MLOps * DevOps (CSE 5121)

* VS Code
* Dockers → Containerization.
* Git & Git Hub → version Control system
* kubernetes.

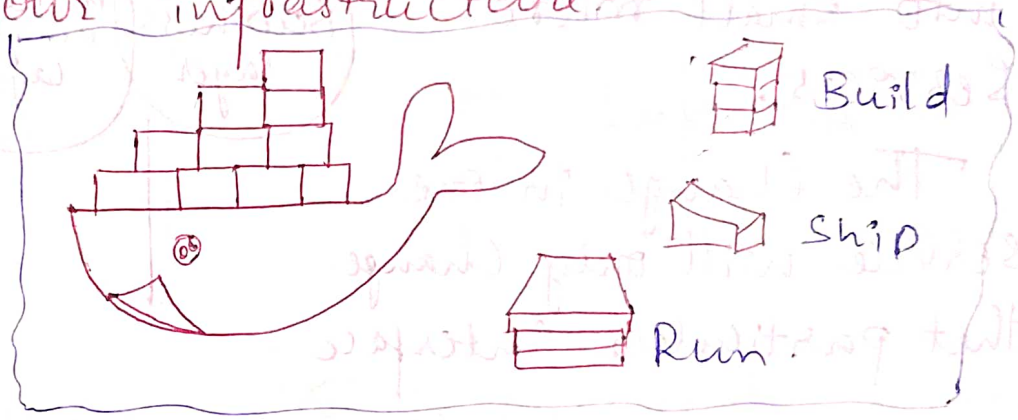
DevOps tools

* SCM → Software Configuration Manage-ment
* CI/CD → Continous Integration and Continous Delivery
* Container
* IDE/Editor
* Communication
* API Manager Store
* Software Management tools
* Configuration Management and Continous Delivery
* Cloud Technologies
* Logging
* Service Management
* Project Management

Docker :-

→ open platform

⇒ Enables you to separate your application from your infrastructure.

