AMRITA VISHWA VIDYAPEETHAM | School of Engineering

Department of Mechanical Engineering

21ARE301 – Introduction to Data Science

B. Tech – 5th Semester - ARE

Name of the Student	K Vinoth Kumar
Registration Number	CH.EN.U4ARE22011
Branch	Automation and Robotics Engineering
Submission Date	22 – 07 -2024

OBJECTIVES :

The objective of this assignment is to demonstrate a foundational understanding of Python programming concepts, such as *user-defined functions, input handling from user and working with basic datatypes in python.*

The following program is written and executed in Jupyter Notebook environment and can be found here.

CODE :

Program #1 – To Calculate the Radius of a Circle

```
from math import pi

def calc_area(rad):
    #Gets the Radius and returns the Area
    area = pi*(r**2)
    return area

r = float(input("Radius of Circle: "))            # Gets Input radius
area = calc_area(r)
print(f'The area of circle is {area:.2f} sq.units')
```

```
[1]  ✓ 2.7s
... The area of circle is 102.07 sq.units
```

Input : 5.7 units

Output : 102.07 sq. Units

Program #2 - Sorting numbers

```python
# Function starts with getting input from user in single line and appends them to a list.
# Then it sorts them and prints the output.
lst = []    #Empty list for storing elements
[a,b,c,d,e,f,g,h,i,j] = map(int,input("Enter any 10 random integers: ").split())  #Gets input in single line
lst.append(a)
lst.append(b)
lst.append(c)
lst.append(d)
lst.append(e)
lst.append(f)
lst.append(g)
lst.append(h)
lst.append(i)
lst.append(j)

print('initial Order: ')
for i in range(10):
    print(lst[i], end=' ')     #Prints elements one by one before sorting
print('\n')


#SORTING in ASCENDING ORDER
lst.sort()
print('After Sorting: ')
for i in range(10):
    print(lst[i], end=' ')
print('\n')

ls = sum(lst)                  #Computes the Sum and prints it
print(f'Sum: {ls}\n')
```

8m 58.6s

```
initial Order:
2 4 43 23 54 22 11 34 14 9

After Sorting:
2 4 9 11 14 22 23 34 43 54

Sum: 216
```

Input: 2 4 43 23 54 22 11 34 14 9

Program #3 - Temperature Check

```python
temp = {'Chennai': 35.3,                 #Cities and temperatures are predefined
        'Delhi': 37.4,
        'Goa': 30.7,
        'Kolkata':32.1,
        'Mumbai': 36.5,
        'Ahmedabad': 33.7,
        'Hyderabad': 38.7,
        'Nagpur': 33.0,
        'Patna': 37.9}
while True:                             #Runs Unitl a proper execution
    inp = input('Enter the city number to check the temperature: ')
    if inp in temp:                     #Checks the city name in the dictionary
        print(f'The temperature in {inp} is {temp[inp]}°C')
        break
    else:
        print('Invalid city name. Try again')
```

[6] 2.7s

```
The temperature in Mumbai is 36.5°C
```

Input : Mumbai

LEARNING OUTCOMES :

1. Gather and convert user-provided data for program execution.
2. Learn to create and utilize functions for specific tasks.
3. Create, modify, and access elements in a list.
4. Store and retrieve data using key-value pairs in a dictionary.

Department of Mechanical Engineering

21ARE301 – Introduction to Data Science

B. Tech – 5th Semester - ARE

Name of the Student	K Vinoth Kumar
Registration Number	CH.EN.U4ARE22011
Branch	Automation and Robotics Engineering
Submission Date	22 – 07 -2024

AIM :

- The aim of the code is to perform various data manipulation and preprocessing tasks using pandas and scikit-learn libraries.
- The Code used in this assignment can be found here.

CODE :

- Import necessary libraries for data manipulation, numerical operations, and preprocessing.
- Load the dataset from a CSV file into a pandas DataFrame and print its contents.

```
#Import Necessary Libraries
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler,StandardScaler

df  = pd.read_csv('datasets/dataset1.csv')      # Read the file
print(df)
```

```
   Person   Age       City Bool  Marks
0  Person2  24.0  Bangalore   No     47
1  Person3  19.0      Delhi   No     89
2  Person4   NaN     Mumbai  Yes     93
3  Person5  24.0  Hyderabad   No     85
4  Person1  17.0    Chennai  Yes     98
```

- *df.describe* - Generate descriptive statistics
- *df.shape* - Representing the dimensionality of the DataFrame
- *df.isnull* - Return a boolean same-sized object indicating if the values are NA

```
print(df['Age'])     # Access Coloumn
print('\n')

print(df.describe())  # Generate descriptive statistics4
print('\n')

print(df.shape)                    #representing the dimensionality of the DataFrame
print('\n')

print(df.isnull())                 #Return a boolean same-sized object indicating if the values are NA
```

```
0    24.0
1    19.0
2     NaN
3    24.0
4    17.0
Name: Age, dtype: float64


             Age      Marks
count   4.000000   5.000000
mean   21.000000  82.400000
std     3.559026  20.366639
min    17.000000  47.000000
25%    18.500000  85.000000
50%    21.500000  89.000000
75%    24.000000  93.000000
max    24.000000  98.000000


(5, 5)


   Person    Age   City   Bool  Marks
0   False  False  False  False  False
1   False  False  False  False  False
2   False   True  False  False  False
3   False  False  False  False  False
4   False  False  False  False  False
```

```python
print(df['Age'])    # Access Coloumn
print('\n')

print(df.describe())  # Generate descriptive statistics4
print('\n')

print(df.shape)                    #representing the dimensionality of the DataFrame
print('\n')

print(df.isnull())                 #Return a boolean same-sized object indicating if the values are NA
```

```
[18]  ✓ 0.0s

0    24.0
1    19.0
2     NaN
3    24.0
4    17.0
Name: Age, dtype: float64


             Age      Marks
count   4.000000   5.000000
mean   21.000000  82.400000
std     3.559026  20.366639
min    17.000000  47.000000
25%    18.500000  85.000000
50%    21.500000  89.000000
75%    24.000000  93.000000
max    24.000000  98.000000


(5, 5)


   Person    Age   City   Bool  Marks
0   False  False  False  False  False
1   False  False  False  False  False
2   False   True  False  False  False
3   False  False  False  False  False
4   False  False  False  False  False
```

