Designing a production-grade Realtime ML Inference Endpoint

-> Scope of our application
-> Exploring the functionalities[Demo]
-> Project structure and components
-> Discussing Common Project Essentials
-> Project and Credentials Configurations Format
-> Deep dive in the workflow
-> I/O Format of the Inference Endpoint
-> Packaging and Running the application

Scope of our application

- A Python Flask server serving predictions from trained serialized machine learning models based on a micro-service architecture.
- Requests containing testing feature data.
- A Cloud native application build with Docker and compatible with orchestration systems like Docker Swarm, AWS ECS, Kubernetes.

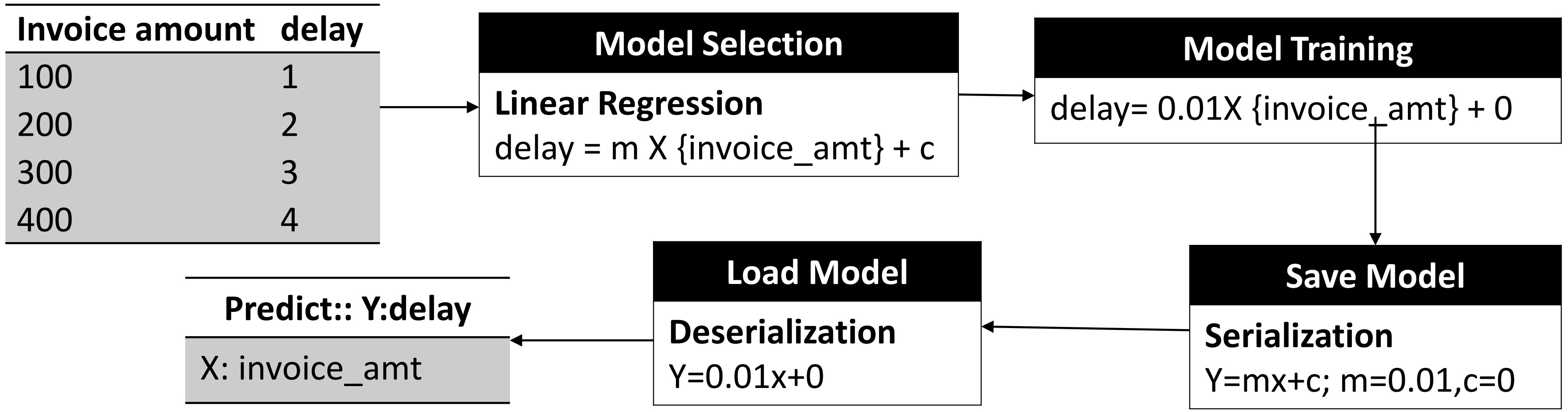


Fig 1: An Example of an ML Pipeline

Exploring the functionalities

- It is able to handle parallel requests for performing predictions in python real time.
- It supports and can extend various storage options like AWS S3, Azure Blob, NAS for saving or loading the model artifacts.
- Can implement any algorithm in python which can serve the predictions within synchronous time duration of a HTTP connection.

