

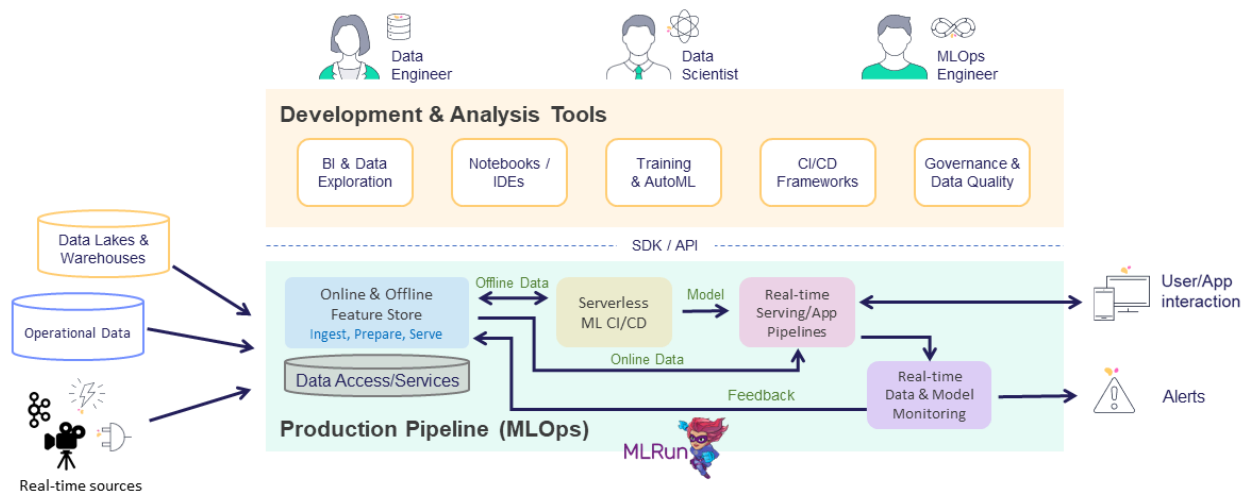
STT Assignment-9

Tanish Yelgoe 23110328, Laksh Jain 23110185

Github: <https://github.com/tanishy7777/MLRun-CI-CD-Workflow>

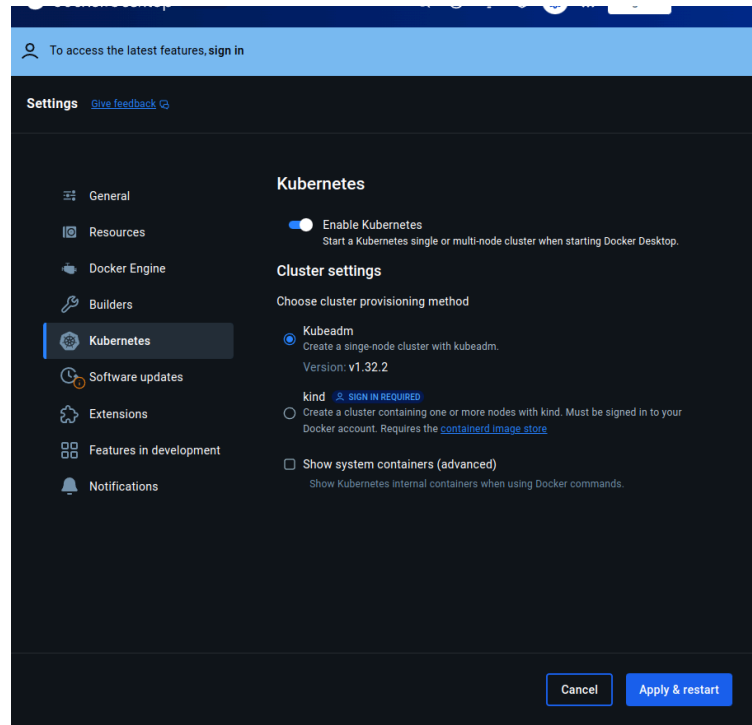
Architecture of MLRun

Source: <https://docs.mlrun.org/en/v1.1.1/>



1. Installation

- Docker Desktop: Installed from previous assignment.
- Enabling Kubernetes from Docker Desktop



```
tanish ➜ ~ 18:30
→ curl -LO https://dl.k8s.io/release/v1.32.0/bin/linux/amd64/kubectl
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %             Dload  Upload   Total   Spent    Left   Speed
100 138    100    138    0     0    418      0  0:00:02  0:00:02  0:00:02  419
100 54.6M  100 54.6M    0     0   23.3M      0  0:00:02  0:00:02  0:00:00  31.5M

tanish ➜ ~ 18:31
→ curl -LO https://dl.k8s.io/release/v1.32.0/bin/linux/amd64/kubectl.sha256
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %             Dload  Upload   Total   Spent    Left   Speed
100 138    100    138    0     0    398      0  0:00:02  0:00:02  0:00:02  400
100 64    100    64    0     0    141      0  0:00:02  0:00:02  0:00:00  141

tanish ➜ ~ 18:31
→ echo "$(cat kubectl.sha256) kubectl" | sha256sum --check
kubectl: OK

tanish ➜ ~ 18:31
→ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
[sudo] password for tanish:

tanish ➜ ~ 18:32
→ kubectl version --client
Client Version: v1.32.0
Kustomize Version: v5.5.0
```

c. Installing kubectl and verifying using sha256

d. Installing helm from binary releases

Installation and Upgrading

Download Helm v3.17.3. The common platform binaries are here:

- [MacOS amd64 \(checksum / 20ef8df4671349a6fc556a621be1170dd709c6c0cf5f7e83a2d9fb0515fd97fc\)](#)
- [MacOS arm64 \(checksum / 89aec43ce07b06239f1bba4a6507236bb48ae487bc5065a8e254d3ce58a16997\)](#)
- [Linux amd64 \(checksum / ee88b3c851ae6466a3de507f7be73fe94d54cbf2987cbaa3d1a3832ea331f2cd\)](#)
- [Linux arm \(checksum / 60d76d1e12d3e058a9e9a8209eff748a6fab5948028a1f0860f48e141243d33d\)](#)
- [Linux arm64 \(checksum / 7944e3defd386c76fd92d9e6fec5c2d65a323f6adc19bfb5e704e3eee10348e\)](#)
- [Linux i386 \(checksum / 51742d78c066437e23b3ca98370df341f9136b408381fe5a150d70b9d9bf24d7\)](#)
- [Linux ppc64le \(checksum / b821885a502b2fa159e3ef3afe9cde6e6c9876d4a623f18868829c3ee4a3c64c\)](#)
- [Linux s390x \(checksum / 71a9c6058e29a7eef0bc72a61843ccbad11997e383dd3e13e1a591ddff8598\)](#)
- [Linux riscv64 \(checksum / 4e4563d43a593e11533024c7a0ddb79fb7d1dec85f9a9f8417ed1bacda0d7d0e\)](#)
- [Windows amd64 \(checksum / 8ea93e2f6285e649dede583ac90ff8cdb938ca53ec6cf5fe909f2303fbc22d96\)](#)
- [Windows arm64 \(checksum / 70ce9dfdbc1ce6142626a829dbdc5920405146f3ce4dc6f6e6739dd308cc7baf\)](#)

```
helm-v3.17.3-linux-amd64.tar.gz
# tanish ~/ 18:37
→ tar -zxvf helm-v3.17.3-linux-amd64.tar.gz
linux-amd64/
linux-amd64/LICENSE
linux-amd64/helm
linux-amd64/README.md
# tanish ~/ 18:38
→ mv linux-amd64/helm /usr/local/bin/helm
mv: cannot move 'linux-amd64/helm' to '/usr/local/bin/helm': Permission denied
# tanish ~/ 18:38
→ sudo mv linux-amd64/helm /usr/local/bin/helm
[sudo] password for tanish:
# tanish ~/ 18:38
→ helm help
The Kubernetes package manager

Common actions for Helm:

- helm search: search for charts
- helm pull: download a chart to your local directory to view
- helm install: upload the chart to Kubernetes
- helm list: list releases of charts

Environment variables:

| Name | Description
```

