	[2.3 2.3 2.3 2.3 2.3]
[2.48 2.48 2.48 2.48 2.48]	[2.4 2.5 2.5 2.5 2.5]	[2.5 2.5 2.5 2.5 2.5]
[2.52 2.52 2.52 2.52 2.52]	[2.5 2.5 2.5 2.5 2.6]	[2.5 2.5 2.5 2.5 2.5]
[2.62 2.62 2.62 2.62 2.62]	[2.6 2.6 2.6 2.6 2.7]	[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.  ]]
```