Task 1: Train a Regression Model and Tune Hyperparameters

Github , LinkedIn, email : mannarmannan02@gmail.com

Assigned project Documentation:Boston_model-hparam tuning and CI/CD Implementation

Model selection: XG-Boost

Parallelism: Kubernetes Jobs (Hyper-parameter-tuning)

Introduction:

I implemented a model to predict house prices based on the provided features. And, I developed an end-to-end CI/CD pipeline for this model using GitHub Actions. Below is the file structure and contents for a detailed evaluation.

File 1: train.py

```
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

column_names = [
    "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX",
    "PTRATIO", "B", "LSTAT"
]

boston_df = pd.DataFrame(data, columns=column_names)
boston_df['MEDV'] = target

X = boston_df.drop('MEDV', axis=1)
y = boston_df['MEDV']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Arguments to be passed from Kubernetes into model container
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--n_estimators", type=int, required=True)
parser.add_argument("--max_depth", type=int, required=True)
parser.add_argument("--learning_rate", type=float, required=True)
parser.add_argument("--subsample", type=float, required=True)
args = parser.parse_args()

model = XGBRegressor(
    n_estimators=args.n_estimators,
    max_depth=args.max_depth,
    learning_rate=args.learning_rate,
    subsample=args.subsample
)
```

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"MSE: {mse}")

model_filename = f"/data/model_{mse}.joblib"
metrics_filename = f"/data/metrics_{mse}.txt"

joblib.dump(model, model_filename)
with open(metrics_filename, "w") as f:
    f.write(str(mse)+"\n")
```

Code Description:

1. **Arguments:**
   The script takes the following hyperparameters as arguments, which are passed via Kubernetes:
   - **n_estimators * max_depth *learning_rate * subsample.**
2. **Training and Splitting Data:**
   The Boston housing dataset is split into **80% training and 20% testing**.
3. **Model and MSE Scores:**
   The trained model is saved as **model_{mse}.joblib**. The mean squared error (MSE) is saved as **metrics_{mse}.txt** for further analysis.

File 2: Docker File (Dockerfile.train)

```
FROM python:3.8-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY train.py .
ENTRYPOINT ["python", "train.py"]
```

File 3: Requirements.txt

```
pandas==2.0.3
scikit-learn==1.3.0
xgboost==1.7.6
joblib==1.3.2
flask==2.0.1
numpy==1.21.2
```

File 4: Kubernetes Job Template (job-hyperparameter.yaml)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: trial-{{N_ESTIMATORS}}-{{MAX_DEPTH}}-{{LEARNING_RATE}}-{{SUBSAMPLE}}
  labels:
```

```
    app: xgb-hyperparameter-tuning
    trial: "true"
spec:
  template:
    metadata:
      labels:
        app: trial
        n_estimators: "{{N_ESTIMATORS}}"
        max_depth: "{{MAX_DEPTH}}"
        learning_rate: "{{LEARNING_RATE}}"
        subsample: "{{SUBSAMPLE}}"
    spec:
      containers:
      - name: xgb-trainer
        image: mannarn/model-train:latest
        args:
        - "--n_estimators={{N_ESTIMATORS}}"
        - "--max_depth={{MAX_DEPTH}}"
        - "--learning_rate={{LEARNING_RATE_RAW}}"
        - "--subsample={{SUBSAMPLE_RAW}}"
        volumeMounts:
        - name: data-volume
          mountPath: /data
      restartPolicy: Never
      volumes:
      - name: data-volume
        hostPath:
          path: /data
          type: DirectoryOrCreate
```

Code Description:

1. **Kind**
   The Job resource in Kubernetes ensures that a pod runs to completion.
2. **Metadata (Job Name and Labels):**
   The job name follows a structured naming convention using hyperparameters:
   trial-{{N_ESTIMATORS}}-{{MAX_DEPTH}}-{{LEARNING_RATE}}-{{SUBSAMPLE}}
   Example: trial-200-5-0-1-0.6
3. **Pod Metadata:**
   Hyperparameters (**n_estimators, max_depth, learning_rate, subsample**) are assigned as pod labels for easy identification.
4. **Passing Hyperparameters as Arguments:**
   These arguments are passed to the container when it starts, allowing the container to configure the XGBoost model dynamically.