e. MLRun installation

```
# tanish ~/ 18:41
→ kubectl create namespace mlrun
namespace/mlrun created

# tanish ~/ 18:41
→ helm repo add mlrun-ce https://mlrun.github.io/ce
"mlrun-ce" has been added to your repositories

# tanish ~/ 18:41
→ helm repo list
NAME      URL
mlrun-ce  https://mlrun.github.io/ce

# tanish ~/ 18:41
→ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "mlrun-ce" chart repository
Update Complete. *Happy Helming!*

# tanish ~/ 18:41
→
```

Create access token

A personal access token is similar to a password except you can have many tokens and revoke access to each one at any time. [Learn more](#)

Access token description
kubectlkey

Expiration date
None

Optional
Access permissions
Public Repo Read-only

Public Repo Read-only tokens allow to view, search, and pull images from any public repositories.

Cancel Generate

Type this

```
kubectl --namespace mlrun create secret docker-registry registry-credentials
--docker-server https://index.docker.io/v1/ --docker-username
your_docker_username --docker-password your_docker_password
--docker-email your_email_id
```

or

```
kubectl --namespace mlrun create secret docker-registry registry-credentials \

--docker-server https://registry.hub.docker.com/ \

--docker-username tanishy7777\

--docker-password <password> \

--docker-email tanishyelgoe777@gmail.com
```

```
tanish ~ 18:41
→ docker login -u tanishy7777
Password:
Login Succeeded

tanish ~ 18:49
→ kubectl --namespace mlrun create secret docker-registry registry-credentials \
--docker-server https://registry.hub.docker.com/ \
--docker-username tanishy7777\
--docker-password [REDACTED] \
--docker-email tanishyelgoe777@gmail.com

secret/registry-credentials created

tanish ~ 18:51
→ helm --namespace mlrun install mlrun-ce --wait --timeout 1800s --set global.registry.url=index.docker.io/saileshpanda97
try.secretName=registry-credentials --set kube-prometheus-stack.enabled=false mlrun-ce/mlrun-ce
```

To see the progress bar run:

watch -n 2 '

TOTAL=\$(kubectl get pods -n mlrun --no-headers 2>/dev/null | grep -v Completed | wc -l)

READY=\$(kubectl get pods -n mlrun --no-headers 2>/dev/null | grep "Running" | wc -l)

PERCENT=\$((100 * READY / (TOTAL == 0 ? 1 : TOTAL)))

echo -ne "🚀 MLRun Setup Progress: \$READY/\$TOTAL pods running (\$PERCENT%)\n"

,

```
Every 2.0s: deathwave: 06:56:19 PM IST
🚀 MLRun Setup Progress: 10/21 pods running (47%)
```

This refreshed every 2 seconds

```
secret/registry-credentials created
tanish ~ 18:51
→ helm --namespace mlrun install mlrun-ce --wait --timeout 1800s --set global.registry.url=index.docker.io/saileshpanda97 --set global.registry.secretName=registry-credentials --set kube-prometheus-stack.enabled=false mlrun-ce/mlrun-ce

NAME: mlrun-ce
LAST DEPLOYED: Thu Apr 10 18:52:41 2025
NAMESPACE: mlrun
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
You're up and running!

Jupyter UI is available at:
localhost:30040

Nuclio UI is available at:
localhost:30050

MLRun UI is available at:
localhost:30060

MLRun API is available at:
localhost:30070

Minio UI is available at:
localhost:30090
- username: minio
- password: minio123

Minio API is available at:
localhost:30080

Pipelines UI is available at:
localhost:30100

Happy ML0PSing!!! :)
tanish ~ 19:06
```

