Assignment 1

Working Code is available in *github* repository and also attached as *ZIP*.

Code is written using **Python** Orchestrated using **Dagster** and deployed using **Docker** and **GitHub Actions.**

Q1. How would you assess if a code is ready for production? What do you think about the given code in model.ipynb?

Few things I noticed in model.ipynb:

1. Historical Data is split into *Train*(training set) and *Test*(holdout set), but they are transformed using the **fit_transform** method of ***StandardScaler().***
    a. Ideally only *Train* data should be *fit_transform*
    b. *Test* data should just be *transformed*

    It gives us more realistic results if *Transformation* is applied after *Split*. Also, *shuffle* split could give better representation of model performance.

    ```
    X_train, X_test, y_train, y_test = train_test_split(
       X, y, test_size=0.3, random_state=99
    )
    transformModel =  StandardScaler()
    X_train = transformModel.fit_transform(X_train)
    X_test = transformModel.transform(X_test)
    ```

2. I would use the Sklearn's **pipeline** for the classifier rather than using the classifier directly. That applies the transformation on *inference* data as well.
3. Inference data is ***not*** *transformed* using ***StandardScaler(),*** but the model is trained on *transformed* data which will result in *Inconsistent representations* and *Performance degradation.*

In general, code does not seem ready for the Production,

- **Description of data/features** are not provided and so it may be difficult for ML Engineers to include the data source and apply the necessary transformations.
- The **Technical** as well as **Business KPI's** are not mentioned and so it's difficult to know if Model is fulfilling its purpose(Only model accuracy is not enough to go to production).

Q2. ***Would you change anything in the code? How and why?***

- ***Hyperparameter tuning*** could improve the results.
- Some other Parametric models like Tree, Bagging based models could help understand the feature importance.

- I would spend little time finding the ***explanation*** of the model results.

Q3. *How would you rearrange code for the production system?*
I would rearrange the code a little bit.(functional code is provided)
- A separate transformation function
- Separate the action of Training, Evaluation and Inference.

Q4. *Discuss training and inference data provided*
- Training data seem non-linearly separable.
- Inference data seem to be from the same distribution as of training data.

Data Sources:
- ❖ Data can be delivered from multiple sources and so in this case I would go for a feature-store structure. Which can collect data from multiple sources and transform them specific for modeling and keep them organized in one source(cache or db or files).
- ❖ Having a feature store is beneficial for reproducible results and models could detach them themselves from the source of data.
- ❖ Using proper storage for feature-store can enrich the model triggering, which can be constant pooling or event triggered based on use case.

Q5. How would you organize the codebase management. E.g. how do you deliver code to the production environment?

- Code should be able to be containerized(using docker or any other tool).
- Code should be able to be orchestrated, separate action flows for Training, Evaluation and Inference to enable **Continuous Evaluation** and **Continuous Training**.
- There are 2 ways to separate the model, as per below:
    - Have one repository per use case but different models but have an *Orchestration* tool to control the interdependent runs and Schedules.
        - Single repository per use case will help to bring down the **COST** and **Maintenance Effort.**
        - There could be multiple models per use case, they can stick together and orchestrated together.(*Output* of one model as *Input* of another model)
        - The schedules and triggers can be managed effectively in a *cohesive* model environment.
    - Have a separate repository for every model.
        - Separate **CI/CD** flow maintaining **microservice** architecture
        - If orchestration is needed, a universal *Orchestration* tool can be utilized to directly use the Containers for orchestrating.
- Save the final result in a **Permanent Storage** for better **monitoring** and **Visualization.**

Optional part

Q1. Would you consider any alternatives to the used algorithm and if so, why?

Current GaussianProcessClassifier algorithm is a non parametric algorithm which looks like a good choice here as data is not linearly separable. However, I would also try out *Tree-based* algorithms as that would give me as they are *faster* and more *explainable*.

Q2. Discuss pros and cons of your chosen algorithms as well as evaluation metrics.

GaussianProcessClassifier is a bit of a slow algorithm but I think that should not be a problem in this particular use case.
Accuracy might not be a very useful evaluation matrix in this particular case as here it is more important to identify the classes which require the maintenance rather than the one which does not require maintenance.