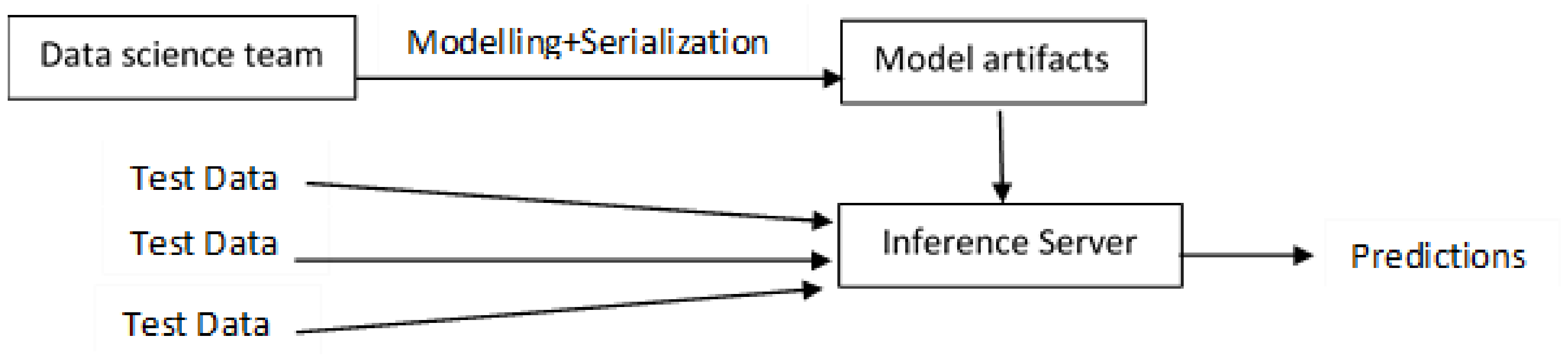


Fig 2: ML Inference Endpoint

Project structure and components

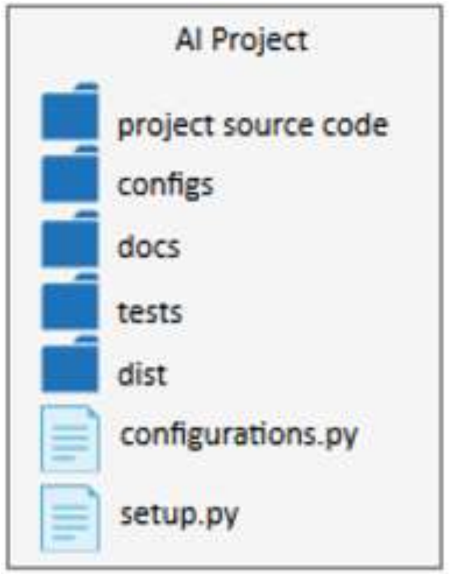


Fig 3: Project Structure

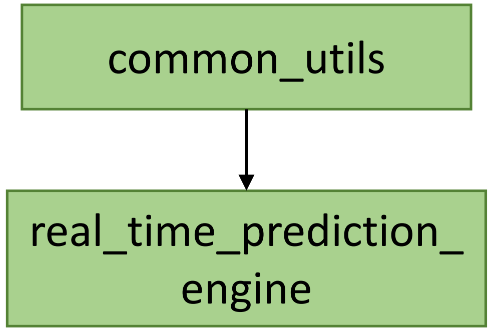


Fig 4: Project Dependencies

- The project source code contains the actual code.
- Configs folder contains various project level configurations like logging levels.
- The docs folder contains documentation created using ***sphinx*** tool as shown***:*** sphinx-build <docs-source-path> <docs-html-path>
- The tests folder contain all the test cases related to the project. The test cases are written using ***pytest*** module. Following command asserts the testcases: python -m pytest -v –s
- The dist folder contains the binary distribution of the project containing the wheel file. It can be used to resolve dependencies. Following command shows how to upload a wheel file in a repository using ***twine***. twine upload --repository-url <repo-url> -u <user> -p <pass> <whl file>.
- The setup.py is a module which is used to create the wheel file using the following command: python setup.py sdist bdist_wheel. The dependent package can be enlisted in the requirements.txt file and can be installed via ***pip*** with an additional url index of the repo. pip install -r requirements.txt --index-url <repo-url>.
- The configurations.py initializes the credentials config and the project configurations according to the SDLC environment it is running in.

