Introduction to Machine Learning

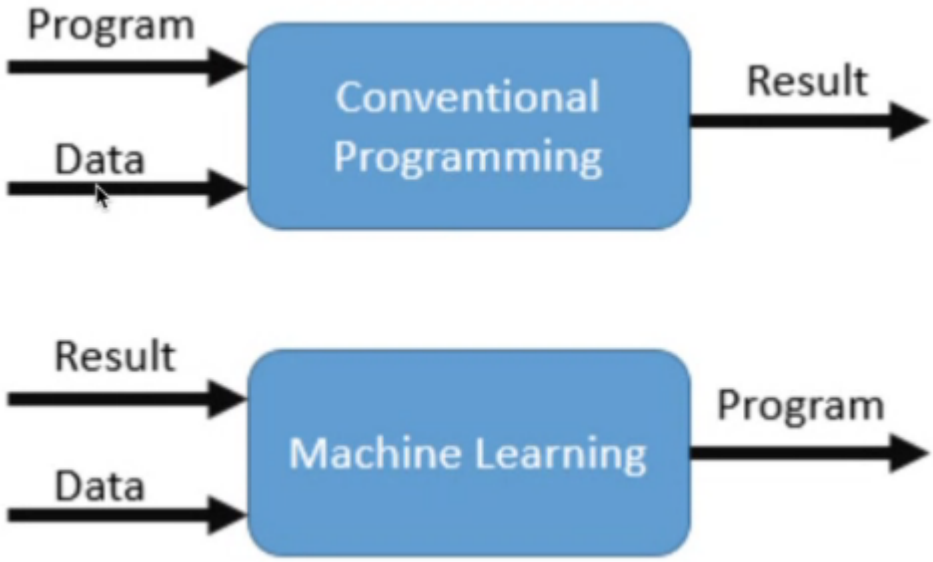
There are two types of AI

- AGI > Artificial General Intelligence
    - Can do all tasks that a human can do.
- ANI > Artificial Narrow Intelligence
    - Machine Translations
    - Classifications
    - Computer Vision etc.

Self Learning

- Python with Type Hints
- SQLModel
- Vector Databases (Pinecone etc.)
- FastAPI
- Docker (Write Once, Run Anywhere)
- CNAI (Cloud Native AI)

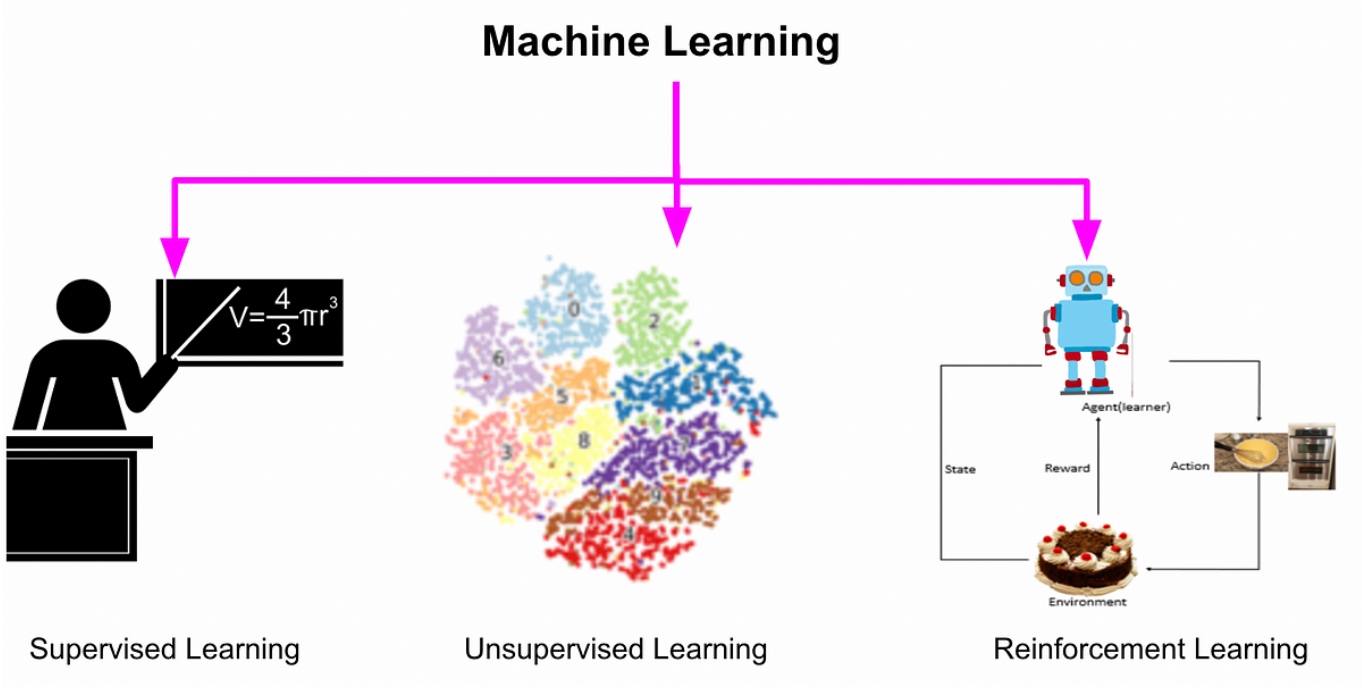
Conventional Programming vs Machine Learning



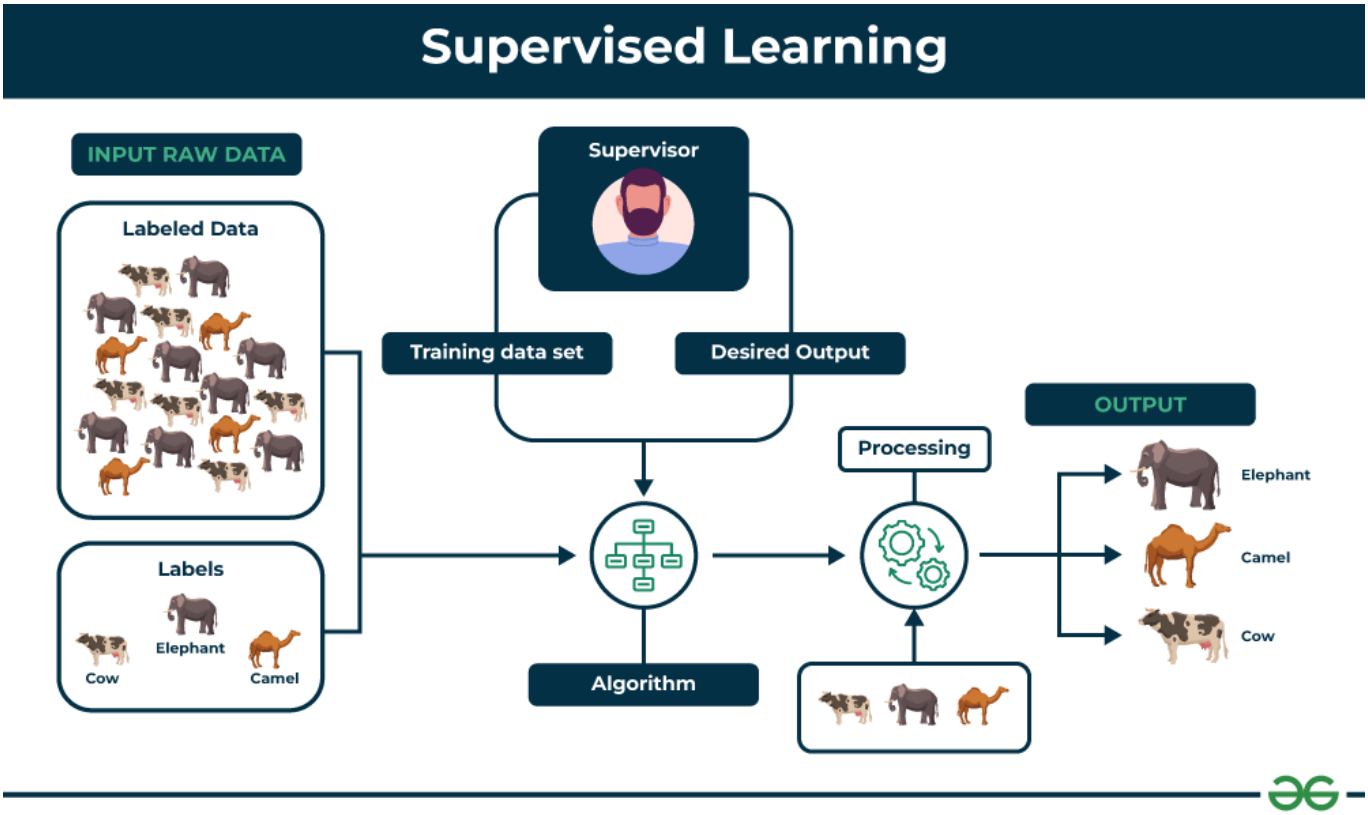
- ML Model > Knowledge Graph

Types of Machine Learning

*There are **03** types of Machine Learning*



01- Supervised Learning



- Supervisor/ML Engineer will filter out all the non-essential information in input.
- Supervised Learning is limiting because Human Intervention is required.
- ML Engineer will perform *Feature Extraction / Feature Engineering*.
- Solution is to use Deep Learning which uses ANN (Artificial Neural Networks) which automatically performs Feature Engineering.

Example of Feature Extraction

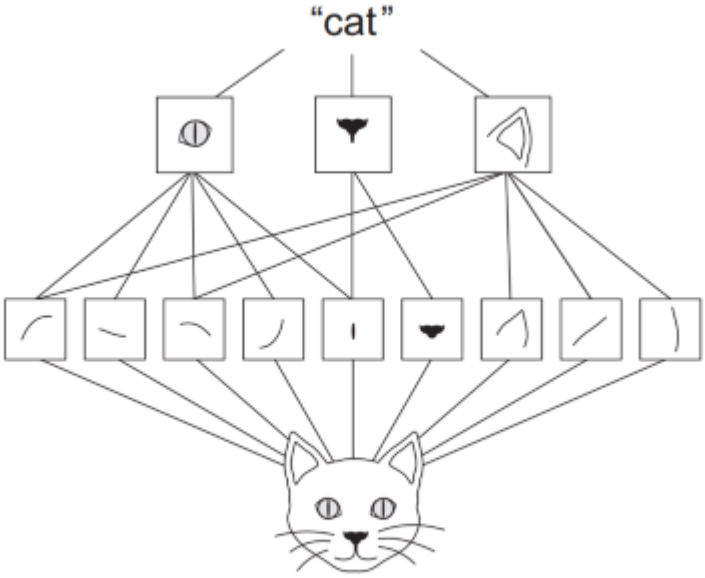
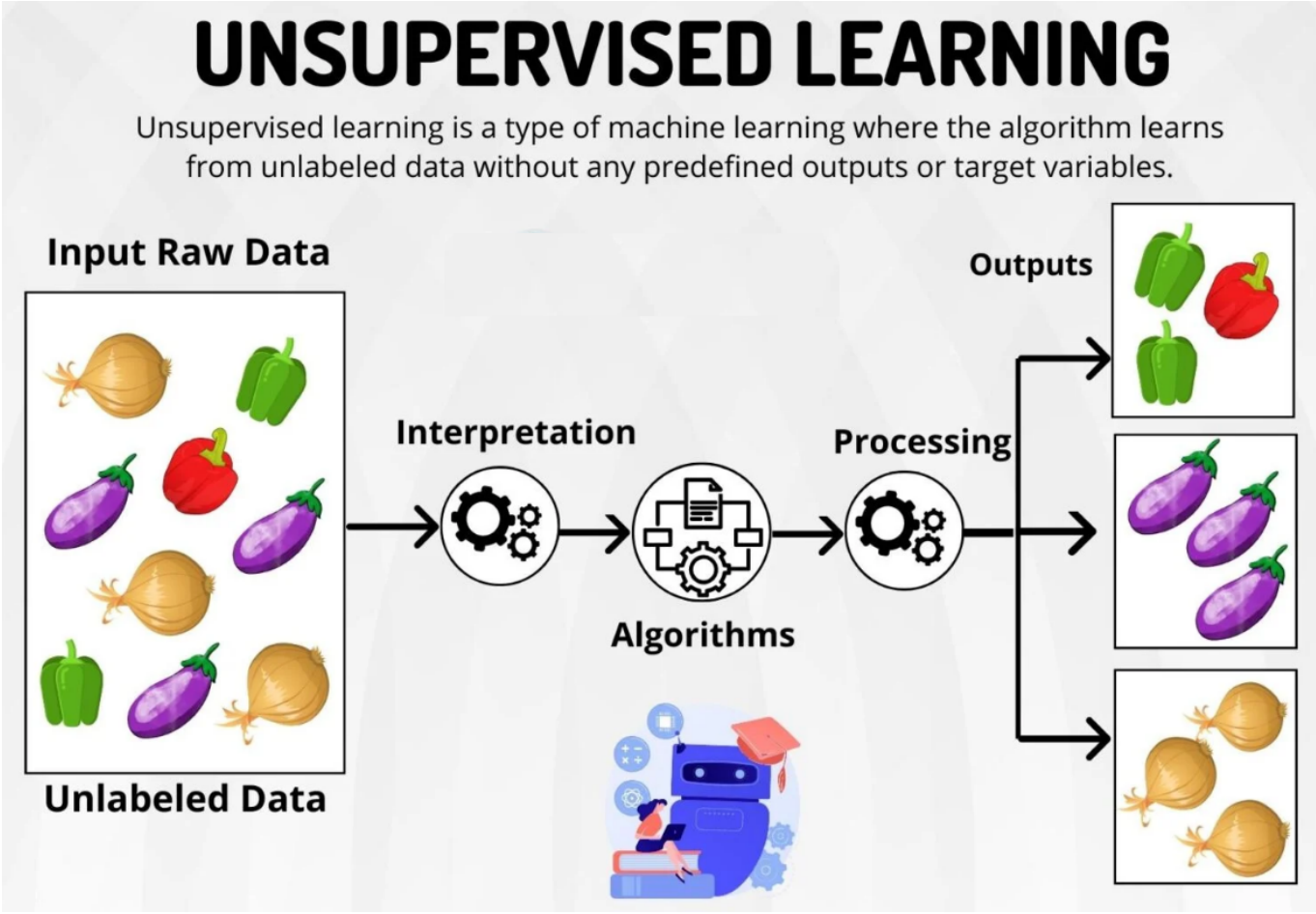


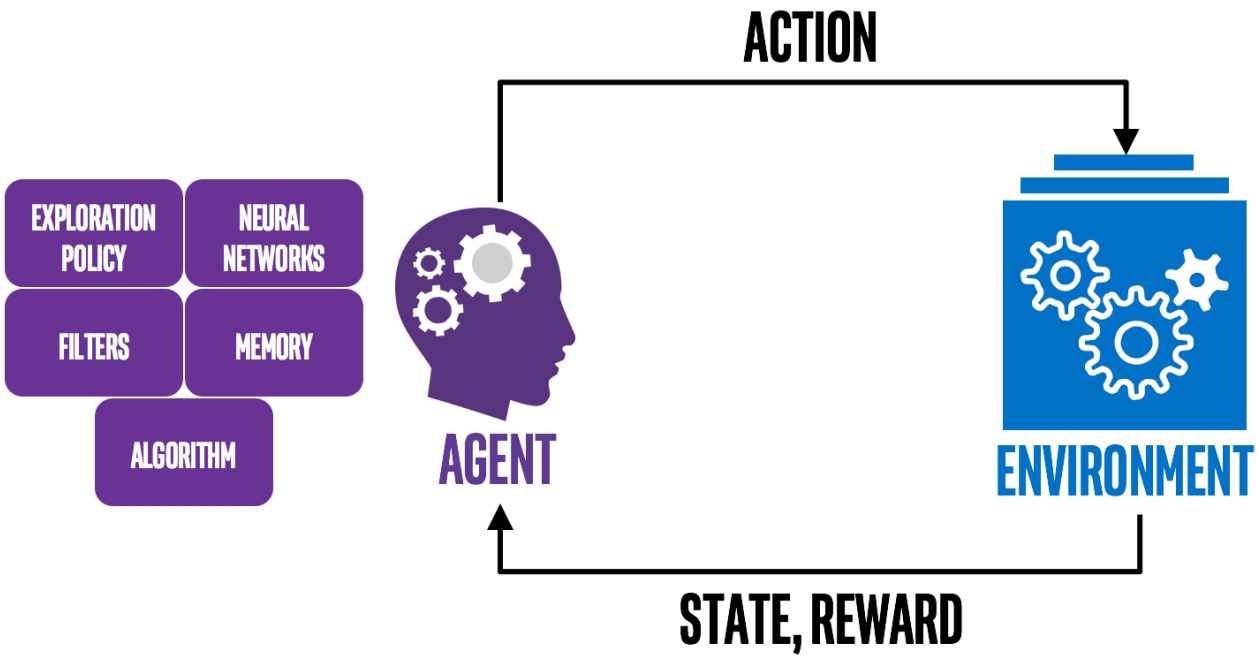
Figure 5.2 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as "cat."

02- Unsupervised Learning



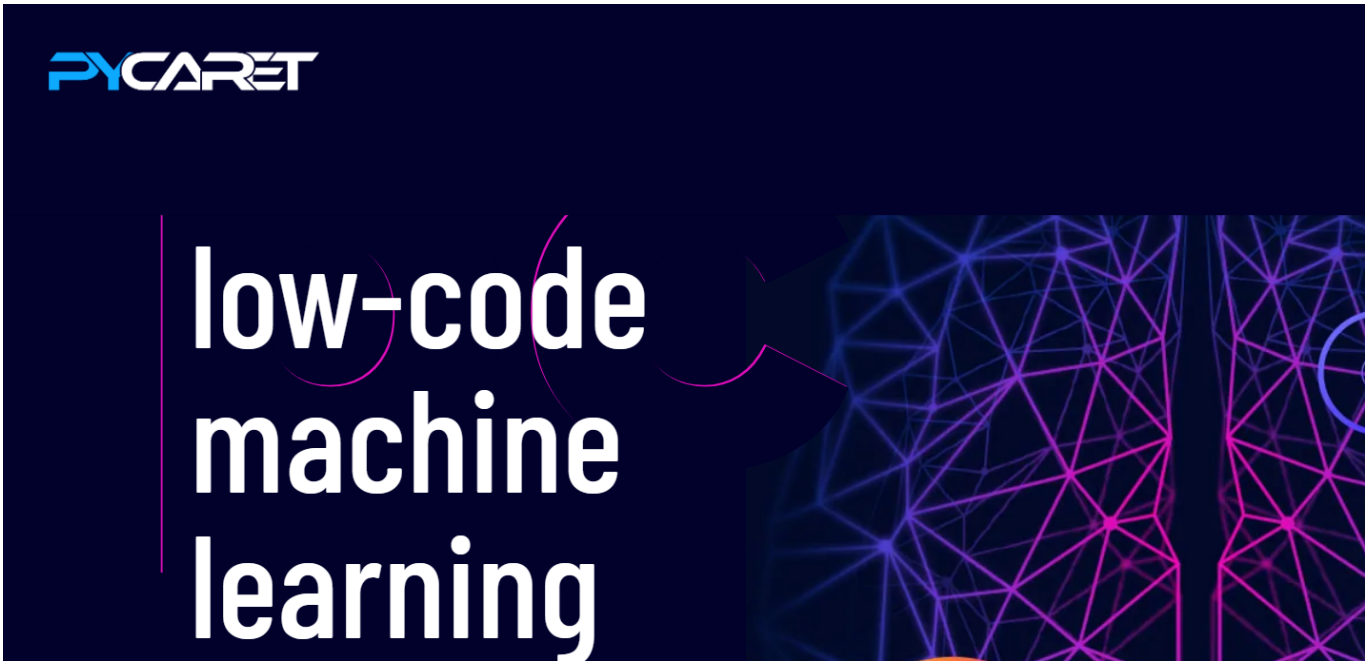
- We do not provide labels/result with Data.
- ML Algorithms apply labels itself.

03- Reinforcement Learning



- AI Agent will get current State from Environment.
- AI Agents perform some Action on Environment.
- Environment will give some Reward (either Positive / Negative)
- The AI Agent will change its future Actions based on previous Rewards.

PyCaret



- Provide ML Solutions in a single library.
- Low Code solution.

Notes Taken By

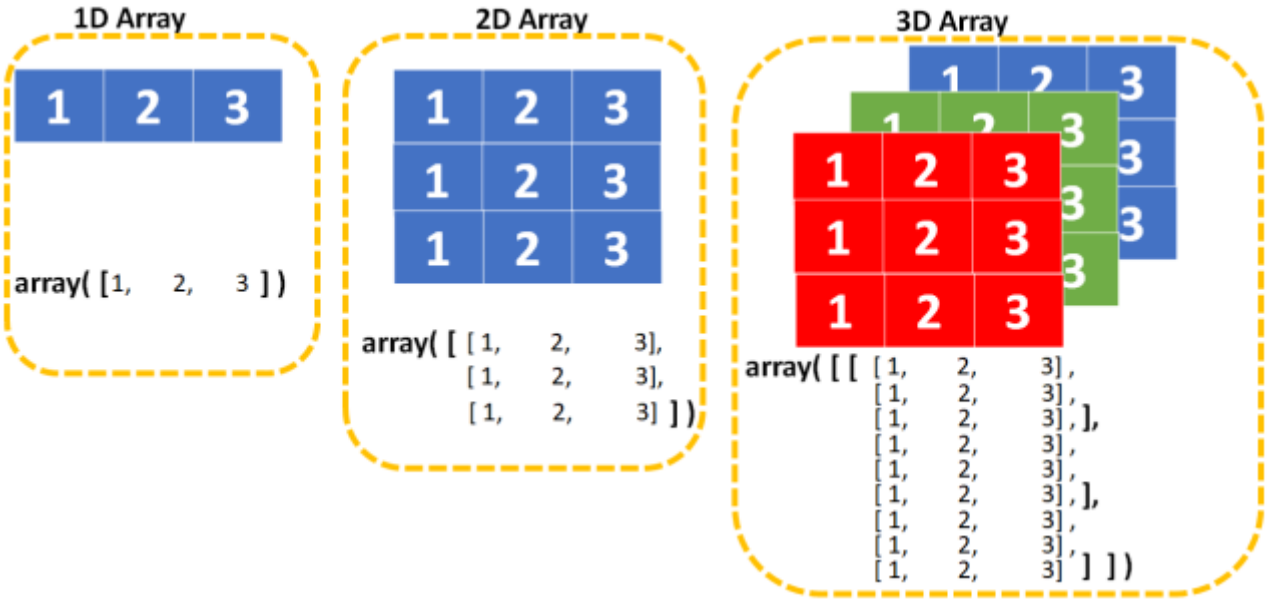
- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Vectorization

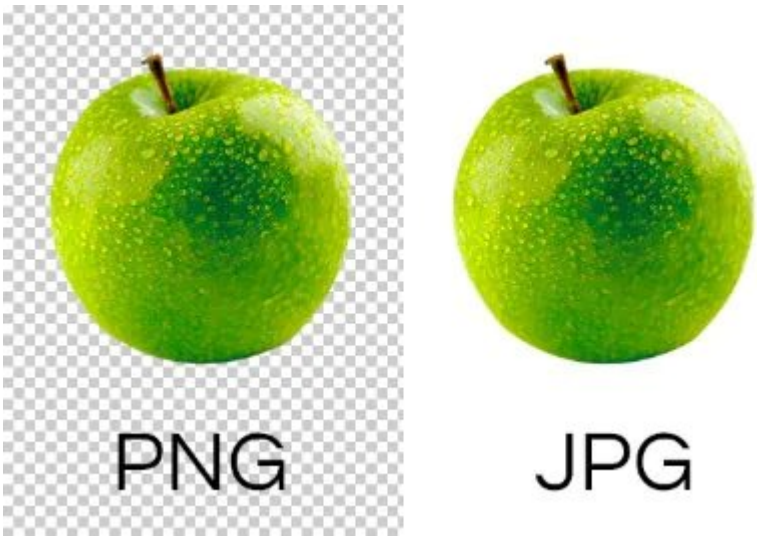
Conversion to an Raw Data into an array of numbers is called vectorization

An image is converted into 03 dimensions.