- The system that requires maintenance should be identified correctly as nearly 100% accuracy.
- Looking at **sensitivity/Recall** would make more sense as we do not want to miss the systems that need maintenance.

Q3. Discuss Big O notation of the algorithms, its algorithmic and memory complexity.

For GaussianProcessClassifier **O(Nd)** will be the complexity given **d** as dimension and **N** as number of dataset.

Q4. List libraries in various programming languages that you know or have used that implement those algorithms.

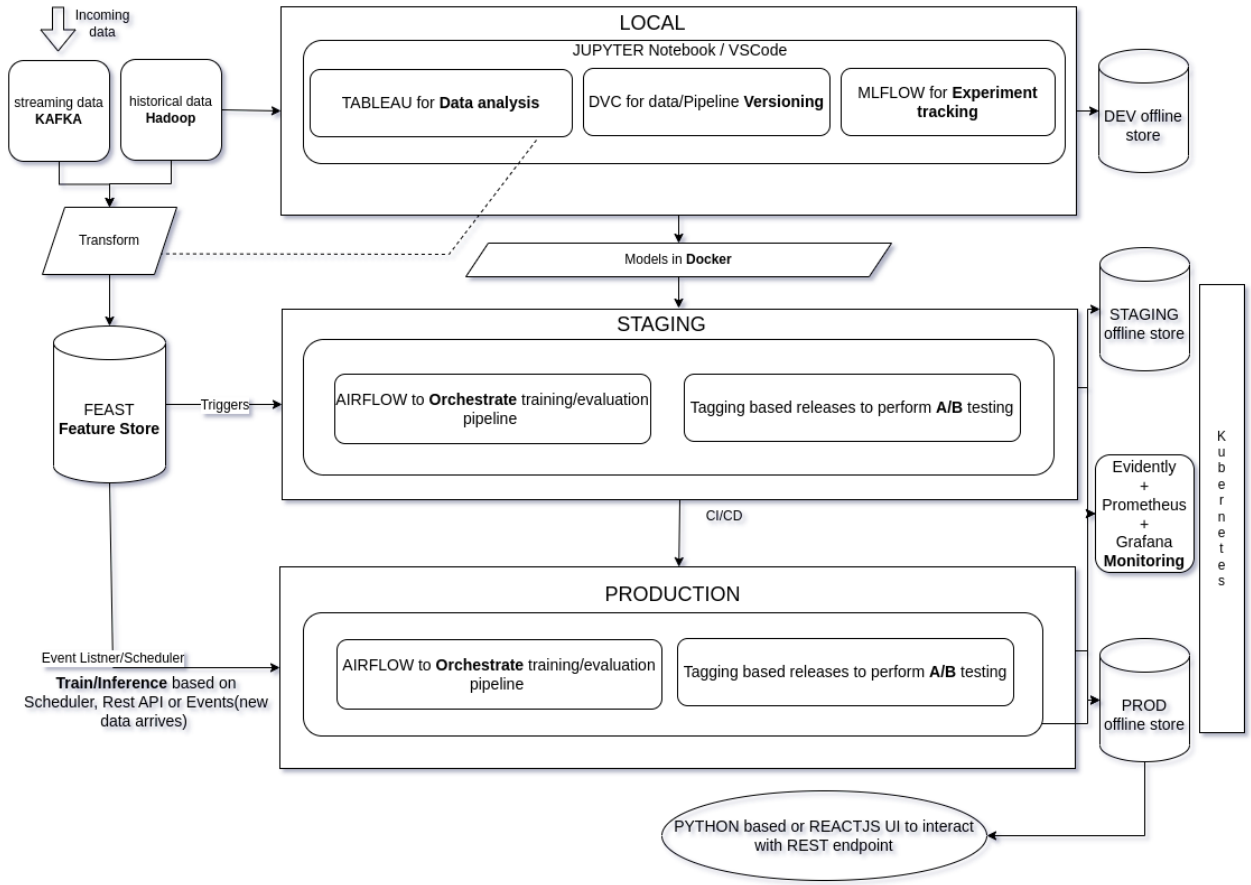
GaussianProcessClassifier : SKLearn : Python
KDependentBayesClassifier : Java-ML : Java

Q5. Would your setup change if these were Databricks notebooks with Spark compute clusters behind them, written in PySpark?