File 5: Script to Launch Trials (generate-jobs.ps1)

```
$n_estimators_list = @(100, 200)
$max_depth_list = @(5, 7)
$learning_rate_list = @(0.01, 0.1)
$subsample_list = @(0.6, 0.8)
```

```powershell
# All combination of h-parameters
foreach ($n in $n_estimators_list) {
    foreach ($depth in $max_depth_list) {
        foreach ($lr in $learning_rate_list) {
            foreach ($sub in $subsample_list) {
                # Sanitize values with dots (e.g., 0.01 → 0-01 for Kubernetes naming)
                $sanitized_lr = "$lr".Replace(".", "-")
                $sanitized_sub = "$sub".Replace(".", "-")

                $yamlContent = (Get-Content job-hyperparameter.yaml -Raw) `
                    -replace '\{\{N_ESTIMATORS\}\}', $n `
                    -replace '\{\{MAX_DEPTH\}\}', $depth `
                    -replace '\{\{LEARNING_RATE\}\}', $sanitized_lr `
                    -replace '\{\{SUBSAMPLE\}\}', $sanitized_sub `
                    -replace '\{\{LEARNING_RATE_RAW\}\}', $lr `
                    -replace '\{\{SUBSAMPLE_RAW\}\}', $sub

                Write-Output "Submitting job: trial-$n-$depth-$sanitized_lr-$sanitized_sub"
                $yamlContent | kubectl apply -f -
            }
        }
    }
}
```

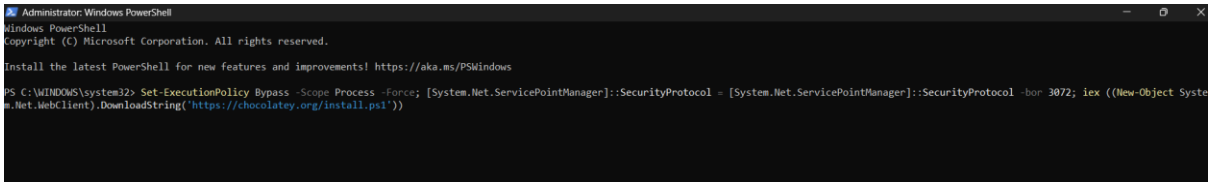
Code Description:

1. **Automated Job Submission:**
   This PowerShell script generates Kubernetes job configurations for all possible hyperparameter combinations.
2. **Sanitizing Values for Kubernetes Naming:**
   Since Kubernetes does not allow dots (.) in resource names, we replace them with hyphens (-):
   $sanitized_lr = "$lr".Replace(".", "-")
   $sanitized_sub = "$sub".Replace(".", "-")
   This ensures compliance with **DNS-1123 Kubernetes naming rules**File_4: Dockerfile.train

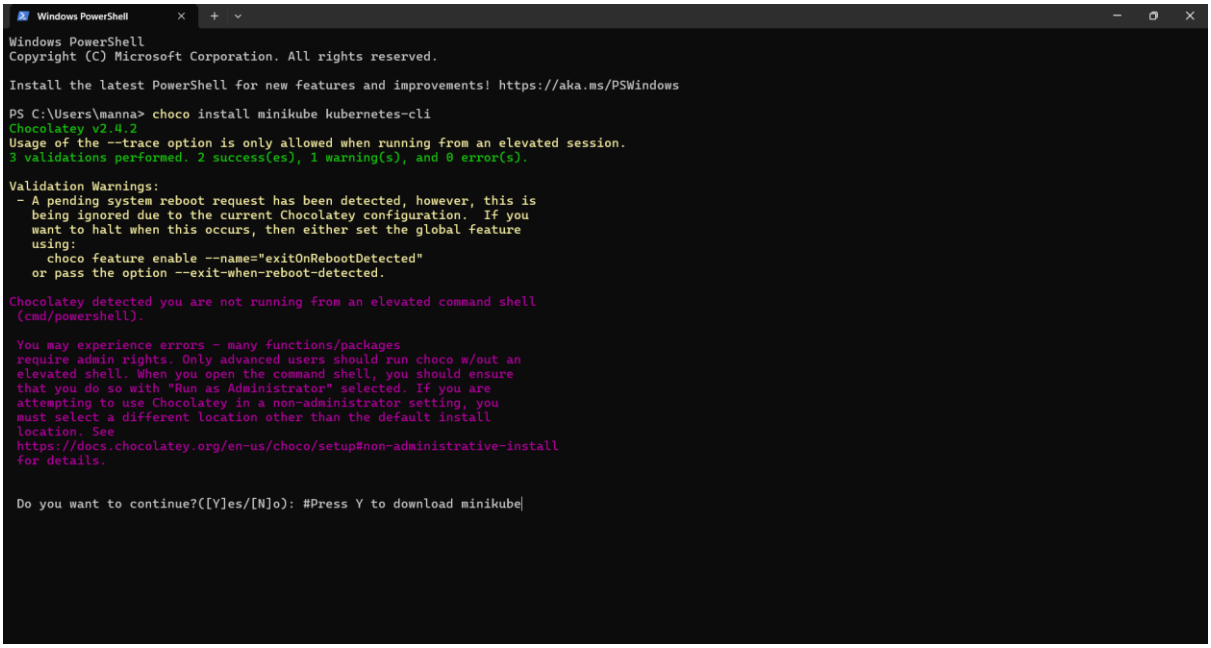
STEPS TO RUN Hyperparameter tuning in Local machine

#Install dependencies:

1.Chocoley or Winget to download Mini Kube:



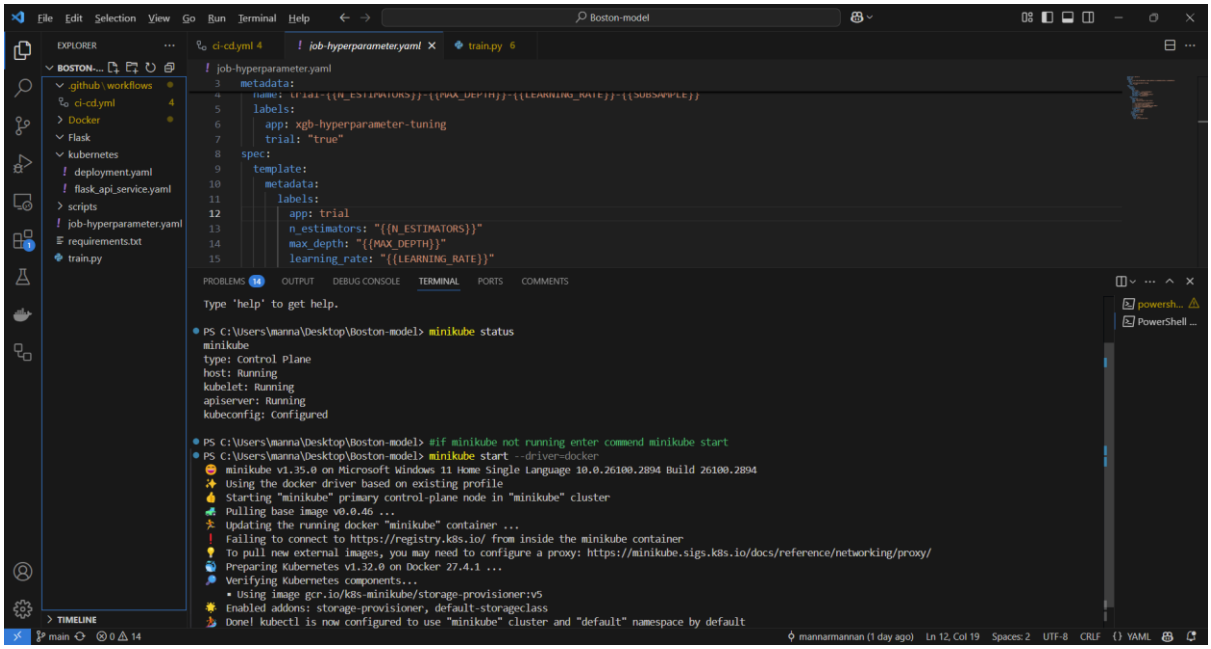
2. Install Mini Kube and Kubernetes CLI tools to create a local development cluster.



3. Start Mini Kube Cluster

>>minikube start --driver=docker

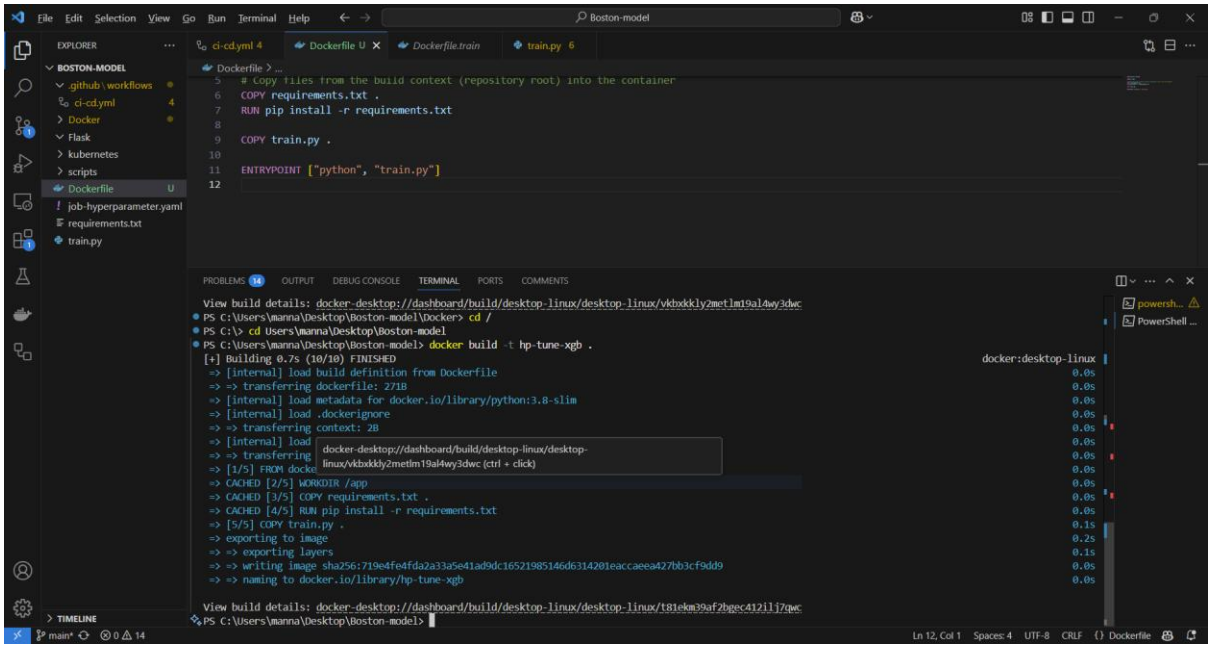
Initializes a local Kubernetes cluster using Docker driver for container orchestration.



