MEASURE ENERGY CONSUMPTION

Date	18/10/2023
Team ID	Proj_212174_Team_1
Project Name	Measure Energy consumption
Maximum Mark	

Data Virtualization:

Data Visualization is a technique of presenting data graphically or in a pictorial format which helps to understand large quantities of data very easily. This allows decision-makers to make better decisions and also allows identifying new trends, patterns in a more efficient way.

Matplotlib and Seaborn:

Matplotlib and Seaborn are python libraries that are used for data visualization. They have inbuilt modules for plotting different graphs. While Matplotlib is used to embed graphs into applications, Seaborn is primarily used for statistical graphs.

But when should we use either of the two? Let's understand this with the help of a comparative analysis. The table below provides comparison between Python's two well-known visualization packages Matplotlib and Seaborn.

Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. In this tutorial, we will be discussing four such libraries.

- Matplotlib
- Seaborn
- Bokeh
- Plotly

We will discuss these libraries one by one and will plot some most commonly used graphs.

CODE AND OUTPUT:

```
%matplotlib inline
```

#we are going to import our data analysis library, pandas. Since we are going to write pandas all the time, we shorten it to pd for brevity:

```
import pandas as pd
```

#We also import the pyplot library, which will be very useful to visualise our data with charts and plots.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

#Seaborn is a library for making statistical graphis in phyton.it

Builds on top of matplotlib and integrates closely with pandas

Data stuctures.seaborn helps you explore and understand your data.

```
#plt.style.use('ggplot')
```

For reading the data set in csv file we use the syntax pd.read_csv()for treading the dataset.

```
df = pd.read_csv("PJME_hourly.csv")
```

Histogram ploting:

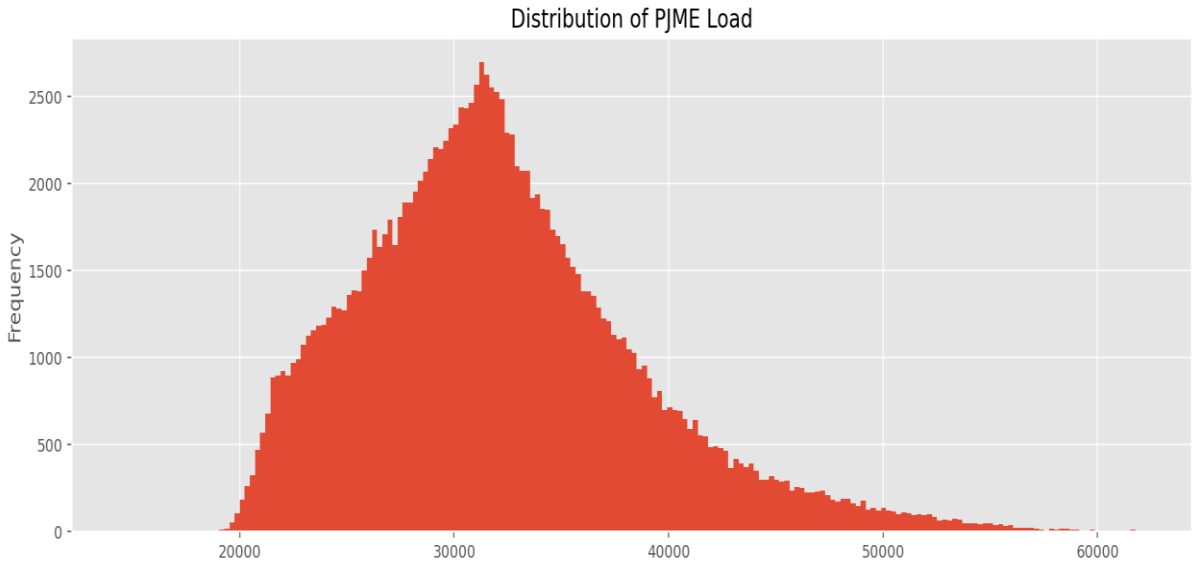
```
f1=data['PJME_MW'].plot.hist(figsize=(15, 5), bins=200,
title='Distribution of PJME Load')
```

#A histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval.

The following represents the distribution od PJME Load and the frequency

We use the histogram ploting for this data set.