Discussing common_utils Project Essentials

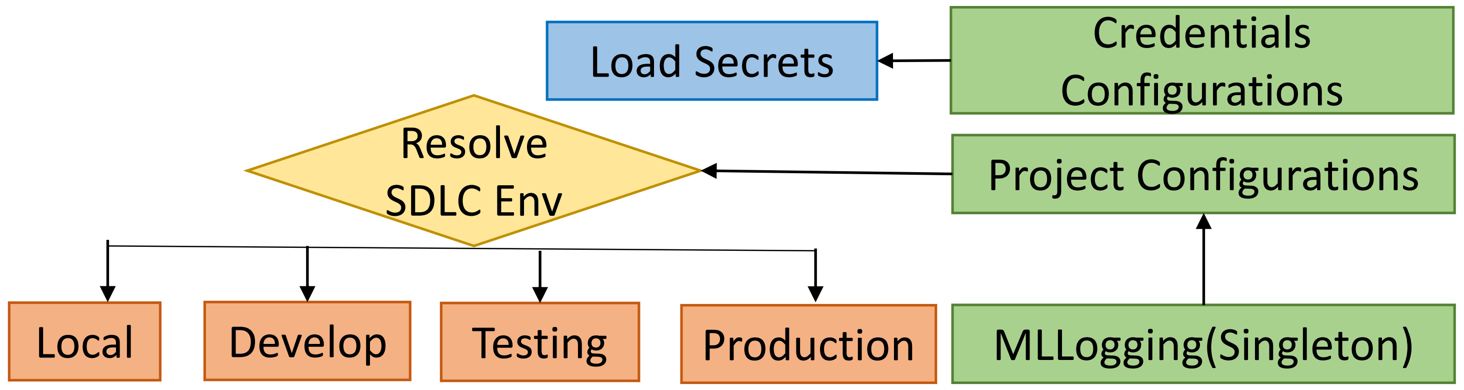


Fig 5: Components for Configuration Setup of Project and Credentials

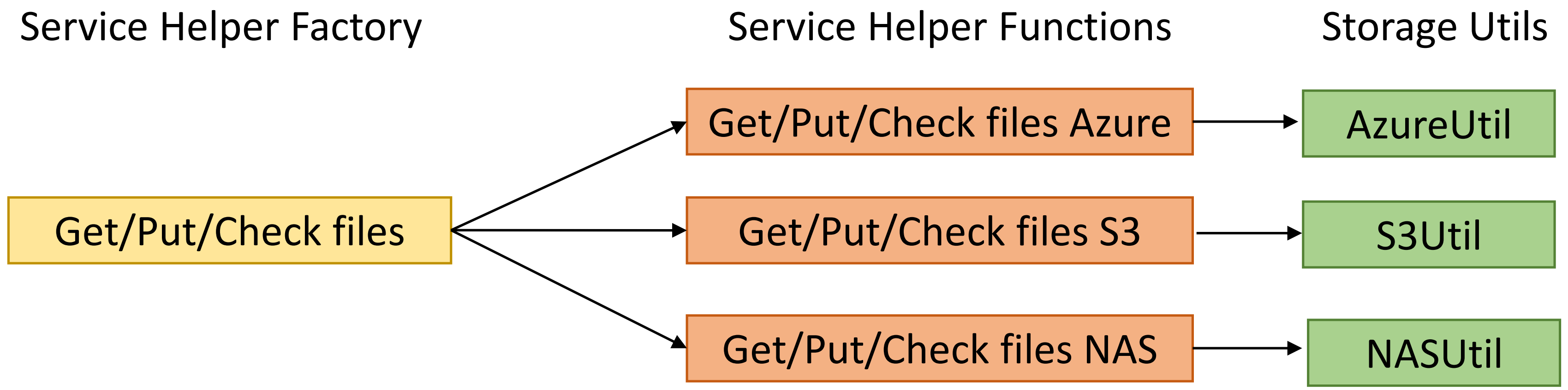


Fig 6: File Handling abstraction in common_utils

Project and Credentials Configurations Format

```
[S3]
S3_local_folder_path =
/ml_webservice/storage/

[Azure]
Azure_local_folder_path =
/ml_webservice/storage/
storage_account_name=asc

[NAS]
NAS_local_folder_path =
/ml_webservice/storage/

[Logs]
log_file_path = /ml_webservice/logs/
log_file_prefix = ml_webservice_log
log_level_file = DEBUG
log_level_console = DEBUG
logger_default_name =
DataScienceLogging
logger_type= FileHandler
log_time_rotator = d
log_time_interval = 1
logging_channel=File
```

Fig 7: Project Configurations

```
[AWS Credentials]
access_key=ABCDEFGH
secret_key=XXXXXXXXX

[Azure Credentials]
account_key=xxxxxxxxxxxxxxx
```