4 Build Training Image

>>docker build –t hp-xgboost .

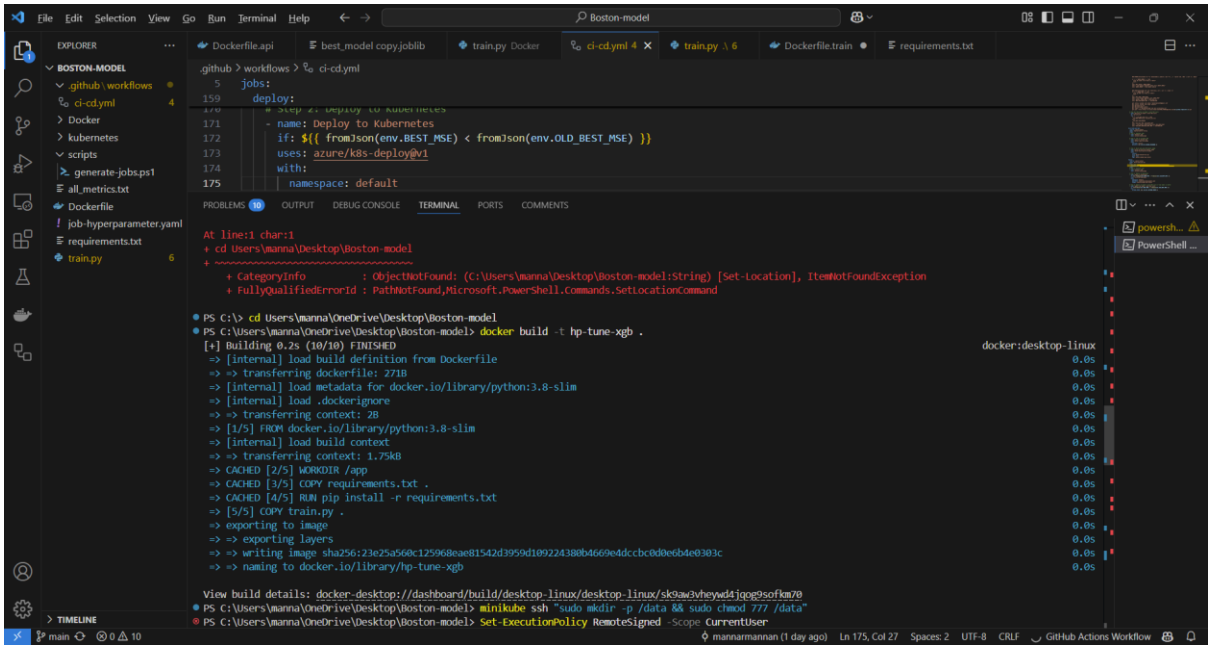
Creates Docker image with training environment and XGBoost dependencies.



5. Create Shared Volume

>>minikube ssh "sudo mkdir -p /data && sudo chmod 777 /data"

Sets up persistent storage in Minikube VM for model artifacts and metrics.



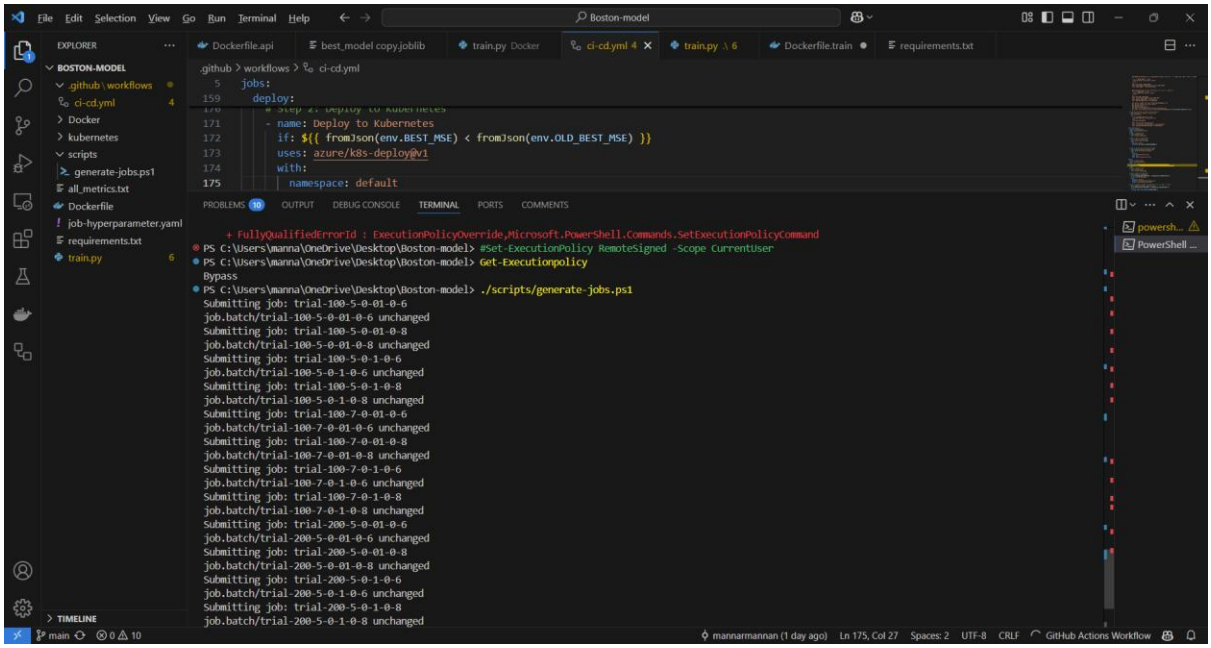
5. Enable PowerShell Script Execution & Launch Parallel Training Jobs

>>Set-ExecutionPolicy Bypass -Scope Process

Temporarily allows PowerShell script execution for hyperparameter job generation.

>>./scripts/generate-jobs.ps1

Submits 16 Kubernetes jobs with different hyperparameter combinations for distributed tuning.

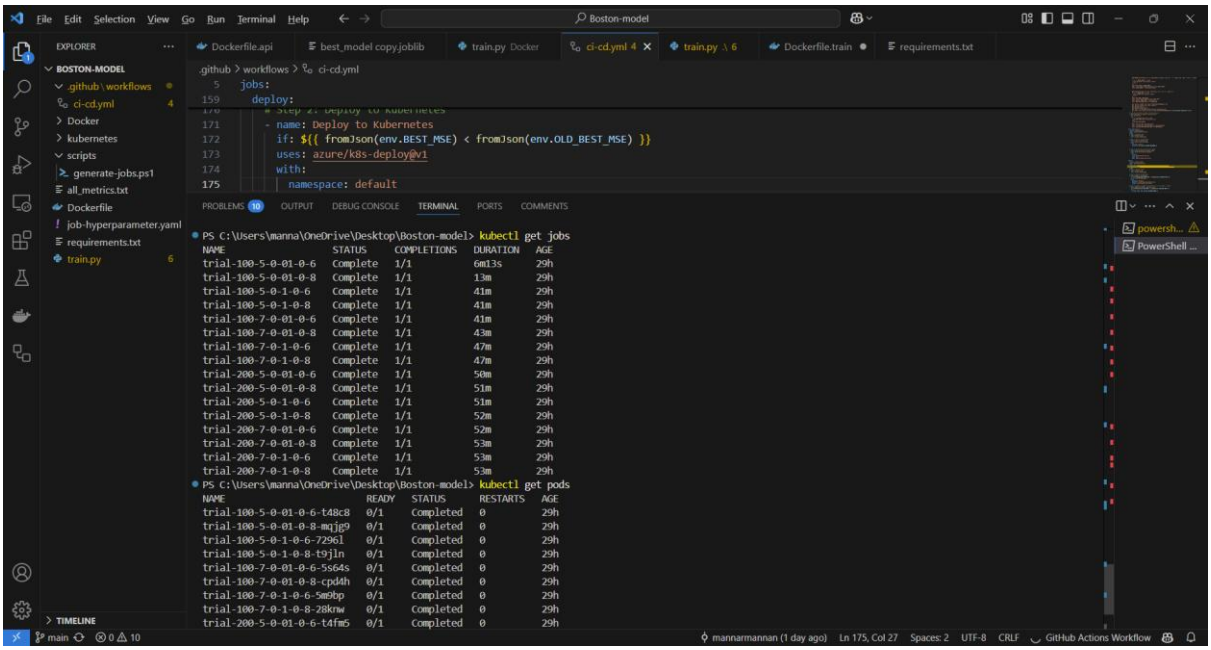


6. Monitor Job Progress

>>kubectl get jobs

>>Kuberctl get pods

Tracks job completion status in real-time through Kubernetes watch mode.