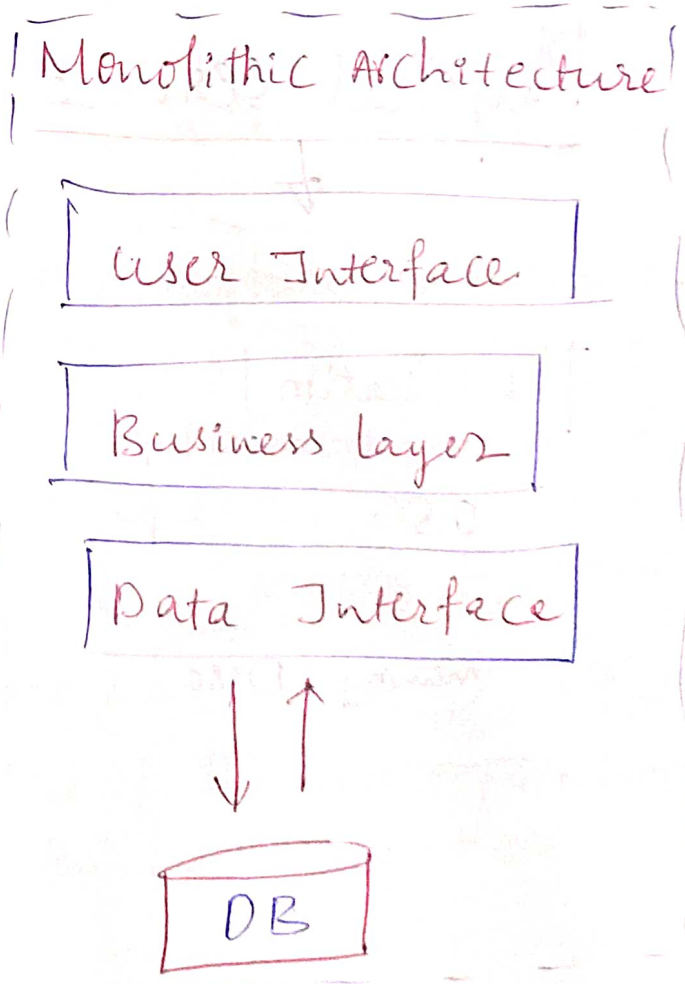


Monolithic :-

→ Self-Contained

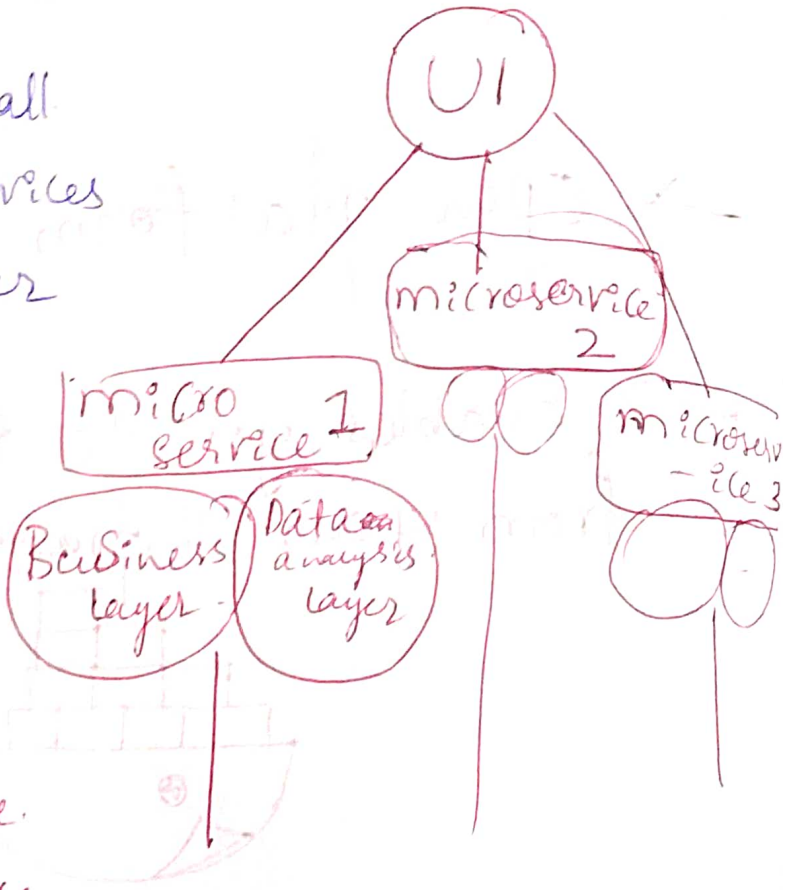
→ Components of the program are interconnected & interdependent

(*) Change at one layer will Changes the others layers



:- Microservices :-

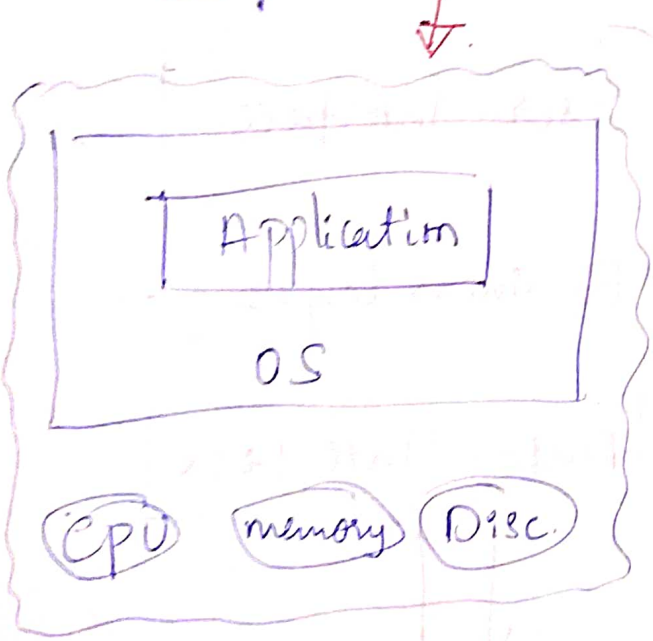
As a collection of small loosely coupled services that operate together