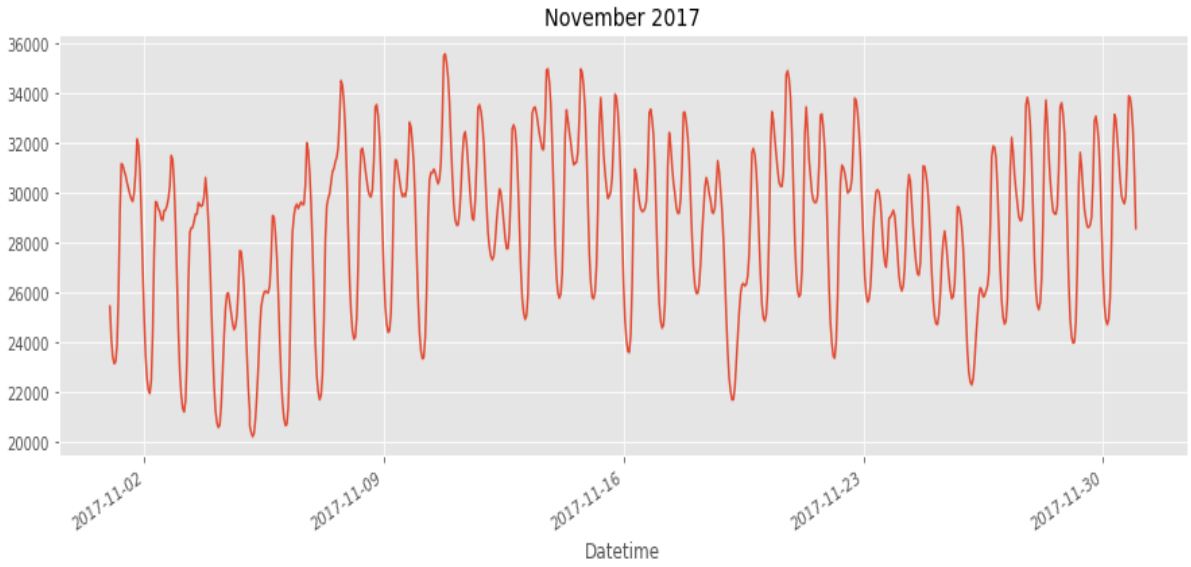


Summer Demand & Winter Demand:

The dips mid-day in the winter months. Conversely in summer months the daily load is more bell shaped. This is due to high mid-day energy consumtion by air conditioning. In winter months people tend to use less energy mid-day.

```
#data1=df['PJME'].loc[(df['PJME'].index >= '2017-11-
01') &
(df['PJME'].index < '2017-12-01')]
\.plot(figsize=(15, 5), title = 'November 2017')
```

```
This plot shows that the energy consumption between
the summer period and winter period.
The energy consumption between this two seasons are
Different so we ploting this difference for the
Data visulaisation.
It is easy for understanding between this two
seasons.
```