- X-Axis
- Y-Axis
- RGB Values



For PNG images, there is an additional channel i.e. **Alpha**. This indicates the transparency.



- To identify an image, the colors may or may not be required.

- If colors are not important, we can use *Grayscale* images as it will reduce Computation Power requirements.

OpenCV

OpenCV is a library of programming functions mainly for real-time computer vision.

- Used for Pre-Processing of Image Data.

Image Segmentation

In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple image segments, also known as image regions or image objects.

Tensorflow

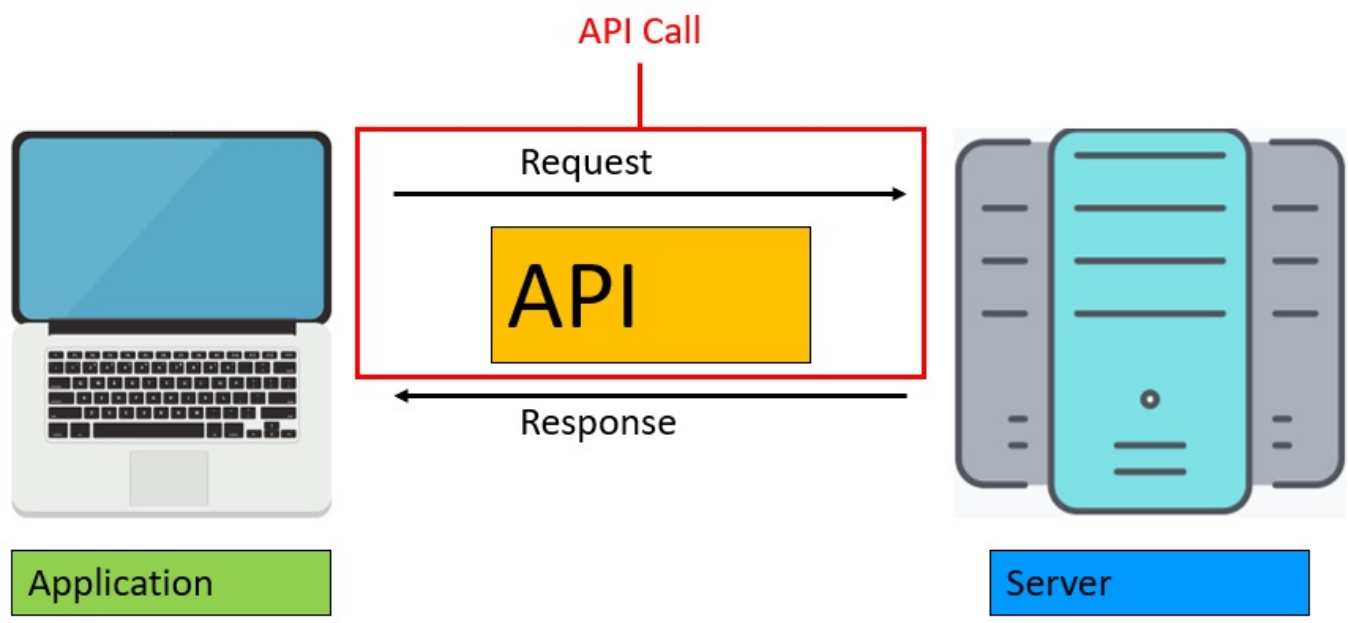
TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

Training Data Ratios

- 70% for training 30% for testing
- 80% for training 20% for testing
- 60% for training 40% for testing
- 60% for training, 20% validation, 20% for testing

API Calls

Application programming interfaces (APIs) are a way for one program to interact with another. API calls are the medium by which they interact. An API call, or API request, is a message sent to a server asking an API to provide a service or information.



API Standards

- JSON Input JSON Output

PyPI

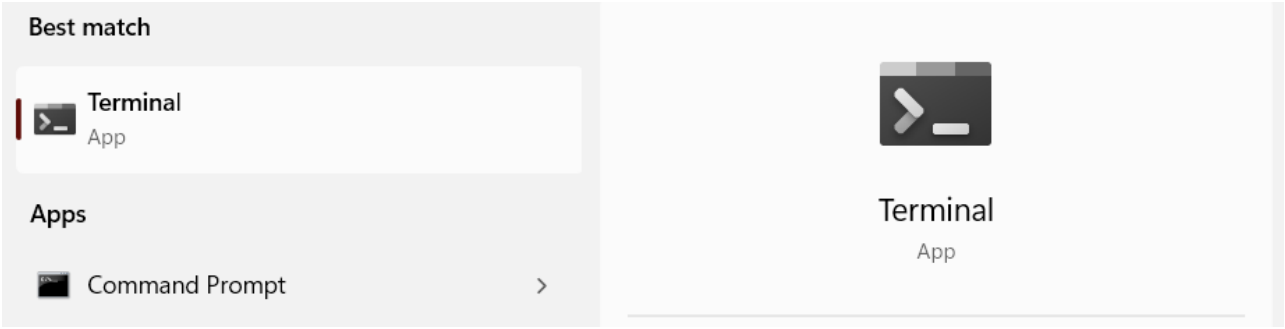
The Python Package Index (PyPI) is a repository of software for the Python programming language.

https://pypi.org/

Installation of Poetry Package

Install Scoop

- Open Terminal in Windows



- Run Following Commands in Sequence
  - `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`
  - `irm get.scoop.sh | iex`

```
PS C:\Users\raaid> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
PS C:\Users\raaid> irm get.scoop.sh | iex
Initializing...
Downloading...
Creating shim...
Adding ~\scoop\shims to your path.
Scoop was installed successfully!
Type 'scoop help' for instructions.
```

Installing Pipx

- Run following highlighted command in Terminal

```
PS C:\Users\raaid> scoop install pipx
Installing 'pipx' (1.5.0) [64bit] from 'main' bucket
pipx.pyz (311.8 KB) [=========================================================================================] 100%
Checking hash of pipx.pyz ... ok.
Running pre_install script...
Linking ~\scoop\apps\pipx\current => ~\scoop\apps\pipx\1.5.0
Creating shim for 'pipx'.
'pipx' (1.5.0) was installed successfully!
'pipx' suggests installing 'python'.
PS C:\Users\raaid> pipx ensurepath
Success! Added C:\Users\raaid\.local\bin to the PATH environment variable.

Consider adding shell completions for pipx. Run 'pipx completions' for instructions.

You will need to open a new terminal or re-login for the PATH changes to take effect.

Otherwise pipx is ready to go! ✨ 🌟 ✨
```

Installing Poetry

- Run following highlighted command in Terminal

```
PS C:\Users\raaid> pipx install poetry
'poetry' already seems to be installed. Not modifying existing installation in 'C:\Users\raaid\pipx\venvs\poetry'.
Pass '--force' to force installation.
PS C:\Users\raaid>
```

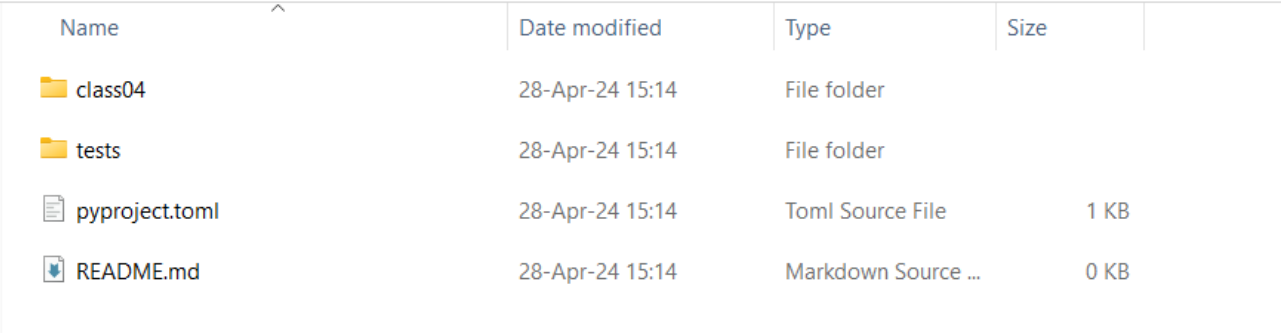
Using Poetry

Creating Package

- Open Terminal in you Project directory and run below command

```
E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry>poetry new class04
Created package class04 in class04

E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry>
```

- After successful command, below directories will be created.



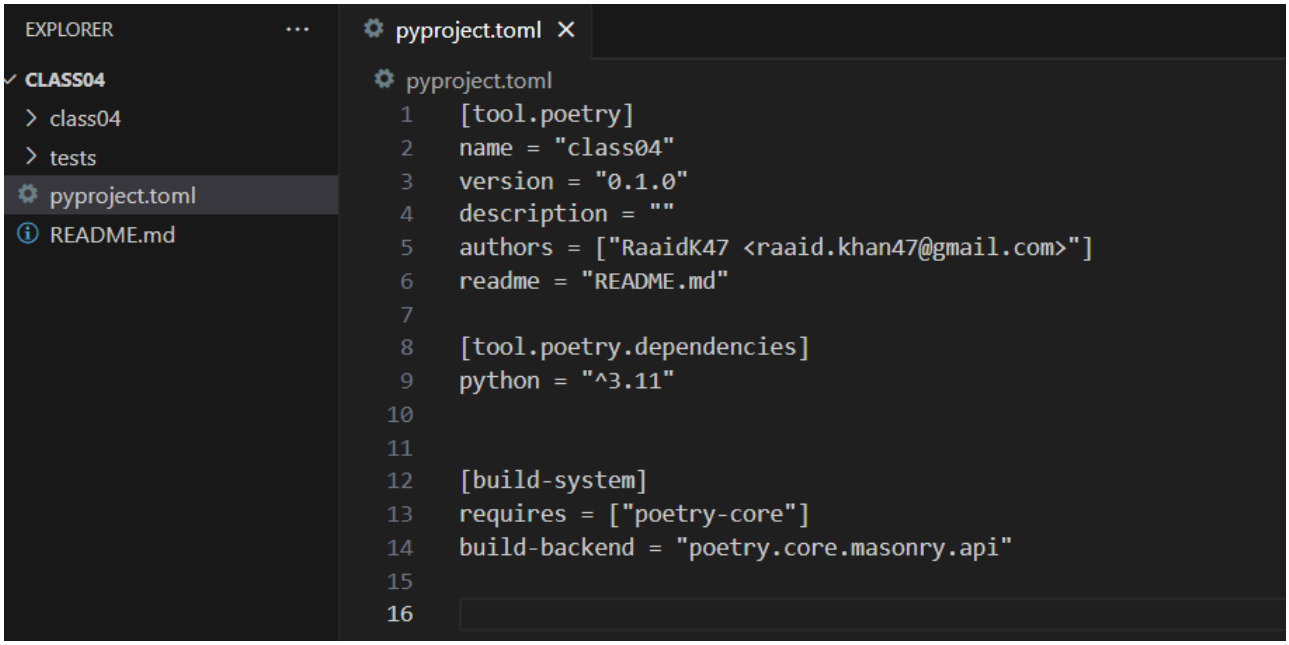
Name	Date modified	Type	Size
class04	28-Apr-24 15:14	File folder	
tests	28-Apr-24 15:14	File folder	
pyproject.toml	28-Apr-24 15:14	Toml Source File	1 KB
README.md	28-Apr-24 15:14	Markdown Source ...	0 KB

- Change Directory to Project Folder which Contains TOML (Tom's Obvious, Minimal Language) File.
    - If conda is installed, deactivate it using `conda deactivate`.
    - Run Poetry Shell in this folder using command `poetry shell`

```
E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>poetry shell
Creating virtualenv class04-FLkWC2od-py3.11 in C:\Users\raaid\AppData\Local\pypoetry\Cache\virtualenvs
Spawning shell within C:\Users\raaid\AppData\Local\pypoetry\Cache\virtualenvs\class04-FLkWC2od-py3.11

E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>()

(class04-py3.11) E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>
```

TOML File

- Open Project in VS Code
- Open `pyproject.toml` file



EXPLORER

CLASS04
- class04
- tests
- pyproject.toml
- README.md

pyproject.toml

```toml
[tool.poetry]
name = "class04"
version = "0.1.0"
description = ""
authors = ["RaaidK47 <raaid.khan47@gmail.com>"]
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.11"


[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"

```

- `.toml` contains MetaData (Data about Data) of our Project.
    - Author Details
    - Project Dependencies
    - Python `^3.11` (Version 3 is fixed (^), .11 can be changed)

Installing Dependencies in Project

- Open any terminal in Project Folder containing `.toml` file.

- Install dependencies with command `poetry add pandas`

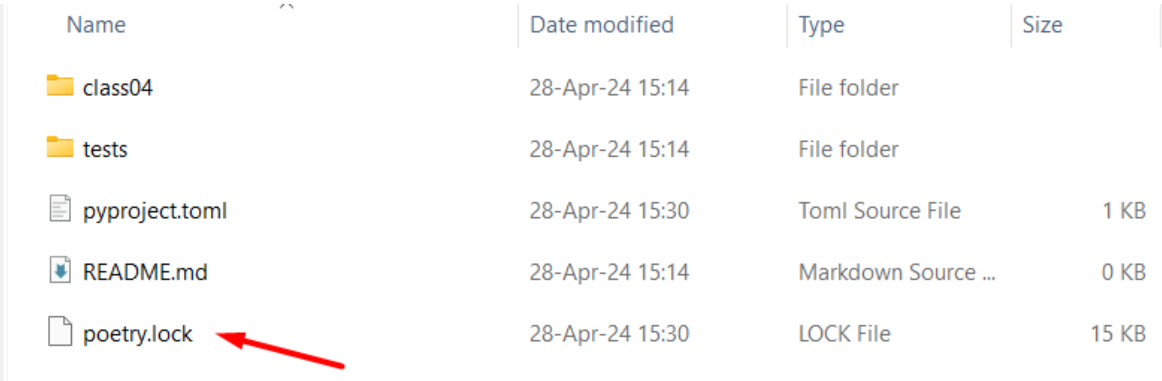
```
E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>poetry add pandas
Using version ^2.2.2 for pandas

Updating dependencies
Resolving dependencies... (1.5s)

Package operations: 6 installs, 0 updates, 0 removals

  - Installing six (1.16.0)
  - Installing numpy (1.26.4)
  - Installing python-dateutil (2.9.0.post0)
  - Installing pytz (2024.1)
  - Installing tzdata (2024.1)
  - Installing pandas (2.2.2)

Writing lock file

E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>
```

- After Installation, `.toml` file will be change

pyproject.toml

```toml
[tool.poetry]
name = "class04"
version = "0.1.0"
description = ""
authors = ["RaaidK47 <raaid.khan47@gmail.com>"]
readme = "README.md"

[tool.poetry.dependencies]
python = "^3.11"
pandas = "^2.2.2"


[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

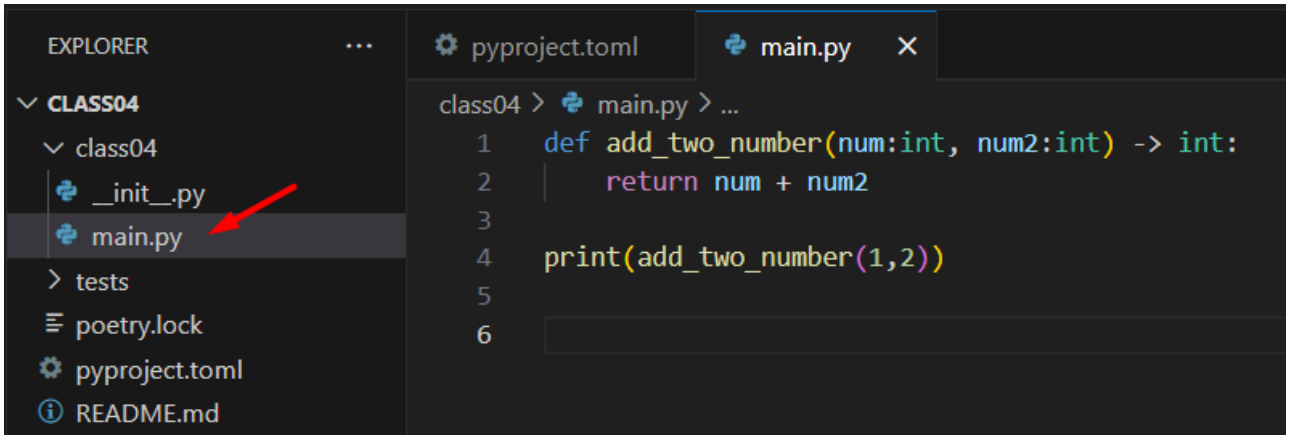
- A `poetry.lock` file will also be created.



Creating the Project

- Go to project folder i.e. `class04`

- Create `main.py` file.



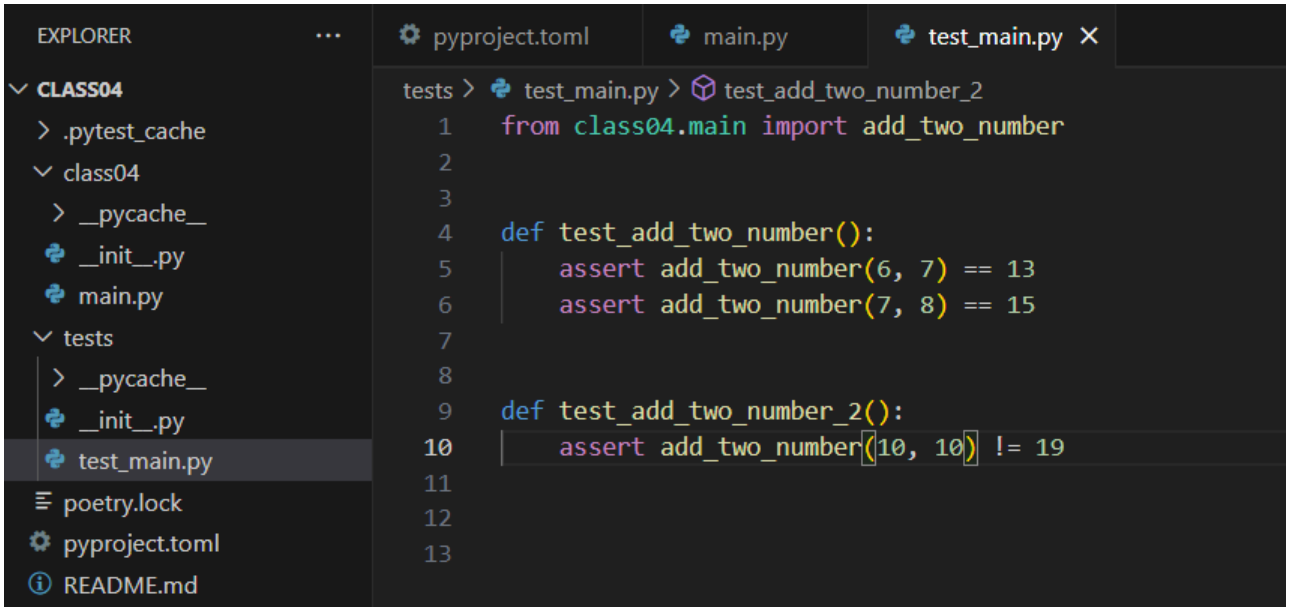
Running Project

- Open Terminal in Main Folder
- Run Following command

```
E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>poetry run python ./class04/main.py
3
```

Writing Tests

- Create a Test in `tests` folder.



- Test Application using `Poetry`

- First install `pytest` in Poetry

```
E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>poetry add pytest
Using version ^8.2.0 for pytest

Updating dependencies
Resolving dependencies... (1.2s)

Package operations: 5 installs, 0 updates, 0 removals

  - Installing colorama (0.4.6)
  - Installing iniconfig (2.0.0)
  - Installing packaging (24.0)
  - Installing pluggy (1.5.0)
  - Installing pytest (8.2.0)

Writing lock file
```

- Run tests using following command

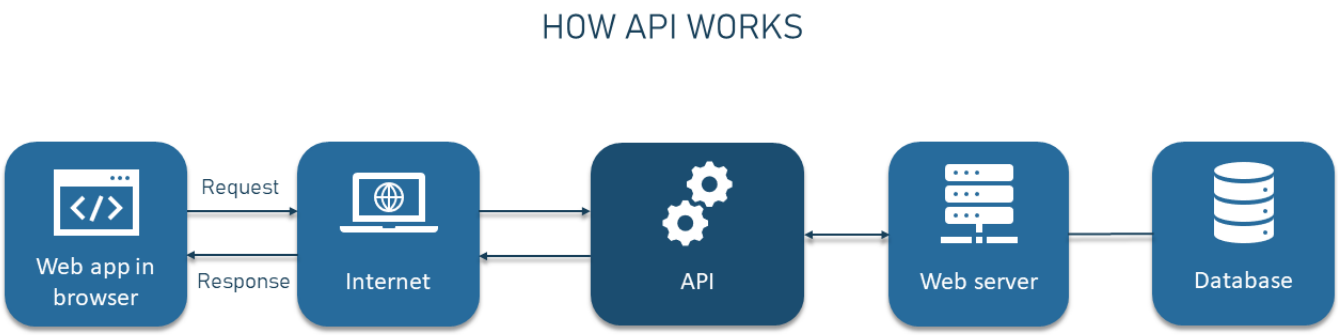
```
E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04>poetry run pytest -v
============================================= test session starts =============================================
platform win32 -- Python 3.11.5, pytest-8.2.0, pluggy-1.5.0 -- C:\Users\raaid\AppData\Local\pypoetry\Cache\virtualenvs\c
lass04-FLkWC2od-py3.11\Scripts\python.exe
cachedir: .pytest_cache
rootdir: E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L03-04\Code\Poetry\class04
configfile: pyproject.toml
collected 2 items

tests/test_main.py::test_add_two_number PASSED                                                       [ 50%]
tests/test_main.py::test_add_two_number_2 PASSED                                                     [100%]