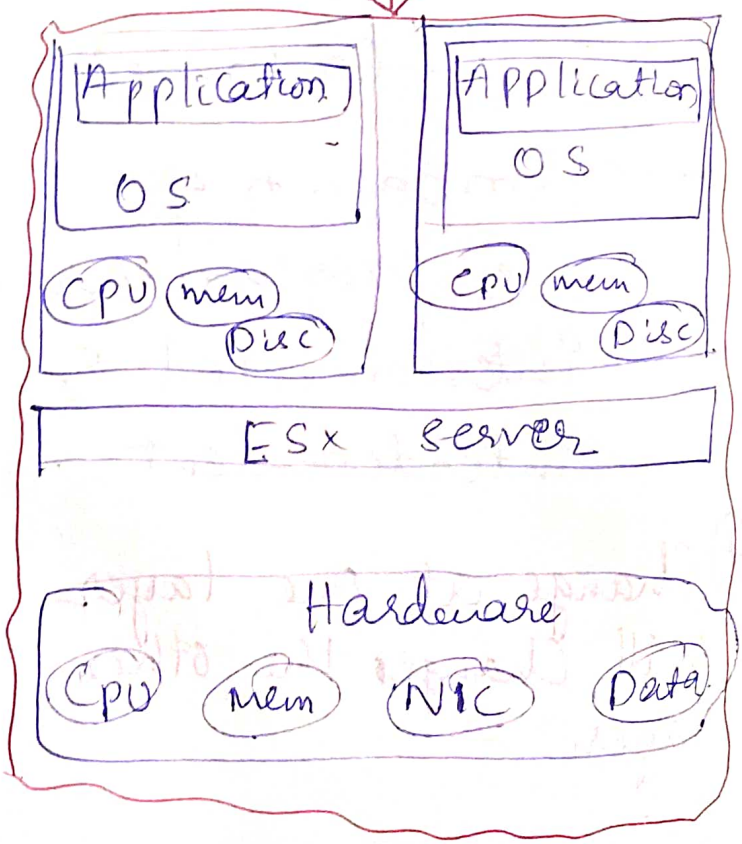


(*) Here job is divided into small micro services.

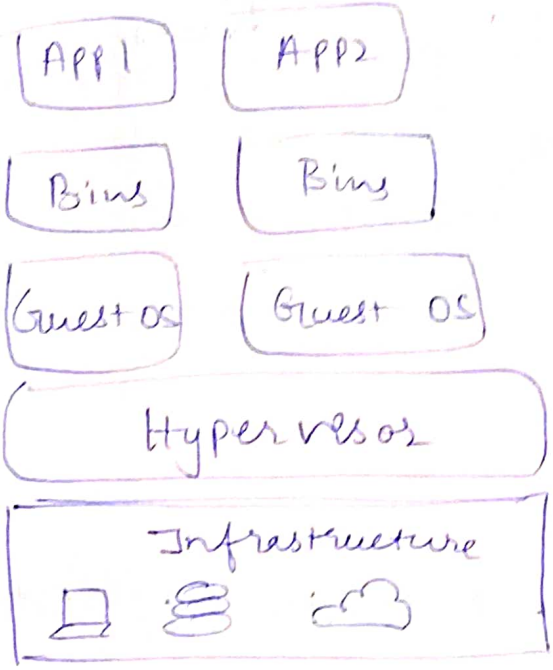
(*) The change in one service will only change that particular's interface