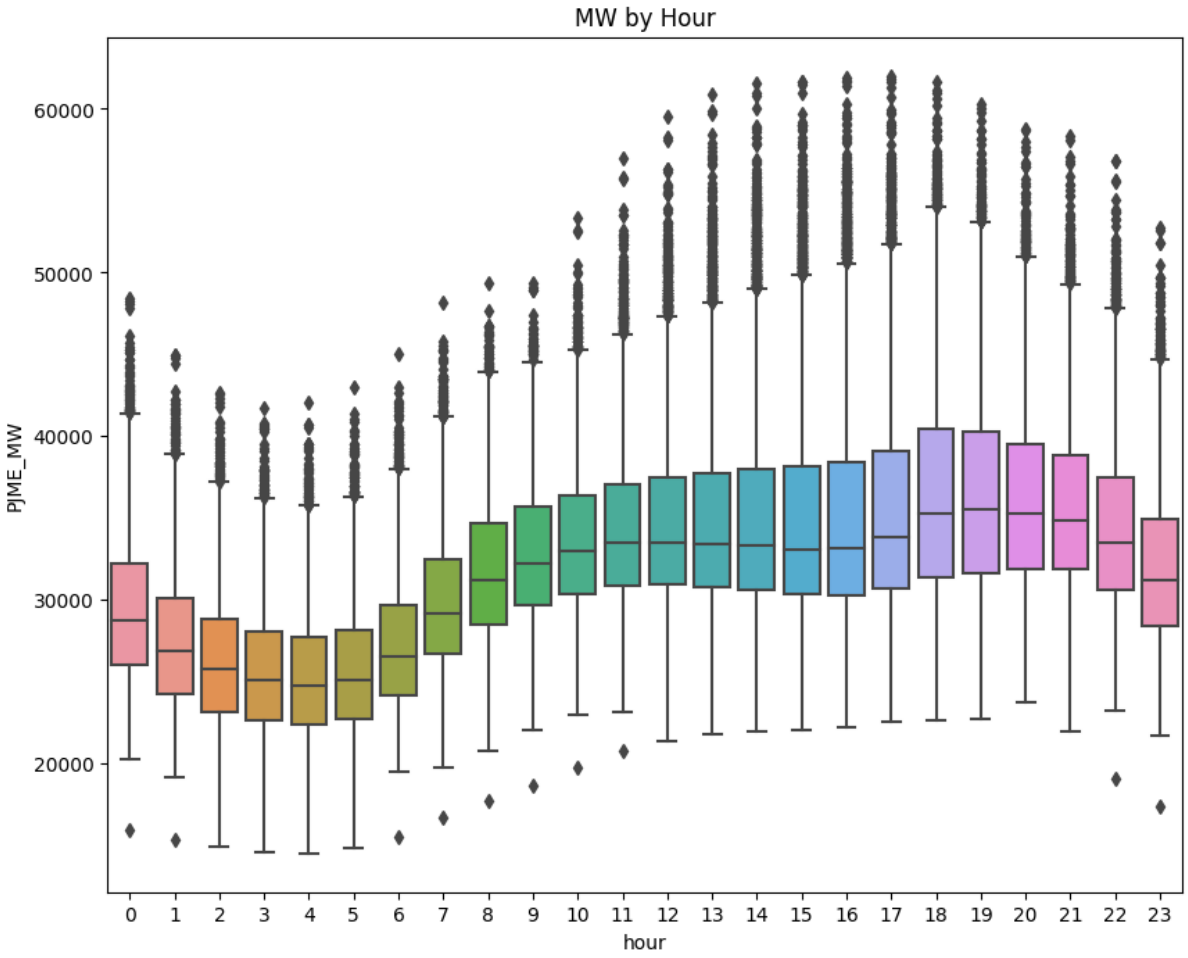


Box Ploting:

A box plot is also known as Whisker plot is created to display the summary of data values having properties like minimum,first quartile,median,third quartile and maximum.

It displays the five number summary of a set od data.

B ox plot is used for show distribution of numeric data values,especially when you want to compare them between multiple groups.



#In the next snippet we define a function, timeparser, which will convert our columns "date" and "time" into a Python datetime object. It basically tells Python in which format our "date" and "time" columns are.

```
plt.style.use('fivethirtyeight')
timeparser = lambda x: pd.datetime.strptime(x, '%Y-%m-%d %H:%M')
```

#We are ready to read the data. We will merge the columns "date" and "time" into a single column "datetime", using the parser function timeparser that we defined above.

```
df = pd.read_csv('energy-consumption.csv', parse_dates = {'datetime': ['date', 'time']}, date_parser = timeparser)
```

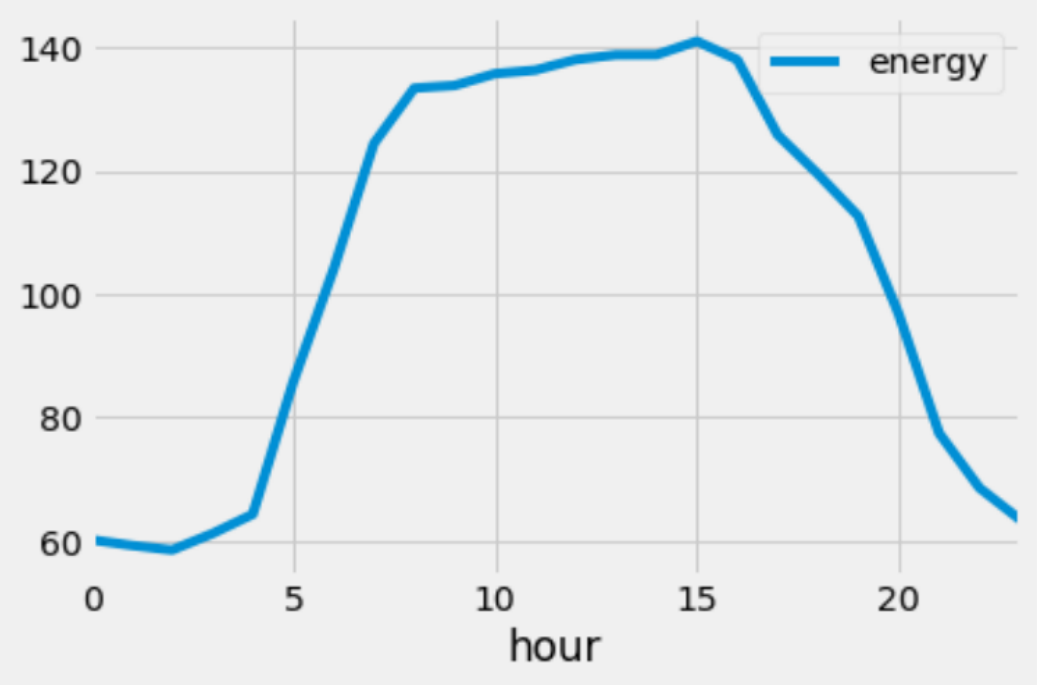
#here first thing we might want to know is how does the total energy profile looks like in an average day. This means that we want to consider both residential and commercial buildings of all types, and we do not discrimnate between weekdays and weekends. This graph is simple to produce, but