- *df.head* - Returns first n rows
- *df.loc* - Access a group of rows and columns by label(s)
- *df.iloc* - Purely integer-location based indexing for selection by position.

```python
print(df); print('\n')

print(df.head(3))                          # Returns first n rows
print('\n')

print(df.loc[df['City']=='Bangalore'])  # Access a group of rows and columns by label(s)
print('\n')

print(df.iloc[1,1])                        # Purely integer-location based indexing for selection by position.
```

```
[13]  ✓ 0.0s

    Person   Age       City Bool  Marks
0  Person2  24.0  Bangalore   No     47
1  Person3  19.0      Delhi   No     89
2  Person4   NaN     Mumbai  Yes     93
3  Person5  24.0  Hyderabad   No     85
4  Person1  17.0    Chennai  Yes     98


    Person   Age       City Bool  Marks
0  Person2  24.0  Bangalore   No     47
1  Person3  19.0      Delhi   No     89
2  Person4   NaN     Mumbai  Yes     93


    Person   Age       City Bool  Marks
0  Person2  24.0  Bangalore   No     47


19.0
```

df.groupby - Group DataFrame using a mapper or by a Series of columns.

```
x = df.groupby(['Bool'])         # Group DataFrame using a mapper or by a Series of columns.
print(x.sum().reset_index())
print('\n')

print(x['Age'].agg(np.mean))
print('\n')

print(x.get_group('No'))         # Print Rows of a specific group
```

```
[25]  ✓ 0.0s
```

```
  Bool                Person   Age                City  Marks
0   No  Person2Person3Person5  67.0  BangaloreDelhiHyderabad    221
1  Yes         Person4Person1  17.0           MumbaiChennai    191


Bool
No     22.333333
Yes    17.000000
Name: Age, dtype: float64


    Person   Age       City Bool  Marks
0  Person2  24.0  Bangalore   No     47
1  Person3  19.0      Delhi   No     89
3  Person5  24.0  Hyderabad   No     85
```

Simple Imputer : Replace missing values using a descriptive statistic (e.g. mean, median, or most frequent) along each column or using a constant value.

```
impute = SimpleImputer(missing_values=np.nan, strategy="mean",fill_value='F')
print(df); print('\n')

data = impute.fit_transform(df['Age'].values.reshape(-1,1))[:,0]
print(data)
```

```
[24]  ✓ 0.0s
```

```
    Person   Age       City Bool  Marks
0  Person2  24.0  Bangalore   No     47
1  Person3  19.0      Delhi   No     89
2  Person4   NaN     Mumbai  Yes     93
3  Person5  24.0  Hyderabad   No     85
4  Person1  17.0    Chennai  Yes     98


[24. 19. 21. 24. 17.]
```

get_dummies() - Each variable is converted in as many 0/1 variables as there are different values. Columns in the output are each named after a value; if the input is a DataFrame, the name of the original variable is prepended to the value.

```
df1 = pd.read_csv('datasets/cars.csv')                    #Reads cars.csv file

new_cars = pd.get_dummies(df1[['Model']])
print(new_cars.to_string())
#print(new_cars)
```

[26] ✓ 0.0s Python

	Model_1	Model_2	Model_3	Model_5	Model_500	Model_A-Class	Model_A1	Model_A4	Model_A6	Model_Astra	Model_Aygo	Model_B-Max	Model_C-Class	Model_CLA	Model_Citigo	Model_Civic	Model_Cooper	Model_E-Cl
0	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False	Fa
3	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	Fa
5	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
6	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
7	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	False	Fa
8	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
9	False	False	False	False	False	False	True	False	False	False	False	False	False	False	False	False	False	Fa
10	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
11	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
12	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
13	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	True	False	Fa
14	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
15	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	False	False	Fa
16	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
17	False	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
18	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
19	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
20	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
21	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
22	False	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False	Fa
23	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
...																		
32	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False	False	False	Fa
33	False	True	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
34	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa
35	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False	Fa

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

MinMaxScaler :

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by::

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min

where min, max = feature_range.

This transformation is often used as an alternative to zero mean, unit variance scaling.

StandardScaler :

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample `x` is calculated as:

z = (x - u) / s

where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

```
cars = pd.read_csv('datasets/cars.csv')
Vol_data = cars[['Volume']]
Weight_data = cars[['Weight']]

scaler = MinMaxScaler()
stdscaler = StandardScaler()
```
[31] ✓ 0.0s

MinMaxScaler.fit_transform - Fit to data, then transform it.

StandardScaler.fit_transform - Fit to data, then transform it.

```
scaler.fit(Vol_data)
mm_data_vol = scaler.transform(Vol_data)
print(mm_data_vol)
print('\n')

scaler.fit(Weight_data)
mm_data_weight = scaler.transform(Weight_data)
print(mm_data_weight)
print('\n')

stdscaler.fit(Vol_data)
stddata_vol = stdscaler.transform(Vol_data)
print(stddata_vol)
print('\n')

stdscaler.fit(Weight_data)
stddata_weight = stdscaler.transform(Weight_data)
print(stddata_weight)
```
[39] ✓ 0.0s

```
[[0.0625]
 [0.1875]
 [0.0625]
 [0.    ]
 [0.375 ]
 [0.0625]
 [0.3125]
 [0.375 ]
 [0.375 ]
 [0.4375]
 [0.125 ]
 [0.25  ]
 [0.0625]
 [0.4375]
 [0.4375]
 [0.4375]
 [0.4375]
 [0.8125]
 [0.4375]
 [0.6875]
 [0.4375]
 [0.6875]
 [0.75  ]
 [0.4375]
 [0.6875]
```

<u>RESULT</u> :

The results show successful execution of data preprocessing steps including loading, inspecting, exploring, and transforming data.

Key operations such as handling missing values and scaling numerical features were performed, preparing the data for subsequent analysis or machine learning tasks.