============================================== 2 passed in 0.02s ==============================================
```

Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Application Programming Interfaces (APIs)

An application programming interface (API) is a way for two or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software.



FastAPI

FastAPI is a modern web framework for building APIs *(AI powered APIs)* with Python 3.7+ that is gaining widespread popularity. It is known for its high performance, auto-generated interactive documentation, and simplicity. FastAPI uses Python type hints for defining request and response models, which enables automatic validation and interactive documentation.

FastAPI vs Django REST Framework

FastAPI and Django Rest Framework are both powerful tools for building APIs in Python, but they cater to different use cases and preferences.

- If you need a high-performance, async-ready API with a shallow learning curve and interactive documentation, FastAPI is an excellent choice.
- If you are developing a full-stack Django application and need a comprehensive set of tools for API development, Django Rest Framework is a proven and reliable choice.

FastAPI Application

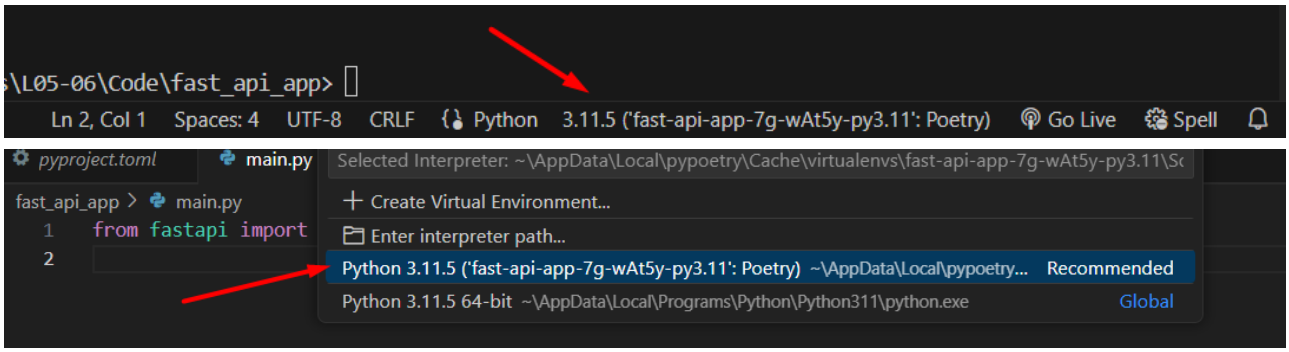
- Create a `Poetry` environment.

- Install following dependencies (reference from book)

```
(fast-api-app-py3.11) PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L05-06\Code\fast_api_app> poetry add fastapi uvicorn httpie requests httpx
Using version ^0.111.0 for fastapi
Using version ^0.29.0 for uvicorn
Using version ^3.2.2 for httpie
Using version ^2.31.0 for requests
Using version ^0.27.0 for httpx
```

- Create `main.py` file in app sub-folder.

- In case of Intellisense error, change environment to Poetry



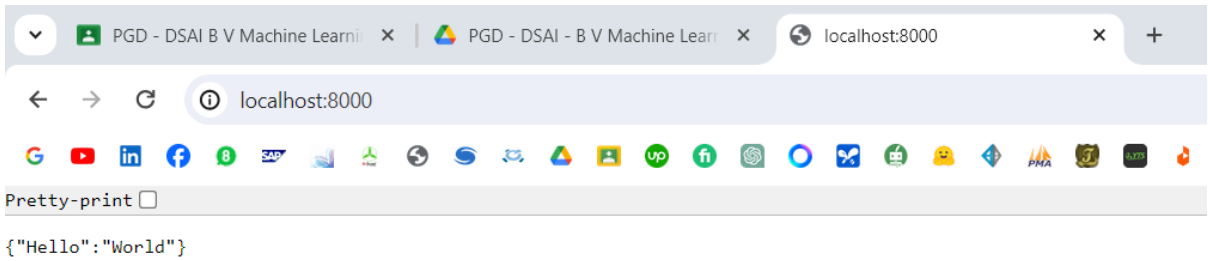
- Run FastAPI Server [Method-01 (Without Poetry)]
  - Open terminal in `main.py` folder
  - Run Server using `uvicorn`

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L05-06\Code\fast_api_app\fast_api_app> uvicorn main:app
INFO:     Started server process [13256]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     127.0.0.1:63335 - "GET / HTTP/1.1" 200 OK
```

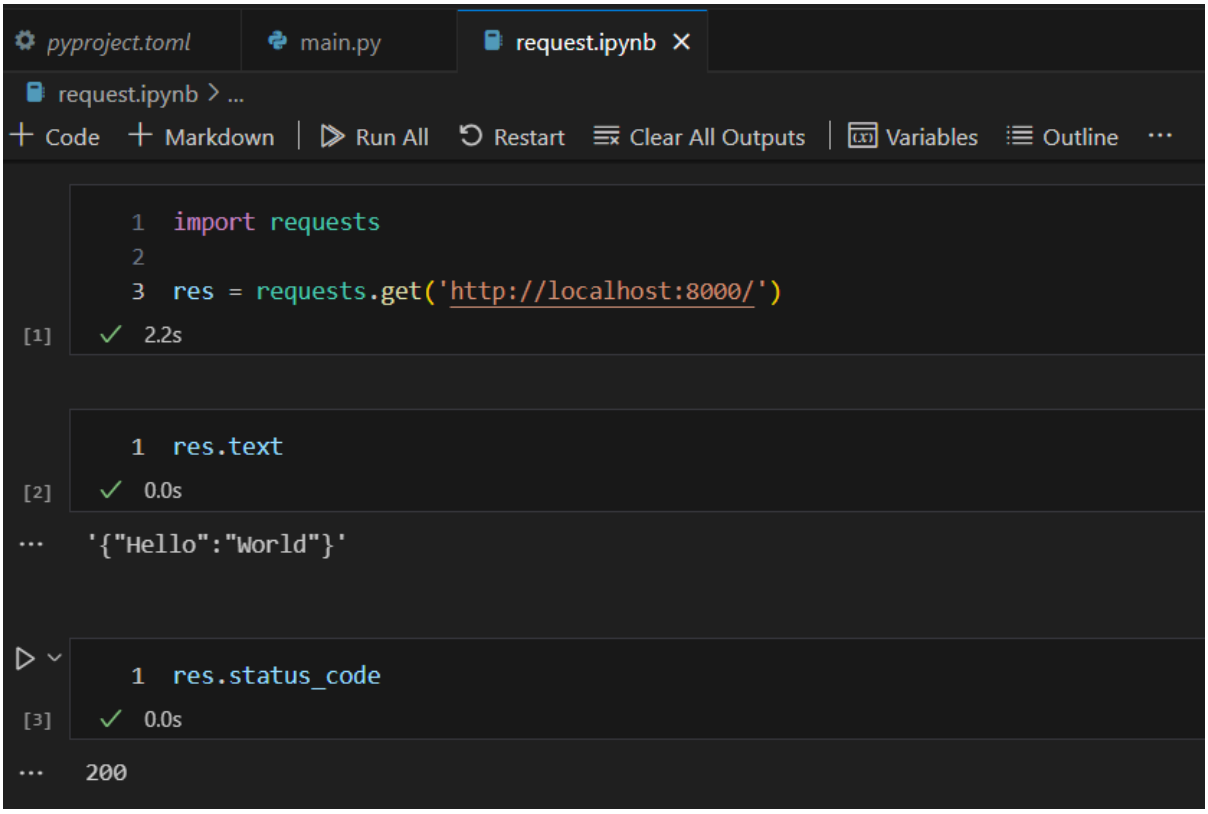
- Run FastAPI Server [Method-02 (With Poetry)]
  - Open terminal in folder with `.toml` file.

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L05-06\Code\fast_api_app> poetry run uvicorn fast_api_app.main:app --reload
INFO:     Will watch for changes in these directories: ['E:\\PGD-CCEE\\C04 - Machine Learning\\Lecture Notes\\L05-06\\Code\\fast_api_app']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [8340] using WatchFiles
INFO:     Started server process [17492]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:64211 - "GET / HTTP/1.1" 200 OK
```

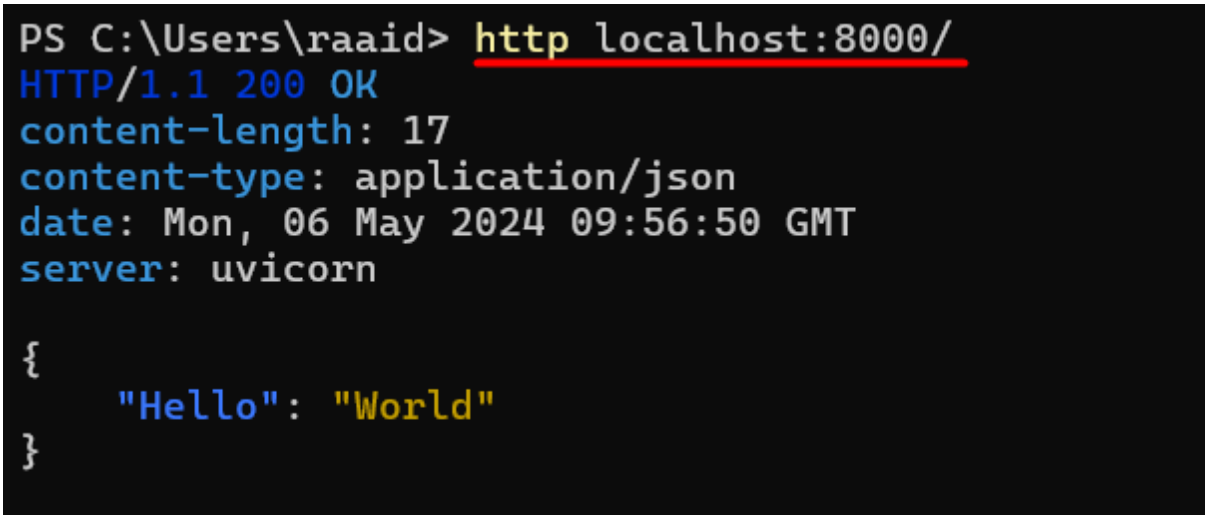
  - `--reload` flag restarts server if `main.py` file is changed.
- This Server can be accessed from
  - Browsers



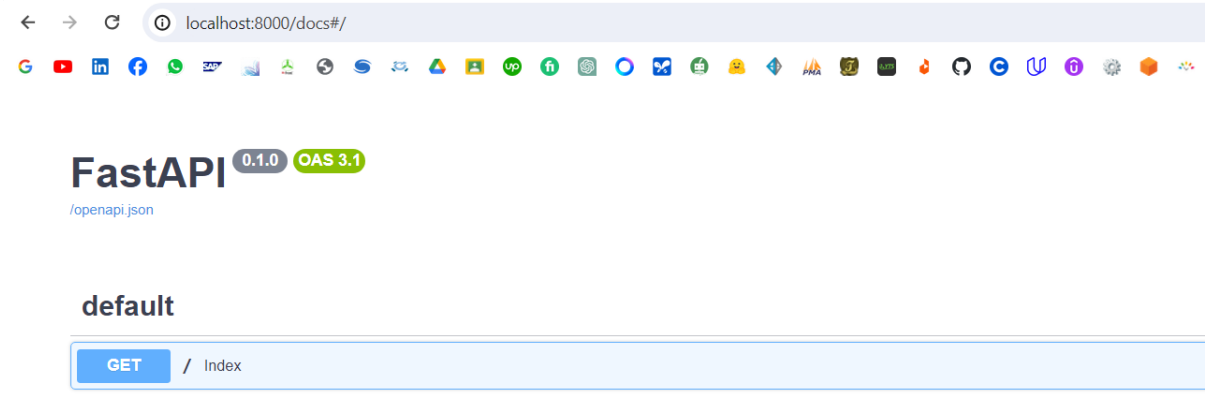
- Using `requests` package.



- Using `httpie` package [Terminal]

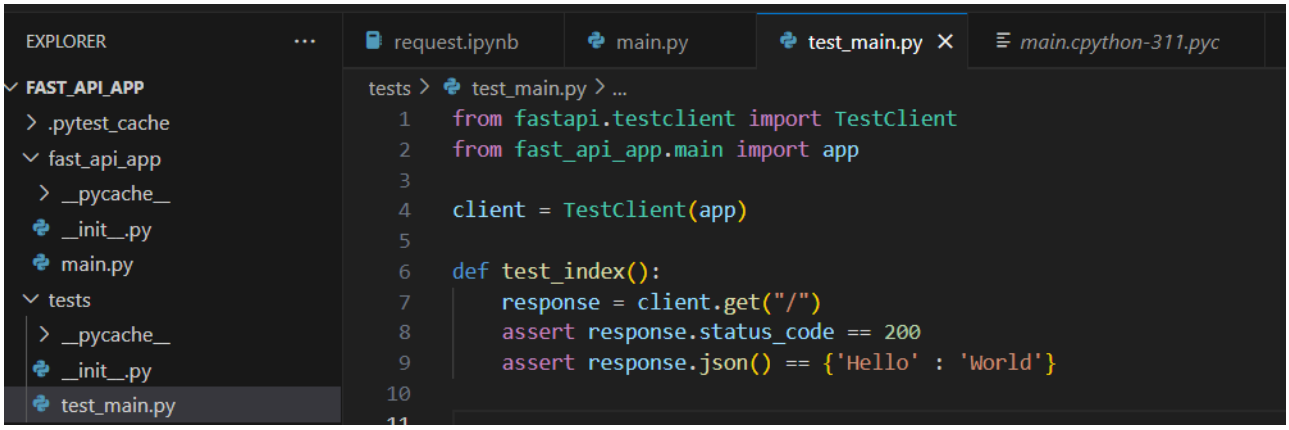


- Accessing Documentation [Swagger]
  - `http://localhost:8000/docs`

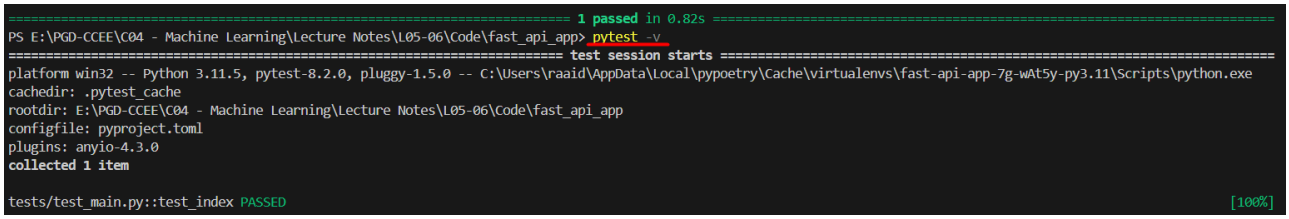


Writing Tests

- Create `test_main.py` file in *tests* folder.



- Test using `pytest` in terminal



Sync vs Async

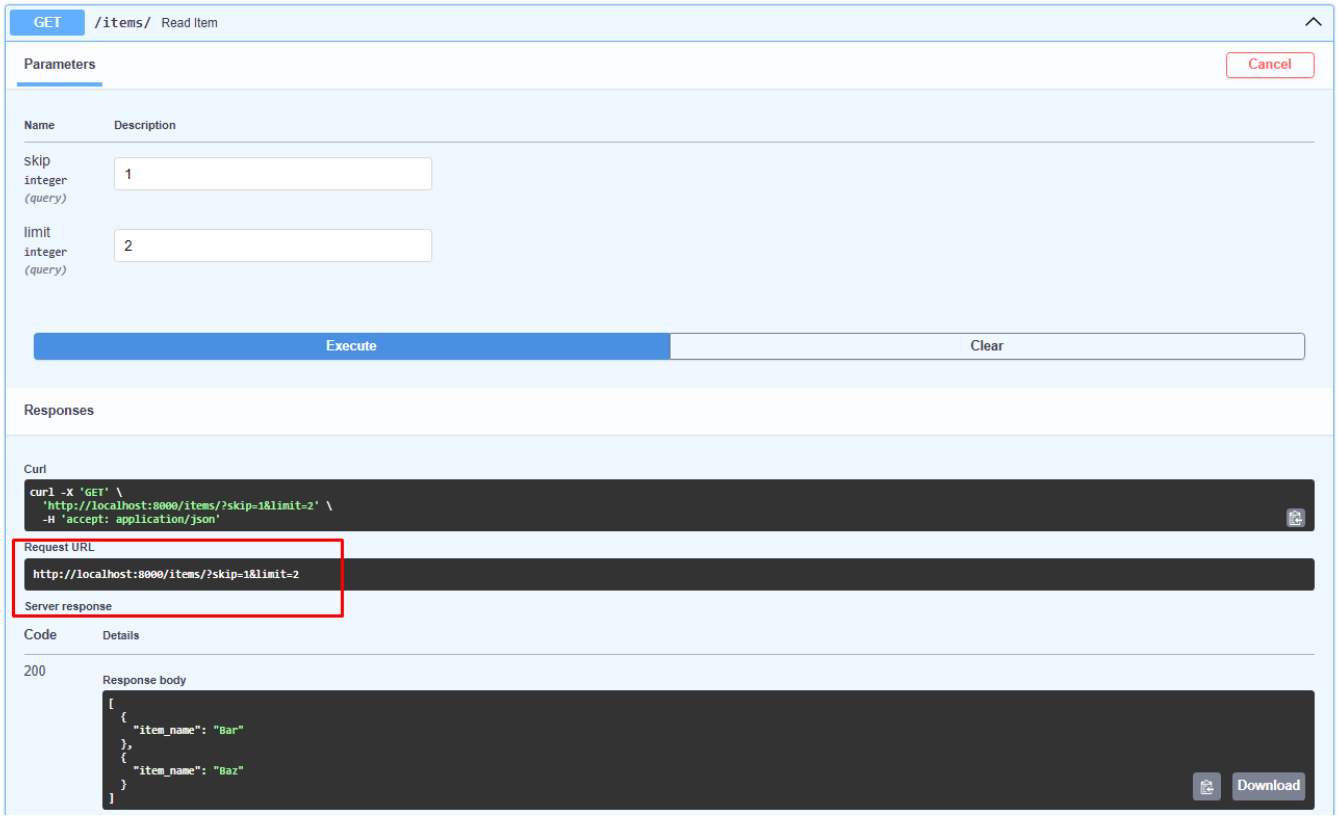
Synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed. It is blocking in nature.

Asynchronous is a non-blocking architecture, which means it doesn't block further execution while one or more operations are in progress.

```
@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Passing multiple Query Parameters

```
# Query Parameters : Can be used to pass data to the function
fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]

# Passing two query parameters to the function
@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]
```

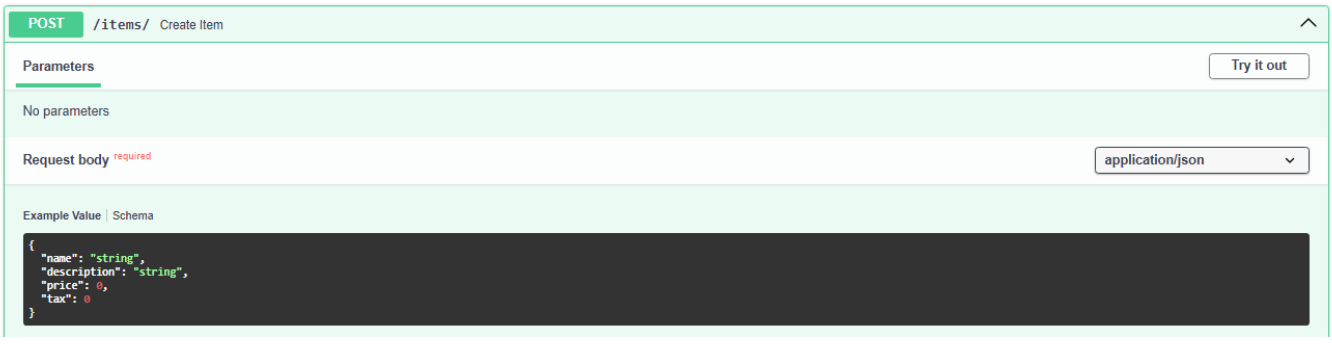


Pydantic for Data Validation

Pydantic is a Python library for data validation and settings management using Python-type annotations. It ensures that the data you work with matches your specified data types, simplifying error handling and data parsing in Python applications.

```python
# Using Pydantic to validate the data (Pdantic shows the response type)
class Item(BaseModel):
    name: str
    description: str | None = None
    price: float
    tax: float | None = None

@app.post("/items/")
async def create_item(item: Item):
    return item
```

Client can see the structure/schema of Request from the Documentation by using Pydantic.



Dependency Injection

"Dependency Injection" means, in programming, that there is a way for your code (in this case, your *path operation functions*) to declare things that it requires to work and use: "dependencies".

And then, that system (in this case **FastAPI**) will take care of doing whatever is needed to provide your code with those needed dependencies ("inject" the dependencies).

This is very useful when you need to:

- Have shared logic (the same code logic again and again).
- Share database connections.
- Enforce security, authentication, role requirements, etc.
- And many other things...

All these, while minimizing code repetition.