perhaps not so informative: in reality there is no such thing as an average building or an average day.

Since we would like to have a plot with the hour of the day on the x-axis and the energy consumption on the y-axis, we first add a column to our dataset, which extracts the hour of the day from the "datetime" columns:

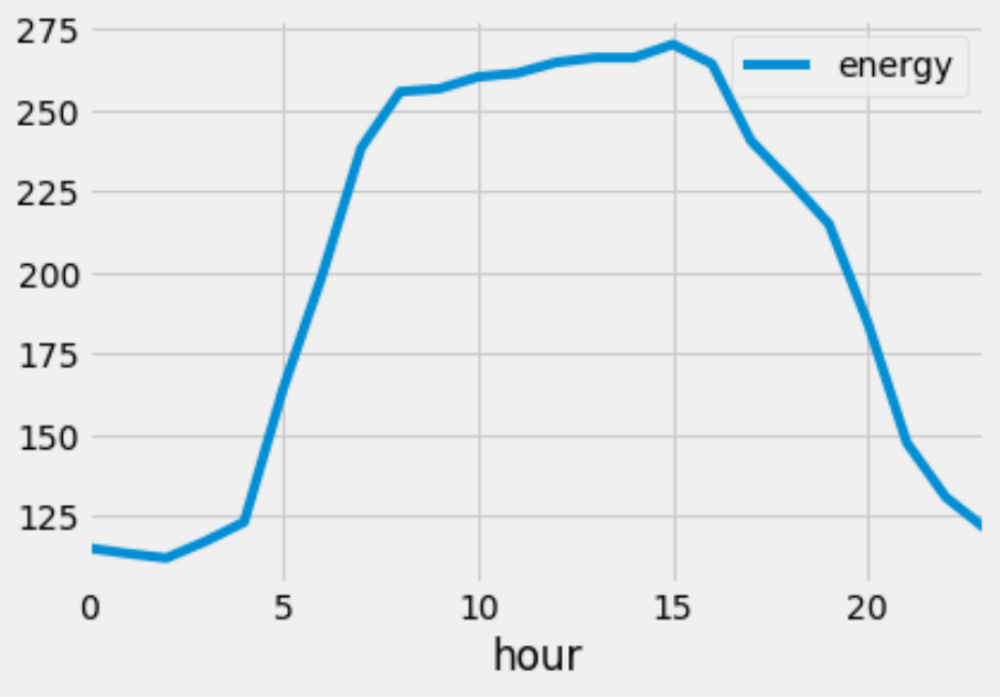
```
df['hour'] = df['datetime'].apply(lambda x: x.hour)

df.groupby('hour').mean().plot()
```

OUTPUT:




```
#df[df.category == 'commercial'].groupby('hour').mean().plot()
```

OUTPUT:



FIXING THE DATA:

#Fixing the data is going to be easy. With a couple of lines of code we can account for the DST shift. We will add a column adj_energy which will give the adjusted energy consumption, taking into account the time shift.

df['shift_energy'] = df['energy'].shift(1)

df['adj_energy'] = df.apply(lambda row: row['shift_energy'] if row['dst'] else row['energy'], axis = 1)