Installed successfully

Can check the status of pods with

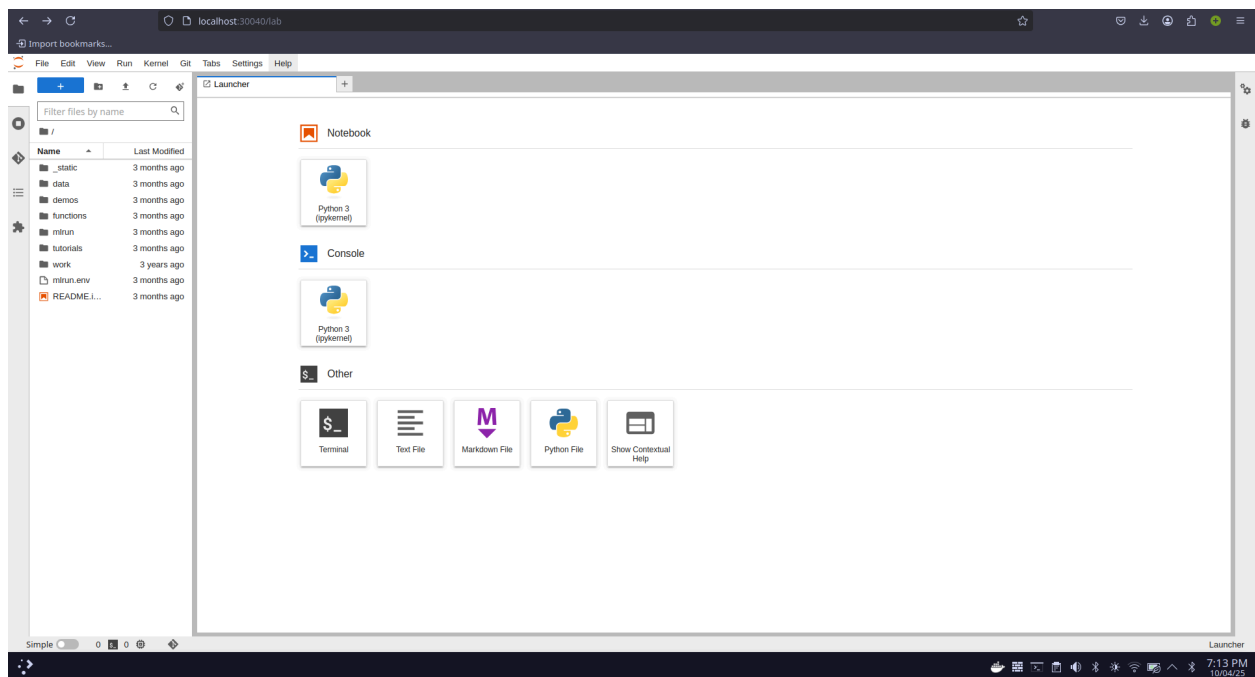
```
→ kubectl -n mlrun get pods
NAME                                READY   STATUS    RESTARTS   AGE
metadata-envoy-deployment-86b9c85656-kc49d    1/1     Running   0           16m
metadata-grpc-deployment-5dfd66bb8d-fpvfn      1/1     Running   7 (9m42s ago)  16m
metadata-writer-7b7fc9f477-bc6dh              1/1     Running   3 (5m30s ago)  16m
minio-65c9b8dc44-xbqpq                      1/1     Running   0           16m
ml-pipeline-5d7998ffc5-qv2xp                 1/1     Running   7 (5m53s ago)  16m
ml-pipeline-persistenceagent-69446f4bdd-pmbx6  1/1     Running   4 (5m56s ago)  16m
ml-pipeline-scheduledworkflow-8564b98f5b-zn2qw 1/1     Running   0           16m
ml-pipeline-ui-5d9ff6dc69-r42s5              1/1     Running   0           16m
ml-pipeline-viewer-crd-6668f68cd7-8kv5m       1/1     Running   0           16m
ml-pipeline-visualizationserver-6878c5b646-9m94p 1/1     Running   0           16m
mlrun-api-chief-6d6868867f-w5h9x             2/2     Running   1 (6m8s ago)  16m
mlrun-db-6c85956db5-zzqpb                   1/1     Running   0           16m
mlrun-jupyter-85768bb958-q6g9v               1/1     Running   0           16m
mlrun-ui-744bcb7f44-trpdk                    1/1     Running   0           16m
mpi-operator-97dd956c5-q5j4                  1/1     Running   0           16m
mysql-9dc78bdd7-7v9j7                        1/1     Running   0           16m
nuclio-controller-94fff55d7-f6gc4             1/1     Running   0           16m
nuclio-dashboard-b6cfbc88d-djljk             1/1     Running   0           16m
spark-operator-controller-855dc75b69-hhg6s    1/1     Running   0           16m
spark-operator-webhook-587b54f56-7n6jc        1/1     Running   0           16m
workflow-controller-789b47d74-cnwnv           1/1     Running   0           16m
```

All are running so,

<http://localhost:30040>

or

(for linux, ip addr show)



Jupyter UI

```
%pip install mlrun scikit-learn~=1.5.1 numpy~=1.26.4
```

1. Create a mlrun project(10%):

```
[10]: import numpy

[11]: import mlrun

project = mlrun.new_project("stt9", "./", user_project=True, description="STT AI assignment 9", overwrite=True)

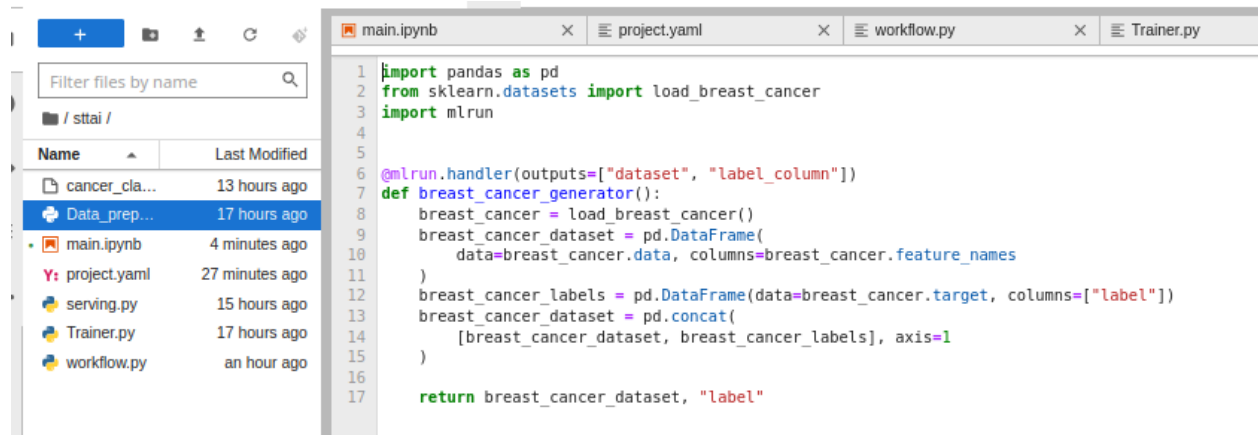
> 2025-04-11 09:43:05,905 [info] Overwriting project (by deleting and then creating): {"name":"stt9-jovyan"}
> 2025-04-11 09:43:05,931 [info] Waiting for project to be deleted: {"project name":"stt9-jovyan"}
> 2025-04-11 09:43:08,972 [info] Project deleted: {"project name":"stt9-jovyan"}
> 2025-04-11 09:43:09,023 [info] Created and saved project: {"context":"./","from_template":null,"name":"stt9-jovyan","overwrite":true,"save":true}

[12]: !python --version

Python 3.9.13
```

2. Create the following Python script:

- Data_prep.py: This fetches data using sklearn.datasets import `load_breast_cancer`. (Add screenshot of the code)[10 %]

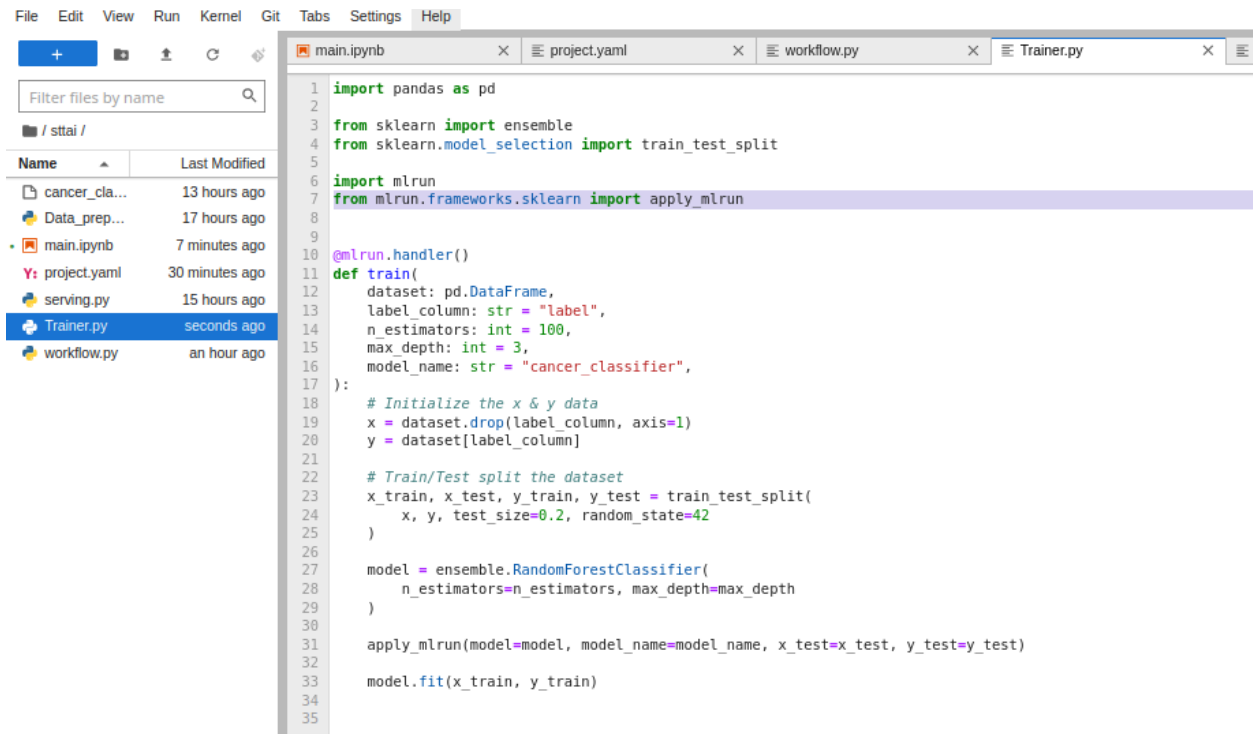


```
1 import pandas as pd
2 from sklearn.datasets import load_breast_cancer
3 import mlrun
4
5
6 @mlrun.handler(outputs=["dataset", "label_column"])
7 def breast_cancer_generator():
8     breast_cancer = load_breast_cancer()
9     breast_cancer_dataset = pd.DataFrame(
10         data=breast_cancer.data, columns=breast_cancer.feature_names
11     )
12     breast_cancer_labels = pd.DataFrame(data=breast_cancer.target, columns=["label"])
13     breast_cancer_dataset = pd.concat(
14         [breast_cancer_dataset, breast_cancer_labels], axis=1
15     )
16
17     return breast_cancer_dataset, "label"
```

Registering the function from Data_prep script in main.ipynb

```
[8]: data_gen_fn = project.set_function(
    "Data_prep.py",
    name="Data-prep",
    kind="job",
    image="mlrun/mlrun",
    handler="breast_cancer_generator",
)
```

- Trainer.py: Split the data into train test (10% test data). Train a model using the training data. Use Random forest classifier. Wrap the model with `apply_mlrun` from `mlrun.frameworks.sklearn`. (Add screenshot of the code)[10 %]



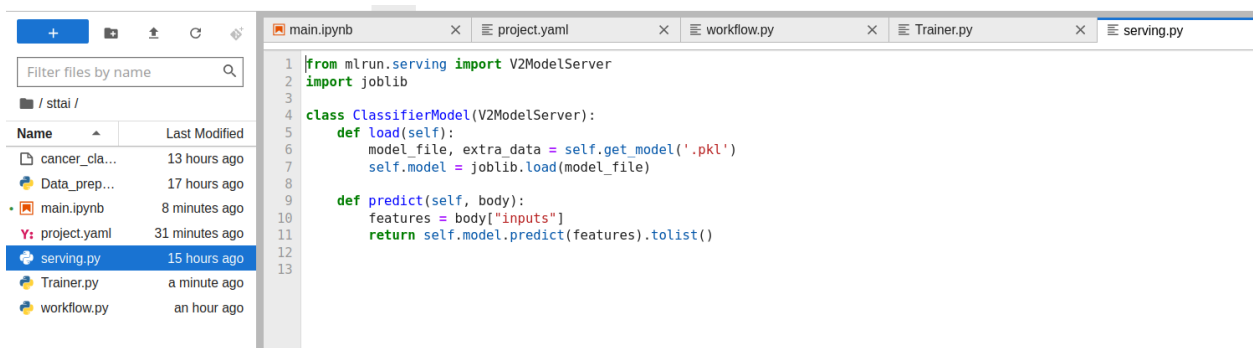
Registering function from Trainer.py in main.ipynb

```

[25]: trainer = project.set_function(
      "Trainer.py", name="trainer", kind="job", image="mlrun/mlrun", handler="train"
      )

```

- c. serving.py: Create a model class that will inherit from `mlrun.serving.V2ModelServer`, enabling automatic support for model lifecycle methods like `load()` and `predict()`. (Add screenshot of the code)[10 %]



Registering function from serving.py in main.ipynb


```
[15]: project.set_function(
    "serving.py",
    name="serving",
    kind="serving",
    image="mlrun/mlrun"
)
```

```
[15]: <mlrun.runtimes.nuclio.serving.ServingRuntime at 0x7f5ca21d8e20>
```

```
[16]: project.set_workflow("main", "workflow.py")
```

- d. workflow.py: Create a Python script that defines an MLRun pipeline using the `@dsl.pipeline` decorator. It should include the following steps:
 - i. Data Ingestion(Add screenshot of the code) [3 %]
 - ii. Model Training: Experiment with different hyperparamters(`n_estimators`: [10, 100, 200], `max_depth`=[2, 5, 10]). Keep max accuracy as selector. (Add screenshot of the code)[4 %]
 - iii. Model Deployment: Deploy the model into the base kubernetes image(`mlrun/mlrun`) using `mlrun.deploy_function` (Add screenshot of the code)[3 %]

Data ingestion, model training with different hyperparameters and max accuracy as selector. Also model deployment into base kubernetes.

```

1 from kfp import dsl
2 import mlrun
3
4 @dsl.pipeline(name="Breast Cancer Classifier Pipeline")
5 def pipeline():
6     project = mlrun.get_current_project()
7
8     data_prep = project.run_function(
9         "Data-prep",
10        outputs=["dataset"]
11    )
12
13    train = project.run_function(
14        "trainer",
15        outputs=["model"],
16        inputs={"dataset": data_prep.outputs["dataset"]},
17        hyperparams={
18            "n_estimators": [10, 100, 200],
19            "max_depth": [2, 5, 10]
20        },
21        selector="max.accuracy",
22    )
23
24    deploy = mlrun.deploy_function(
25        "serving",
26        models=[{"key": "my_model", "model_path": train.outputs["model"], "class_name": "ClassifierModel"}],
27    )
28
  
```

Expanded screenshots from workflow.py:

Data ingestion:

```
data_prep = project.run_function(
    "Data-prep",
    outputs=["dataset"]
)|
```

Model training with different hyperparameters

```
train = project.run_function(
    "trainer",
    outputs=["model"],
    inputs={"dataset": data_prep.outputs["dataset"]},
    hyperparams={
        "n_estimators": [10, 100, 200],
        "max_depth": [2, 5, 10]
    },
    selector="max.accuracy",
)
```

Model deployment using `mlrun.deploy_function`

```
deploy = mlrun.deploy_function(
    "serving",
    models=[{"key": "my_model", "model_path": train.outputs["model"], "class_name": "ClassifierModel"}],
)
```

Registering function from workflow.py in main.ipynb

```
[30]: project.set_workflow("main", "workflow.py")
```

```
[32]: project.run("main", watch=True)
```

```
> 2025-04-10 22:31:12,621 [warning] it is recommended to use k8s secret (specify secret_name),
_secret_key directly is unsafe
> 2025-04-10 22:31:12,630 [warning] it is recommended to use k8s secret (specify secret_name),
_secret_key directly is unsafe
> 2025-04-10 22:31:12,633 [warning] it is recommended to use k8s secret (specify secret_name),
_secret_key directly is unsafe
> 2025-04-10 22:31:13,075 [info] Pipeline submitted successfully: {"id": "6e1675be-1052-4281-b15
e": "stt9-jovyan-main 2025-04-10 22-31-12"}
> 2025-04-10 22:31:13,075 [info] Pipeline run id=6e1675be-1052-4281-b154-35c19cf1987d, check UI
Workflow started in project stt9-jovyan id=6e1675be-1052-4281-b154-35c19cf1987d
```

[click here to view progress](#)

Pipeline running (id=6e1675be-1052-4281-b154-35c19cf1987d), [click here](#) to view the details in MLRun UI

```
> 2025-04-10 22:31:13,105 [info] Started run workflow stt9-jovyan-main with run id = 6e1675be-
```

3. Run the workflow using **project.run**:

1. Add the screenshot of the workflow graph [10 %]

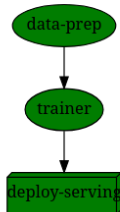
```
[16]: project.set_workflow("main", "workflow.py")

[17]: project.run("main", watch=True)

> 2025-04-11 09:43:10,443 [warning] it is recommended to use k8s secret (specify secret_name), specifying the aws_access_key/aws_secret_key directly is unsafe
> 2025-04-11 09:43:10,447 [warning] it is recommended to use k8s secret (specify secret_name), specifying the aws_access_key/aws_secret_key directly is unsafe
> 2025-04-11 09:43:10,449 [warning] it is recommended to use k8s secret (specify secret_name), specifying the aws_access_key/aws_secret_key directly is unsafe
> 2025-04-11 09:43:10,884 [info] Pipeline submitted successfully: {"id": "440b8e8c-7ba2-4eb8-94aa-7097f7315b82", "pipeline_name": "stt9-jovyan-main 2025-04-11 09-43-10"}
> 2025-04-11 09:43:10,885 [info] Pipeline run id=440b8e8c-7ba2-4eb8-94aa-7097f7315b82, check UI for progress
Workflow started in project stt9-jovyan id=440b8e8c-7ba2-4eb8-94aa-7097f7315b82
click here to view progress

Pipeline running (id=440b8e8c-7ba2-4eb8-94aa-7097f7315b82), click here to view the details in MLRun UI

> 2025-04-11 09:43:11,037 [info] Started run workflow stt9-jovyan-main with run id = '440b8e8c-7ba2-4eb8-94aa-7097f7315b82' by kfp engine
> 2025-04-11 09:43:11,038 [info] Waiting for pipeline run completion: {"project": "stt9-jovyan", "run_id": "440b8e8c-7ba2-4eb8-94aa-7097f7315b82"}
```



Run Results

[info] Workflow 440b8e8c-7ba2-4eb8-94aa-7097f7315b82 finished, state=Succeeded

click the hyper links below to see detailed results

uid	start	state	kind	name	parameters	results
...15cac7e3	Apr 11 09:43:48	completed	run	trainer		best_iteration=7 accuracy=0.9736842105263158 f1_score=0.9790209790209791 precision_score=0.9722222222222222 recall_score=0.9859154929577465
...d683ee76	Apr 11 09:43:21	completed	run	data-prep		label_column=label

[17]: 440b8e8c-7ba2-4eb8-94aa-7097f7315b82

[10]: project.save()

The screenshot shows the MLRun web interface. On the left, a sidebar contains navigation icons. The main area displays the workflow graph with three steps: 'data-prep', 'trainer', and 'deploy-serving', connected by arrows. To the right of the graph, the 'serving' job details are shown, including a timestamp 'Apr 11, 2025, 03:13:10 PM' and a 'Build Log' section. The build log contains the following text:

```
2025-04-11 09:44:41 (info) Deploying function
2025-04-11 09:44:41 (info) Building
2025-04-11 09:44:41 (info) Staging files and preparing base images
2025-04-11 09:44:41 (warn) Using user provided base image, runtime interpreter version is provided by the base image
2025-04-11 09:44:41 (info) Building processor image
2025-04-11 10:16:17 (info) Build complete
2025-04-11 10:18:15 (info) Function deploy complete
```

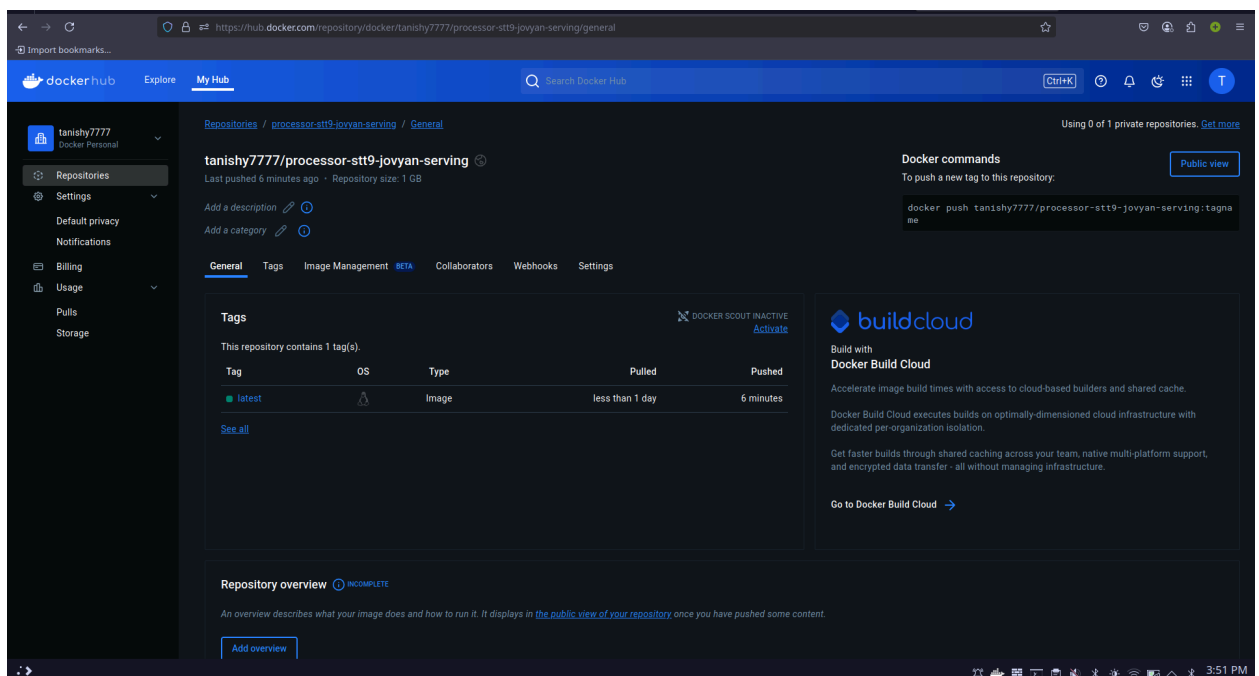
Logs:

Running the following command shows:

```
kubectll logs
nuclio-kanikojob.processorstt9jovyan-servinglatest.aunpjz
vbgv6nd -n mlrun -f
```

This shows image was pushed to dockerhub

```
and want to suppress this warning.
INFO[1806] Taking snapshot of full filesystem...
INFO[1828] No files were changed, appending empty layer to config. No layer added to image.
INFO[1828] CMD [ "processor" ]
INFO[1828] No files changed in this command, skipping snapshotting.
INFO[1828] Pushing layer 'index.docker.io/tanishy7777/processor-stt9-jovyan-serving/cache:cc97660ca30be12977f711dd198cf5d86f11c3d5ae7258c58aee038f14340915' to ca
ow
INFO[1828] Pushing image to index.docker.io/tanishy7777/processor-stt9-jovyan-serving/cache:cc97660ca30be12977f711dd198cf5d86f11c3d5ae7258c58aee038f14340915
WARN[1830] Retrying operation after 1s due to HEAD https://index.docker.io/v2/tanishy7777/processor-stt9-jovyan-serving/cache/manifests/cc97660ca30be12977f711dd
f5d86f11c3d5ae7258c58aee038f14340915: unexpected status code 401 Unauthorized (HEAD responses have no body, use GET for details)
WARN[1832] Retrying operation after 2s due to HEAD https://index.docker.io/v2/tanishy7777/processor-stt9-jovyan-serving/cache/manifests/cc97660ca30be12977f711dd
f5d86f11c3d5ae7258c58aee038f14340915: unexpected status code 401 Unauthorized (HEAD responses have no body, use GET for details)
WARN[1836] Retrying operation after 4s due to HEAD https://index.docker.io/v2/tanishy7777/processor-stt9-jovyan-serving/cache/manifests/cc97660ca30be12977f711dd
f5d86f11c3d5ae7258c58aee038f14340915: unexpected status code 401 Unauthorized (HEAD responses have no body, use GET for details)
WARN[1841] Error uploading layer to cache: failed to push to destination index.docker.io/tanishy7777/processor-stt9-jovyan-serving/cache:cc97660ca30be12977f711dd
cf5d86f11c3d5ae7258c58aee038f14340915: HEAD https://index.docker.io/v2/tanishy7777/processor-stt9-jovyan-serving/cache/manifests/cc97660ca30be12977f711dd198cf5d
1c3d5ae7258c58aee038f14340915: unexpected status code 401 Unauthorized (HEAD responses have no body, use GET for details)
INFO[1841] Pushing image to index.docker.io/tanishy7777/processor-stt9-jovyan-serving:latest
INFO[1879] Pushed 'index.docker.io/tanishy7777/processor-stt9-jovyan-serving:sha256:06adf0cf9836428c98af6676d86f93015b050b22bd263a8c626188ed860edde'
```



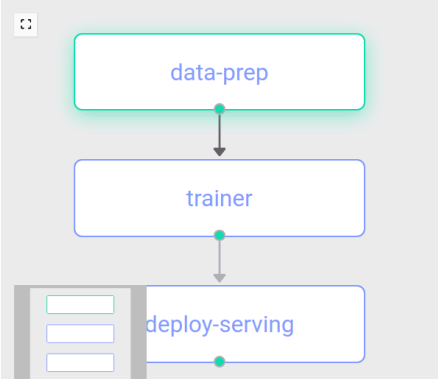
<https://hub.docker.com/repository/docker/tanishy7777/processor-stt9-jovyan-serving/general>

2. Add the screenshot of the Data_prep artifact and take the screenshot of the dataframe. [10 %]

Data_Prep artifact screenshot

Monitor Jobs Monitor Workflows Schedule

← main 2025-04-11 09-43-10



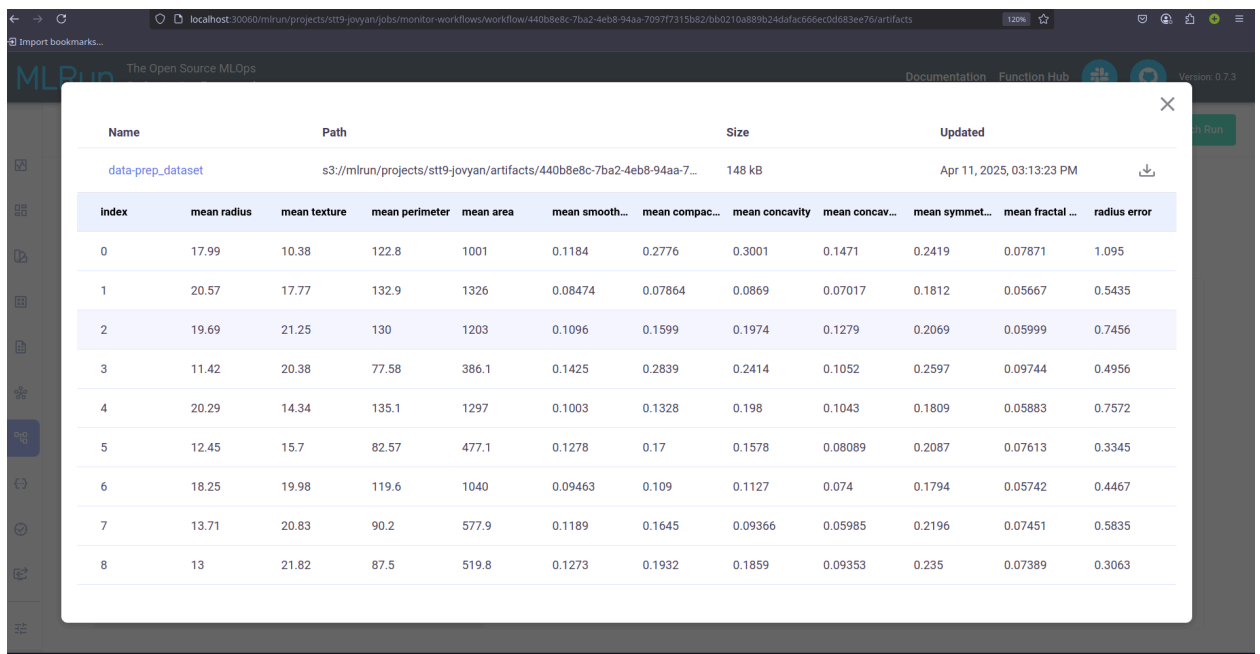
The workflow diagram shows a sequence of steps: 'data-prep' (highlighted with a green border), followed by 'trainer', and then 'deploy-serving'. Arrows indicate the flow from data-prep to trainer, and from trainer to deploy-serving. A small icon of a server rack is shown next to the deploy-serving step.

data-prep
Apr 11, 2025, 03:13:23 PM

Overview Inputs Artifacts Results Logs Pods

Name ↑	Path	Size	Updated
data-prep_dataset	s3://mlrun/projects/stt9-jovyan/artifacts/4...	148 kB	Apr 11, 2025,...

Dataframe screenshot



The screenshot shows a dataframe artifact named 'data-prep_dataset' with 12 columns. The first column is 'index' (0-8). The next 10 columns are feature metrics: 'mean radius', 'mean texture', 'mean perimeter', 'mean area', 'mean smooth...', 'mean compac...', 'mean concavity', 'mean concav...', 'mean symmet...', and 'mean fractal ...'. The final column is 'radius error'. The data is presented in a table with alternating light blue and white rows.

Name	Path	Size	Updated
data-prep_dataset	s3://mlrun/projects/stt9-jovyan/artifacts/440b8e8c-7ba2-4eb8-94aa-7...	148 kB	Apr 11, 2025, 03:13:23 PM

index	mean radius	mean texture	mean perimeter	mean area	mean smooth...	mean compac...	mean concavity	mean concav...	mean symmet...	mean fractal ...	radius error
0	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871	1.095
1	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435
2	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999	0.7456
3	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744	0.4956
4	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883	0.7572
5	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613	0.3345
6	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742	0.4467
7	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451	0.5835
8	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389	0.3063

3. Add the screenshot of the confusion-matrix artifact of train.py [10 %]

Trainer artifacts

Monitor JobsMonitor WorkflowsSchedule

← main 2025-04-11 09-43-10

data-prep

↓

trainer

↓

deploy-serving

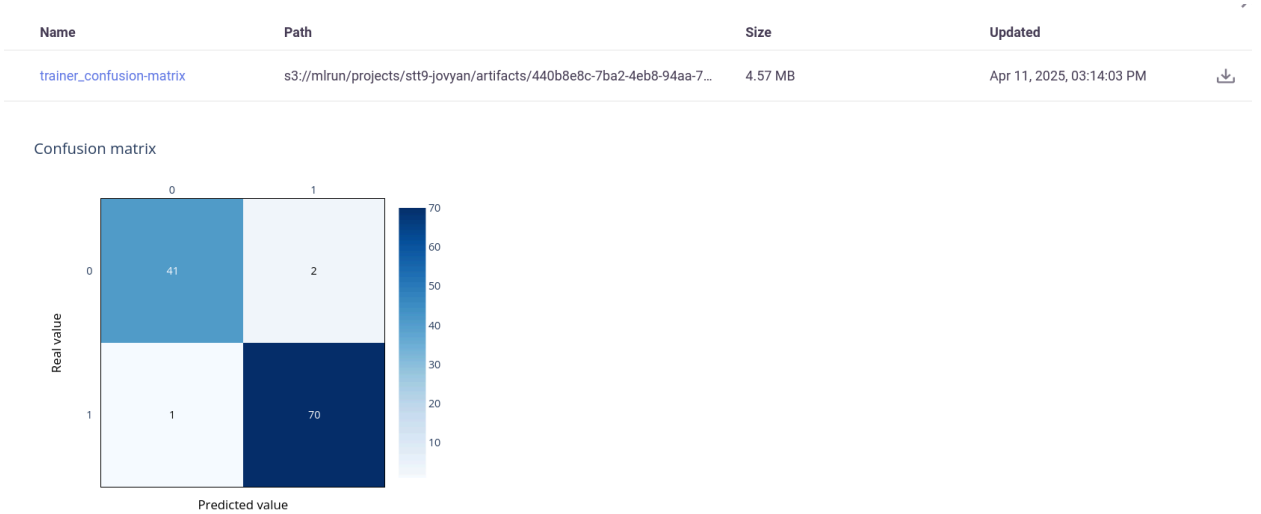
trainer

Apr 11, 2025, 03:14:10 PM

OverviewInputsArtifactsResultsLogsPods

Name ↑	Path	Size	Updated	
cancer_classifier	s3://mlrun/projects/stt9-jovyan/artifacts/4...	37 kB	Apr 11, 2025,...	
trainer_calibration-cu...	s3://mlrun/projects/stt9-jovyan/artifacts/4...	4.57 MB	Apr 11, 2025,...	
trainer_confusion-m...	s3://mlrun/projects/stt9-jovyan/artifacts/4...	4.57 MB	Apr 11, 2025,...	
trainer_feature-impor...	s3://mlrun/projects/stt9-jovyan/artifacts/4...	4.57 MB	Apr 11, 2025,...	
trainer_roc-curves	s3://mlrun/projects/stt9-jovyan/artifacts/4...	4.57 MB	Apr 11, 2025,...	

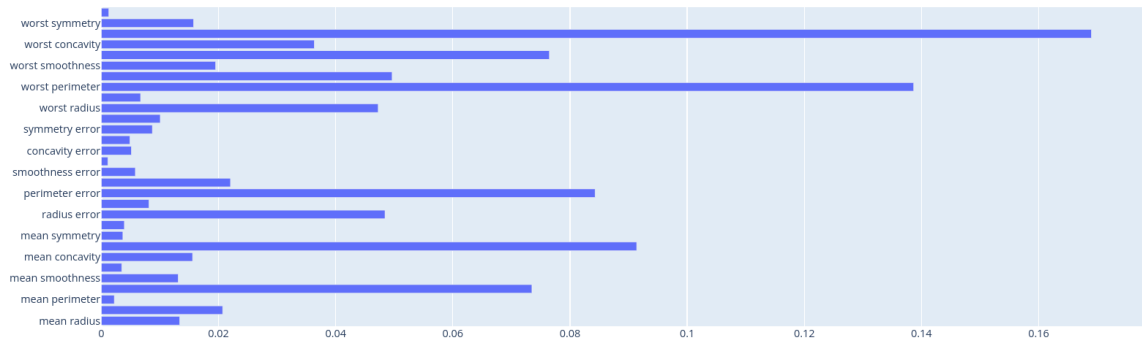
Screenshot of the confusion matrix artifact



4. Add the screenshot of feature selection artifact. [10 %]
Feature selection/importance artifact

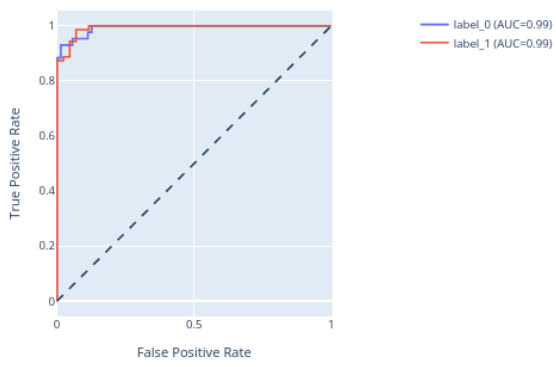
Name	Path	Size	Updated
trainer_feature-importance	s3://mlrun/projects/stt9-jovyan/artifacts/440b8e8c-7ba2-4eb8-94aa-7...	4.57 MB	Apr 11, 2025, 03:14:02 PM

