# Project Overview

## OBJECTIVE

1. Design complete solution to demonstrate end-to-end pipeline development and manage a machine/ deep learning project
2. Develop a understanding of all stages of a machine learning project lifecycle
3. Demonstrate understanding of challenges encountered during the project development and provide ways to tackle them
4. Showcase understanding of software engineering best practices while developing the project

## PROBLEM STATEMENT

Classify News Articles into categories - With information overload today users are inundated with news articles of all topics, even the ones which may not be relevant to users. Design a system which can classify incoming news articles and appropriately tag the corresponding category. Develop a data pipeline which includes the all the following stages of Machine Learning Project Life Cycle –

1. Data Ingestion
2. Data Preparation
3. Data segregation & Model Training
4. Model Deployment
5. Model Prediction

## SETUP DETAILS

Stages are elaborated as follows –

1. DATA INGESTION

Project Name - data-ingestion-service

The objective of this project is to source new data to re-train the model.

This project should connect to at least 2 different news sources. Ingestion could be done either as follows –

1. Via REST APIs (search news APIs available on rapidapi, for e.g., <https://rapidapi.com/newscatcher-api-newscatcher-api-default/api/free-news/>),
2. Via RSS feed (For e.g. by using BeautifulSoup to scrap RSS feed)
3. Create your own mock news generator service

**Please ensure that the collected news has “category” or “topic” as a part of raw data irrespective of the source.**

**Data Cleaning** - Since different news sources could have different response formats, transform the API response for all news sources to same raw format, which should include following fields –

1. title
2. date/ time
3. summary
4. topic/ category
5. source

**Scheduling** - Schedule all news source services to collect news using a process scheduler (every few mins/ hours), or with custom thread management code.

**Data Storage** – Use a Publisher – Subscriber model to collect data and push it to the store. Create a kafka topic and use it to queue cleaned responses from all sources. The sink for the stream should be MySQL/ MongoDB.

Diagram

Description automatically generated with medium confidence

1. DATA PREPARATION (PROPROCESING), SEGREGATION AND MODEL TRAINING   
   Project Name – model-training-service

The objective of this project is to trigger model re-training and deployment on-demand. Over a period, the performance of models degrades, and it becomes important to retrain them.

Before using the raw data can be used for model training/ retraining it needs to be preprocessed to relevant structure.

* 1. Load the data from “raw\_data” source (MySQL/ MongoDB) into Spark by using relevant connector for PySpark
  2. Perform data cleaning and preprocessing, followed by segregation to train and test datasets. (Necessary pretraining steps to be shared with participants)
  3. Perform model re-training (a pre-pretrained model along with code to retrain the model could be provided to participants)
  4. Serialize the model and save it to a location (or push the model to a model registry like MLFlow), which can later be used for model retrieval model deployment

1. Model Prediction –   
   Project Name – model-prediction-service  
   A separate classifier project picks up the trained model either from a location or from the model registry, and exposes it for prediction in following modes –
   1. Real Time REST (flask) API – allow user to initiate a classification request in JSON or any other format using a REST client and returns the classified category in real time.  
      This API should be a POST request and should accept JSON in request and return JSON as response.  
      Request body should contain 2 fields - the title and description/summary of a single news article  
      Response should contain the predicted category as a field
   2. Batch Mode (IF TIME PERMITS) – Allow user to pass bulk news articles, and project returns a bulk classification response. It could also be an API which accepts multi-part request containing excel in a pre-determined format, and simply submits the request to bulk process. Response would be a unique id. Another API retrieves the processed excel with classified responses, once the processing is complete, or returns “WIP” if the processing is still in progress.

**User Interface** - A simple Streamlit app or a basic HTML page containing a form can be exposed in this project.

1. It takes the title and the description/ summary of the article, calls the flask API for prediction, and shows the predicted classification.
2. It provides an option to upload a batch file and response can be processed and stored at a folder location.

## INFRASTRUCTURE/ DEPLOYMENT/ ADDITIONAL GUIDELINES

1. Docker images may be created for all the 3 projects. These images can then be used for deployment as containers.
2. Deployment can be orchestrated by using docker-compose (optional)
3. Flask APIs (wherever required) should use Gunicorn/ Bjoern/ CherryPy as a WSGI server.
4. Use PEP guidelines for python code standards.

## ARCHITECTURE/ DESIGN EXPLAINABILITY

An important aspect of ML software development pipeline is explainability. Use UML (Unified Modelling Language) to explain the project architecture and design. Following diagrams are expected –

1. Use Case Diagram – identify the actors in the system, and associated actions
2. Activity Diagram – prepare logical workflows which illustrate the flow of control in the system.
3. Sequence Diagram – prepare sequence of interactions with the system.
4. Component/ Deployment Diagram – diagram to show how components have been organized.

## ML TOOLING

Following tools to be used for project setup –

1. PyCharm as Python IDE
2. PlantUML for UML diagrams documentation – install it as a plugin in PyCharm
3. Virtual Environment – use venv or virtualenvwrapper to setup separate environments for all projects described
4. Instead of using Conda, participants can also perform a minimal installation of jupyter setup
5. MySQL/ MongoDB as datastore
6. PySpark for stream processing
7. POSTMAN for testing Flask APIs.
8. Apache Zookeeper + Kafka for message queue/ streams
9. Tensorboard for monitoring the progress of model retraining
10. MLFlow for model versioning + hyper-parameters versioning
11. Python cookiecutter templates may be used for setting up the project

## MILESTONES

There would be 4 milestones –

1. Week 1 –
   1. Participants are expected to complete documentation and architectural design by creating the UML diagrams.
   2. Prepare the local environment with the right tools and installations
   3. Ingest data source and prepare data-ingestion-service which populates “raw\_data”
2. Week 2 –
   1. Setup the model-training-service project
   2. Setup appropriate connectors to load data from MongoDB/ MySQL into PySpark RDD.
   3. Understand cleaning + preprocessing steps necessary to transform the raw data and complete data preparation step.
3. Week 3 –
   1. Segregate the preprocessed data into training and test
   2. complete model training and deployment (and model registry)
   3. Convert the project to On-Demand service, to be able to retrain the model as needed. This could be a flask API which could trigger the entire pipeline in model-training-service (loading data into PySpark, preprocessing, model training)
   4. Serialize the re-trained model.
   5. Push this model to model-registry (MLFlow) along with hyperparameters
4. Week 4 –
   1. Expose model via model-prediction-service in the form of flask API.
   2. Dockerize all the projects by adding appropriate Dockerfile
   3. Prepare a simple HTML page containing a form to take an article as input and print the predicted category.