#Let's now print the last two plots again, to verify that this time there is no shift:

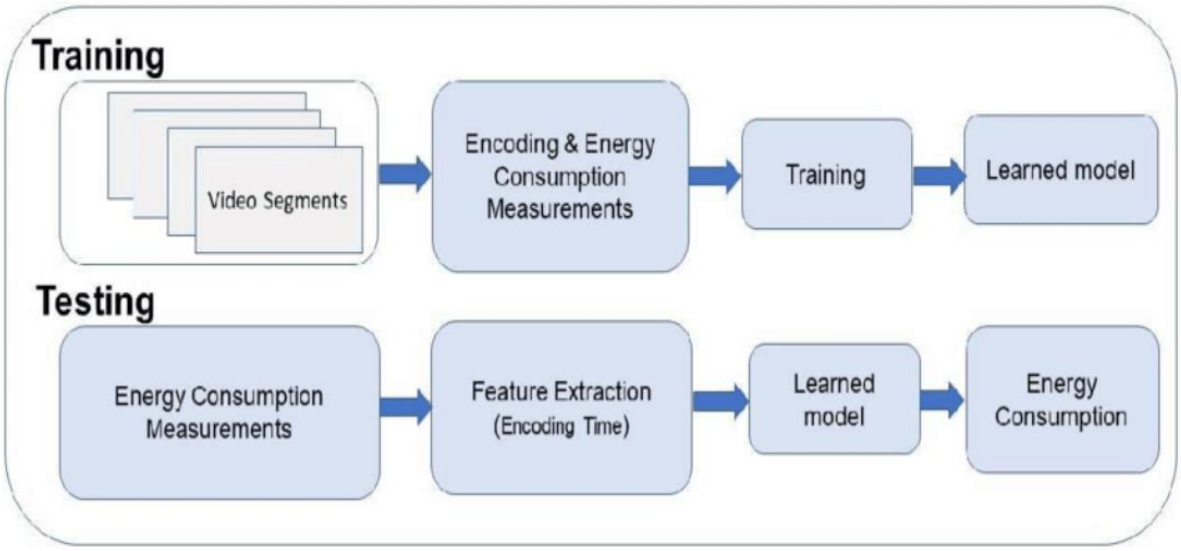
fig, ax = plt.subplots()

for dst in [True, False]:

ax = df[(df.category == 'residential') & (df.dst == dst)].groupby('hour').mean().plot(ax = ax, y = 'adj_energy', label = str(dst))

OUTPUT:

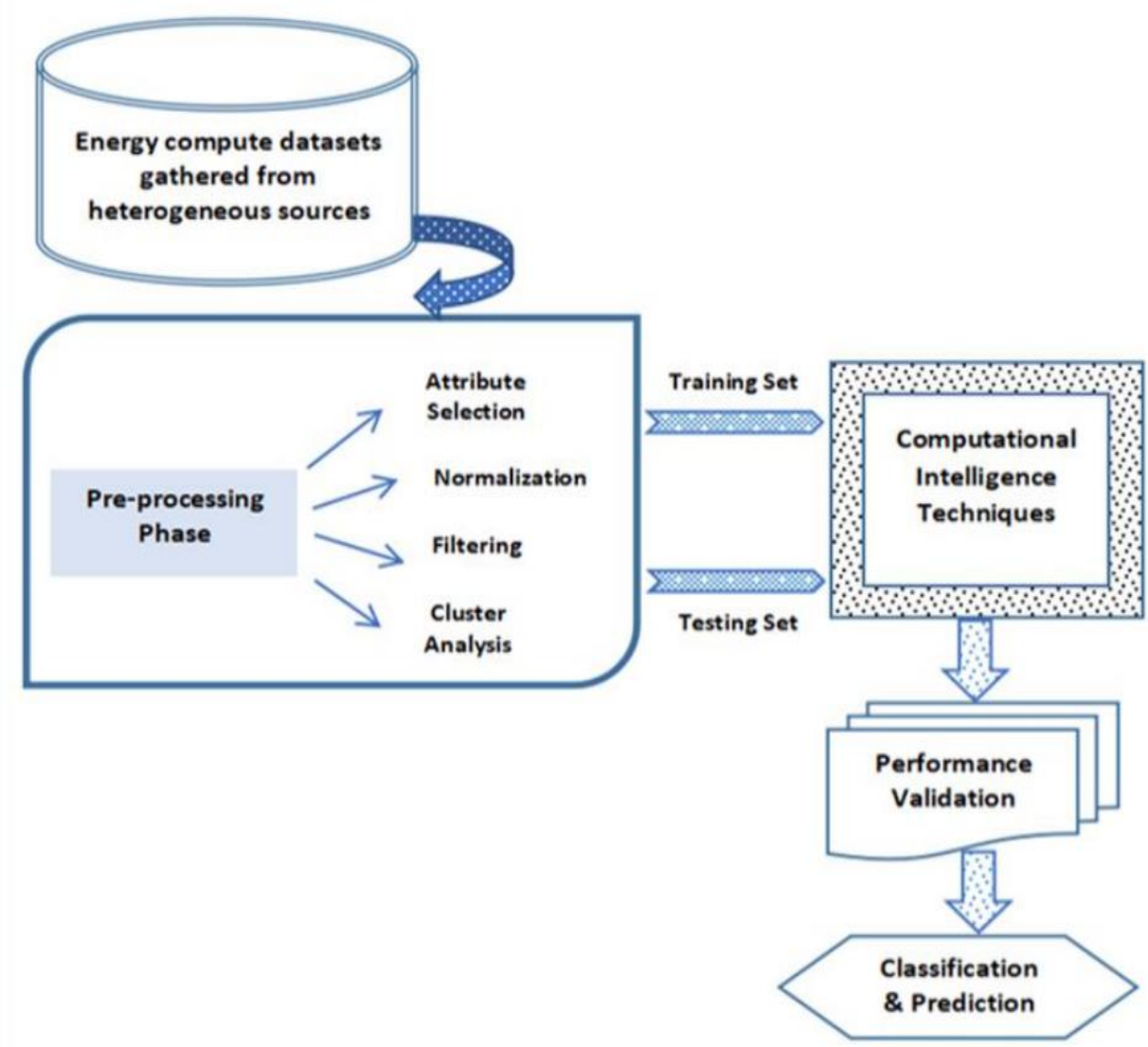
ENERGY CONSUMPTION ARCHITECTURE DIAGRAM:



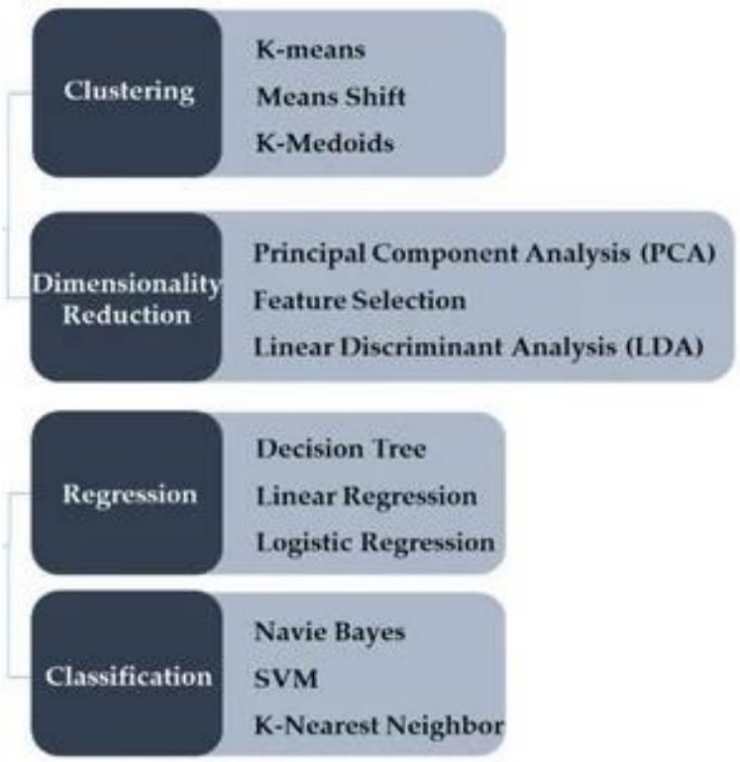
There are 3 modules,

1)DATA PREPROCESSING:

Data preprocessing in Python refers to the process of cleaning, transforming, and organizing raw data into a format suitable for analysis or machine learning. Python provides various libraries and tools to perform data preprocessing tasks efficiently.



2)Algorithms used:



3)Data visualization: