Task 1: Train a Regression Model and Tune Hyperparameters

Dataset: https://scikit-learn.org/1.0/modules/generated/sklearn.datasets.load_boston.html

Objective: Train a model to predict house prices based on the provided features. Use hyperparameter tuning to optimize the model's performance. The candidate will also be required to run the tuning process in parallel using Kubernetes or Vertex AI.

Instructions:

1. **Model Selection**:
   - Use a machine learning model such as **XGBoost** or **Random Forest** for predicting house prices.
2. **Hyperparameter Tuning**:
   - Set up hyperparameter tuning to optimize for the following parameters:
     - For **XGBoost**: `n_estimators`, `max_depth`, `learning_rate`, `subsample`
     - For **Random Forest**: `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`
   - Define a range for each hyperparameter. (To keep the task manageable, feel free to define a small range that minimizes computation and can easily be handled within the free GCP credits available.)
3. **Parallelism**:
   - Run the hyperparameter tuning trials in parallel using either:
     - **Vertex AI Hyperparameter Tuning**: Set up a bayesian hyperparameter tuning job in Vertex AI and autoscale the resources needed for parallel execution.
     - **Kubernetes Jobs**: Set up a Kubernetes cluster to run multiple training jobs in parallel. Configure auto-scaling to handle the load dynamically.
4. **Evaluation**:
   - Evaluate the model's performance using **Mean Squared Error (MSE)** or **R-squared (R^2)** on a validation set.
   - Store each trial of hyperparameter tuning as artefact along with their evaluation metrics.
   - Choose the best and deploy for inference/prediction.
5. **Submission**:
   - Submit your notebook or Python script, and include:
     - Detailed instructions (with screenshots) of how you implemented parallel hyperparameter tuning (using Vertex AI or Kubernetes).

Expected Deliverables:

- A well-documented Python script (or Jupyter notebook) that includes:
  - Training code with hyperparameter tuning logic.

    - Instructions on how to reproduce the pipelines (Vertex AI or Kubernetes) including artefact storage.
    - Instructions on how you deployed the best model.

Task 2: Build a CI/CD Pipeline

Use the model from Task 1.

Set up a CI/CD pipeline that:

    - Runs tests on model code (Any changes to model code runs validation and compares accuracy with previous version).
    - Triggers a build when new code is pushed to a Git repository.
    - Deploys the model to production when tests pass (new model outperforms old).

Instructions:

1. **CI/CD Setup**:
    - Use **Git** for version control and a CI tool of your choice (e.g., **GitHub Actions**, **Cloud Build**, **Jenkins**, **GitLab CI**).
    - Implement automated tests for the model code (e.g., testing the model predictions).
2. **Submission**:
    - Share your Git repository that contains the CI/CD configuration (Dockerfile, Kubernetes YAML, pipeline script).
    - Include a report (README.md) that describes the architecture and tools used, along with any trade-offs or challenges you encountered.

Expected Deliverables:

- Well-documented Git repository with clear instructions on how to reproduce the pipeline.

Note: The candidate will not be evaluated on their machine learning skills but rather on their ability to design and deploy GCP services, set up autoscaling, parallelization, and implement CI/CD processes.

Submission Instructions:
Send all documents and deliverables to **careers@tymestack.com**.
Optionally, you can use loom (https://www.loom.com/) for documenting your approach & explanations. *(This approach is desirable as it provides a clearer understanding of your thought process.)*