**MEASURE ENERGY CONSUMPTION**

|  |  |
| --- | --- |
| Date | 17/10/2023 |
| Team Leader | Preethi Sandal |
| Team id | 212174 |
| Project Name | Measure Energy consumption |
| Maximum Marks |  |

**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](https://www.simplilearn.com/top-python-libraries-for-data-science-article) 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:**  **[DATA VISUALIZATION]**

%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

#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

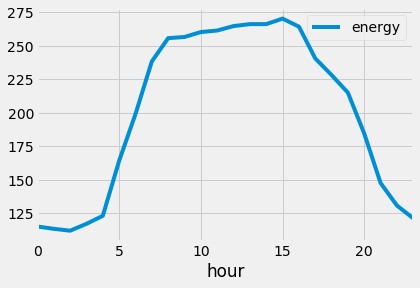
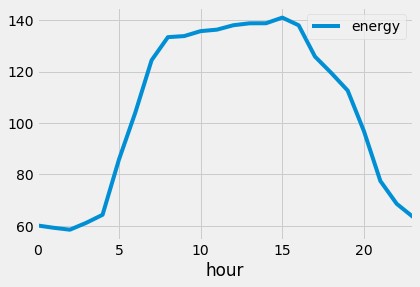
()

OUTPUT:

<

matplotlib.axes.\_subplots.AxesSubplot at 0x7f15b4e

8e978>



df[df.category == 'residential'].groupby('hour').mean().plot()

**OUTPUT:**

<

matplotlib.axes.\_subplots.AxesSubplot at 0x7f15b2ca

9240>

The situation is already more interesting here. The first thing we can notice is that

commercial buildings consume much more energy than residential ones. At the lowest,

they consume around 110kW/h and have peaks of more than 270kW/h. Residential

buildi

ngs, on the other hand, have a consumption in the range from 0.6 to 2.0kW/h.

This explains why the first "overall" graph and the second "commercial

-

only" have a

similar shape (commercial consumption dominates), but different values on the y

-

axis

(

due to av

eraging with the very low residential values).

Also notice that the shape of the consumption curve is totally different for residential

buildings: they have a first peak around 07:00, when people wake up, and then a larger

peak at 18:00

-

21:00

, when most p

eople come back home from work.

Another thing we can do is to plot, instead of the

*average*

energy consumption,

the

*total*

one, i.e. the sum of all consumptions. Since there is a big difference in

magnitude between the consumptions of commercial and resident

ial buildings, we expect

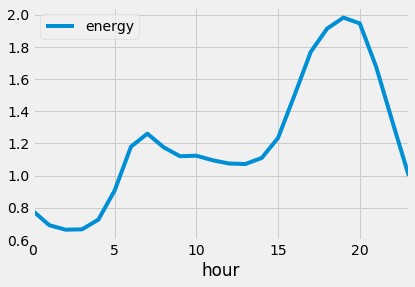
this plot to ave a shape similar to the residential one and

-

in fact

-

this is the case:



df.groupby('hour').sum().plot()

**OUTPUT:**

<

matplotlib.axes.\_subplots.AxesSubplot at 0x7f15b2c

32a20>

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:

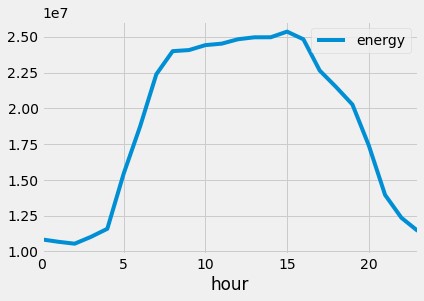
fig, ax = plt.subplots()

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:**

fig, ax = plt.subplots()

for dst in [True, False]:

ax = df[(df.category == 'commercial') & (df.dst ==

dst)].groupby('hour').mean().plot(ax = ax, y = 'adj\_energy', label = str(dst))

plt.show()

OUTPUT:



HOW

CAN PYTHON BE USED FOR DATA VISUALIZATION?

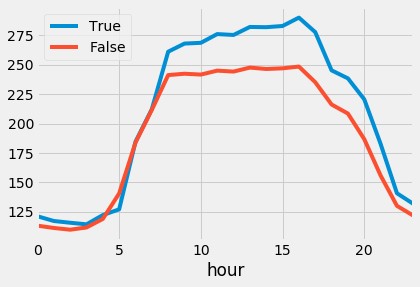
The process of finding trends and correlations in our data by representing it

pictorially is called Data Visualization. To perform data visualization in python, we

can

use various python data visualization modules such as Matplotlib, Seaborn, Plotly,