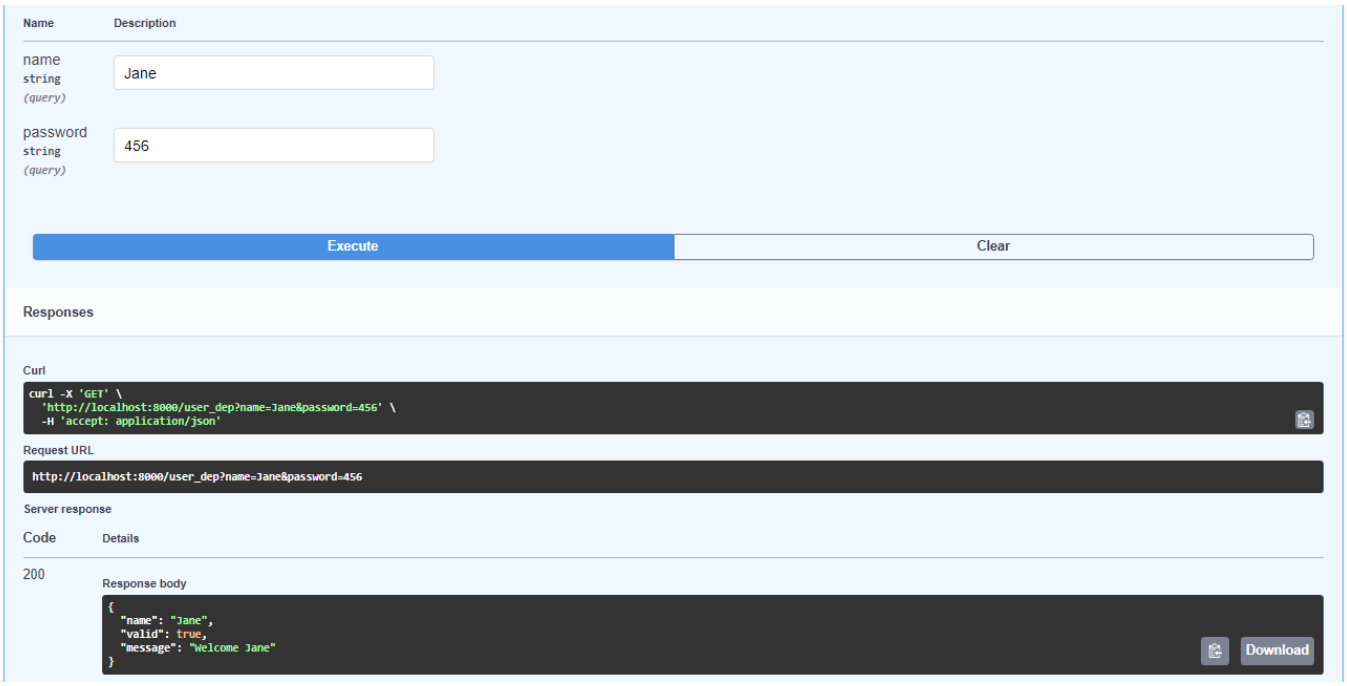
```python
from fastapi import FastAPI, Depends, Params

app = FastAPI()

# the dependency function:
def user_dep(name: str = Params, password: str = Params):
    return {"name": name, "valid": True}

# the path function / web endpoint:
@app.get("/user")
def get_user(user: dict = Depends(user_dep)) -> dict:
    return user
```

- The get_user() depends on user_dep() , it will run after user_dep() is executed.
- Depends(user_dep) means that user_dep() will be executed only when get_user() function is **Used** and will NOT execute when get_user() function is **Defined**.

Hands-On

```python
users : list[dict[Any,Any]] = [
    {'name' : 'John', 'password' : '123'},
    {'name' : 'Jane', 'password' : '456'}
]

# the dependency function:
def user_dep(name: str = params, password: str = params):
    for d in users:
        if (d['name'] == name and d['password'] == password):
            return {"name": name,
```

```
                    "valid": True,
                    "message": "Welcome " + name}
    else:
        return {"name": name, "valid": False}

# the path function / web endpoint:
@app.get("/user_dep")
def get_user(user: Annotated[dict, Depends(user_dep)]) -> dict:
    return user
```



Streamlit

- Create a Poetry Project
- Add `Streamlit` library
- Copy/Create your Streamlit Application in Poetry Folder.
- Create `__init__.py` file in your streamlit application folder
- Run streamlit app using following command
    - `poetry run streamlit run ./00_helloworld/hello.py`

Notes

- `Session` dictionary is created on Server.
- `Cookies` are created on Client.

Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Docker

Docker is an open source platform that enables developers to build, deploy, run, update and manage containers — i.e. standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

- **Docker Image:** A reusable, shareable file used to create containers. A blueprint of your Container.
- **Docker Container:** A runtime instance; a self-contained software. Created from an Image.

Docker Commands

- `docker version`
- `docker run hello-world`
    - Will download a Dummy Image to test proper Docker Installation
- Pull Anaconda Image
    - docker pull continuumio/anaconda3
- `docker images`
    - All images created in Docker
- `docker run -it continuumio/anaconda3:latest /bin/bash`
    - `-it` Run docker in *Interactive Mode*
    - Name of Image and its version (latest)
    - Run terminal in root directory (/bin/bash)
- `CTRL+PQ`
    - Exit container without terminating it.
- `docker container ls`
    - Details of all running containers
- `docker exec -it <container_name/container_id> bash`
    - Go into a Docker container that is already running. [Will not make another container]
- `docker ps`

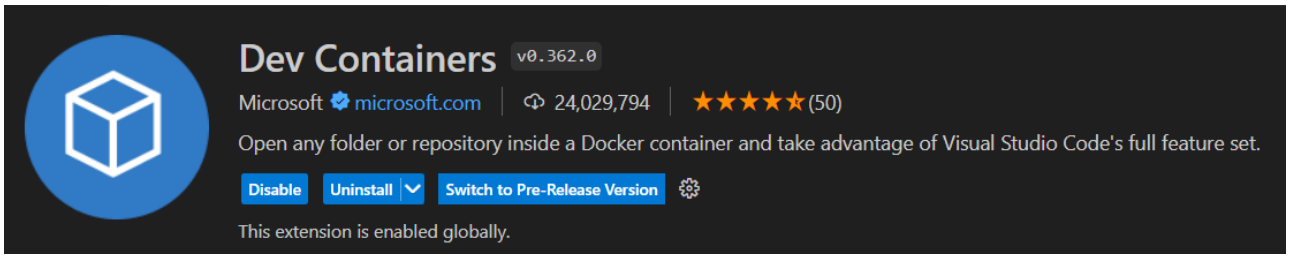
    - List All Running Containers
- `docker ps a`
    - List the Containers, even those that are in stopped state
- `docker stop <container_name/container_id>`
    - Stop container
- `docker rm container_name`
    - Deleting a Docker Container. [Container must be stopped first]

Linux Flags

- Single Dash (-) > Each letter will be considered as separate command
    - `- it` (i and t are separate commands)
- Double Dash (--) > Complete Word will be considered as a command
    - `--reload`

Dev Container Extension

- Install following extension in VsCode



- This will show all containers, you can explore file system of Container as well
- This will allow you to code in VsCode that will execute in Docker Container.

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.12

LABEL maintainer="ameen-alam"
# Set the working directory in the container [Folder is created]
WORKDIR /code
# Install system dependencies required for potential Python packages
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Install Poetry
```

```
RUN pip install poetry

# Copy the current directory contents into the container at /code
COPY . /code/

# Configuration to avoid creating virtual environments inside the Docker container
RUN poetry config virtualenvs.create false

# Install dependencies including development ones
RUN poetry install

# Make port 8000 available to the world outside this container
EXPOSE 8000

# Run the app. CMD can be overridden when starting the container.
# Command is passed inside and array. [Separated by Spaces]
CMD ["poetry", "run", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--reload"]
```

Building a Docker Image

- Building from a Simple Dockerfile

```
docker build -t my-image .
```

- Building from .dev or .prod Dockerfile

```
docker build -f Dockerfile.dev -t my-image .
```

Running a Docker Container from Image

```
docker run -d --name container-1 -p 8000:8000 my-image
```

- `-d` = detach [Container will run in background but you will remain in terminal of base operating system]
- `8000:8000` = Expose port 8000 of Container to port 8000 of Host Computer.

Databases

Data Sanitization

Making sure that input Query does not consist of any Harmful material that can damage the Database and its Data

- SQL Alchemy

Data Validation

Making sure that Data in Query consist of only valid data and is following Schema of Database

- Pydantic

SQL Model

*A Python Package that performs both **Data Sanitization** and **Data Validation***

*SQLModel is an **ORM (Object Relational Mapper)** that converts Objects of OOP to that of Objects of Databases*

Types of Database

- Structured Database (SQL)
  - PostgreSQL
  - MySQL
- Non-Structured Database (NoSQL)
  - Key Value
  - Column Based
  - Document Based
  - Graph Databases

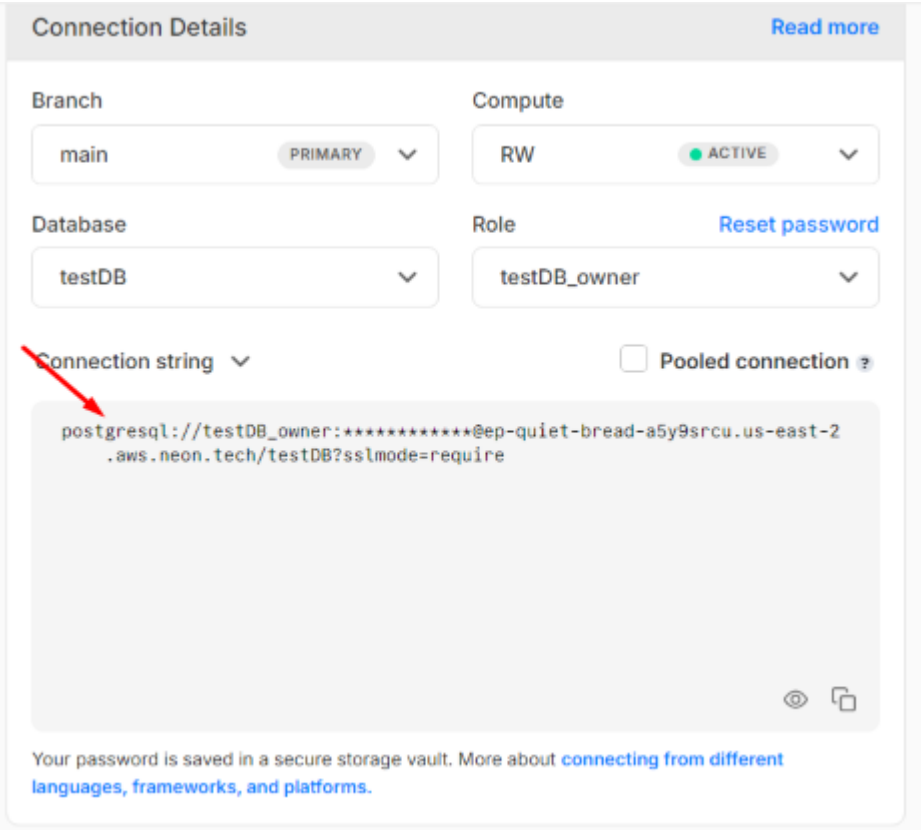
PostgreSQL

- Free
- Open Source
- Distributed

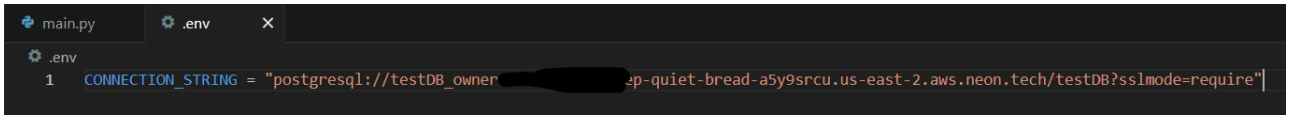
Working with SQL Model

- Go to `neon.tech` and create account.
- Create a `Test` Database

- Go to Connectivity of Database in Dashboard and copy Connection String



- Create a Poetry Project.
- Create a `.env` file in root of project.
    - Add this .env file to `.gitignore`
- Save your credentials in this `.env` file.



- You can retrieve this secret into your program vai Environment Variables using following code:

```
from dotenv import load_dotenv, find_dotenv
import os

#read .env file and load into environment
_ : bool = load_dotenv(find_dotenv())

# Get Secret value from you OS Environment
conn_string = os.environ.get("CONNECTION_STRING")

print(conn_string)
```

- Import SQLModel

```
from sqlmodel import Field, SQLModel, create_engine
```

- Create an Object of SQLModel

```
# A Table with Name Hero will be created in the Database
class Hero(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    name: str
    secret_name: str
    age: int | None = None
```

  - `table=True` means that object will of Pydantic + SQLAlchemy
  - `table=False` means that object will be of Pydantic only.

- Create the Database Engine to establish DB Connectivity.

```
engine = create_engine(conn_string, echo=True)
```

  - `echo = True` will show underlying SQL Queries

- Create a Table of your Object with Following code

```
SQLModel.metadata.create_all(engine)
```

- Create objects of you Hero Class

```
hero_1 = Hero()
hero_1.name = "Deadpond"
hero_1.secret_name = "Dive Wilson"
hero_1.age = 48

hero_2 = Hero(name="Spider-Boy", secret_name="Pedro Parqueador")
hero_3 = Hero(name="Rusty-Man", secret_name="Tommy Sharp", age=48)
```

- Create a Session, add Objects to DB and Commit Changes to DB.

```
session = Session(engine)

session.add(hero_1)
session.add(hero_2)
session.add(hero_3)
```

```
session.commit()
```

- Retrieve Data from Database

```
with Session(engine) as session:
statement = select(Hero)
results = session.exec(statement)
for hero in results:
    print(hero)
```

    - This will retrieve all Data from the Hero table in our Database

Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Environment Variable

- Information Stored in `.evn` file.
- Information is loaded into Operating System (OS)
- To get this information, the Hacker will have to Hack the Operating System or Cloud Platform on which that application is Hosted.

Docker Compose Commands

- `docker compose up`
    - Build all Services in compose.yaml file.
    - First Time
- `docker compose up --build`
    - Re-Build Services in case of any change in Source Code.

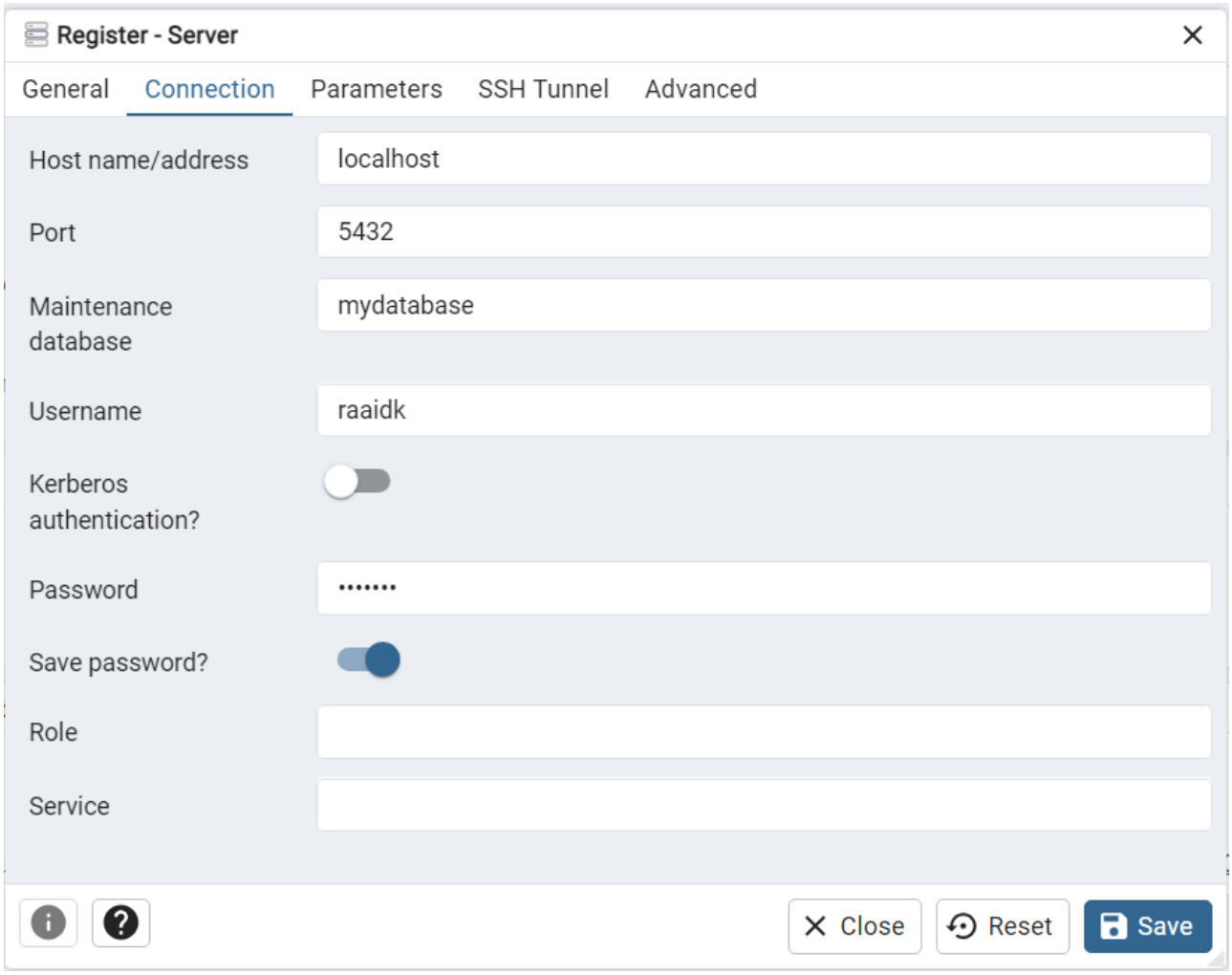
Docker Database Connection

```
conn_url = 'postgresql+psycopg2://yourUserDBName:yourUserDBPassword@yourDBDockerContainerName/yourDBName
```

pgAdmin

- Create a Database connection using environment variable of Postgres_db service from our `compose.yaml` file.

```yaml
  postgres_db:
    image: postgres:latest
    restart: always
    container_name: PostgresCont  # Name Used for Database Connectivity

    # These parameters will be made part of OS Environment on Docker Runtime
    # In Actual, we use .env file for Security Purpose
    environment:
      - POSTGRES_USER=raaidk
      - POSTGRES_PASSWORD=pass123
      - POSTGRES_DB=mydatabase
```



`Note:` Port = **5433** (i.e. Port of Local Computer on which the Docker Database is Mapped)

- This will connect pgAdmin to `PostgresCont` in Docker.

Database for Testing

- Create a Separate Database for Testing Functions.

```python
from app.main import app

def test_write_main():
```

```
    connection_string = str(settings.TEST_DATABASE_URL)

    engine = create_engine(
        connection_string, connect_args={"sslmode": "require"}, pool_recycle=300)

    SQLModel.metadata.create_all(engine)

    with Session(engine) as session:

        def get_session_override():
                return session

        # Override the Database session in Main App to TEST_DATABASE
        app.dependency_overrides[get_session] = get_session_override

        client = TestClient(app=app)

        todo_content = "buy bread"

        response = client.post("/todos/",
            json={"content": todo_content}
        )

        data = response.json()

        assert response.status_code == 200
        assert data["content"] == todo_content
```

Event Driven Architecture

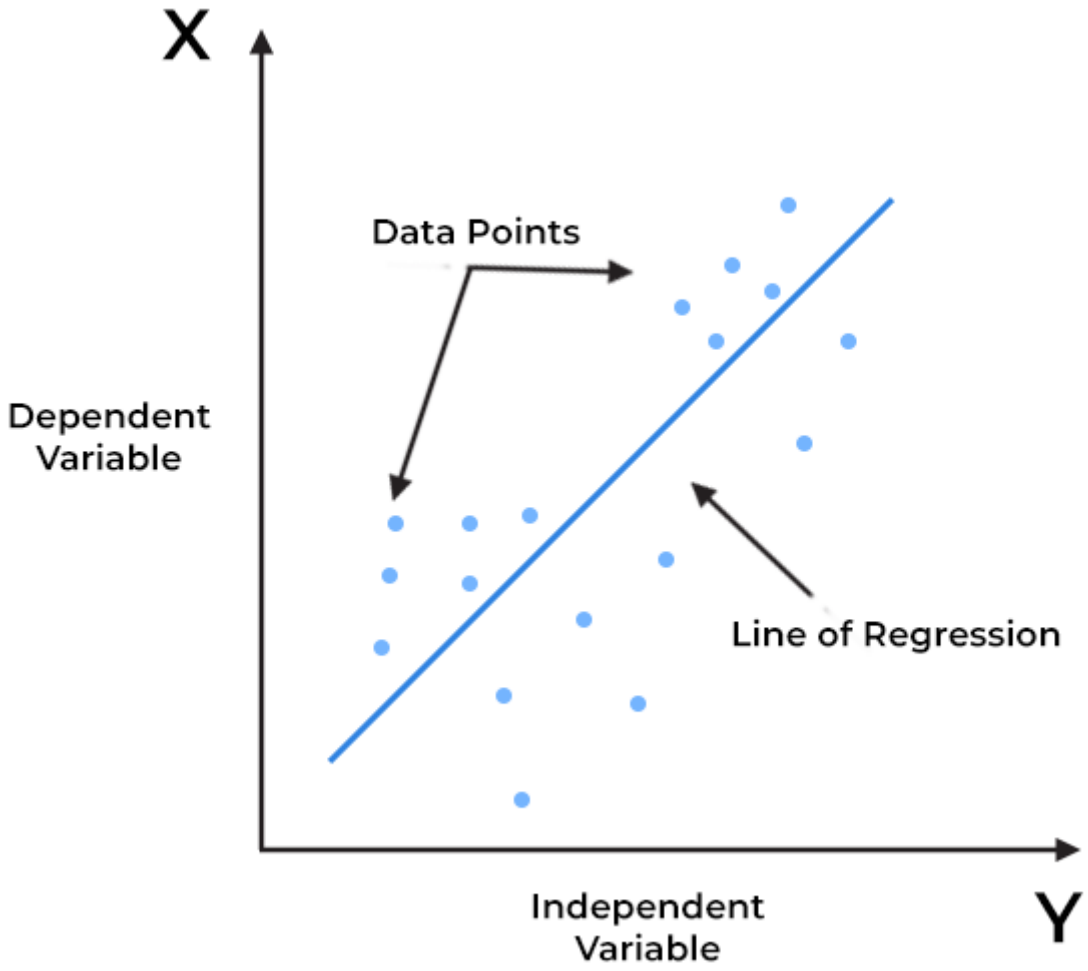
- Apache Kafka
- Home Work

Project

- At least 03 Microservices
    - FastAPI
    - Database
    - User Interface
- All services to start with single `docker compose up` command.

Regression

- X = Independent Variable
- Y = Dependent Variable



Equation of Line

$y = wx + b$

- w & b are weights of Linear Regression
    - w = Slope of Line
    - b = Y Intercept of Line
- Machine Learning find the appropriate values of *w* & *b*
- This can be done using Libraries such as *Tensorflow*.

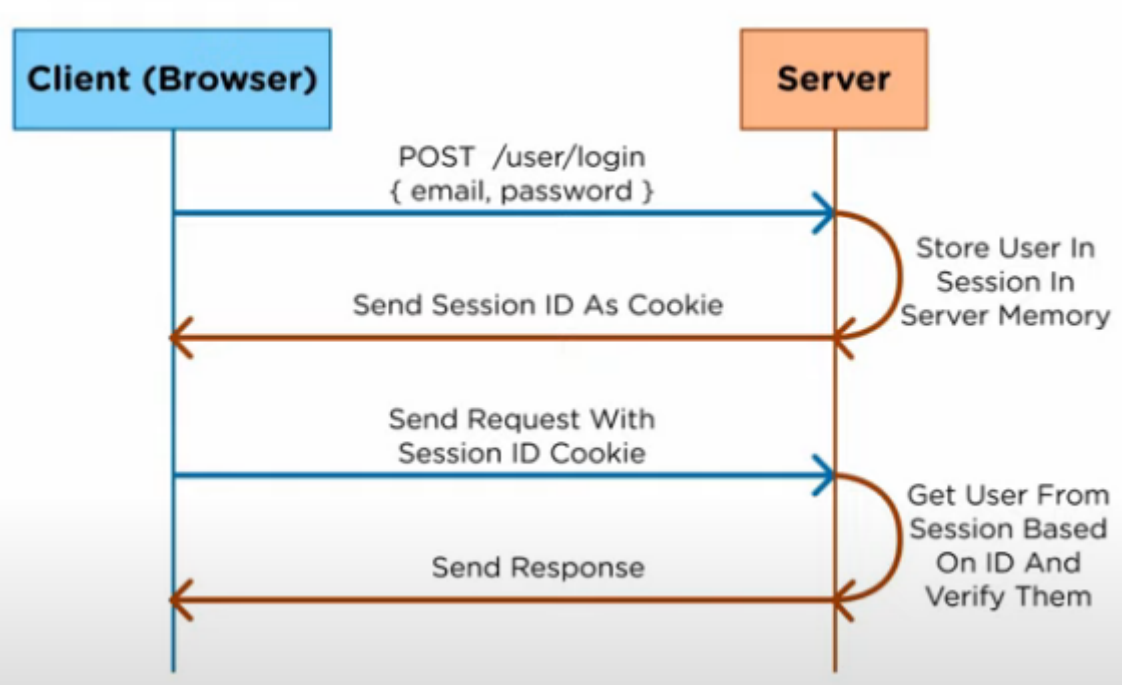
Types of Data

- Qualitative Data
- Quantitative Data
    - Discrete
        - Ordinal (In Order)
        - Nominal
        - Categorical Data
    - Continuous

Authentication & Authorization

Session Based

- Verifying User from Username & Password
- Create **Sessions** on Server using Database / Memory.
- Sends Session ID as Cookie to Client.
- Cookie is stored on Client (Front End)
- Users can interact with Backend after Authentication using Session Cookie.
    - i.e. Authorized

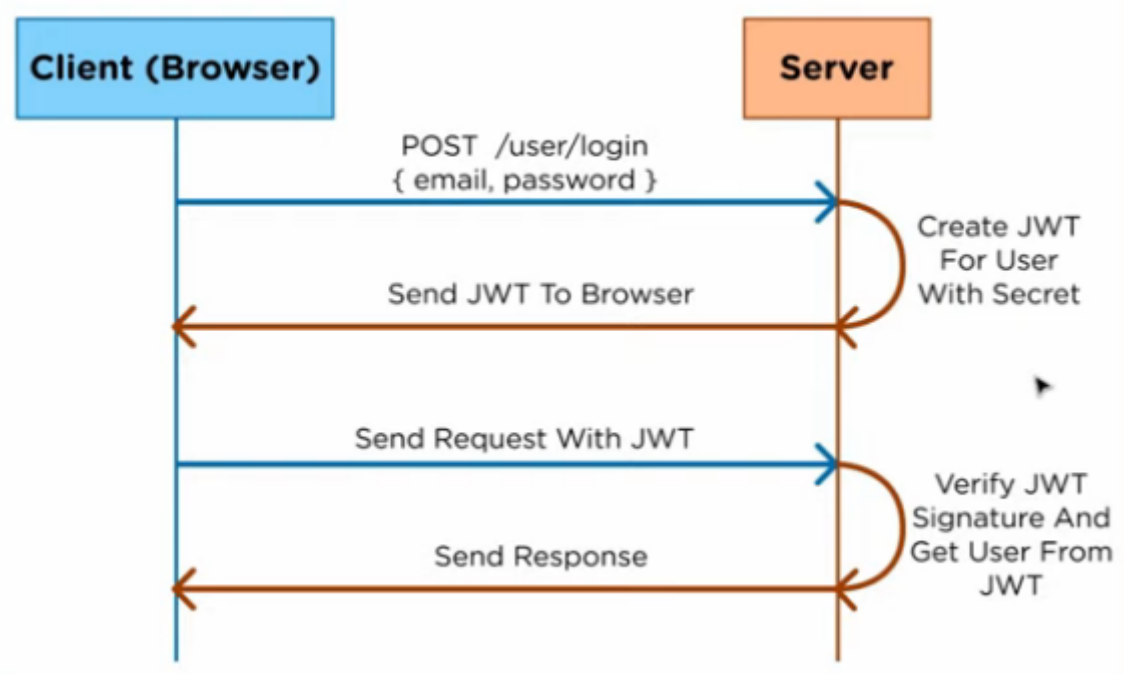


Cons:

- Server has to do lookup in Database to get the correct user. (Will take time for a large System)

Token Based

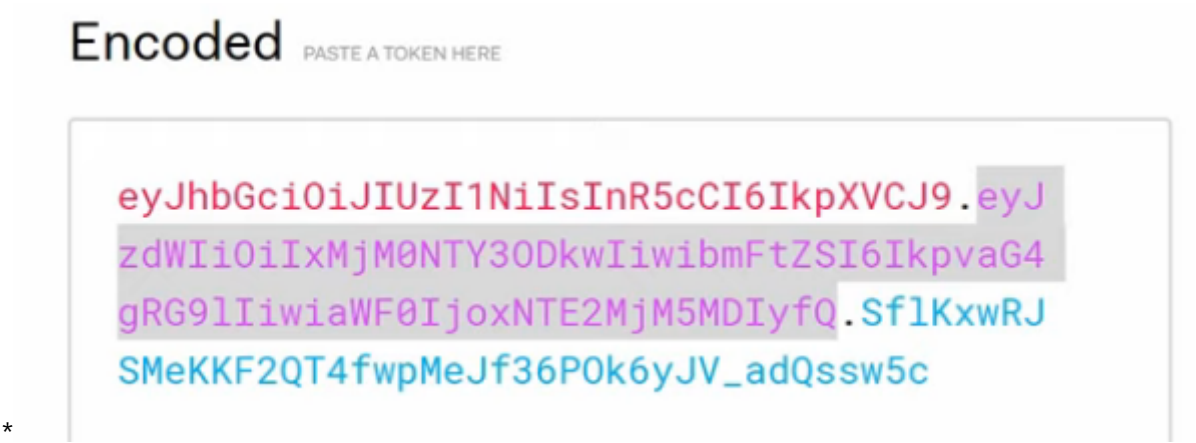
- Verifying User from Username & Password.
- Server creates a **Json Web Token (JWT)** and sign it with its own secret key.
    - Since JWT is signed with secret Key of Server, it will become *invalid* if it is tampered.
    - No information is stored on Server.
    - JWT has all the information about the user.
- JWT is sent back to Client.
    - Client can store JWT in any way i.e. Cookie / Local etc.
- Future requests from Client will contain JWT.
- Server will get User information after De-Coding (De-Serializing) the JWT.



How JWT Works

Encoding

- How to Encode/Decode is defined in Header
- Encode Header and Payload using Header Info.
- Use `HMACSHA256` to create a Verification Signature of (Header + Payload) using Secret Key of server and append it to last (blue) section of JWT.



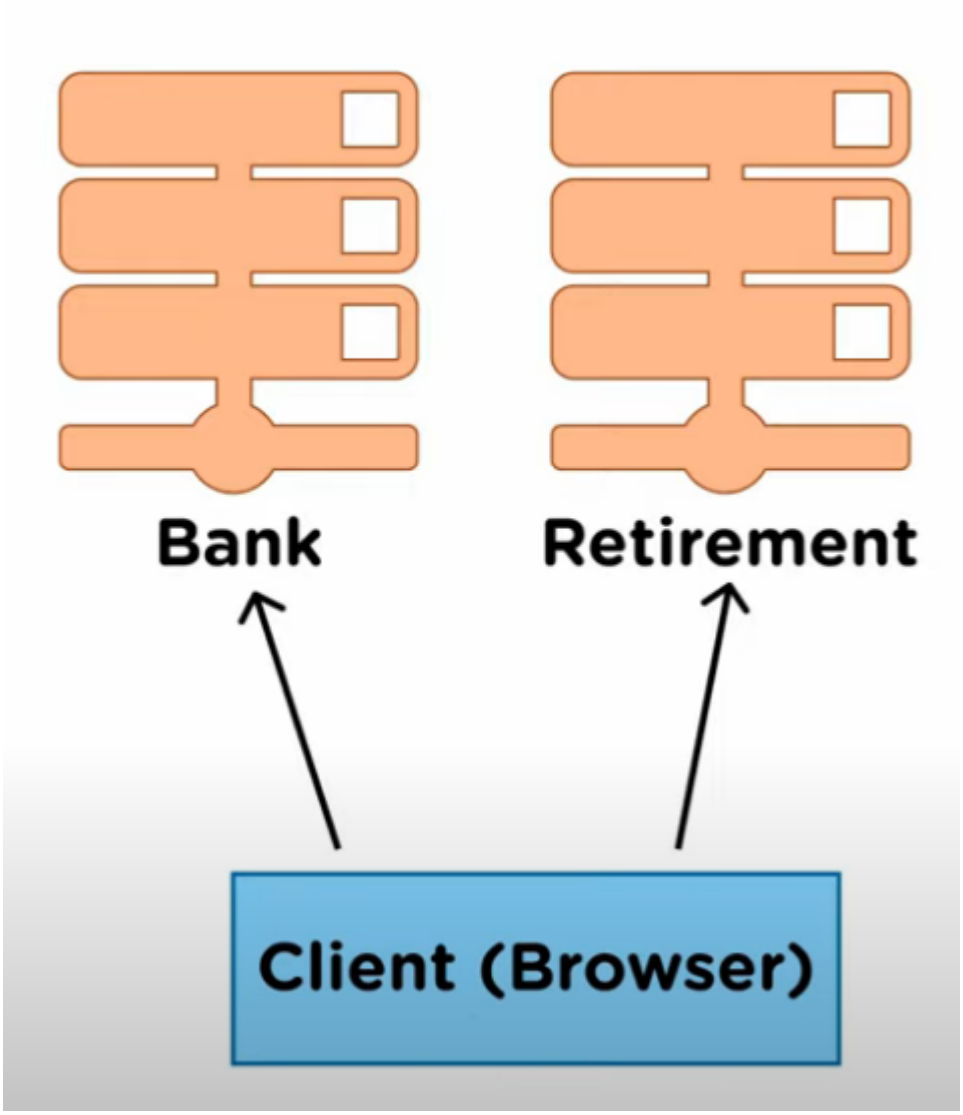
*

Decoding

- Decode Header
- Decode Payload
- Create a Verification Signature (using its Secret Key) and Match it with the signature provided in JWT.
- Secret Key has to be safely stored on Server.

Why Use JWT

- If a Client logs in to One Server, it will get a JWT.
- Client can login in to another Server of same bank with authentication by using the same JWT.
    - Since login information is stored on Client and not on Server.


Bank
Retirement
Client (Browser)

Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Apache Kafka

Apache Kafka is an open-source distributed event streaming platform developed by the Apache Software Foundation. It is designed for building real-time data pipelines and streaming applications. Kafka is capable of handling high throughput and low-latency data transmission, making it a popular choice for organizations needing to process large volumes of data quickly.

Kafka's ability to handle large-scale message streaming with high fault tolerance and scalability makes it a cornerstone technology for many modern data architectures. It is widely used in industries such as finance, retail, healthcare, and technology for applications ranging from log aggregation and monitoring to real-time analytics and event-driven microservices.

Why Use *Event Drive Architecture (EDA)*

Loose Coupling

All services/components of your application were used to run on a single Server. In case of server failure, complete application was out of service (due to Tight Coupling i.e. all services on a single Server). Tight coupling is easy from developer point of view.

Loose Coupling is a concept of having **Microservices** that are combined to create a whole application. In this, if one service is down, other services are not effected and that service is restarted by creating a new Container and deploying it.

Scalability

If one service becomes loaded, we can Horizontally Scale **this service only** by adding containers of this service in the system.

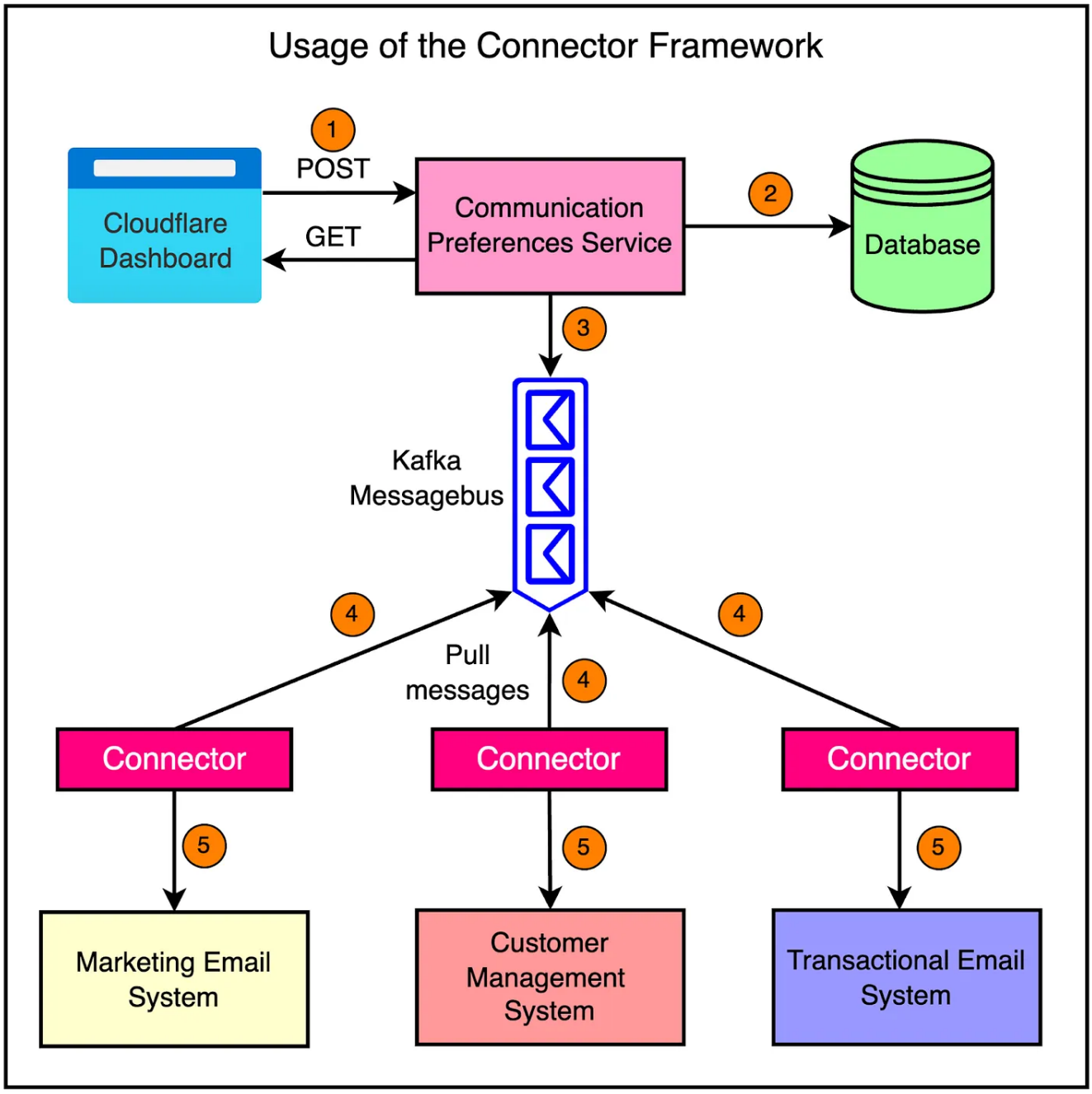
Real Time Processing

Previously an empty request was sent after some time (few seconds) to backend to retrieve if there is any new data in backend database. This will increase load to Server.

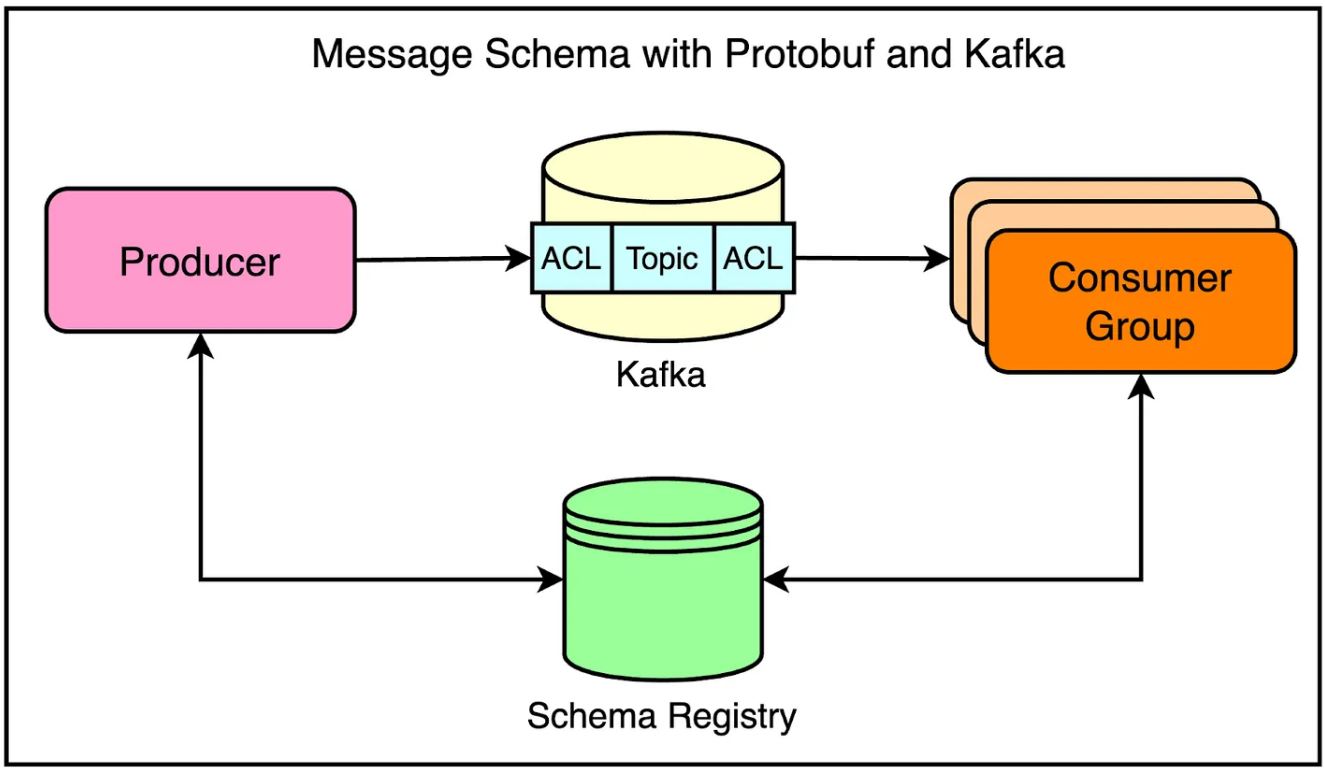
With EDA, if a new message is receive in backend, the server send the Payload to user. (Without being specifically asked.) i.e. **Events can trigger actions as they happen.**

Reducing Load on Database and Server

With Kafka, a message is passed to all its subscribers without involving Server and Database.



Kafka Terminologies



1. **Producers**: These are clients that publish data (messages) to Kafka topics. Producers can send data to one or more topics, and they have the ability to choose which partition within a topic the data is sent to, allowing for load balancing and data organization.

2. **Consumers**: These clients read data from Kafka topics. Consumers can be part of a consumer group, which allows multiple consumers to read from the same topic concurrently, dividing the data load among them.

3. **Access Control Lists (ACLs)**: Access Control Lists (ACLs) are a security feature in Apache Kafka used to control who can access and perform operations on Kafka resources such as topics, consumer groups, and the Kafka cluster itself. ACLs help enforce security by specifying which users or applications (identified by their principals) have permission to perform certain actions.

4. **Topics**: A topic is a category or feed name to which records are sent. Topics in Kafka are partitioned and replicated across multiple nodes, providing scalability and fault tolerance.

5. **Schema Registry**: The Schema Registry is a service for managing and enforcing schemas for data stored in Kafka topics. It provides a centralized repository for schemas and ensures that data producers and consumers adhere to a common schema format. The Schema Registry supports schema evolution, allowing schemas to evolve over time while maintaining compatibility.

6. **Protobuf**: Protocol Buffers, commonly known as Protobuf, is a language-agnostic binary serialization format developed by Google. Protobuf is used to serialize structured data, making it efficient for both storage and communication. It is particularly popular for use in remote procedure calls (RPCs) and for data interchange between systems.

```
syntax = "proto3";

message Person {
    string name = 1;
    int32 id = 2;
    string email = 3;
}
```

BLOG: https://blog.bytebytego.com/p/cloudflares-trillion-message-kafka

Using Kafka

- Install Docker Desktop and Run It
- Pull `Apache Kafka` Docker Image using below command
    - `docker pull apache/kafka:3.7.0`
- Start the Kafka docker container
    - `docker run -p 9092:9092 apache/kafka:3.7.0`
- Switch Terminal to Docker Container inside specific Directory
    - `docker exec -it <container-name> /bin/bash`

```
PS C:\Users\raaid> docker exec -it sharp_blackburn /bin/bash
64fce0f33270:/$ ls
__cacert_entrypoint.sh  lib                     proc                    sys
bin                     lib64                   root                    tmp
dev                     media                   run                     usr
etc                     mnt                     sbin                    var
home                    opt                     srv
64fce0f33270:/$
```

- Switch to directory `opt/kafka/bin` which contains Kafka Commands
    -

```
64fce0f33270:/$ cd opt/kafka/bin
64fce0f33270:/opt/kafka/bin$ ls
connect-distributed.sh              kafka-dump-log.sh                   kafka-server-stop.sh
connect-mirror-maker.sh             kafka-e2e-latency.sh                kafka-storage.sh
connect-plugin-path.sh              kafka-features.sh                   kafka-streams-application-reset.sh
connect-standalone.sh               kafka-get-offsets.sh                kafka-topics.sh
kafka-acls.sh                       kafka-jmx.sh                        kafka-transactions.sh
kafka-broker-api-versions.sh        kafka-leader-election.sh            kafka-verifiable-consumer.sh
kafka-client-metrics.sh             kafka-log-dirs.sh                   kafka-verifiable-producer.sh
kafka-cluster.sh                    kafka-metadata-quorum.sh            trogdor.sh
kafka-configs.sh                    kafka-metadata-shell.sh             windows
kafka-console-consumer.sh           kafka-mirror-maker.sh               zookeeper-security-migration.sh
kafka-console-producer.sh           kafka-producer-perf-test.sh         zookeeper-server-start.sh
kafka-consumer-groups.sh            kafka-reassign-partitions.sh        zookeeper-server-stop.sh
kafka-consumer-perf-test.sh         kafka-replica-verification.sh       zookeeper-shell.sh
kafka-delegation-tokens.sh          kafka-run-class.sh
kafka-delete-records.sh             kafka-server-start.sh
```

- Run following command in bin directory to **Create** a *Kafka Topic*
    - `/opt/kafka/bin/kafka-topics.sh --create --topic <topic name> --bootstrap-server localhost:9092`
    - topic name can be anything

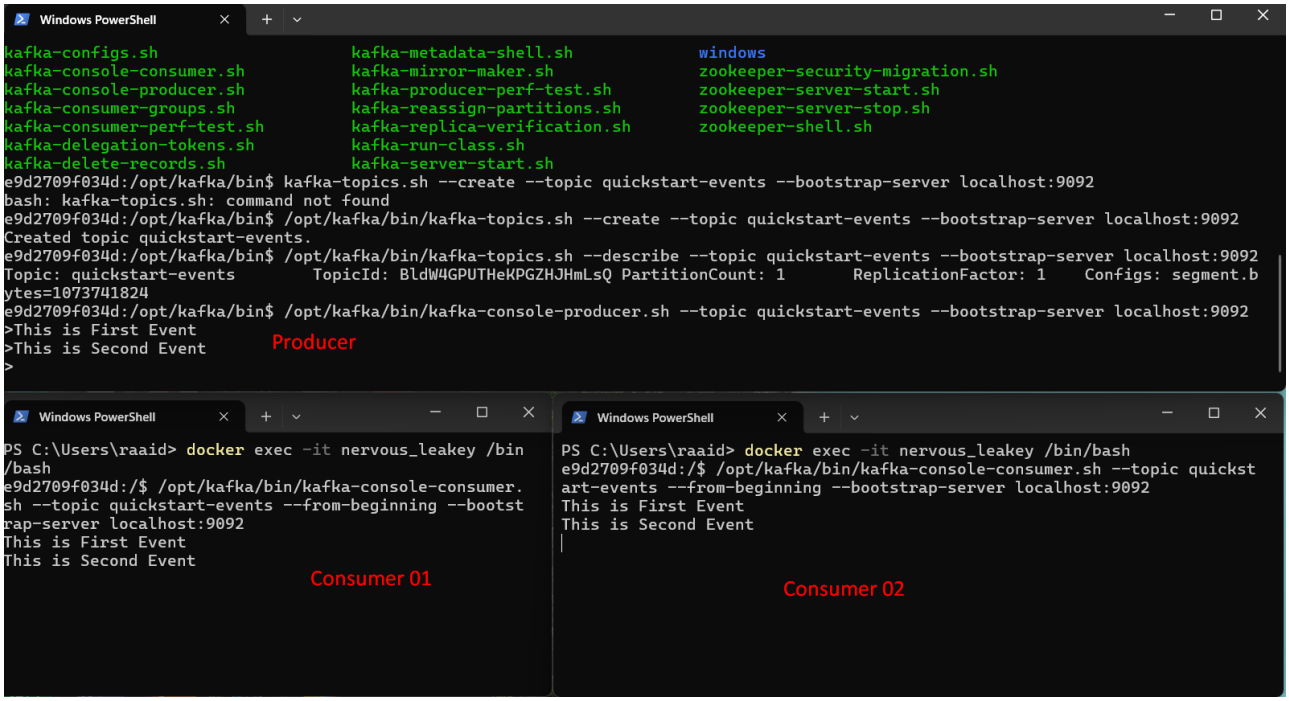
- -bootstrap-server is a server used to run kafka inside internal container. (In our case Docker Container)

```
e9d2709f034d:/opt/kafka/bin$ /opt/kafka/bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhost:9092
Created topic quickstart-events.
e9d2709f034d:/opt/kafka/bin$
```

- Run following command to **Describe** a *Kafka Topic*
  - `/opt/kafka/bin/kafka-topics.sh --describe --topic <topic name> --bootstrap-server localhost:9092`

```
e9d2709f034d:/opt/kafka/bin$ /opt/kafka/bin/kafka-topics.sh --describe --topic quickstart-events --bootstrap-server localhost:9092
Topic: quickstart-events        TopicId: BldW4GPUTHeKPGZHJHmLsQ PartitionCount: 1       ReplicationFactor: 1    Configs: segment.byt
es=1073741824
        Topic: quickstart-events        Partition: 0    Leader: 1       Replicas: 1     Isr: 1
e9d2709f034d:/opt/kafka/bin$
```

- Run following command to **Create** a *Producer*
  - `/opt/kafka/bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhost:9092`
- Run following command to **Create** a *Consumer*
  - `/opt/kafka/bin/kafka-console-consumer.sh --topic quickstart-events --from-beginning --bootstrap-server localhost:9092`
  - You can create multiple consumers. In our case 02
- What ever you will Produce on Producer will be forwarded to all Consumers.



Kafka UI

- Create a Docker Network with name *kafka-net*
  - `docker network create -d bridge kafka-net`

```
PS C:\Users\raaid> docker network create -d bridge kafka-net
d0e766120bedbe15e8f4980816734bc4dc196a1c2dbf2218a1a6641900890e81
```

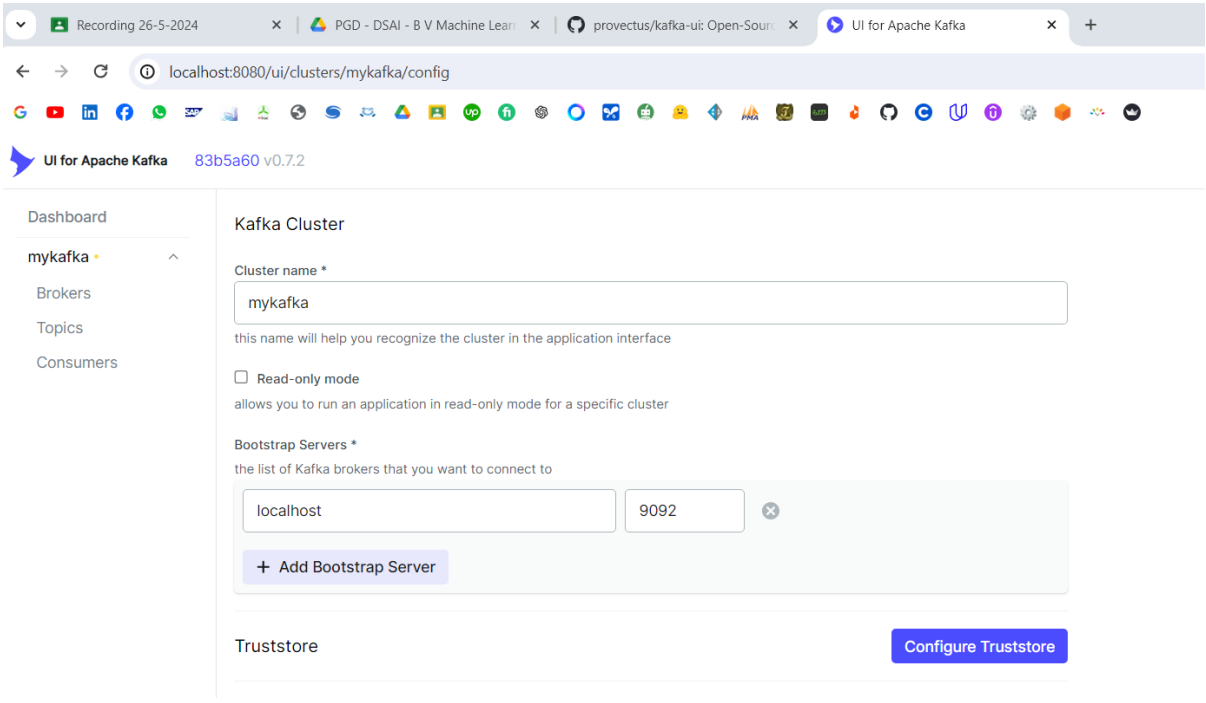
- Show all Docker Networks
    - `docker network ls`

```
PS C:\Users\raaid> docker network ls
NETWORK ID     NAME                      DRIVER    SCOPE
e9112f87877b   bridge                    bridge    local
2f301d438364   host                      host      local
d0e766120bed   kafka-net                 bridge    local
0718242c9597   none                      null      local
f93879644674   todo-app_my-api-net-todo  bridge    local
```

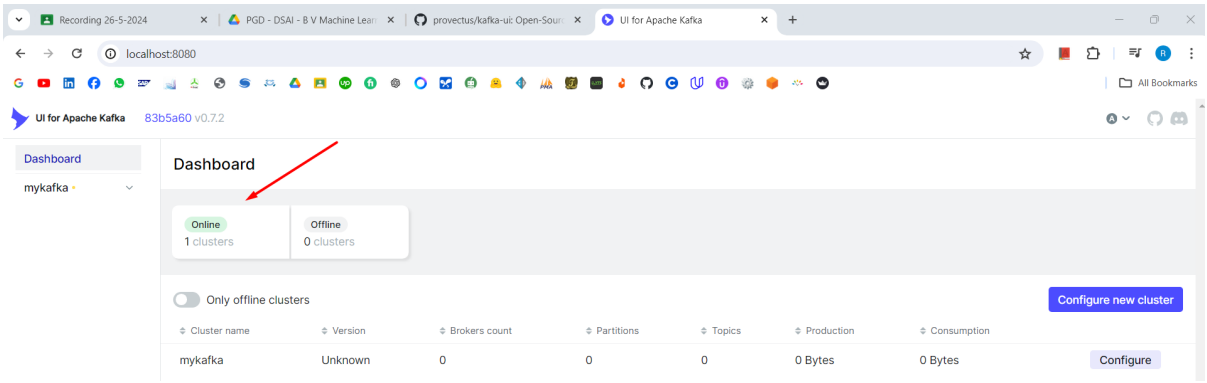
- Stop Containers already running in Docker (Specially if Container is using port 9092)
- Create a New Container named *mykafka* which is using Network *kafka-net*
    - `docker run -p 9092:9092 --network kafka-net --name mykafka apache/kafka:3.7.0`

```
PS C:\Users\raaid> docker run -p 9092:9092 --network kafka-net --name mykafka apache/kafka:3.7.0
===> User
uid=1000(appuser) gid=1000(appuser) groups=1000(appuser)
===> Setting default values of environment variables if not already set.
CLUSTER_ID not set. Setting it to default value: "5L6g3nShT-eMCtK--X86sw"
===> Configuring ...
===> Launching ...
===> Using provided cluster id 5L6g3nShT-eMCtK--X86sw ...
[2024-05-29 10:21:58,466] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2024-05-29 10:21:58,521] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2024-05-29 10:21:58,604] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2024-05-29 10:21:58,606] INFO [ControllerServer id=1] Starting controller (kafka.server.ControllerServer)
```

- Open a new terminal and run following command.
    - `docker run -it -p 8080:8080 --network kafka-net -e DYNAMIC_CONFIG_ENABLED=true provectuslabs/kafka-ui`
    - This will pull ***kafka-ui*** from Docker hub.
    - Go to `localhost:8080` to access *kafka-ui*
    - Enter the Details of Bootstrap Server



- After Successful connection, you will se the online Cluster



- ***This method for learning purpose only, Generally Docker-Compose is used to build Kafka Servers (defined below)***

Kafka-UI (With Docker Compose)

- Create following `docker-compose.yml` file

```
version: '3.8'

x-kafka-common: &kafka-common
  image: 'bitnami/kafka:latest'
  ports:
    - "9092"
  networks:
    - kafka-net
  healthcheck:
    test: "bash -c 'printf \"\" > /dev/tcp/127.0.0.1/9092; exit $$?;'"
    interval: 5s
    timeout: 10s
    retries: 3
    start_period: 30s
  restart: unless-stopped

x-kafka-env-common: &kafka-env-common
  ALLOW_PLAINTEXT_LISTENER: 'yes'
  KAFKA_CFG_AUTO_CREATE_TOPICS_ENABLE: 'true'
  KAFKA_CFG_CONTROLLER_QUORUM_VOTERS: 0@kafka-0:9093,1@kafka-1:9093
  KAFKA_KRAFT_CLUSTER_ID: abcdefghijklmnopqrstuv
  KAFKA_CFG_PROCESS_ROLES: controller,broker
  KAFKA_CFG_CONTROLLER_LISTENER_NAMES: CONTROLLER
  KAFKA_CFG_LISTENERS: PLAINTEXT://:9092,CONTROLLER://:9093
  EXTRA_ARGS: "-Xms128m -Xmx256m"

services:

  kafka-0:
    <<: *kafka-common
```

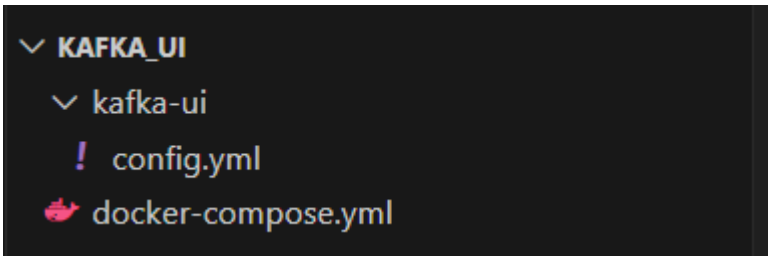
```
    environment:
      <<: *kafka-env-common
      KAFKA_CFG_NODE_ID: 0
    volumes:
      - kafka_0_data:/bitnami/kafka

  kafka-1:
    <<: *kafka-common
    environment:
      <<: *kafka-env-common
      KAFKA_CFG_NODE_ID: 1
    volumes:
      - kafka_1_data:/bitnami/kafka

  kafka_ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    volumes:
      - ./kafka-ui/config.yml:/etc/kafkaui/dynamic_config.yaml
    environment:
      DYNAMIC_CONFIG_ENABLED: 'true'
    depends_on:
      - kafka-0
      - kafka-1
    networks:
      - kafka-net
    ports:
      - '8080:8080'
    healthcheck:
      test: wget --no-verbose --tries=1 --spider localhost:8080 || exit 1
      interval: 5s
      timeout: 10s
      retries: 3
      start_period: 30s

networks:
  kafka-net:

volumes:
  kafka_0_data:
    driver: local
  kafka_1_data:
    driver: local
```

- Create a folder name `kafka-ui` in same Directory and create a `config.yml` file.

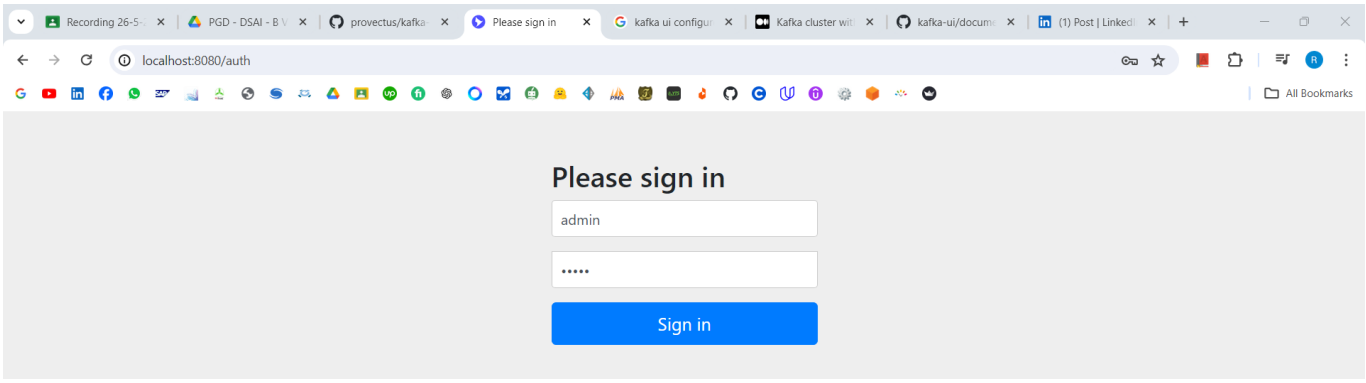
```
auth:
  type: LOGIN_FORM

spring:
  security:
    user:
```

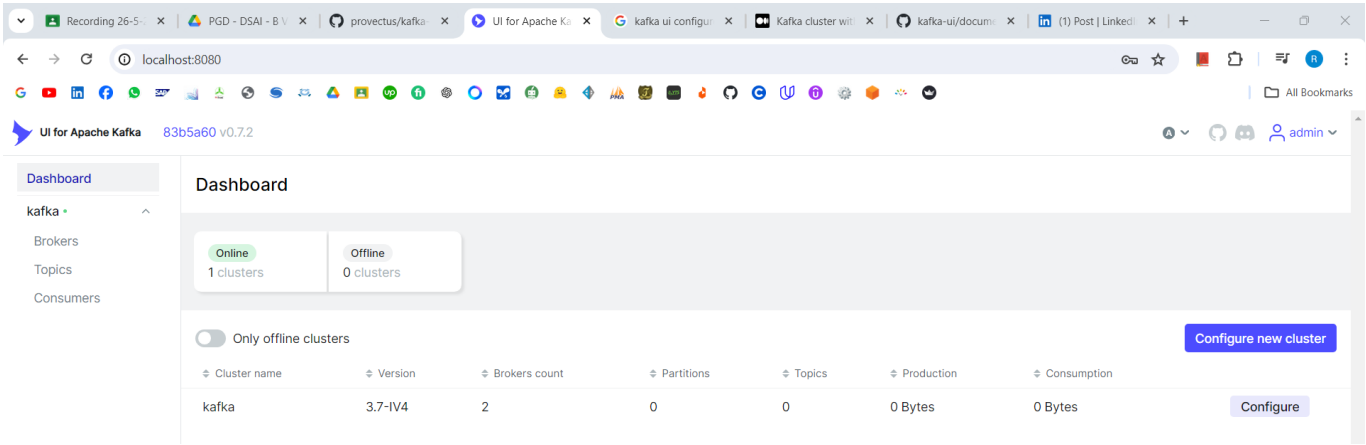
```
      name: admin
      password: admin

kafka:
  clusters:
    - bootstrapServers: kafka-0:9092,kafka-1:9092
      name: kafka
```

- Below if the file structure



- Run all Containers using `docker compose up` command.
- Go to `http://localhost:8080/` and use credentials from `config.yml` file (i.e. admin admin)





Using Kafka in Applications

- You can use Python Library `AIOKafka` to create Producers and Consumers from within Application Code.

Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Kong

Kong is an open-source API Gateway that acts as a **Middleware** between your clients and your backend services. Here's an easy way to understand it:

What is an API Gateway?

An API Gateway is like a receptionist at a hotel. When you arrive, the receptionist helps you check in, provides you with information, and directs you to your room. Similarly, an API Gateway sits in front of your backend services and manages all incoming requests, ensuring they reach the correct service.

Key Features of Kong API Gateway:

1. **Routing**: Kong directs incoming API requests to the appropriate service. For example, if a request comes in for `/users`, Kong knows to route it to the user service.
2. **Security**: It can handle authentication and authorization, ensuring that only authorized users can access certain services. Think of it as the bouncer at a club checking IDs before letting people in.
3. **Rate Limiting**: Kong can limit the number of requests a user can make in a certain time period, preventing abuse. This is like a store manager ensuring that one customer doesn't buy all the stock at once.
4. **Logging and Monitoring**: Kong can keep track of all the requests and responses, providing insights into how your services are being used. It's like having a security camera that records all activities.
5. **Transformation**: It can modify requests and responses on the fly. For example, if a client needs data in a different format, Kong can transform the response before sending it back.

How Kong Works:

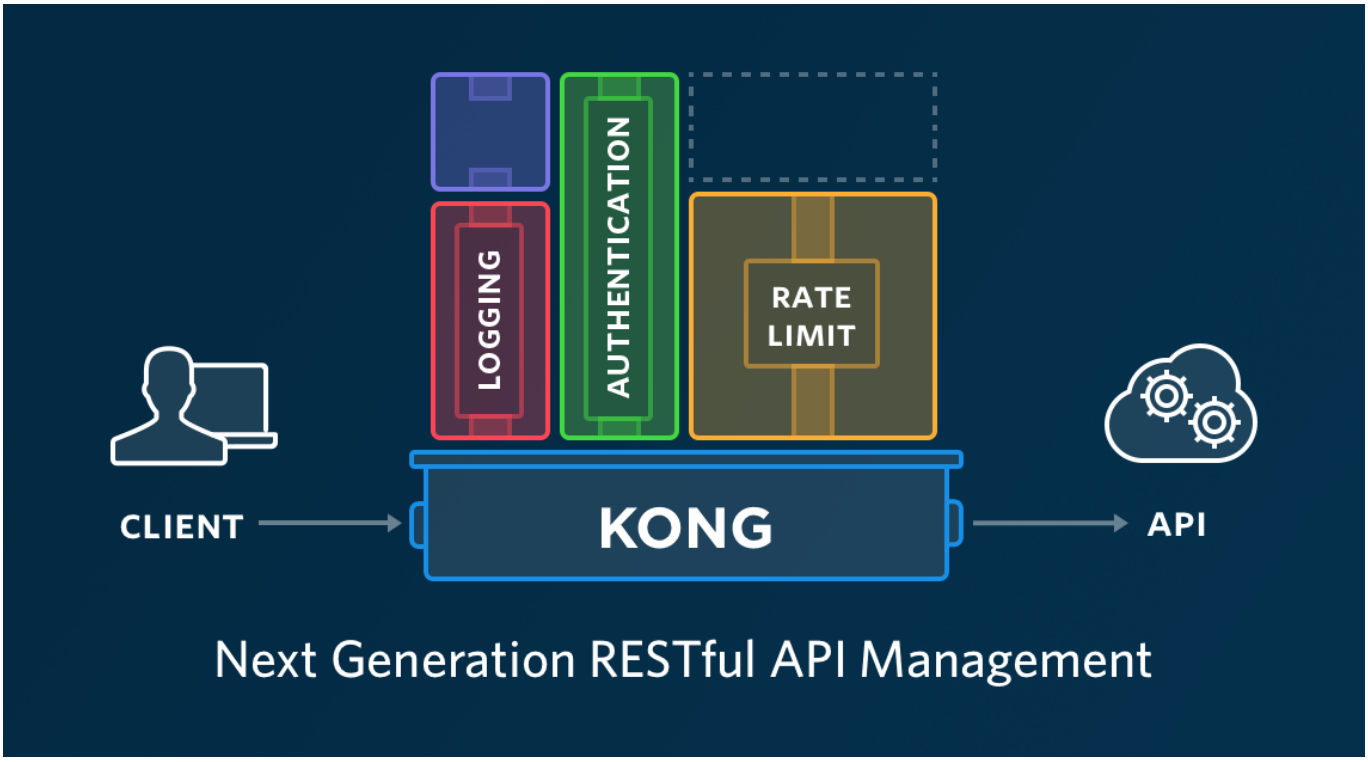
1. **Installation**: Kong is installed and runs as a server.
2. **Configuration**: You configure Kong to know about your backend services (called APIs in Kong).
3. **Plugins**: You can add plugins to Kong to provide extra features like authentication, rate limiting, and logging.
4. **Clients**: When clients (like a mobile app or web app) make requests, they go to Kong first.
5. **Proxying**: Kong forwards these requests to the appropriate backend services, and then it forwards the responses back to the clients.

Example:

Imagine you have an online store with services for users, products, and orders. Without Kong, each client would need to know the details of each service. With Kong, clients only need to interact with Kong, which

handles routing, security, and other concerns, making it easier and more efficient for everyone involved.

In summary, Kong simplifies the management of API traffic, enhances security, and provides additional features to improve the performance and reliability of your services.



Configuring **Kong** and **Konga-UI**

- Create Following `compose.yml` file

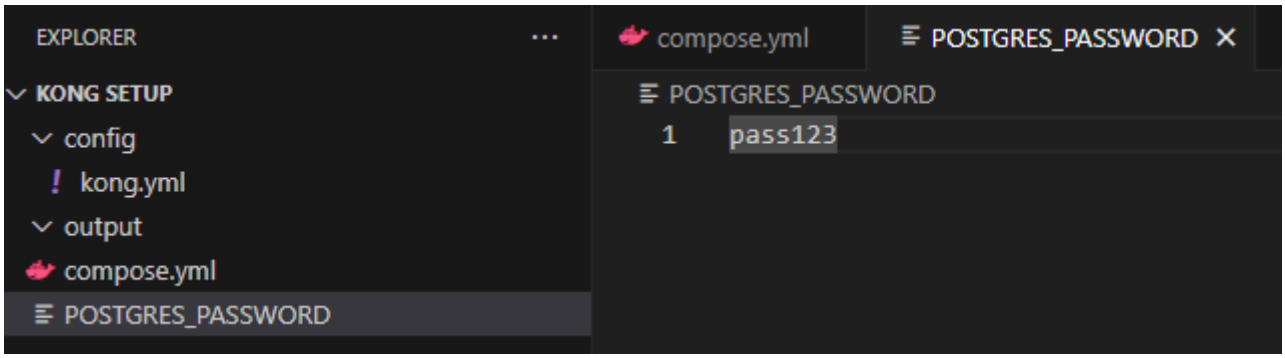
```
version: '3.7'

volumes:
  kong_data: {}
  kong_prefix_vol:
    driver_opts:
     type: tmpfs
     device: tmpfs
  kong_tmp_vol:
    driver_opts:
     type: tmpfs
     device: tmpfs

networks:
  kong-net:
    external: false

services:
  kong-migrations:
    image: "${KONG_DOCKER_TAG:-kong:latest}"
    command: kong migrations bootstrap
```