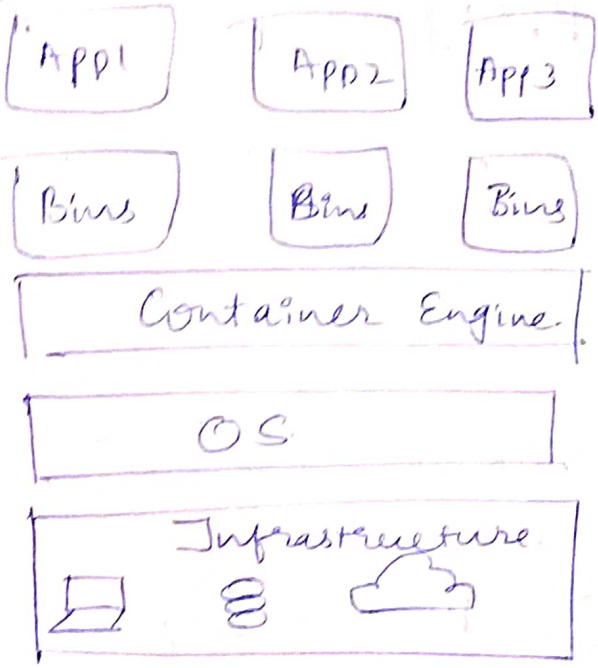
Physical servers vs Virtual Machine





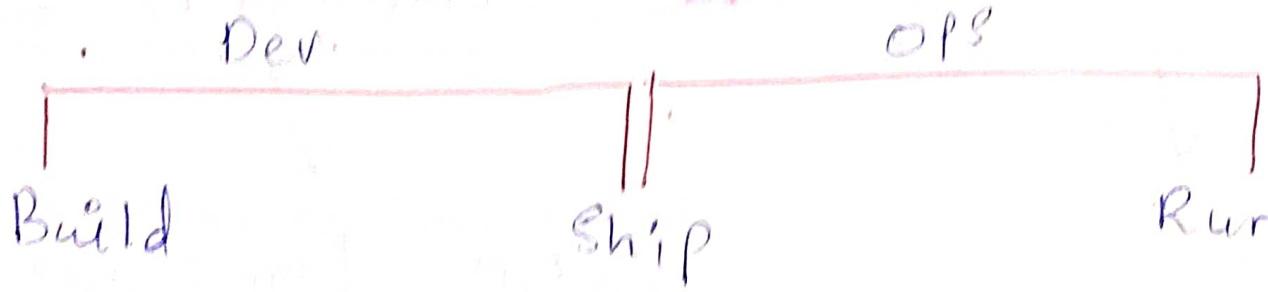
Virtual Machine VS Containers





* In virtual machine. Enables to install multiple OS using Hypervisor
* V.M enables the working in a requ-ired. O.S
* It does it in hardware level

* It enables the seperation of the application with the infrastructure/hardw
* The softwares used in the process is stored in the container and can be used in any OS by any ~~other~~ others, upto end user
* One of the most popular container is docker.

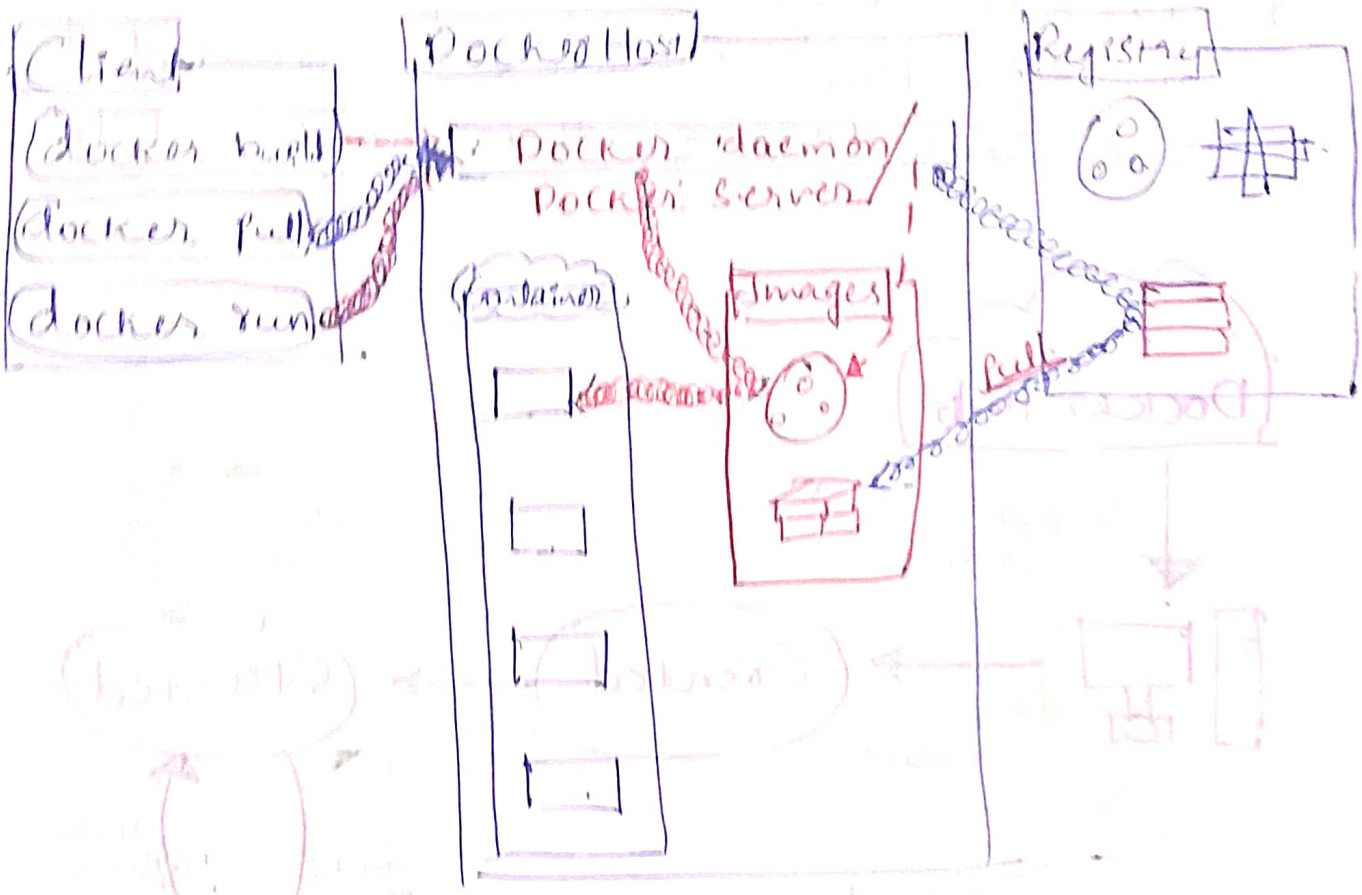


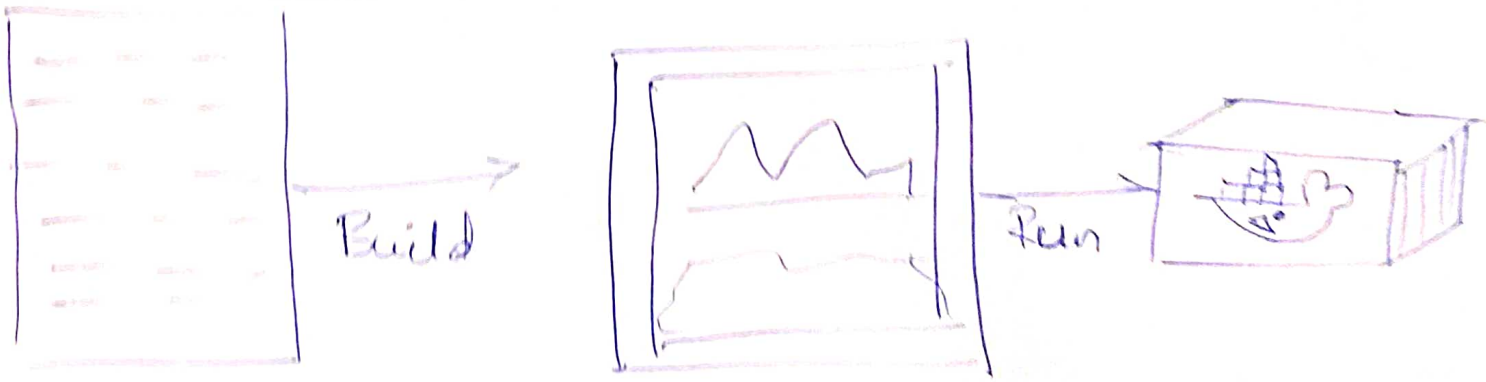
Docker Engine:-

(*) Docker engine is a client-server application with these major components

- A server
  - → Creates and manages Docker objects, such as Images, Containers, Networks & Volumes
- A Rest API
  - Communicate to server through API.
- Client
  - ↳ Docker CLI

Docker Architecture



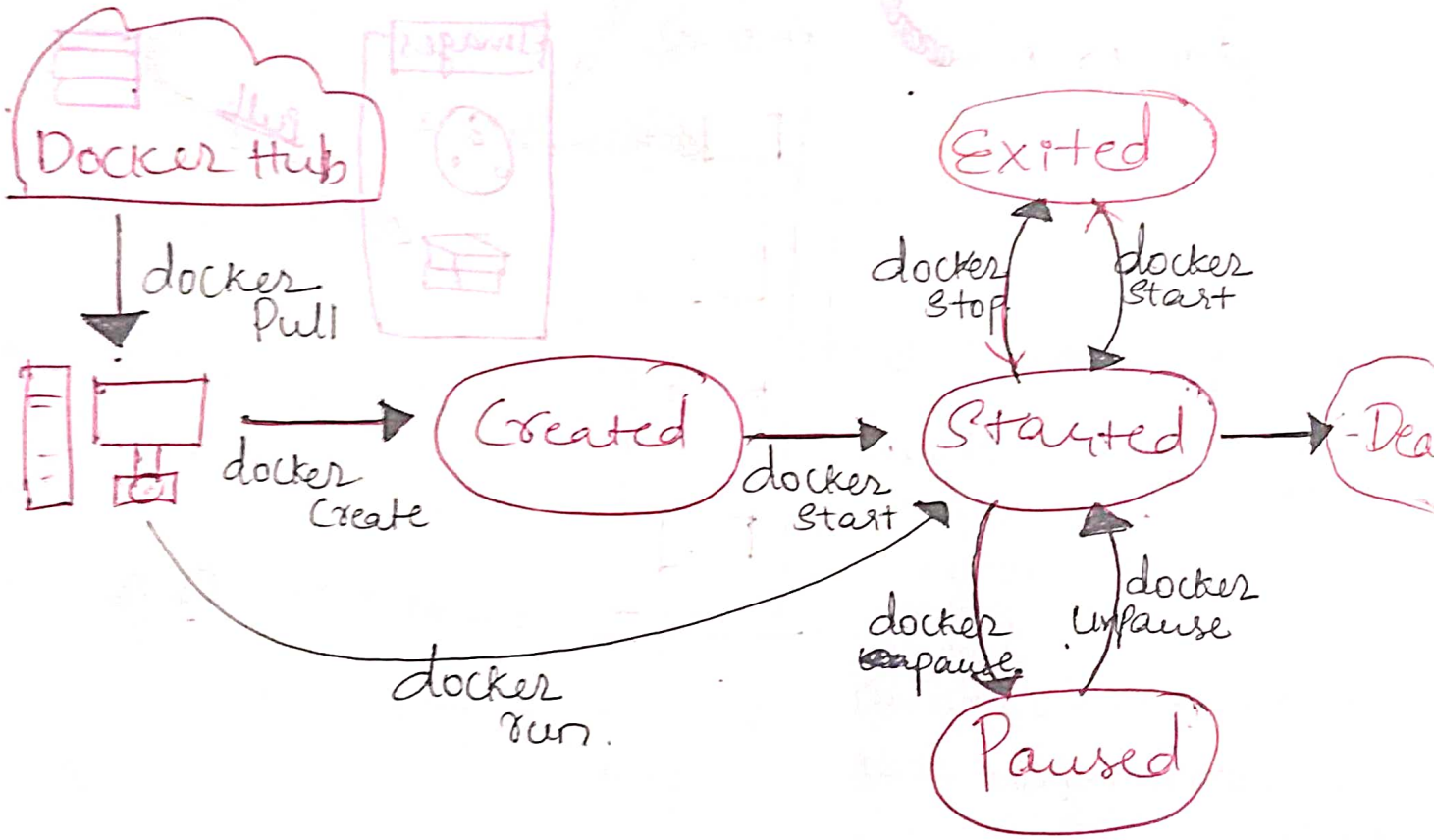


Docker File

D. Image

Docker Container

* Docker Image is create by reading the .dockerfile.
* Docker Image is a template containing multiple lines which are created by using the each line in the Docker File.