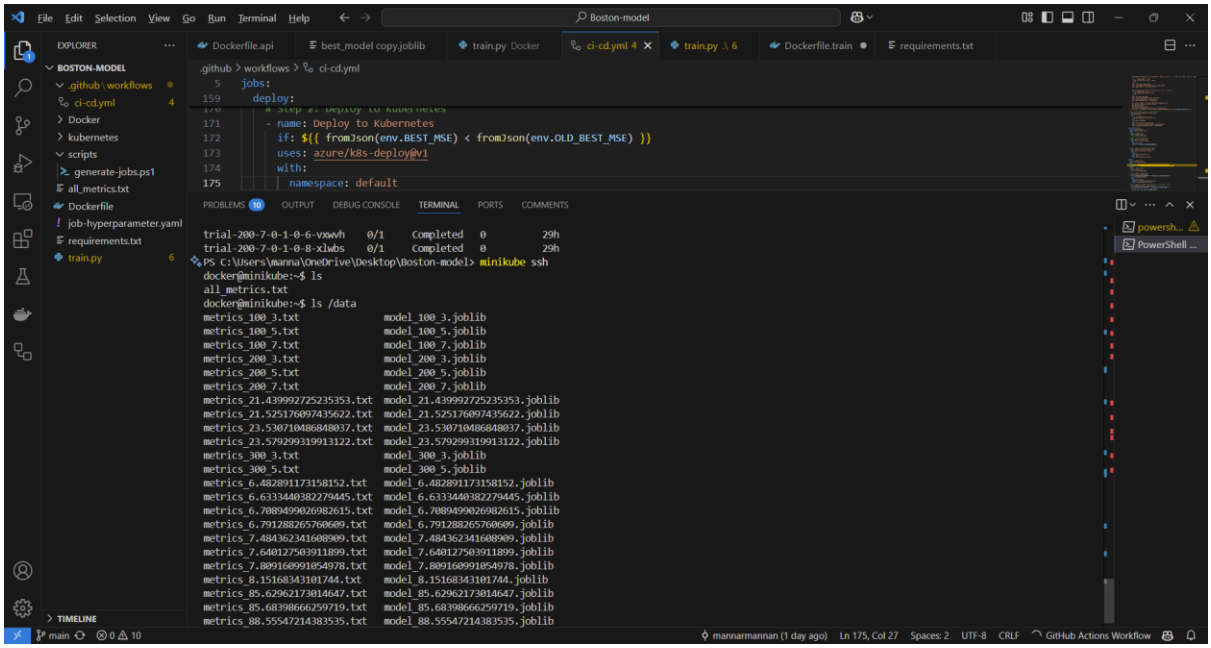
7.Login into shared memory of Virtual machine

>>Minikibe ssh

Inspect Training Results

>>minikube ssh "ls -lh /data"

Verifies artifact creation in shared volume with model files and metrics.



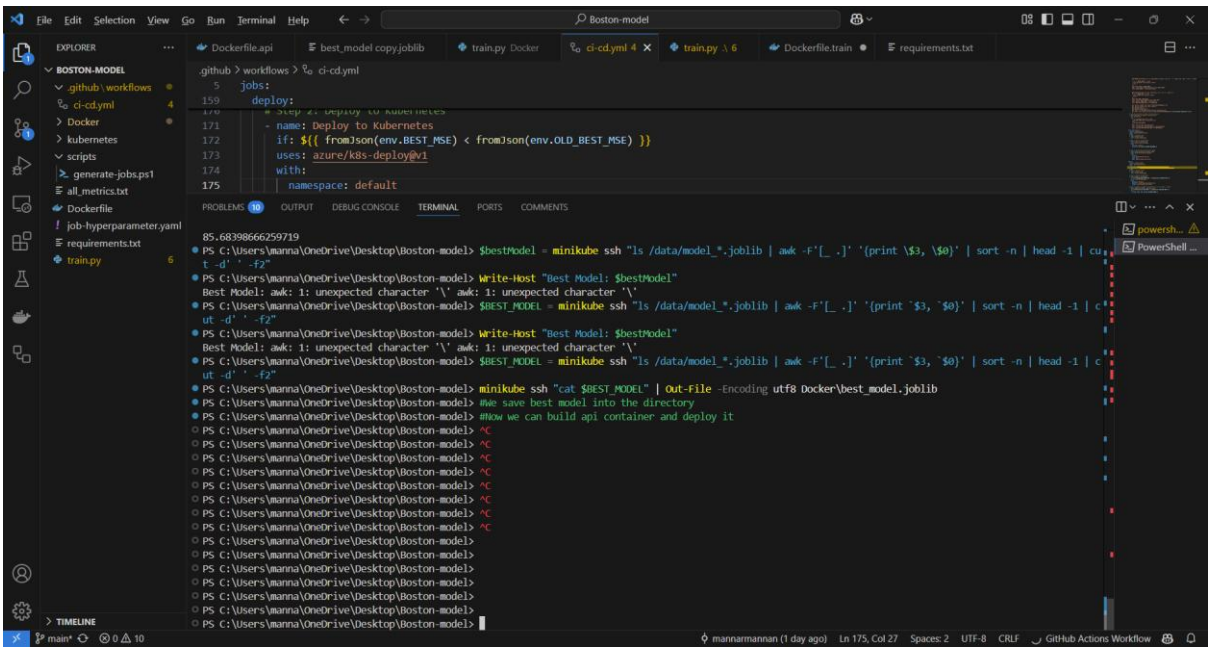
9. Retrieve Best Model

>>minikube ssh "ls /data/model_*.joblib | sort -V | head -1"