I will probably use **MLLib** which directly works on distributed data.

Assignment 2

Q1. What would a cloud architecture look like and why?



Q2. What tools and implementation approach would you take?

In this architecture I have tried to use mostly Open Source tools except some of them but, using a cloud provider could also be used.

Using Cloud Provider vs Ensemble tools

Comparison

Cloud Provider	Ensembel tools
Less maintanance	**More maintance required**
Useful with less people in the team	**Useful with a dedicated team**
Could be more expensive	**Can be planned wisely to be less expensive**
Less suitable for specific use cases	**Can be architected specific to the use case**

AZURE

- Data Storage : Azure Blobs
- Database : Microft SQL/T-SQL
- Servers : AKS(Azure Kubernetes Service)
- Data pipelines : Microsoft Fabrics
- Model Developemnt and Deployment : Azure Machine Learning

AWS

- Data Storage : S3
- Database : Athena or Redshift
- Servers : ECS or EKS
- Data pipelines : AWS Glue
- Model Developemnt and Deployment : Azure Sagemaker

Ensemble

- Data Storage : Hadoop/S3 or even MongoDB(depending on usecase)
- Database : Postgres
- Cache DB : Redis
- Servers : Kubernetes cluster/EKS/AKS
- Data pipelines : Airflow/Dagster/Prefect
- Model Development and Deployment : Jupyter | MLFlow, DVC | Airflow/Dagster/ZenML

Q3. Which pipelines would you build there? How?

- A feature store pipeline, which will collect the data from different sources, Transform them and make them available to the Models.
- A ML pipeline, to facilitate run of multiple models based on Schedule, Trigger and Events. And also to support inter model dependencies.

An Orchestrator like Airflow, Dagster, ZenML would help achieve achieve these things

Q4. Which assumptions have you made?

1. Historical Data is already gathered and stored in data warehouse/lake based storage.
2. Infrastructure is available to process Real time data and made available to be processed by model.
3. High volume of data is expected.
4. Model training/evaluation should be either manually triggered or triggered based on schedule/event.
5. Model Prediction triggers are request/scheduled/event based.
6. MLOps infrastructure needs to be set up from scratch.

Q5. How can code management be done?

1. Using a GIT based repository.
2. Containerizing the Model in Docker.
3. Creating CI/CD flow to push images to Container
4. hose images can be picked by Orchestrator for serving

Q6. Discuss maintenance aspects.

Two types of maintenance would be needed

1. *Model Maintenance:* Maintaining model requires *continuous monitoring* and when there is any kind of drift detected, may *continuous Evaluation/Training* can help to re-deploy the model fast.
2. *Code and Infrastructure Maintenance:*
    a. This is more of a technical maintenance. Concept remains the same, monitor the servers and proactively find the error and fix that.
    b. Code should be written and structured in a way that it can be migrated and changed with minimum effort.

Optional part

Q1. How would you integrate the near real time model updates (online training) with the once-a-day slow full-update (batch training)?

An event-based or *constant pulling* model triggering mechanism can be implemented. Orchestrators these days almost mostly support Event-Based triggers but in a very simple way a constant-pulling can be written(using Asynchronous or multi-threading) to continuously check if new data has been arrived and then trigger the model.

Q2. Discuss organizational aspects and challenges of a multi team setup.

The biggest challenge is to develop something which suits everyone's needs. The best way to go about it is to gather the understanding first and then develop something. Also, keep a room to add new *Tools/Plugins.* A good software practice always helps and I believe that every data scientist should be given a walk through these principles.

In multi-team setup, if one is not careful it could shoot the cost of implementing tech to roof very soon.

Q3. How would your cloud architecture change if some data were external and collected outside the ZEISS organization?

A good architecture should be built as distributed systems(microservices). The one microservice should always be detached from the implementation part and just only care about the output of that service.

The service that would be working on data, would be separated and that implementation can change anytime. The other Service(Model or Reporting) will only care about the output of that data service which could be either a *Data Storage* or simply just a *REST* endpoint.