```
    depends_on:
      - db
    environment:
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-kong}
      KONG_PG_HOST: db
      KONG_PG_USER: ${KONG_PG_USER:-kong}
      KONG_PG_PASSWORD_FILE: /run/secrets/kong_postgres_password
    secrets:
      - kong_postgres_password
    networks:
      - kong-net
    restart: on-failure
    deploy:
      restart_policy:
        condition: on-failure

  kong-migrations-up:
    image: "${KONG_DOCKER_TAG:-kong:latest}"
    command: kong migrations up && kong migrations finish
    depends_on:
      - db
    environment:
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-kong}
      KONG_PG_HOST: db
      KONG_PG_USER: ${KONG_PG_USER:-kong}
      KONG_PG_PASSWORD_FILE: /run/secrets/kong_postgres_password
    secrets:
      - kong_postgres_password
    networks:
      - kong-net
    restart: on-failure
    deploy:
      restart_policy:
        condition: on-failure

  kong:
    image: "${KONG_DOCKER_TAG:-kong:latest}"
    user: "${KONG_USER:-kong}"
    depends_on:
      - db
    environment:
      KONG_ADMIN_ACCESS_LOG: /dev/stdout
      KONG_ADMIN_ERROR_LOG: /dev/stderr
      KONG_PROXY_LISTEN: "${KONG_PROXY_LISTEN:-0.0.0.0:8000}"
      KONG_ADMIN_LISTEN: "${KONG_ADMIN_LISTEN:-0.0.0.0:8001}"
      KONG_CASSANDRA_CONTACT_POINTS: db
      KONG_DATABASE: postgres
      KONG_PG_DATABASE: ${KONG_PG_DATABASE:-kong}
      KONG_PG_HOST: db
      KONG_PG_USER: ${KONG_PG_USER:-kong}
      KONG_PROXY_ACCESS_LOG: /dev/stdout
      KONG_PROXY_ERROR_LOG: /dev/stderr
```

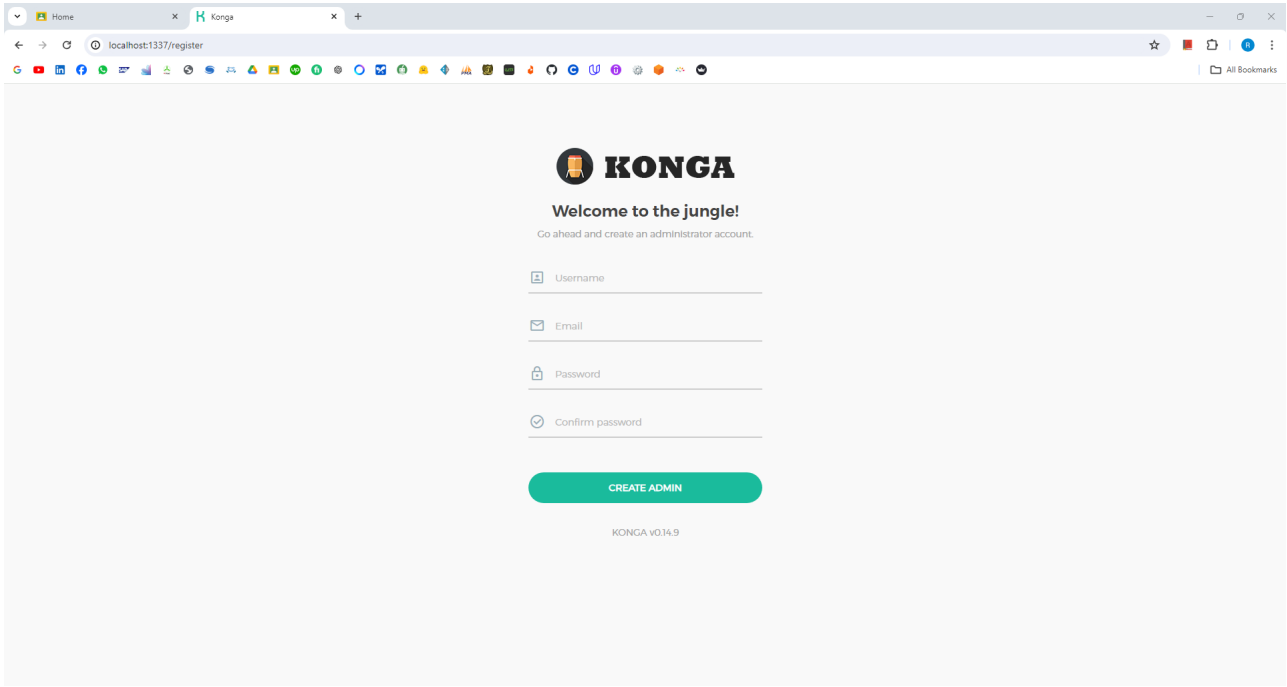
```
      KONG_PG_PASSWORD_FILE: /run/secrets/kong_postgres_password
      KONG_PREFIX: ${KONG_PREFIX:-/var/run/kong}
    secrets:
      - kong_postgres_password
    networks:
      - kong-net
    ports:
      # The following two environment variables default to an insecure value
(0.0.0.0)
      # according to the CIS Security test.
      - "${KONG_INBOUND_PROXY_LISTEN:-0.0.0.0}:8000:8000/tcp"
      - "${KONG_INBOUND_SSL_PROXY_LISTEN:-0.0.0.0}:8443:8443/tcp"
      # Making them mandatory but undefined, like so would be backwards-breaking:
      # - "${KONG_INBOUND_PROXY_LISTEN?Missing inbound proxy host}:8000:8000/tcp"
      # - "${KONG_INBOUND_SSL_PROXY_LISTEN?Missing inbound proxy ssl
host}:8443:8443/tcp"
      # Alternative is deactivating check 5.13 in the security bench, if we
consider Kong's own config to be enough security here

      - "127.0.0.1:8001:8001/tcp"
      - "127.0.0.1:8444:8444/tcp"
    healthcheck:
      test: ["CMD", "kong", "health"]
      interval: 10s
      timeout: 10s
      retries: 10
    restart: on-failure:5
    read_only: true
    volumes:
      - kong_prefix_vol:${KONG_PREFIX:-/var/run/kong}
      - kong_tmp_vol:/tmp
    deploy:
      restart_policy:
        delay: 50s
        condition: on-failure
        max_attempts: 5
        window: 10s
      resources:
        limits:
          cpus: ${KONG_CPU_LIMIT:-2}
          memory: ${KONG_MEMORY_LIMIT:-2g}
    security_opt:
      - no-new-privileges

  db:
    image: postgres:9.5
    environment:
      POSTGRES_DB: ${KONG_PG_DATABASE:-kong}
      POSTGRES_USER: ${KONG_PG_USER:-kong}
      POSTGRES_PASSWORD_FILE: /run/secrets/kong_postgres_password
    secrets:
      - kong_postgres_password
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "${KONG_PG_USER:-kong}"]
```

```
      interval: 30s
      timeout: 30s
      retries: 3
    restart: on-failure
    deploy:
      restart_policy:
        condition: on-failure
    stdin_open: true
    tty: true
    networks:
      - kong-net
    volumes:
      - kong_data:/var/lib/postgresql/data


  konga-prepare:
    image: pantsel/konga:latest
    command: "-c prepare -a postgres -u postgresql://kong:pass123@db:5432/konga"
    networks:
      - kong-net
    restart: on-failure
    secrets:
      - kong_postgres_password
    depends_on:
      - db
    volumes:
      - kong_data:/var/lib/postgresql/data

########################################
  # Konga: Kong GUI
########################################

  konga:
    image: pantsel/konga:latest
    restart: always
    networks:
        - kong-net
    environment:
      DB_ADAPTER: postgres
      DB_URI: postgresql://kong:pass123@db:5432/konga
      NODE_ENV: production
    depends_on:
      - db
    ports:
      - "1337:1337"

secrets:
  kong_postgres_password:
    file: ./POSTGRES_PASSWORD
```

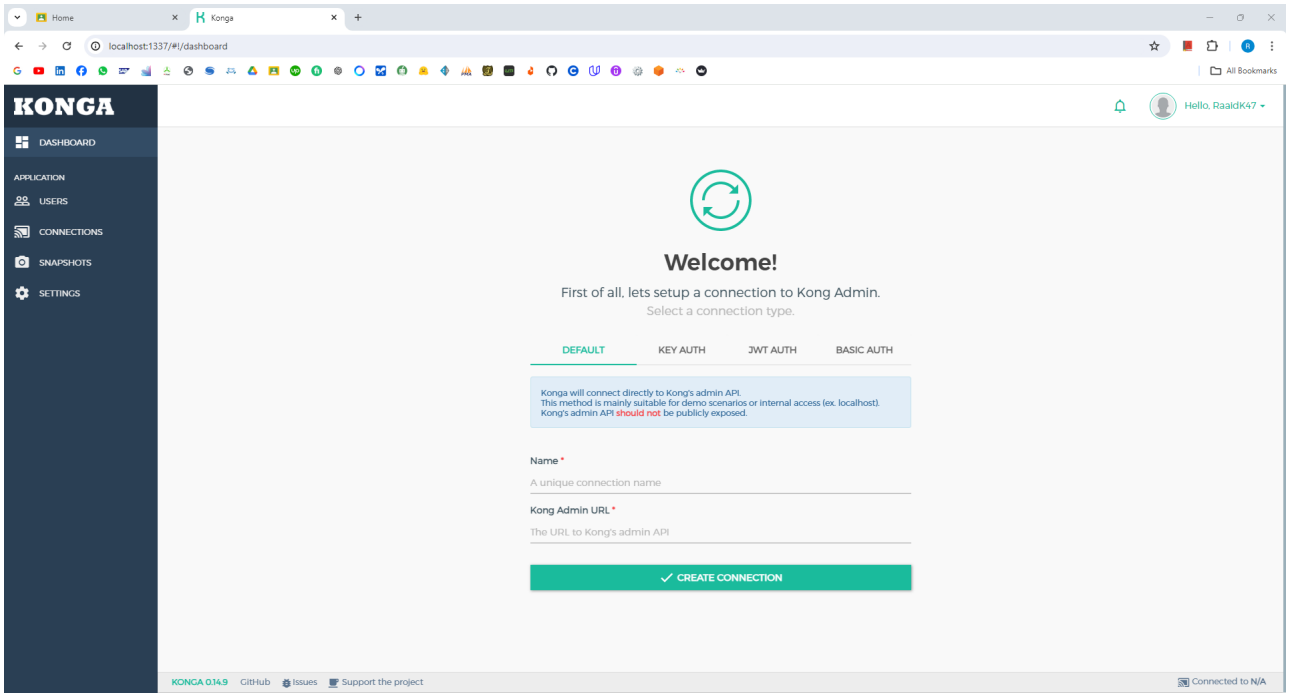
- Create a `POSTGRES_PASSWORD` file and place Password in it

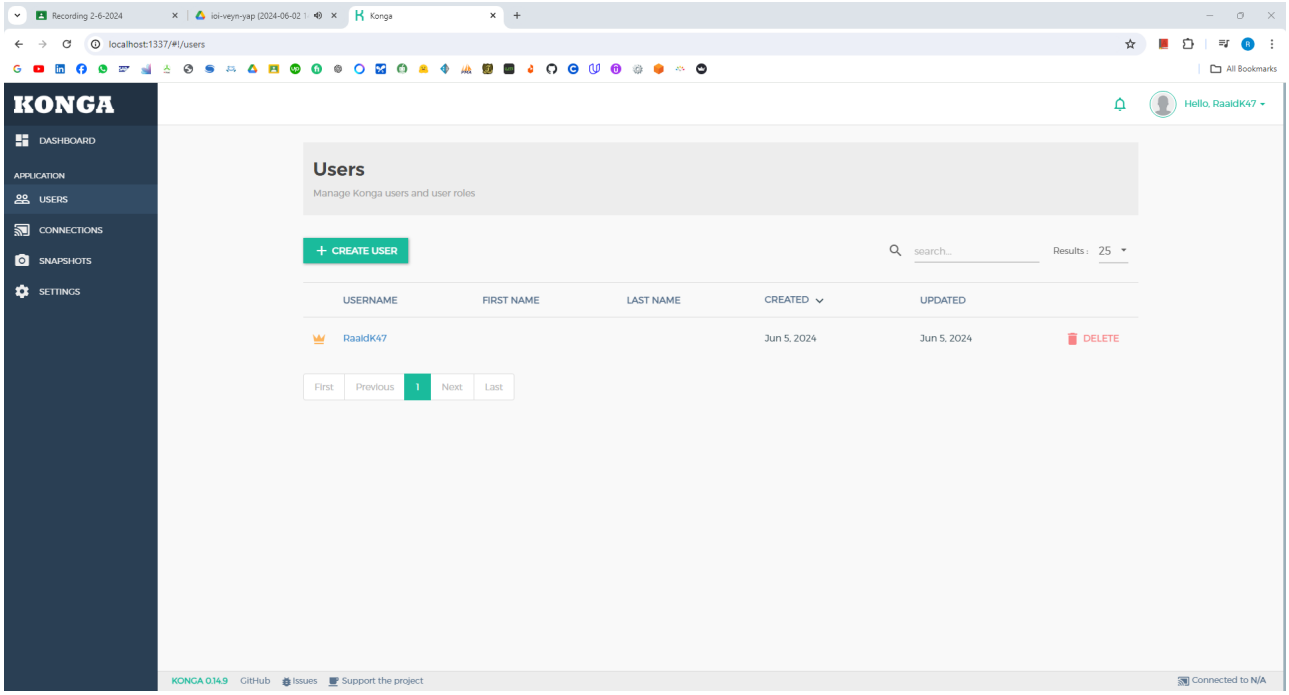


- Run `docker compose up` command to start **Kong** and **Konga-UI**
- Go to `http://localhost:1337/` to access UI



- Register with *Username* and *Password* for an Admin Account.
- Log-In to Admin Account to access the UI.





PyCaret

PyCaret is an open-source, low-code machine learning library in Python that simplifies the process of building, deploying, and maintaining machine learning models. Here's a straightforward explanation:

Key Features of PyCaret:

1. **Low-Code**: It requires very little coding to create machine learning models. This makes it accessible even for those who are not experts in programming or machine learning.
2. **Automates Processes**: PyCaret automates many tasks that would typically require multiple lines of code, such as data preprocessing, model training, hyperparameter tuning, and model evaluation.
3. **End-to-End Machine Learning**: PyCaret covers the entire machine learning lifecycle from data preparation to model deployment, all within a single, cohesive framework.

How It Works:

1. **Data Preparation**: PyCaret helps clean and prepare data for modeling. It can handle missing values, encode categorical variables, and scale numerical data with just a few lines of code.
2. **Model Training**: You can quickly train and compare multiple machine learning models using PyCaret's simple and intuitive functions. It supports a wide range of algorithms out-of-the-box.
3. **Model Evaluation**: PyCaret provides easy-to-understand metrics and visualizations to evaluate model performance, allowing you to select the best model for your needs.
4. **Hyperparameter Tuning**: It offers automated hyperparameter tuning to optimize model performance without manual intervention.
5. **Model Deployment**: PyCaret simplifies the process of deploying machine learning models into production, making it easier to integrate them into applications.

Example:

Here’s a basic example of how you might use PyCaret:

```
# Importing the regression module
from pycaret.regression import *

# Loading a dataset
data = get_data('insurance')

# Setting up the environment in PyCaret
reg = setup(data, target='charges')

# Comparing different models
best_model = compare_models()

# Finalizing the best model
final_model = finalize_model(best_model)

# Making predictions on new data
predictions = predict_model(final_model, data=new_data)
```

In this example, PyCaret helps load the data, set up the machine learning environment, compare different models to find the best one, finalize the model, and make predictions—all with just a few lines of code.

Benefits:

- **User-Friendly**: Ideal for beginners and non-experts in machine learning.
- **Efficiency**: Saves time by automating repetitive tasks.
- **Versatility**: Can be used for a variety of machine learning tasks, including classification, regression, clustering, and anomaly detection.

Overall, PyCaret is a powerful tool that democratizes machine learning by making it more accessible and less time-consuming.

Creating a PyCaret Project with Poetry

- Create a New `Poetry` Project
- Change Directory to Poetry Project Folder
- Open `Poetry Shell`
    - If a `.venv` folder is created, you can delete this folder.
    - OR - You make sure that this folder is not copied into Docker Container
- Set the `python` property to `">=3.9,<3.13"` in `.toml` file.

```
8   [tool.poetry.dependencies]
9   python = ">=3.9,<3.13"
```

- Install PyCaret with command `poetry add pycaret`

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L13-14 - Kong\Code\PyCaret\pycaret_project> poetry add pycaret
Using version ^3.3.2 for pycaret

Updating dependencies
Resolving dependencies... Downloading https://files.pythonhosted.org/packages/33/1a/1142c3d27dd2134157f9d6cf1fed5a566b2f
Resolving dependencies... Downloading https://files.pythonhosted.org/packages/33/1a/1142c3d27dd2134157f9d6cf1fed5a566b2f
Resolving dependencies... Downloading https://files.pythonhosted.org/packages/33/1a/1142c3d27dd2134157f9d6cf1fed5a566b2f
Resolving dependencies... (40.2s)

Package operations: 95 installs, 1 update, 0 removals

  - Installing six (1.16.0)
  - Installing attrs (23.2.0)
  - Installing colorama (0.4.6)
  - Installing markupsafe (2.1.5)
  - Installing numpy (1.26.4)
  - Installing python-dateutil (2.9.0.post0)
  - Installing pytz (2024.1)
  - Installing rpds-py (0.18.1)
  - Installing tzdata (2024.1)
  - Installing asttokens (2.4.1)
  - Installing joblib (1.3.2)
  - Installing certifi (2024.6.2)
  - Installing charset-normalizer (3.3.2)
```

- You may get following error.

```
  RuntimeError

  Unable to find installation candidates for kaleido (0.2.1.post1)

  at ~\pipx\venvs\poetry\Lib\site-packages\poetry\installation\chooser.py:74 in choose_for
       70
       71                  links.append(link)
       72
       73          if not links:
    →  74              raise RuntimeError(f"Unable to find installation candidates for {package}")
       75
       76          # Get the best link
       77          chosen = max(links, key=lambda link: self._sort_key(package, link))
       78

Cannot install kaleido.
```

- Run command `poetry add kaleido==0.2.1` to Install required Dependency. It may take some time.

