Assignment - ML Systems and Operations

The effort will be centered around exercising the "testing process" for non trivial datasets and resulting models that is associated with typical production ML deployments.

Dataset selection: Pick a dataset from either kaggle or hugging face. The dataset should

- represent a real business process or user collected information, and have associated documentation explaining what the features/attributes are, and how the dataset is sourced.

- have in the range of ~20-25k or more tabular records, combining numeric, categorical as well as textual attributes

Task 1): *Pre-training tests* - create at least two "tests" on the quality of data. Use a "unit test" style of testing as provided by data or model testing libraries (i.e. popular libraries such as great-expectations, evidently, etc.). The tests should be provided as python modules/files (not as notebook files). Library dependencies if any, should be provided as Dockerfile definitions, (or requirement.txt) and be executable locally.

- Create a test that measures missing or null values, potentially on a segment of the original dataset compared to an "expectation" of acceptable quality. Define where that "expectation" is derived from and provide the reasoning behind why you picked that expectation as documentation to your test code. Note: the expectation definition and the reasoning behind that choice will also be scored by the reviewers alongside the test itself (weight of expectation definition 0.4, executable code 0.6).

- Pick *two* relevant attributes of your choice in the dataset (e.g. gender, and price) and create a test for their distribution (one per each attribute) compared to an "expectation" of acceptable values or value ranges. Same as prior, alongside the test code, provide an explanation as to how and why you picked the specific expectation of what is acceptable/normal behavior.

Task 2): *Pre-deployment tests* - create one model trained on potentially a segment of the original dataset to optimize for a target variable in your dataset. Describe what your input and target variable is for the model training (alternatively input/output *schemas* should be provided as part of the versioning exercise below). The type, size or sophistication of the model training is for the scope of this exercise irrelevant and serves only the purpose of having an artifact type and an outcome to validate.

- Use a locally executable orchestrator/flow engine (options: *Metaflow* (local), *Prefect* (local), *ZenML* (local or local_docker orchestrator) or even a simple self-written one if interested with instructions how to run) and execute the data tests from Task 1 as one separate step and the training code of your model as a separate second step. The model training should be included as a *second step* right after the data tests. The output of the model training step should be a *serialized model* artifact stored as a file. Use a sensible serialization format (ideally a better format than pickle). The flow should be executable locally. Steps should have their own distinct (and only necessary) dependencies listed as Dockerfile(s) or as requirement.txt files.

- *Versioning models*: Once the step for training and serialization is complete, version the serialized model, meaning make sensible choices over the location where to store all the model artifacts, including a list of all available models. You are allowed to use (open) libraries for this (*mlflow*), or alternatively write a very simple own model versioning library that can a) *list available models* from your exercise and their locations; b) *load a particular model* via its id; c) *each model has metadata over input/output* as a schema, and its *code dependencies*.

- Add a *third step* to your local flow that validates “robustness” of the trained model compared to an expectation or existing benchmark. For this step you need to execute the previously trained model, via loading, deserializing and initializing it. Define and reason on your choice of expectation/acceptable behavior for the model in regards to robustness. Comparison to other more established models' behavior is also permissible here as an option. Note: how well your trained model passes or fails the test, is not the focus, rather your "test setup" should be relevant, sensible and catches "undesired" behavior. Hence the choice and reasoning of the behavior "expectation" are again the focus in this exercise.

- Induce at the training step an artificial error, (i.e. the training data size is too small (< 1k records), or the system is interrupted during training). Introduce logic/configs in your code (or flow) that handles/responds to these or system error scenarios. Document about your choice of the error handling as a paragraph in the documentation of your training "step".

Task 3): *Post-deployment tests* - In this task you will perform a "drift" test on a segment of the data that *has not been exposed during model training*. In addition, in this task an *a/b test* will be exercised. Explain the choice of the unseen data segment for these tests as documentation in your code. More details for this task will follow after the second Exercise Lab.

- (1) More details will follow.

- (2) More details will follow.

- (3) More details will follow.

Logistics: Each task should be submitted one week (by Thursday EOD) following the corresponding lab session. During the exercise labs, students will present and get feedback on their existing task solution. The expectation is that a lab session is used for presenting, feedback, discussing and fixing your solutions. You should have already prepared and done most of the work prior to the lab.

- On each task you should get at least 35% to pass

- Overall you need to get 50% to pass.

- Submission can also be done with a delay of one week, past the due date with a penalty of 20%