Model Building Program:

1. Read the data in to the pandas DataFrame

2. Exploratory Analysis
   - ↳ Shape
   - ↳ dtypes
   - ↳ describe()
   - ↳ Convert the attributes to appropriate type (as.type)
   - ↳ Find missing values
   - ↳ plots, etc

2a. Split the data into train and test

3. Feature Engineering

4. Imputation. (Filling the missing values)

5. Cat ⟶ Num : OneHotEncoding

6. Num ⟶ Standarizing / Normalize.

7. Building the model on train data and will be testing it performance on both train and test

9. Pick the best model and the pre-processing steps that are required to build the best model

---

1. Import the required class
2. Instantiate / Create an object.
3. Train the model using fit function
4. Predict / Transform
5. Class the error matric

Required Libraries

SVM → Support vector machine.

Sklearn → It contains many ML algorithm
Scikit learn package

* Simple Imputer → To fill the missing values. (categorical)

* StandardScaler
OneHotEncoder,
LabelEncoder
↳ Yes or no
} → To fill the missing values (numerical)

* One HotEncoder will create the dummy columns of the unique attributes.

import os → used to interact with system level command → path, cwd(), join

Classification Problem → Bank Marketing

import configparser → used to read config file

import numpy as np

import pandas as pd

from sklearn.impute import SimpleImputer → To fill the missing values of categorical

from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder

StandardScaler → $\frac{Value - \bar{x}}{S.d}$ ← Bringing the values b/w −3 to 3 or 0 and 1

OneHotEncoder → For categorical attribute it will create a seperate column

LabelEncoder → Yes or no → 1 or 0

from sklearn.model_selection import train_test_split → To split train and test data

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score

```
import warnings
warnings.filterwarnings('ignore')
```
} It will filter all the errors and doesnot displays.

```
PATH = os.getcwd()
```
→ Set the current working directory

```
print('\n Reading Config file...')
config = configparser.ConfigParser()
config.read(PATH + '/conf/config.ini')
```
} Reading config.ini file. Since config.ini file has the information about MySQL → host, portnumber ... ↓ Unless knowing this info we cannot connect to the mysql running on another container.

```
host = Config['MySQL']['host']
Port = Config['MySQL']['Port']
User = Config['MySQL']['user']
Password = Config['MySQL']['password']
db = Config['MySQL']['db']
```

```
Cat_Attr_Names = config['Dtypes']['category'].split(',')
Num_Attr_Names = Config['Dtypes']['float64'].split(',')
```

Connector = 'mysql+mysqlconnector://' + str(user) + ':' + str(password) + '@' + str(host) + ':' + str(port) + '/' + str(db)

Print(Connector)

data = pd.read_sql("Select * from bank", Con = Connector)

Understanding the Data

data.shape

data.columns

data.head()

data.tail()

Replace "unknown" with np.nan.

↳ data.replace(to_replace=['unknown'], value=np.nan, inplace=True

∴ Customer_no. is not of much value to dropping it.

data = data.drop(['customer_no'], axis=1)

⇒ Summary Statistics → data.describe().

data.dtypes

↓

In this output we observed that Few attributes such as job, marital, education, default, housing, loan, contact, month, day_of_week, Poutcome and y are CATEGORIC but are interpreted as Object type

∴ TypeCasting – Convert the attribute into appropriate type.

using astype('category') to convert job, marita ---- ---- ---- Each to categorical attribute from existing object datatype.

```
data[Cat_Attr_Names] = data[Cat_Attr_Names].apply(lambda col: col.astype('category'))

data[Num_Attr_Names] = data[Num_Attr_Names].apply(lambda col: col.astype('float64'))
```

⇒ handling missing data.

```
X = data.drop('y', axis=1)   ← storing whole data except 'y' attribute in X
Y = np.array(data['y'])      ← storing the 'y' values in Y
Cat_Attr_Names.remove('y')
```

```
data.isnull().sum()
```

⇒ Categorical attributes distribution

```
for attr in Cat_Attr_Names:
    Print(attr)
    Print(data[attr].value_counts(),"\n")
```

Shows the all type of attributes with the various sub-attributes along with count.

Pd.Value_counts(y) → Displays the numbers of yes & no's