>>$BEST_MODEL = minikube ssh "ls /data/model_*.joblib | awk -F'[_ .]' '{print `$3, `$0}' | sort -n | head -1 | cut -d' ' -f2"

>>minikube ssh "cat $BEST_MODEL" | Out-File -Encoding utf8 Docker\best_model.joblib

Uses version-sorting to identify the model with lowest MSE from filename patterns.



Task 2: Build a CI/CD Pipeline ::Refer repo: Github

CI/CD workflow file for all the above instruction to automate the process

CI/CD Workflow Enhancements:

- Added clearer stage separation (build → tune → deploy)
- Fixed environment variable handling for MSE comparisons
- Improved error handling in model retrieval steps
- Added explicit skip-ci tags to prevent pipeline loops

Key Pipeline Improvements:

1. Parallel Build Stages
   Simultaneous image building and dependency installation for faster execution
2. Atomic Artifact Handling
   Models and metrics stored with MSE in filenames for easy version comparison
3. Rolling Update Strategy
   Conditional deployment only when new model outperforms previous best MSE
4. Persistent Metric Tracking
   Commits best MSE to repository for historical comparison across runs

NOTE: Major efforts not taken in deployment stage

```yaml
name: Model CI/CD Pipeline with Hyperparameter Tuning

on: [push]

jobs:
  build-and-train:
    runs-on: ubuntu-latest
    steps:
    # Step 1: Checkout code
    - name: Checkout code
      uses: actions/checkout@v2

    # Step 2: Set up Docker Buildx
    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2

    # Step 3: Log in to Docker Hub
    - name: Log in to Docker Hub
      uses: docker/login-action@v2
      with:
        username: mannarn
        password: ${{ secrets.DOCKER_PASSWORD }}

    # Step 4: Build and push Train Docker image
```

```yaml
    - name: Build and push Train Docker image
      uses: docker/build-push-action@v3
      with:
        context: .
        file: Docker/Dockerfile.train
        push: true
        tags: mannarn/model-train:latest

  hyperparameter-tuning:
    runs-on: ubuntu-latest
    needs: build-and-train
    outputs:
      best_mse: ${{ steps.getting_results.outputs.best_mse }}
      old_best_mse: ${{ steps.retrieve_old.outputs.old_best_mse }}
    steps:
    # Step 1: Checkout code
    - name: Checkout code
      uses: actions/checkout@v2

    # Step 2: Set up Python
    - name: Set up Python
      uses: actions/setup-python@v2
      with:
        python-version: '3.8'

    # Step 3: Install dependencies
    - name: Install dependencies
      run: pip install -r requirements.txt

    # Step 4: Set up Minikube
    - name: Set up Minikube
      run: |
        curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
        chmod +x minikube
        sudo mv minikube /usr/local/bin/
        minikube start --driver=docker --cpus=4 --memory=8192mb

    # Step 5: Generate Kubernetes Jobs for hyperparameter tuning
    - name: Generate Kubernetes Jobs
      shell: pwsh
      run: |
        chmod +x scripts/generate-jobs.ps1
        ./scripts/generate-jobs.ps1

    # Step 6: Wait for Jobs to complete
    - name: Wait for Jobs to complete
      run: |
```

```
        kubectl wait --for=condition=complete --timeout=1000s job --all ||
true

    # Step 7: Check Job and Pod Status
    - name: Check Job and Pod Status
      run: |
        kubectl get jobs
        kubectl get pods

    # Step 8: Retrieve logs of failed pods
    - name: Retrieve logs of failed pods
      run: |
        for pod in $(kubectl get pods --field-selector=status.phase=Failed -o
jsonpath='{.items[*].metadata.name}'); do
          echo "Logs for pod $pod:"
          kubectl logs $pod
        done

    # Step 9: Get the results of hyperparameter tuning
    - name: Getting results
      id: getting_results
      run: |
        BEST_MODEL=$(minikube ssh -- "ls /data/model_*.joblib | sort -t'_' -
k2,2n | head -1")

        if [ -z "$BEST_MODEL" ]; then
          echo "No model file found in /data/"
          exit 1
        fi
        echo "Best Model: $BEST_MODEL"

        BEST_MSE=$(minikube ssh "cat /data/metrics_*.txt | sort -n | head -1")
        if [ -z "$BEST_MSE" ]; then
          echo "No MSE file found in /data/"
          exit 1
        fi

        echo "Best MSE: $BEST_MSE"
        echo "best_mse=$BEST_MSE" >> $GITHUB_OUTPUT

        git config --global user.name "github-actions[bot]"
        git config --global user.email "github-actions@github.com"

        git add Docker/best_model.joblib