📷 🔍 + 📄 🗨 📧 📄 📄 📄 📄



ROC curve

Name	Path	Size
trainer_roc-curves	s3://mlrun/projects/stt9-jovyan/artifacts/6e1675be-1052-4281-b154-35c19cf1987d/trainer/2/roc-curves.html	4.57 MB

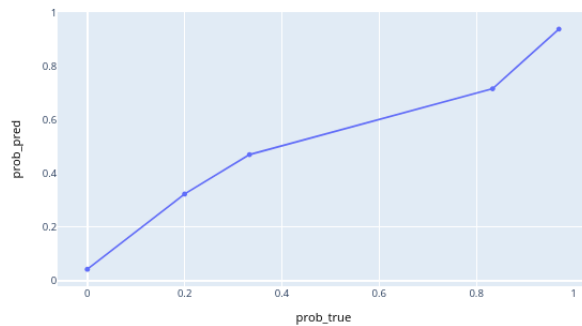


trainer_calibration-curve

s3://mlrun/projects/stt9-jovyan/artifacts/6e1675be-1052-4281-b154-35c19cf1987d/trainer/2/calibration-curve.h...



Calibration Curve



trainer_test_set

s3://mlrun/projects/stt9-jovyan/artifacts/6e1675be-1052-4281-b154-35c19cf1987...

50.2 kB

Apr 11, 2025, 04:01:52 AM



index	mean radius	mean texture	mean perimeter	mean area	mean smooth...	mean compac...	mean concavity	mean concav...	mean symmet...	mean fractal ...	r
0	12.47	18.6	81.09	481.9	0.09965	0.1058	0.08005	0.03821	0.1925	0.06373	C
1	18.94	21.31	123.6	1130	0.09009	0.1029	0.108	0.07951	0.1582	0.05461	C
2	15.46	19.48	101.7	748.9	0.1092	0.1223	0.1466	0.08087	0.1931	0.05796	C
3	12.4	17.68	81.47	467.8	0.1054	0.1316	0.07741	0.02799	0.1811	0.07102	C
4	11.54	14.44	74.65	402.9	0.09984	0.112	0.06737	0.02594	0.1818	0.06782	C

Initializing git repo and pushing

Commands:

```
!git init
```

```
!git status
```

```
!git add .
```

```
!git config --global user.email "tanishyelgoe777@gmail.com"
```

```
!git config --global user.name "Tanish Yelgoe"
```

```
!git commit -m "MLRun project"
```

```
!git status
```

```
!git remote add origin
```

```
https://tanishy7777:<token>@github.com/tanishy7777/MLOps-Assignment.git
```

```
!git push origin master
```

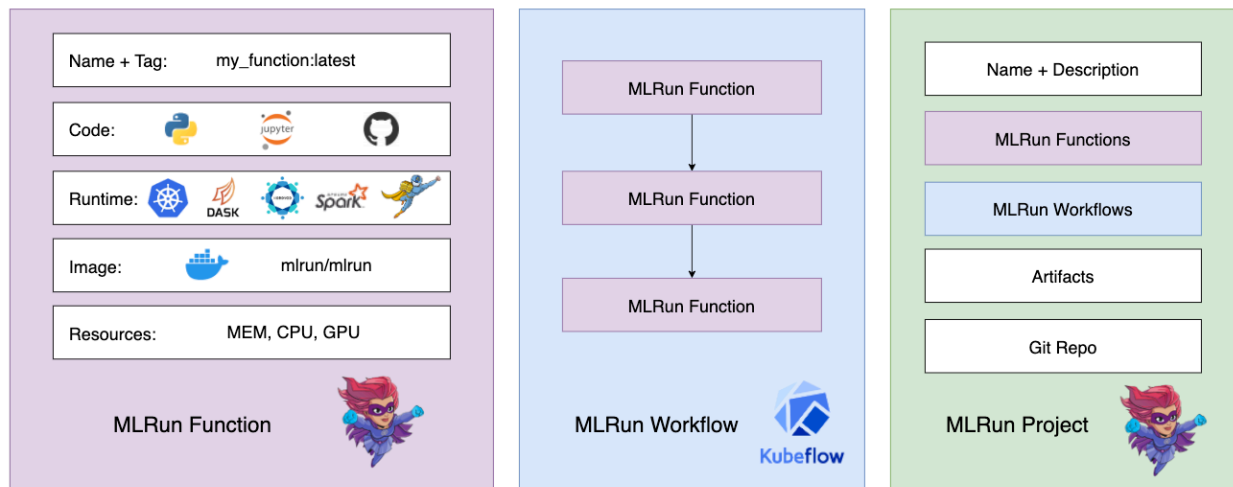
Github repo: <https://github.com/tanishy7777/MLOps-Assignment>

Some important commands for reference:

1. `kubectl get pods -n mlrun`
2. `kubectl get pods -n mlrun --sort-by=.metadata.creationTimestamp`
3. `kubectl delete pod <podname>-n mlrun`
4. `kubectl port-forward -n mlrun svc/nuclio-dashboard 8070:8070 &`
`export NUCLIO_DASHBOARD_URL=http://localhost:8070`
5. `kubectl delete pods --field-selector status.phase=Failed -n mlrun`
6. `kubectl delete pod --field-selector=status.phase=Succeeded -n mlrun`
7. `mlrun logs <pod> -n mlrun`

Note: (Not to be graded, added for self reference and learning)

MLRun Functions



Advantages of MLRun

1. **Serverless Functions:** You write the code, MLRun handles running it anywhere
2. **Auto-logging:** It tracks everything (metrics, artifacts) automatically
3. **Kubernetes Integration:** You can scale effortlessly on cloud clusters
4. **Versioning:** You can go back to old function versions if needed
5. **Reusability via Function Hub:** Use and share functions with others
6. **CI/CD Friendly:** Works great with Git and automation pipelines
7. **Can also be used for hyper parameter tuning**

MLRun Function Comprises of

Code: Your Python script

Required packages: pandas, scikit-learn, etc.

Resources: "I need 2 CPUs and 4GB RAM"

Metadata: Name, labels, version hash

Storage: "Save results to this location"

Deployment config: "Run on spot instance" or not

Why Use Functions?

Let's say you wrote some code to train a model. Instead of just running it like a normal Python script, MLRun lets you wrap it as a function, and now:

1. You can run it on bigger machines with more memory or GPUs (using Kubernetes)
2. You can track everything it does (using MLRun's logging)
3. You can share it with others using a function hub
4. You can version it (MLRun keeps track of changes so you can go back to old versions)

Function Lifecycle

1. You write some code
2. Wrap it as a function in MLRun
3. Register the function in a project (projects keep things organized)
4. Run the function — either in Jupyter, or on a cloud cluster