These steps are crucial for ensuring the quality and consistency of the dataset, ultimately leading to more reliable and accurate modeling outcomes.

<u>LEARNING OUTCOMES</u> :

1. Gained proficiency in loading and manipulating datasets using pandas DataFrame operations.

2. Acquired the ability to inspect datasets for missing values and understand the importance of handling incomplete data.

3. Gained knowledge of different data scaling techniques, including Min-Max scaling and Standard scaling.

4. Learned to fit and transform data using these scalers to prepare features for machine learning algorithms.

5. Enhanced ability to document code and explain its functionality, purpose, and output

Department of Mechanical Engineering

21ARE301 – Introduction to Data Science

B. Tech – 5th Semester - ARE

Name of the Student	K Vinoth Kumar
Registration Number	CH.EN.U4ARE22011
Branch	Automation and Robotics Engineering
Submission Date	22 – 07 -2024

AIM :

The aim of this code is to perform comprehensive data preprocessing and exploratory data analysis on a diabetes dataset.

The Code used in this assignment can be found here

CODE :

1. Import Libraries for performing numerical operations, data manipulation & analysis, for creating plots.

```python
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.impute import SimpleImputer
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```
✓ 0.0s Python

Data – Dataset loaded from 'diabetes.csv'

Columns *'Glucose', 'skin thickness', 'DiabetesPedigreeFunction', 'BloodPressure',* and '*Insulin'* are normalized using **MinMaxScaler.**

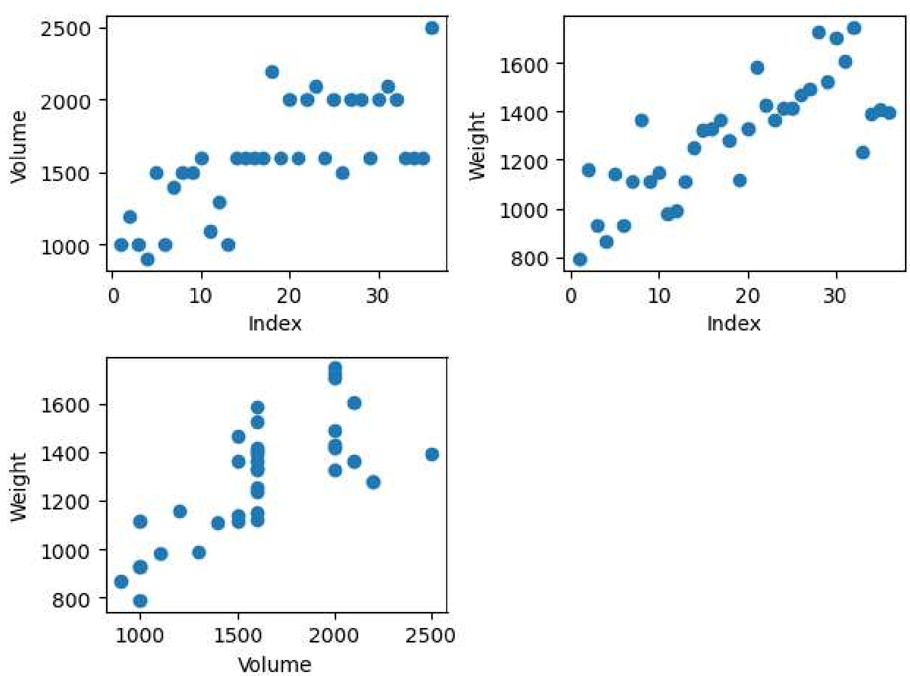
```python
data = pd.read_csv('datasets/diabetes.csv')

Glucose = data[['Glucose']]
Bp = data[['BloodPressure']]
Insulin = data[['Insulin']]
DiaPedFUnc = data[['DiabetesPedigreeFunction']]
SkinThikckness = data[['SkinThickness']]
BMI = data[['BMI']]
Age = data[['Age']]
Outcome = data[['Outcome']]
Pregnancies = data[['Pregnancies']]

scaler = MinMaxScaler()

Glucose_norm = scaler.fit_transform(Glucose)
Bp_norm = scaler.fit_transform(Bp)
SkinThikckness_norm = scaler.fit_transform(SkinThikckness)
Insulin_norm = scaler.fit_transform(Insulin)
DiaPedFUnc_norm = scaler.fit_transform(DiaPedFUnc)
```
[6] ✓ 0.0s Python

Cars – Dataset loaded from 'cars.csv'

```
cars = pd.read_csv('datasets/cars.csv')
Vol_data = cars[['Volume']]
Weight_data = cars[['Weight']]
```

- The first two subplots has Volume and Weight respectively scatter plotted with respect to index.
- The third subplot has a scatter plot with Volume data on the x-axis and Weight data on the y-axis.

```
plt.subplot(2,2,1)
plt.scatter(x,Vol_data)
plt.xlabel('Index')
plt.ylabel('Volume')

plt.subplot(2,2,2)
plt.scatter(x,Weight_data)
plt.xlabel('Index')
plt.ylabel('Weight')

plt.subplot(2,2,3)
plt.scatter(Vol_data,Weight_data)
plt.xlabel('Volume')
plt.ylabel('Weight')
plt.show()
[7]  ✓ 0.1s
```

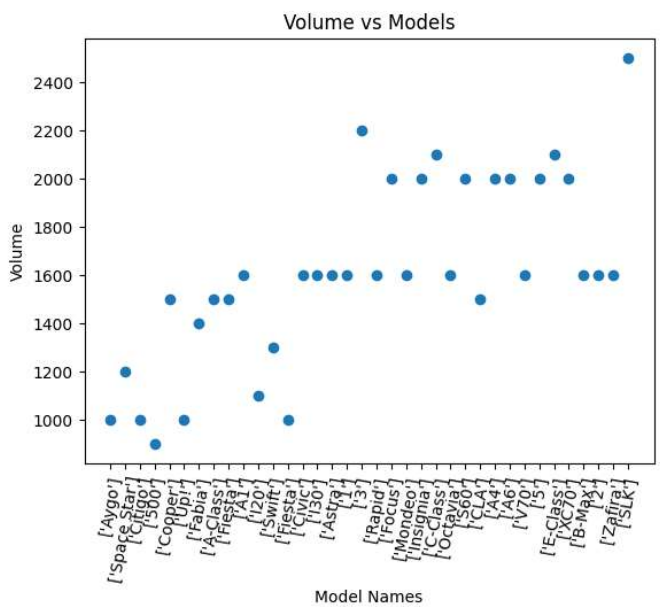


- The model names are extracted to a variable called *model.* It is then converted and reshaped to numpy array to be plotted in y axis.
- First, Scatter plot is plotted between index and Volumes. Then the label is set on the x-axis with a rotation of 80 degrees for better readability.