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L13-14 - Kong\Code\PyCaret\pycaret_project> poetry add kaleido==0.2.1

Updating dependencies
Resolving dependencies... (0.7s)

Package operations: 1 install, 0 updates, 0 removals

  - Installing kaleido (0.2.1)

Writing lock file
```

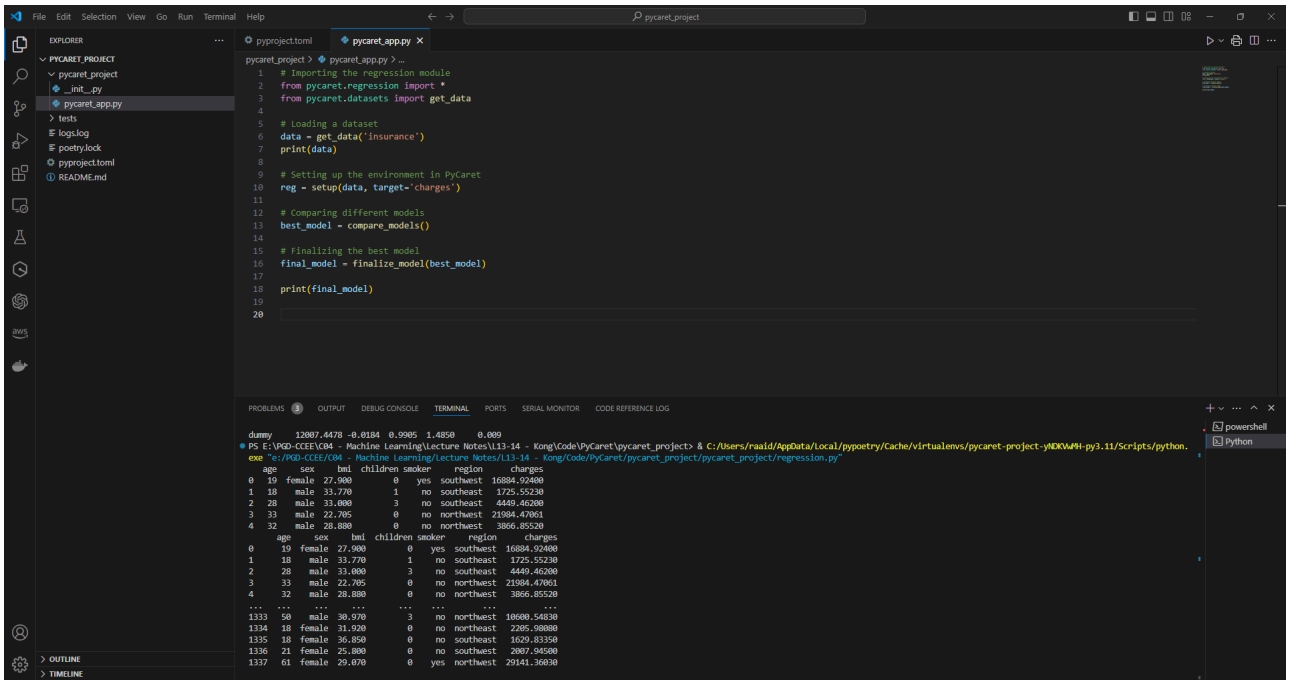
- Run command `poetry add pycaret` again.

```
PS E:\PGD-CCEE\C04 - Machine Learning\Lecture Notes\L13-14 - Kong\Code\PyCaret\pycaret_project> poetry add pycaret
Using version ^3.3.2 for pycaret

Updating dependencies
Resolving dependencies... (8.2s)

Package operations: 1 install, 0 updates, 0 removals

  - Installing pycaret (3.3.2)

Writing lock file
```

- This will successfully install PyCaret.

```
[tool.poetry.dependencies]
python = ">=3.9,<3.13"
kaleido = "0.2.1"
pycaret = "^3.3.2"
```

- You can now write Python Code in Poetry Environment by Importing PyCaret Modules.



Notes Taken By

- Muhammad Raaid Khan
- Data Science and AI (Batch - 05)
- NED - CCEE

Machine Learning with PyCaret

You have to choose a Machine Learning Algorithm based on the output of this Algorithm and its alignment with your Problem Case.

- **Binary Classification**
- **MultiClass Classification**
- **Regression**: Predicting a Numerical Value based on Data
- **Time Series Forecasting**: Predicting a Value based on Time
- **Clustering**: Finding similar Groups from Existing Data, *You do not create Groups*
  - These Groups can be used for Targeted Marketing etc.
- **Anomaly Detection**: To detect a Data Point that is odd one in the data set (Finding Outliers)

Classification

Loading Dataset

```
# load sample dataset
from pycaret.datasets import get_data
data = get_data('diabetes')

# Here `data` is a pandas DataFrame.
```

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Setting up PyCaret

```
# Setting Up Pycaret
from pycaret.classification import *
s = setup(data, target = 'Class variable', session_id = 123)

# Here `target` is the Variable which we want to predict.
# If you do not specify `target` then it will take the last column as target.
# `session_id` is used to ensure that the same model is used for the same data.
# If you do not specify `session_id` then a random number is generated.
# You can perform different experiments with different session_id.
```

	Description	Value
0	Session id	123
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)

	Description	Value
7	Numeric features	8
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	1d1a

Comparing Models

```
best = compare_models()
```

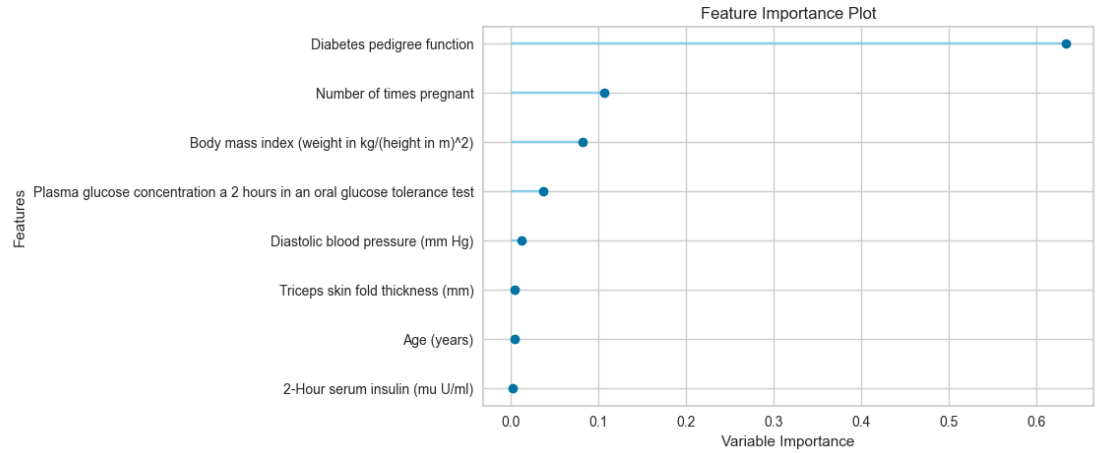
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	0.7689	0.8047	0.5602	0.7208	0.6279	0.4641	0.4736	0.2470
ridge	Ridge Classifier	0.7670	0.8060	0.5497	0.7235	0.6221	0.4581	0.4690	0.0040
lda	Linear Discriminant Analysis	0.7670	0.8055	0.5550	0.7202	0.6243	0.4594	0.4695	0.0050
rf	Random Forest Classifier	0.7485	0.7911	0.5284	0.6811	0.5924	0.4150	0.4238	0.0230
nb	Naive Bayes	0.7427	0.7955	0.5702	0.6543	0.6043	0.4156	0.4215	0.0040
gbc	Gradient Boosting Classifier	0.7373	0.7914	0.5550	0.6445	0.5931	0.4013	0.4059	0.0180
ada	Ada Boost Classifier	0.7372	0.7799	0.5275	0.6585	0.5796	0.3926	0.4017	0.0130
et	Extra Trees Classifier	0.7299	0.7788	0.4965	0.6516	0.5596	0.3706	0.3802	0.0210
qda	Quadratic Discriminant Analysis	0.7282	0.7894	0.5281	0.6558	0.5736	0.3785	0.3910	0.0050
lightgbm	Light Gradient Boosting Machine	0.7133	0.7645	0.5398	0.6036	0.5650	0.3534	0.3580	0.0830
knn	K Neighbors Classifier	0.7001	0.7164	0.5020	0.5982	0.5413	0.3209	0.3271	0.1710
dt	Decision Tree Classifier	0.6928	0.6512	0.5137	0.5636	0.5328	0.3070	0.3098	0.0040
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0040
svm	SVM - Linear Kernel	0.5954	0.5914	0.3395	0.4090	0.2671	0.0720	0.0912	0.0050

Evaluating Best Model

```
evaluate_model(best)
```

Plot Type:

Pipeline Plot	Hyperparameters	AUC	Confusion Matrix	Threshold	Precision Recall	Prediction Error	Class Report	Feature Selection
Learning Curve	Manifold Learning	Calibration Curve	Validation Curve	Dimensions	Feature Importance	Feature Importance...	Decision Boundary	Lift Chart
Gain Chart	Decision Tree	KS Statistic Plot						




```
predict_model(best)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.7576	0.8568	0.5309	0.7049	0.6056	0.4356	0.4447

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable	prediction_label	prediction_score
552	6	114	88	0	0	27.799999	0.247	66	0	0	0.8036
438	1	97	70	15	0	18.200001	0.147	21	0	0	0.9648
149	2	90	70	17	0	27.299999	0.085	22	0	0	0.9394
373	2	105	58	40	94	34.900002	0.225	25	0	0	0.7999
36	11	138	76	0	0	33.200001	0.420	35	0	1	0.6393
...	...	...	...	...	...	...	...	...	...	...	...
85	2	110	74	29	125	32.400002	0.698	27	0	0	0.8002
7	10	115	0	0	0	35.299999	0.134	29	0	1	0.6230
298	14	100	78	25	184	36.599998	0.412	46	1	0	0.5984
341	1	95	74	21	73	25.900000	0.673	36	0	0	0.9244
472	0	119	66	27	0	38.799999	0.259	22	0	0	0.6798

231 rows × 11 columns

```
predictions = predict_model(best, data=data)
predictions.head()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.7773	0.8357	0.5709	0.7321	0.6415	0.4836	0.4915

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable	prediction_label	prediction_score

	Number of times pregnant	Plasma glucose concentration a 2 hours in an oral glucose tolerance test	Diastolic blood pressure (mm Hg)	Triceps skin fold thickness (mm)	2-Hour serum insulin (mu U/ml)	Body mass index (weight in kg/(height in m)^2)	Diabetes pedigree function	Age (years)	Class variable	prediction_label	prediction_score
0	6	148	72	35	0	33.599998	0.627	50	1	1	0.6940
1	1	85	66	29	0	26.600000	0.351	31	0	0	0.9419
2	8	183	64	0	0	23.299999	0.672	32	1	1	0.7976
3	1	89	66	23	94	28.100000	0.167	21	0	0	0.9454
4	0	137	40	35	168	43.099998	2.288	33	1	1	0.8394

Saving Model

```
save_model(best, 'my_best_pipeline')

# This will create a my_best_pipeline.pkl file that can be shared with others to use the model.
# The .pkl file can be used to load the model and make predictions. (On potentially New Data)
```

Loading Model

```
#To load the model back in environment:
loaded_model = load_model('my_best_pipeline')
print(loaded_model)
```

Regression

Loading Dataset

```
# load sample dataset
from pycaret.datasets import get_data
data = get_data('insurance')
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Setting Up PyCaret

```
from pycaret.regression import *
s = setup(data, target = 'charges', session_id = 123)
```

	Description	Value
0	Session id	123
1	Target	charges
2	Target type	Regression
3	Original data shape	(1338, 7)
4	Transformed data shape	(1338, 10)
5	Transformed train set shape	(936, 10)

	Description	Value
6	Transformed test set shape	(402, 10)
7	Numeric features	3
8	Categorical features	3
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Maximum one-hot encoding	25
14	Encoding method	None
15	Fold Generator	KFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	reg-default-name
21	USI	11d3

Getting Best Model

```
best = compare_models()
```

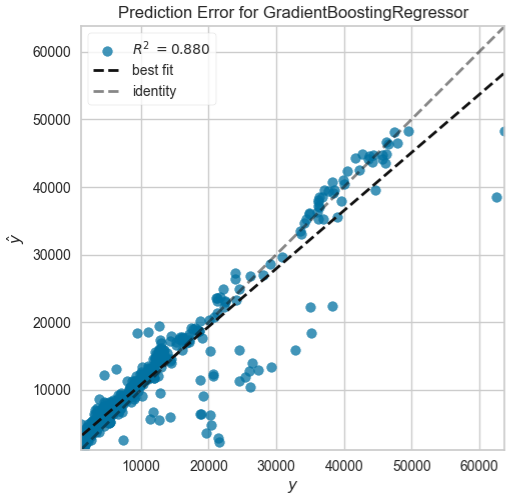
	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2701.9927	23548981.3626	4832.9682	0.8320	0.4447	0.3137	0.0790
rf	Random Forest Regressor	2771.4583	25416502.3827	5028.6343	0.8172	0.4690	0.3303	0.0410
lightgbm	Light Gradient Boosting Machine	2992.1828	25521038.3331	5042.0978	0.8149	0.5378	0.3751	0.3800
et	Extra Trees Regressor	2833.3624	28427844.2412	5305.6516	0.7991	0.4877	0.3363	0.0350
ada	AdaBoost Regressor	4316.0568	29220505.6498	5398.4561	0.7903	0.6368	0.7394	0.0120
lar	Least Angle Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.0090
llar	Lasso Least Angle Regression	4303.7694	38386824.2786	6176.4846	0.7306	0.5952	0.4434	0.0120
br	Bayesian Ridge	4311.2349	38391950.0874	6176.8896	0.7306	0.5910	0.4447	0.0130
ridge	Ridge Regression	4317.6984	38396435.9578	6177.2329	0.7306	0.5891	0.4459	0.0080
lasso	Lasso Regression	4303.7697	38386797.6709	6176.4824	0.7306	0.5952	0.4434	0.1750
lr	Linear Regression	4303.5559	38388058.4578	6176.5920	0.7306	0.5949	0.4433	0.2810
huber	Huber Regressor	3463.2216	48801106.4612	6963.9984	0.6544	0.4927	0.2212	0.0120
dt	Decision Tree Regressor	3383.4916	47823199.0729	6895.7016	0.6497	0.5602	0.4013	0.0130
par	Passive Aggressive Regressor	4537.0122	67346309.9218	8142.7826	0.5422	0.5276	0.3207	0.0100
en	Elastic Net	7372.5238	90450782.5713	9468.3193	0.3792	0.7342	0.9184	0.0100
omp	Orthogonal Matching Pursuit	9089.9268	133439413.5272	11488.4238	0.0884	0.8790	1.1514	0.0130
knn	K Neighbors Regressor	8007.7997	131387268.8000	11425.3695	0.0859	0.8535	0.9232	0.0110
dummy	Dummy Regressor	9192.5418	148516792.8000	12132.4733	-0.0175	1.0154	1.5637	0.0290

Evaluating Best Model

```
evaluate_model(best)
```

Plot Type:

Pipeline Plot | Hyperparameters | Residuals | Prediction Error | Cooks Distance | Feature Selection | Learning Curve | Manifold Learning | Validation Curve
Feature Importance | Feature Importance... | Decision Tree | Interactive Residuals



Making Predictions on New Data

```
predictions = predict_model(best, data=data)
predictions.head()
```

	age	sex	bmi	children	smoker	region	charges	prediction_label
0	19	female	27.900000	0	yes	southwest	16884.923828	18464.334448
1	18	male	33.770000	1	no	southeast	1725.552246	4020.345384
2	28	male	33.000000	3	no	southeast	4449.461914	6555.388388
3	33	male	22.705000	0	no	northwest	21984.470703	9627.045725
4	32	male	28.879999	0	no	northwest	3866.855225	3325.531292

Clustering

Loading Dataset

```
from pycaret.datasets import get_data
data = get_data('jewellery')
```

	Age	Income	SpendingScore	Savings
0	58	77769	0.791329	6559.829923
1	59	81799	0.791082	5417.661426
2	62	74751	0.702657	9258.992965
3	59	74373	0.765680	7346.334504
4	87	17760	0.348778	16869.507130

Setting up PyCaret

```
from pycaret.clustering import *
s = setup(data, normalize = True)
```

	Description	Value
0	Session id	2538
1	Original data shape	(505, 4)
2	Transformed data shape	(505, 4)

	Description	Value
3	Numeric features	4
4	Preprocess	True
5	Imputation type	simple
6	Numeric imputation	mean
7	Categorical imputation	mode
8	Normalize	True
9	Normalize method	zscore
10	CPU Jobs	-1
11	Use GPU	False
12	Log Experiment	False
13	Experiment Name	cluster-default-name
14	USI	354f

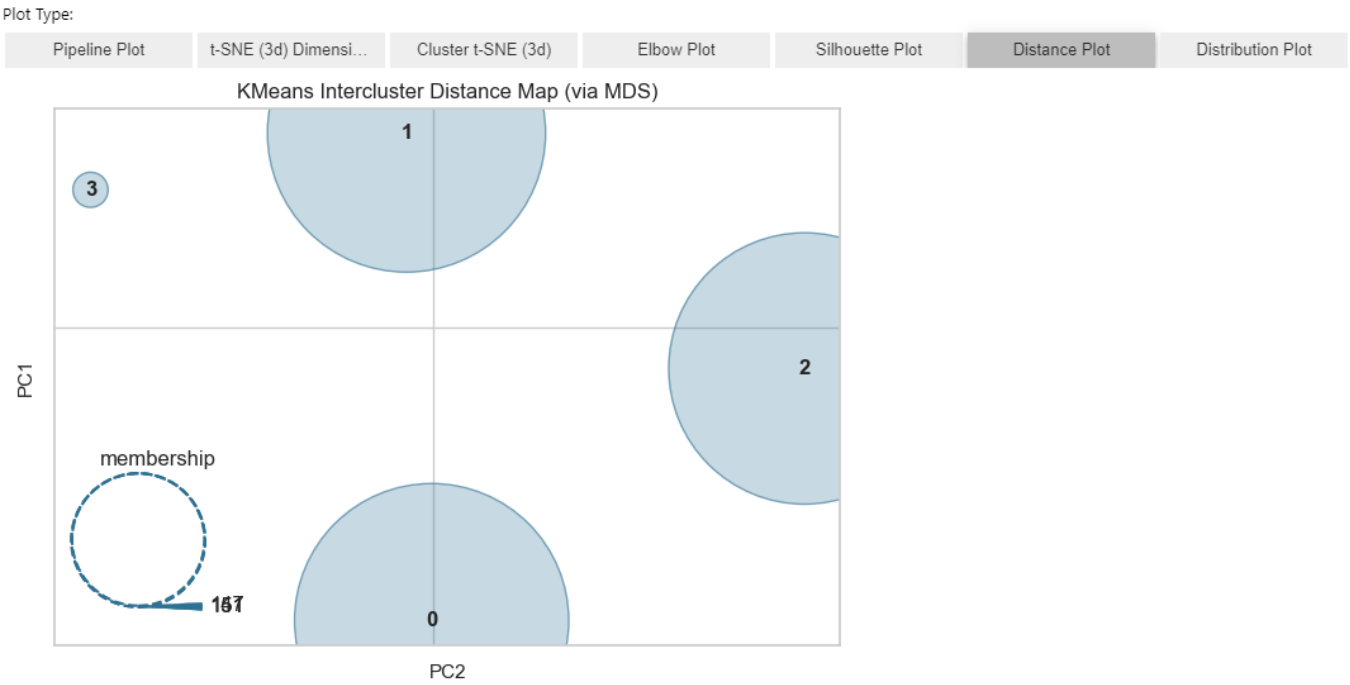
Calculating K-Means

```
kmeans = create_model('kmeans')
```

	Silhouette	Calinski-Harabasz	Davies-Bouldin	Homogeneity	Rand Index	Completeness
0	0.7581	1611.2647	0.3743	0	0	0

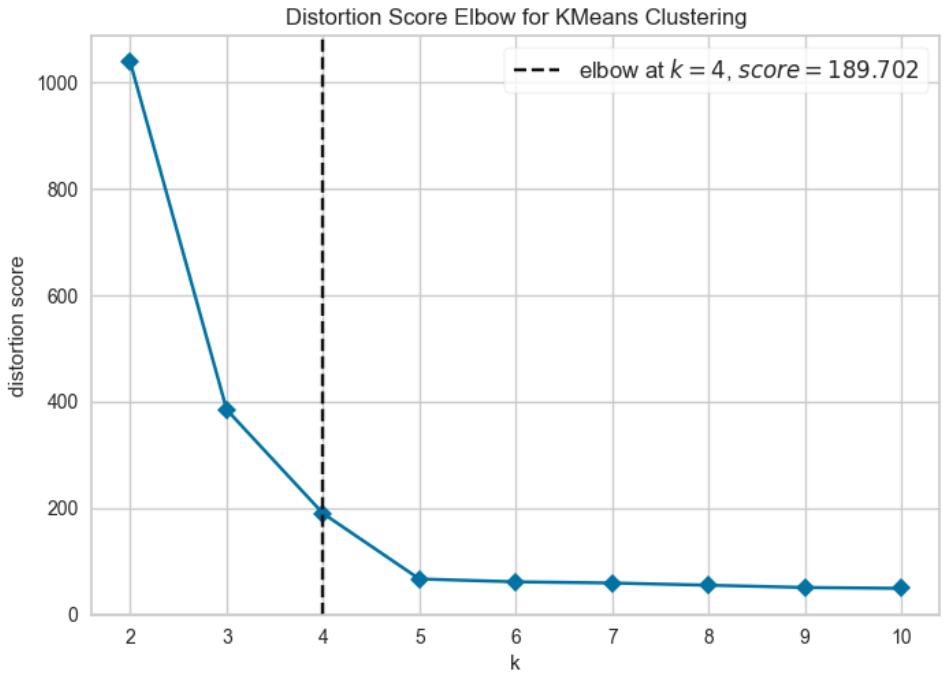
Evaluating Model

```
evaluate_model(kmeans)
```



Plotting Elbow Plot

```
plot_model(kmeans, plot = 'elbow')
```



Assign Clusters to Existing Dataset

```
result = assign_model(kmeans)
result
```

	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.791329	6559.830078	Cluster 1
1	59	81799	0.791082	5417.661621	Cluster 1
2	62	74751	0.702657	9258.993164	Cluster 1
3	59	74373	0.765680	7346.334473	Cluster 1
4	87	17760	0.348778	16869.507812	Cluster 2
...	...	...	...	...	...
500	28	101206	0.387441	14936.775391	Cluster 0
501	93	19934	0.203140	17969.693359	Cluster 2
502	90	35297	0.355149	16091.402344	Cluster 2
503	91	20681	0.354679	18401.087891	Cluster 2
504	89	30267	0.289310	14386.351562	Cluster 2

505 rows × 5 columns

Making Predictions on New Data

```
predictions = predict_model(kmeans, data = data)
predictions
```

	Age	Income	SpendingScore	Savings	Cluster
0	-0.042287	0.062733	1.103593	-1.072467	Cluster 1
1	-0.000821	0.174811	1.102641	-1.303473	Cluster 1
2	0.123577	-0.021200	0.761727	-0.526556	Cluster 1
3	-0.000821	-0.031712	1.004705	-0.913395	Cluster 1
4	1.160228	-1.606165	-0.602619	1.012686	Cluster 2
...	...	...	...	...	...

	Age	Income	SpendingScore	Savings	Cluster
500	-1.286268	0.714535	-0.453557	0.621787	Cluster 0
501	1.409024	-1.545704	-1.164109	1.235201	Cluster 2
502	1.284626	-1.118447	-0.578054	0.855313	Cluster 2
503	1.326092	-1.524929	-0.579866	1.322452	Cluster 2
504	1.243160	-1.258335	-0.831890	0.510463	Cluster 2

505 rows × 5 columns