        git commit --allow-empty -m "Uploading model [skip ci]"
        git remote set-url origin https://x-access-token:${{
secrets.GITHUB_TOKEN }}@github.com/mannarn/Boston-model.git
        git push origin main
```

```yaml
  # Step 10: Retrieve old model's best MSE score
  - name: Retrieve old model's best MSE score
    id: retrieve_old
    run: |
      if [ -f old_best_mse.txt ]; then
        OLD_BEST_MSE=$(cat old_best_mse.txt)
      else
        OLD_BEST_MSE=999999
      fi
      echo "Old Best MSE: $OLD_BEST_MSE"
      echo "old_best_mse=$OLD_BEST_MSE" >> $GITHUB_OUTPUT

build-and-push-api:
  runs-on: ubuntu-latest
  needs: hyperparameter-tuning
  steps:
  # Step 1: Checkout code
  - name: Checkout code
    uses: actions/checkout@v2

  # Step 2: Log in to Docker Hub
  - name: Log in to Docker Hub
    uses: docker/login-action@v2
    with:
      username: mannarn
      password: ${{ secrets.DOCKER_PASSWORD }}

  # Step 3: Build and push API Docker image
  - name: Build and push API Docker image
    uses: docker/build-push-action@v3
    with:
      context: .
      file: Docker/Dockerfile.api
      push: true
      tags: mannarn/model-api:latest

deploy:
  runs-on: ubuntu-latest
  needs: [build-and-push-api, hyperparameter-tuning]
  env:
    BEST_MSE: ${{ needs.hyperparameter-tuning.outputs.best_mse }}
    OLD_BEST_MSE: ${{ needs.hyperparameter-tuning.outputs.old_best_mse }}
  steps:
    # Step 1: Checkout code
    - name: Checkout code
      uses: actions/checkout@v2
```

```
      # Step 2: Convert CRLF to LF
      - name: Convert CRLF to LF
        run: |
          find . -type f -exec sed -i 's/\r$//' {} \;

      # Step 3: Print old and new MSE scores
      - name: Check MSE Scores
        id: check_mse
        run: |
          echo "Old Best MSE: $OLD_BEST_MSE"
          echo "New Best MSE: $BEST_MSE"
          if awk "BEGIN {exit !($BEST_MSE < $OLD_BEST_MSE)}"; then
            echo "New model is better. Deploying..."
            echo "deploy=true" >> $GITHUB_ENV
          else
            echo "New model is not better. Skipping deployment."
            echo "deploy=false" >> $GITHUB_ENV
          fi

      # Step 4: Set up Minikube and Kubernetes context
      - name: Set up Minikube
        if: env.deploy == 'true'
        run: |
          # Install Minikube
          curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
          sudo install minikube-linux-amd64 /usr/local/bin/minikube

          # Start Minikube cluster
          minikube start --driver=docker --cpus=4 --memory=8192mb

          # Verify Minikube status
          minikube status

          # Configure kubectl to use Minikube's context
          mkdir -p ~/.kube
          minikube kubectl -- config view --flatten > ~/.kube/config
          echo "KUBECONFIG=~/.kube/config" >> $GITHUB_ENV

      # Step 5: Deploy to Kubernetes if new MSE is lower
      - name: Deploy to Kubernetes
        if: env.deploy == 'true'
        uses: azure/k8s-deploy@v1
        with:
          namespace: default
          manifests: |
            kubernetes/deployment.yaml
            kubernetes/flask_api_service.yaml
```

```
          images: mannarn/model-api:latest
          kubectl-version: "latest"

      # Step 6: Update old model's best MSE score if new model is better
      - name: Update old model's best MSE score
        if: env.deploy == 'true'
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          echo "$BEST_MSE" > old_best_mse.txt
          git config --global user.name "github-actions[bot]"
          git config --global user.email "github-actions@github.com"
          git add old_best_mse.txt
          git commit --allow-empty -m "Updating old_best_mse [skip ci]"
          git remote set-url origin https://x-access-token:${{
secrets.GITHUB_TOKEN }}@github.com/mannarn/Boston-model.git
          git push origin main
```

Conclusion

In this project, I successfully developed and deployed an end-to-end machine learning pipeline for predicting house prices using the XGBoost regression model. The implementation involved hyperparameter tuning through Kubernetes Jobs, ensuring efficient parallel execution of multiple training trials.

By leveraging Kubernetes for parallelized hyperparameter tuning, I optimized model performance while maintaining scalability and automation. The integration of GitHub Actions enabled a robust CI/CD pipeline, automating the build, tuning, and deployment processes seamlessly.