```
model = cars[['Model']].to_numpy()
model.reshape(1,36)
```

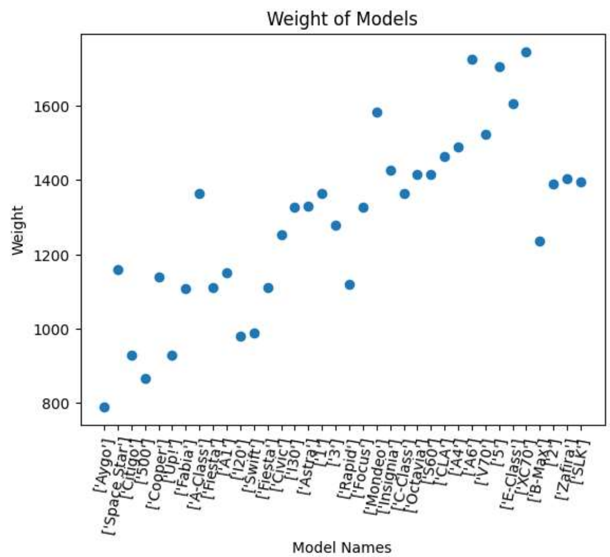
```
array([['Aygo', 'Space Star', 'Citigo', '500', 'Cooper', 'Up!', 'Fabia',
        'A-Class', 'Fiesta', 'A1', 'I20', 'Swift', 'Fiesta', 'Civic',
        'I30', 'Astra', '1', '3', 'Rapid', 'Focus', 'Mondeo', 'Insignia',
        'C-Class', 'Octavia', 'S60', 'CLA', 'A4', 'A6', 'V70', '5',
        'E-Class', 'XC70', 'B-Max', '2', 'Zafira', 'SLK']], dtype=object)
```

```
plt.scatter(x,Vol_data)
plt.xlabel('Model Names')
plt.ylabel('Volume')
plt.title('Volume vs Models')
plt.xticks(x,model,rotation=80)
plt.show()
```



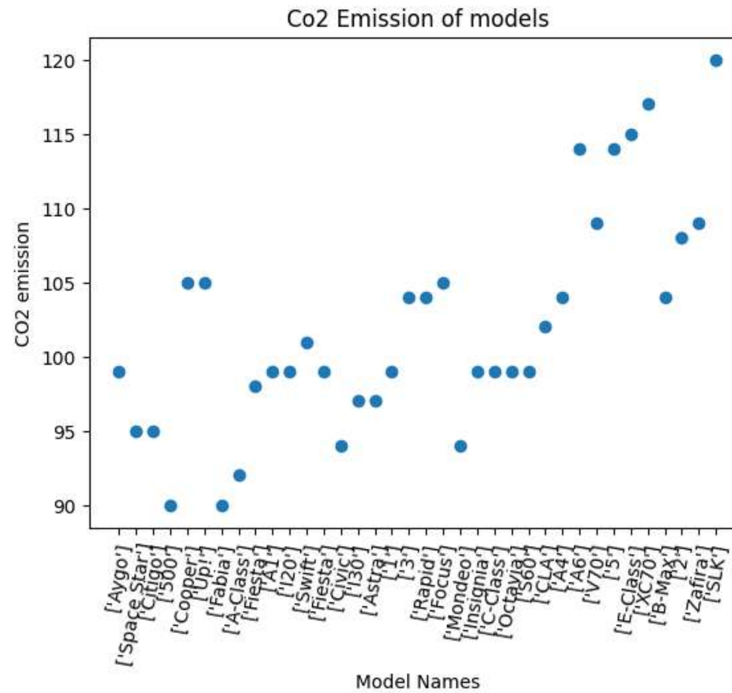
- Here also, a scatter plot has been plotted between Model Name and Weight.
- After that the label is set on the x-axis with a rotation of 80 degrees for better readability.

```python
plt.scatter(x,Weight_data)
plt.xlabel('Model Names')
plt.ylabel('Weight')
plt.title('Weight of Models')
plt.xticks(x,model,rotation=80)
plt.show()
```
✓ 0.1s



- The Co2 emission data are extracted to a variable called *co2*. It is then converted and reshaped to numpy array to be plotted in y axis.
- Then a scatter plot has been plotted between Model names and CO2 Emissions which says the level of emission for different models.

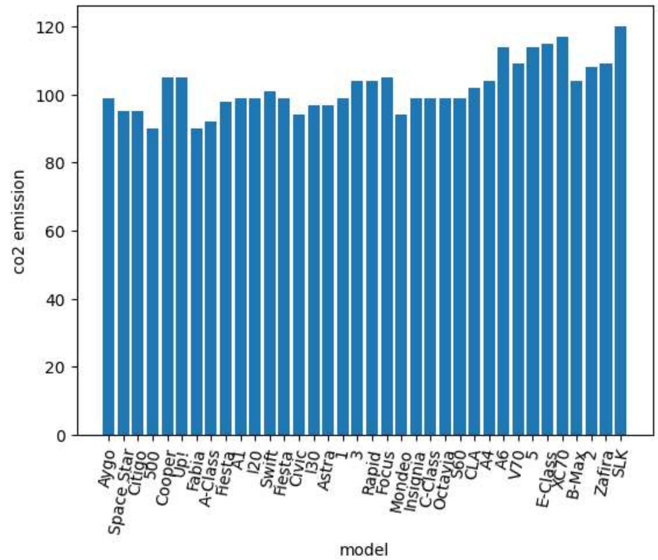
```python
co2 = cars[['CO2']].to_numpy()
co2.reshape(1,36)