etc



**ENERGY CONSUMPTION ARCHITECTURE DIAGRAM:**

There are 3 modules,

**1)**

**DATA**

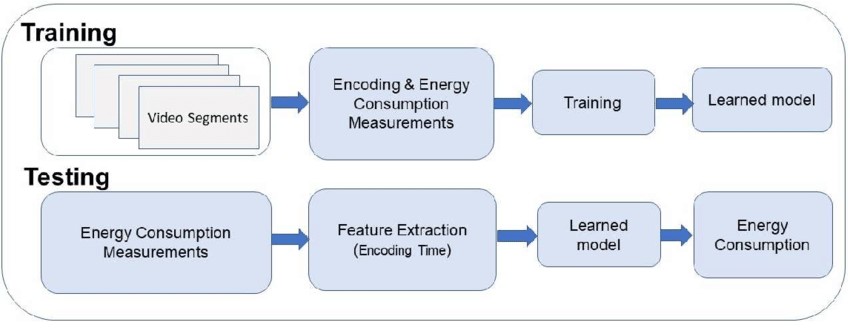
**PREPROCESSING:**

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

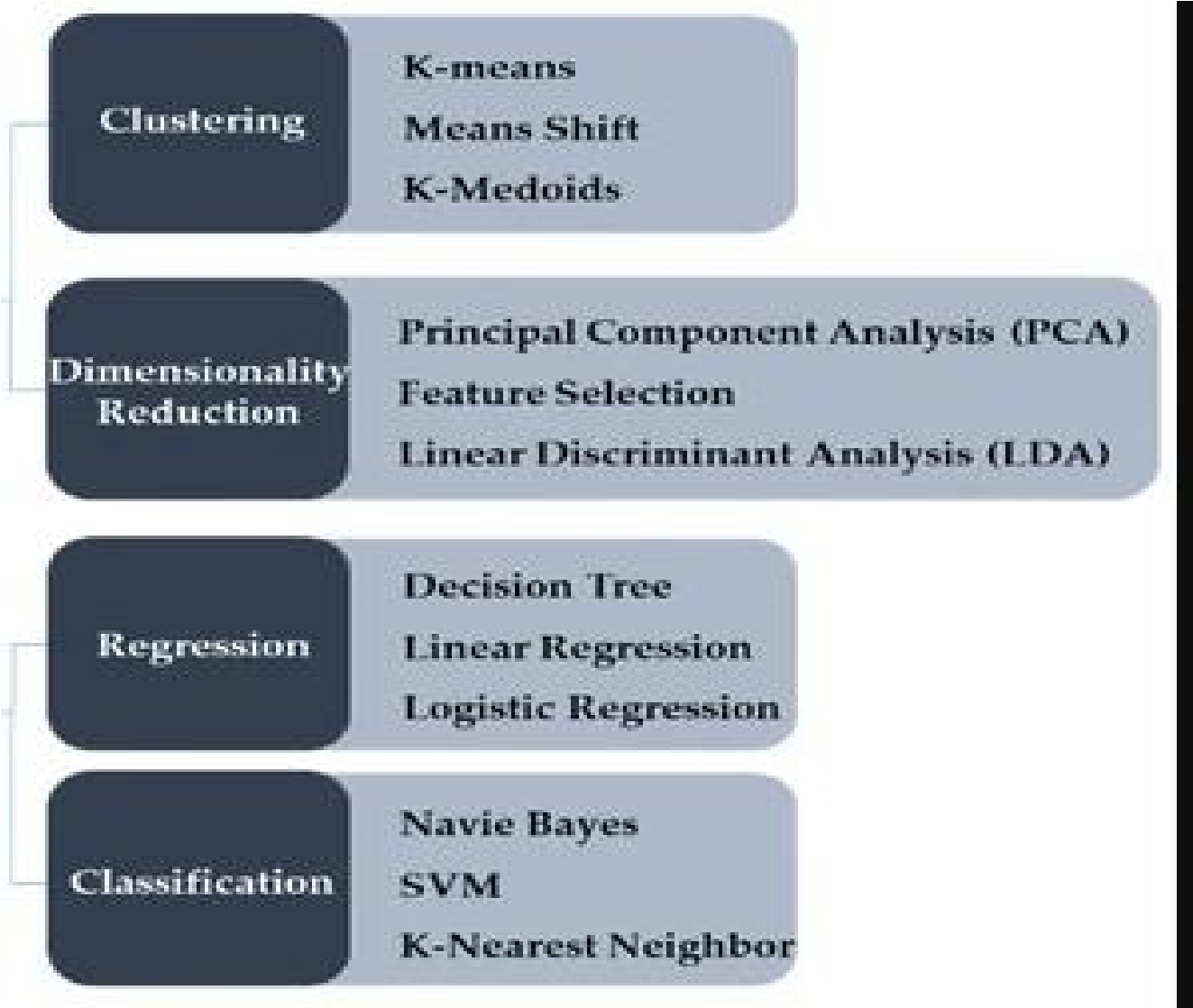
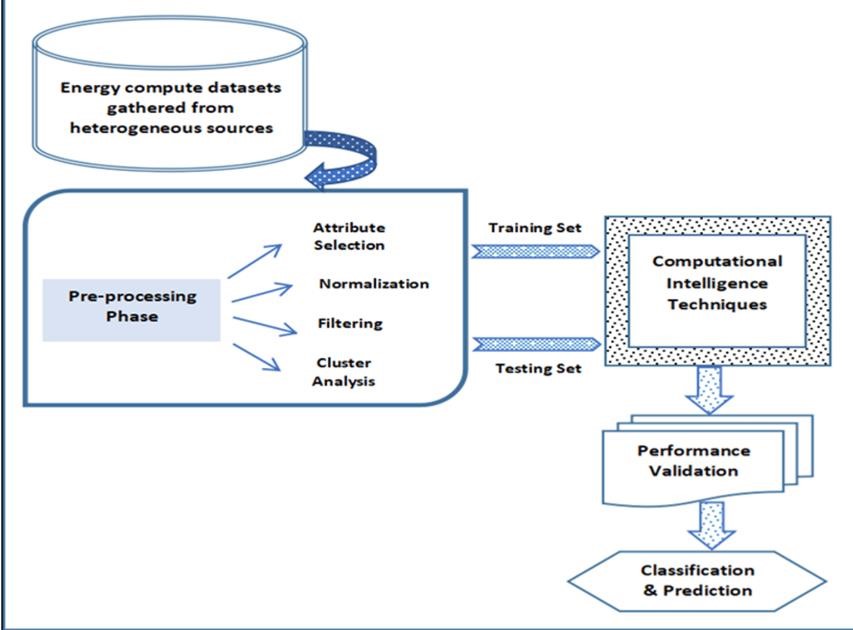
tasks efficiently.



**2)**

**ALGORITHMS USED**

**:**



**3)**

**DATA VISUALIZATION OUTPUT**

**:**