plt.scatter(x,co2)
plt.xlabel('Model Names')
plt.ylabel('CO2 emission')
plt.xticks(x,model,rotation=80)
plt.title('Co2 Emission of models')
plt.show()
```
[19] ✓ 0.1s

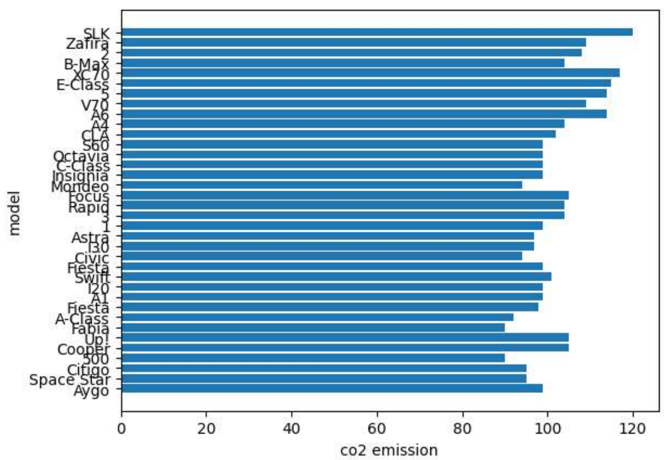


- Volume, Weights and Co2 data are extracted separately stored in variables. Then they are converted to 1D array by flatten().
- 1D arrays are often required for performing computations and plottting.

```
vol = Vol_data.to_numpy()
weight = Weight_data.to_numpy()

vol = vol.flatten()
weight = weight.flatten()
co2 = co2.flatten()
model = model.flatten()
```

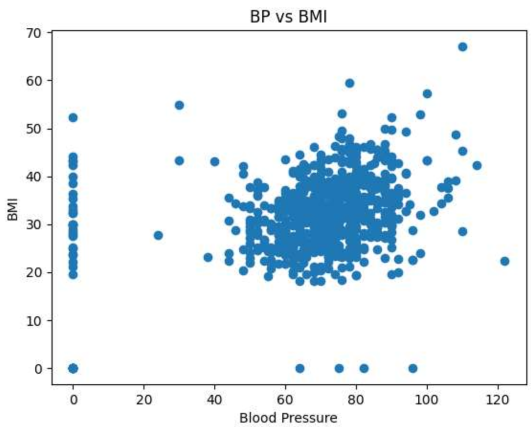
```
plt.bar(x,co2)

plt.xlabel('model')
plt.ylabel('co2 emission')
plt.xticks(x,model,rotation=80)
plt.show()

plt.barh(x,co2)
plt.xlabel('co2 emission')
plt.ylabel('model')
plt.yticks(x,model)
plt.show()
```

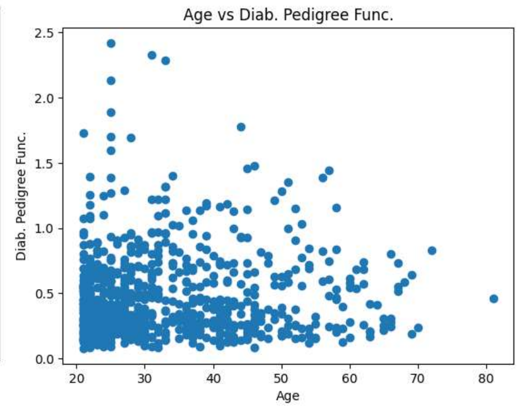




Scatter plotted between Blood Pressure – BMI & Age – Diabetes Pedigree Function to see the relationships between them.

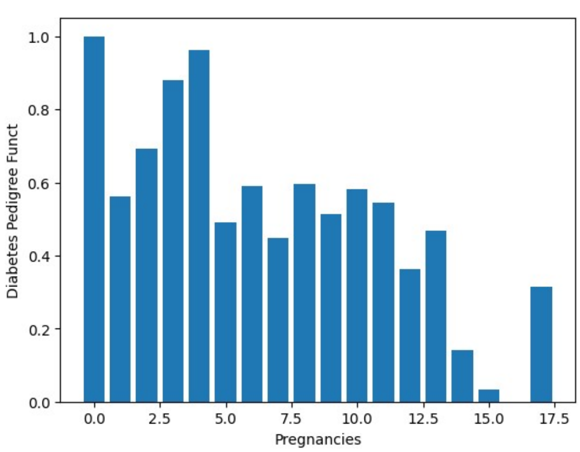
```
plt.scatter(Bp,BMI)
plt.xlabel('Blood Pressure'); plt.ylabel('BMI')
plt.title('BP vs BMI')
plt.show()

plt.scatter(Age,DiaPedFUnc)
plt.title('Age vs Diab. Pedigree Func.')
plt.xlabel('Age'); plt.ylabel('Diab. Pedigree Func.')
plt.show()
```
[7] ✓ 0.2s

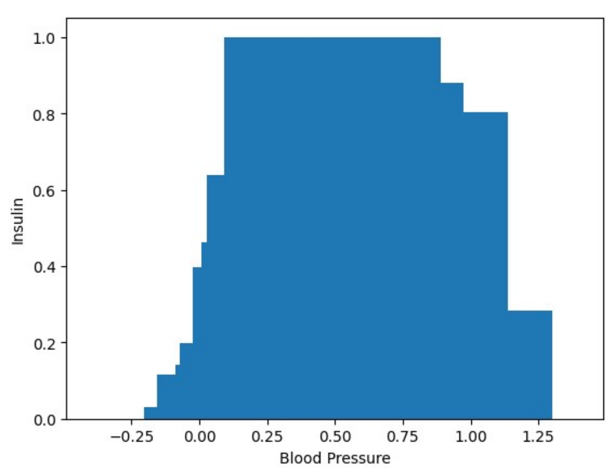


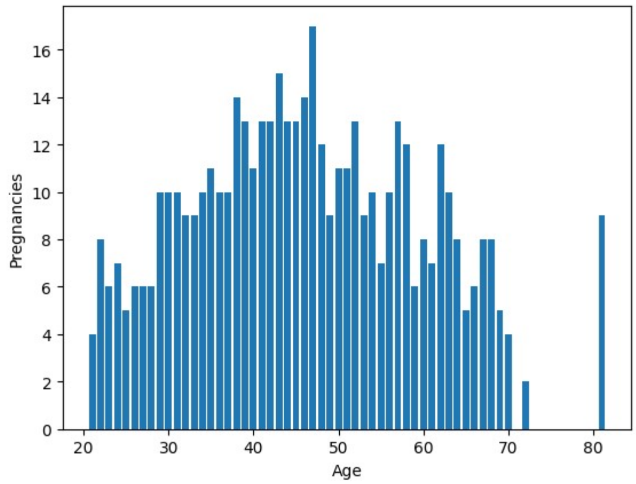


- Here, Pregnancies Count of patients, Diabetes Pedigree Function and Age are converted to numpy arrays.
- Checked the relationship between Pregnancies and Diabetes Pedigree Function, Blood pressure and Insulin using bar graphs.

```
P = Pregnancies.to_numpy().flatten()
DPF = DiaPedFUnc_norm.flatten()
A = Age.to_numpy().flatten()

plt.bar(P,DPF)
plt.xlabel('Pregnancies'); plt.ylabel('Diabetes Pedigree Funct')
plt.show()

# B = Bp.to_numpy().flatten()
# I = Insulin.to_numpy().flatten()
B = Bp_norm.flatten()
I = Insulin_norm.flatten()
plt.bar(B,I)
plt.show()

plt.bar(A,P)
plt.xlabel('Age'); plt.ylabel('Pregnancies')
plt.show()
```





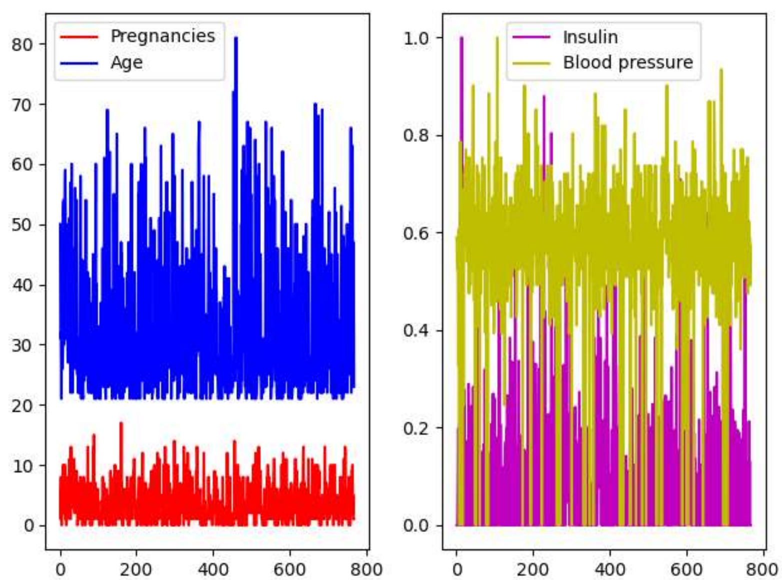


#OVERLAY PLOTS

Created two line plots within a single figure, comparing different sets of data.

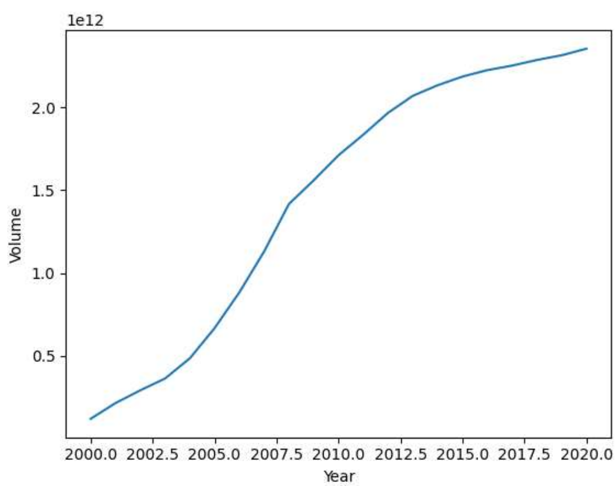
This code snippet results in a figure with two subplots side by side, showing the trends of 'Pregnancies' and 'Age' in the first subplot, and 'Insulin' and 'Blood Pressure' in the second subplot.

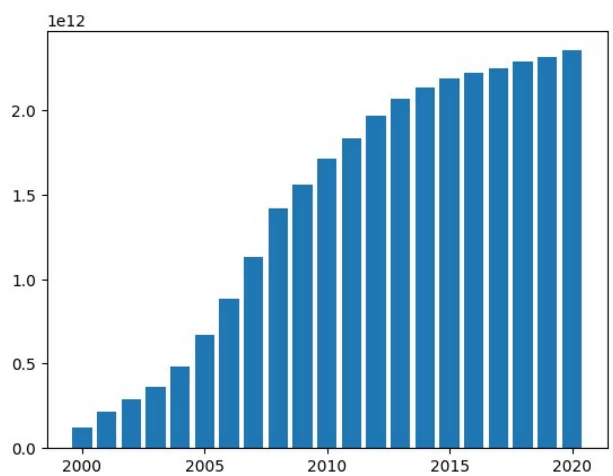
```
x = np.arange(1,769)

plt.subplot(1,2,1)
plt.plot(x,P, color='r', label='Pregnancies')
plt.plot(x,A, color='b', label='Age')
plt.legend()
#lt.show()
plt.subplot(1,2,2)
plt.plot(x,I, color='m', label='Insulin')
plt.plot(x,B, color='y', label='Blood pressure')
plt.legend()
plt.tight_layout()
plt.show()
```



- Apple Dataset is loaded and stored in variable ‘data’.
- Initialize an empty list to store the annual trading volumes. Initialize a variable sum to accumulate the trading volume.
- Iterate over each year from 2000 to 2020. For each year, iterate over the Date column in the dataset.
- If the year part of the date matches the current year, add the corresponding Volume to sum. Append the accumulated volume for the year to the volume list.
- Visualized the volume of models for every year by a line plot and bar graph.

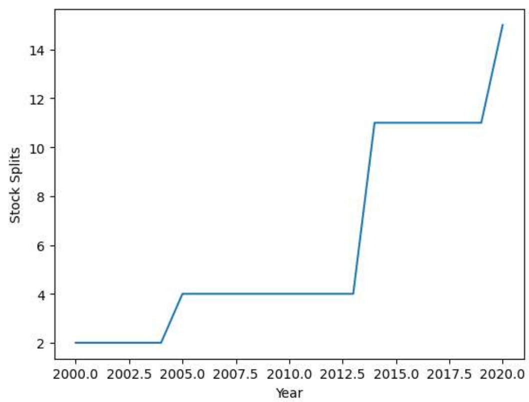
```
data = pd.read_csv('datasets/Apple Stock.csv')

year = np.arange(2000,2021)

volume = []
sum = 0
x = 2000
for x in range(2000,2021):
    for i,j in enumerate(data.Date):
        if int(j.split('/')[2]) == x:
            sum += data.Volume[i]
    volume.append(sum)

plt.plot(year,volume)
plt.xlabel('Year'); plt.ylabel('Volume')
plt.show()

plt.bar(year,volume)
plt.show()
```
[13]

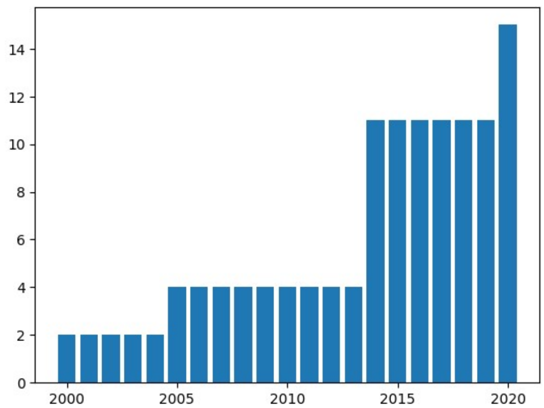




Visualized the volume of models for every year by a line plot and bar graph.

```python
stock = data['Stock Splits']
st = []
sum = 0
for x in range(2000,2021):
    for i,j in enumerate(data.Date):
        if int(j.split('/')[2]) == x:
            sum += stock[i]
    st.append(sum)

plt.plot(year,st)
plt.xlabel('Year'); plt.ylabel('Stock Splits')
plt.show()

plt.bar(year,st)
plt.show()
```





RESULT :

- The results showcase successful execution of various data preprocessing and plotting techniques, providing a comprehensive understanding of the dataset's structure, relationships, and distributions.
- This thorough analysis is foundational for building effective data models and deriving meaningful insights from the data.

LEARNING OUTCOMES :

- Gained proficiency in loading datasets into pandas DataFrames and extracting specific columns for focused analysis.
- Acquired the ability to create various types of plots using matplotlib, including scatter plots, bar graphs, overlay plots.
- Enhanced the ability to perform a comprehensive data analysis by combining multiple preprocessing and visualization techniques to gain deeper insights into the dataset.
- Understood how to visualize relationships between different features, which is essential for exploratory data analysis and identifying patterns or trends in the data.

Department of Mechanical Engineering

21ARE301 – Introduction to Data Science

B. Tech – 5th Semester - ARE

Name of the Student	K Vinoth Kumar
Registration Number	CH.EN.U4ARE22011
Branch	Automation and Robotics Engineering
Submission Date	22 – 07 -2024

AIM :

The aim of this code is to perform comprehensive data preprocessing and exploratory data analysis on a diabetes dataset.

The Code used in this assignment can be found here.

CODE :

Import Libraries for performing numerical operations, data manipulation & analysis, for creating plots.

```python
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from sklearn.impute import SimpleImputer
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

✓ 0.0s Python

Data – Dataset loaded from 'diabetes.csv'

```python
data = pd.read_csv('datasets/diabetes.csv')

Glucose = data[['Glucose']]
Bp = data[['BloodPressure']]
Insulin = data[['Insulin']]
DiaPedFUnc = data[['DiabetesPedigreeFunction']]
SkinThikckness = data[['SkinThickness']]
BMI = data[['BMI']]
Age = data[['Age']]
Outcome = data[['Outcome']]
Pregnancies = data[['Pregnancies']]

scaler = MinMaxScaler()

Glucose_norm = scaler.fit_transform(Glucose)
Bp_norm = scaler.fit_transform(Bp)
SkinThikckness_norm = scaler.fit_transform(SkinThikckness)
Insulin_norm = scaler.fit_transform(Insulin)
DiaPedFUnc_norm = scaler.fit_transform(DiaPedFUnc)
```

[6] ✓ 0.0s Python

- Converted the Outcome data to a flattened NumPy array and categorized the outcomes into 'Yes' (positive) and 'No' (negative).
- Created and displayed a pie chart for the diagnosed results.

- Categorize the number of pregnancies into three categories: less than 5, 5-9, and 10 or more
- Created and displayed a pie chart for the number of pregnancies:

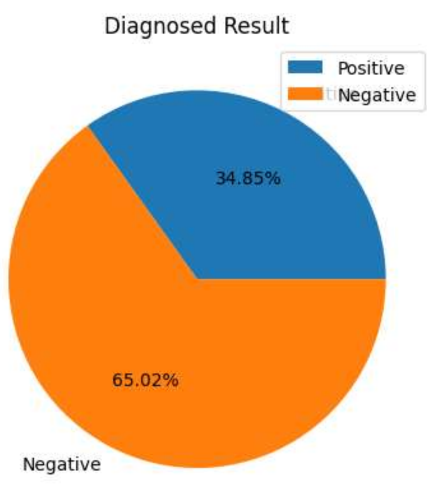
- A function is defined to calculate the percentage for the pie chart labels:
- Created a pie chart with the sizes and labels specified, and used the percentage function for the label percentages.

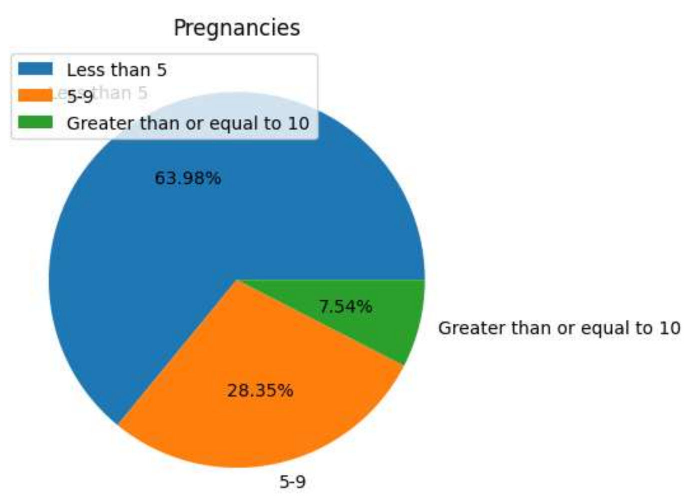
PIE CHART

```python
Out = Outcome.to_numpy().flatten()
Yes  =  [ x for x in Out if x == 1]
No = [ x for x in Out if x == 0 ]
size = [ len(Yes),len(No)]
result = ['Positive','Negative']

def percentage(pcts, allvals):
    abs = float(pcts / 100.*np.sum(allvals))
    abs /= 7.69
    return "{:.2f}%".format(abs)

plt.pie(size,
        labels=result,
        autopct=lambda pct: percentage(pct, size))
plt.title('Diagnosed Result')
plt.legend()
plt.show()

preg_pie = []
count  = [c for c in P  if  c < 5]; preg_pie.append(len(count))
count  = [c for c in P  if  5<=  c < 10]; preg_pie.append(len(count))
count  = [c for c in P if  10 <= c]; preg_pie.append(len(count))
label_per = ['Less than 5', '5-9','Greater than or equal to 10']
plt.pie(preg_pie,
        labels=label_per,
        autopct=lambda pct: percentage(pct, preg_pie))
plt.title('Pregnancies')
plt.legend()
plt.show()
```

[17] ✓ 0.1s

4



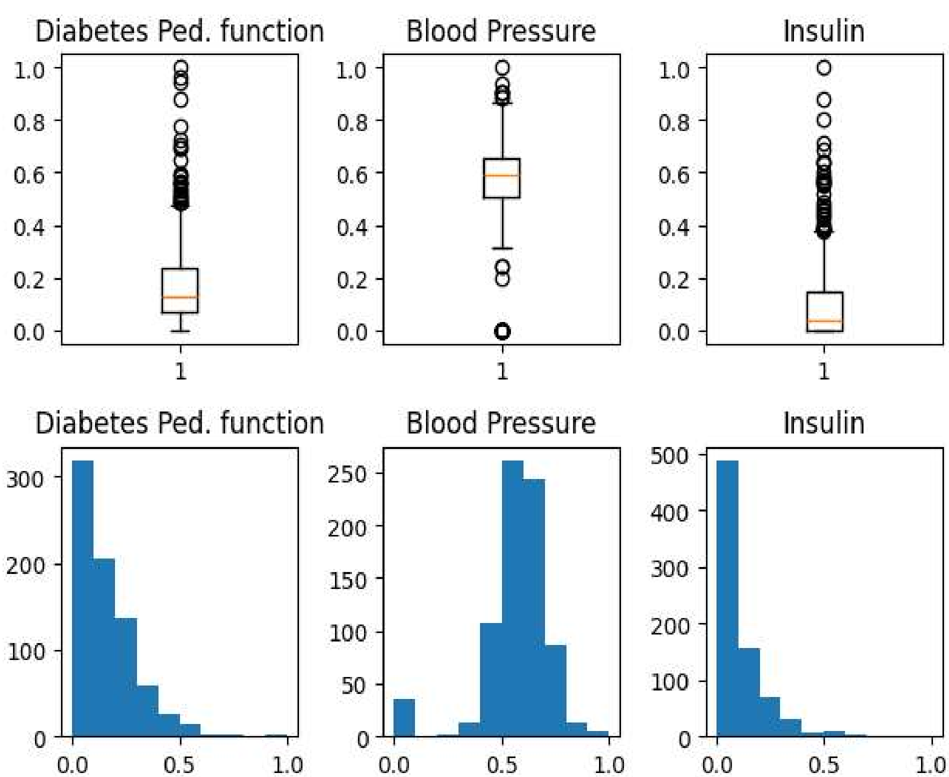


- Plotted Boxplot for Diabetes Pedigree Function, Blood Pressure and Insulin.
- Plotted Insulin for Diabetes Pedigree Function, Blood Pressure and Insulin.

HIST & BOXPLOT

```python
plt.subplot(2,3,1)
plt.boxplot(DiaPedFUnc_norm)
plt.title('Diabetes Ped. function')
plt.subplot(2,3,4)
plt.title('Diabetes Ped. function')
plt.hist(DiaPedFUnc_norm)

plt.subplot(2,3,2)
plt.boxplot(Bp_norm)
plt.title('Blood Pressure')
plt.subplot(2,3,5)
plt.hist(Bp_norm)
plt.title('Blood Pressure')

plt.subplot(2,3,3)
plt.boxplot(Insulin_norm)
plt.title('Insulin')
plt.subplot(2,3,6)
plt.hist(Insulin_norm)
plt.title('Insulin')

plt.tight_layout()
plt.show()
```

[18] ✓ 0.4s



RESULT :

- The results showcase successful execution of various data preprocessing and visualization techniques, providing a comprehensive understanding of the dataset's structure, relationships, and distributions.
- This thorough analysis is foundational for building effective data models and deriving meaningful insights from the data.

LEARNING OUTCOMES :

- Gained proficiency in loading datasets into pandas DataFrames and extracting specific columns for focused analysis.
- Acquired the ability to create various types of plots using matplotlib, including pie charts, histograms and box plots.
- Enhanced the ability to perform a comprehensive data analysis by combining multiple preprocessing and visualization techniques to gain deeper insights into the dataset.
- Understood how to visualize relationships between different features, which is essential for exploratory data analysis and identifying patterns or trends in the data.