Fig 8: Credentials Configurations

Deep dive in the workflow

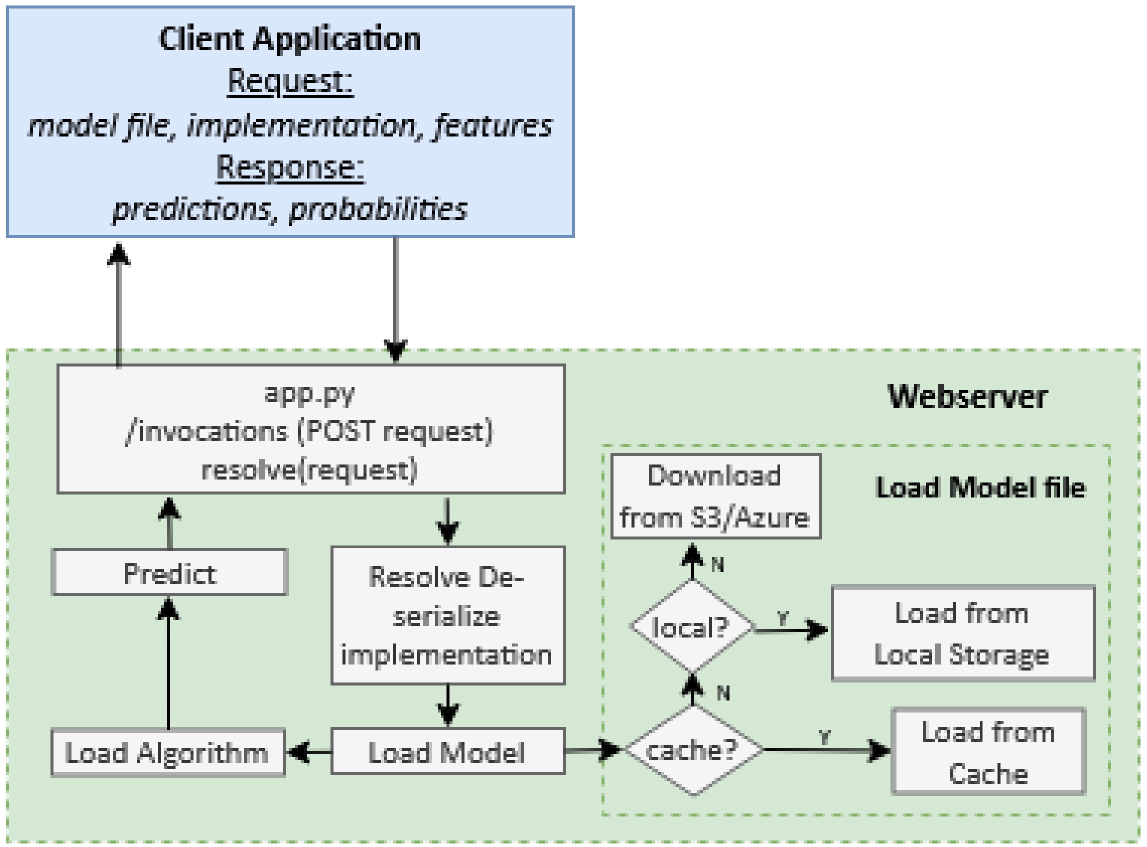


Fig 9: Workflow for inferencing predictions

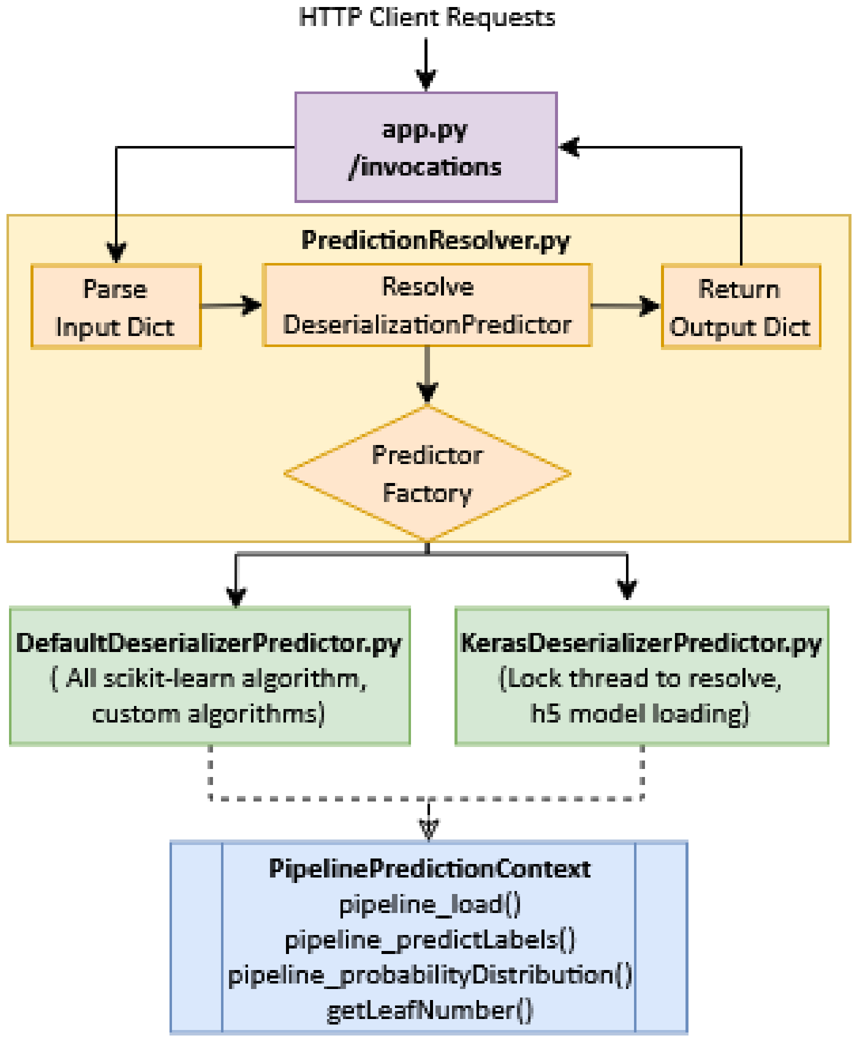
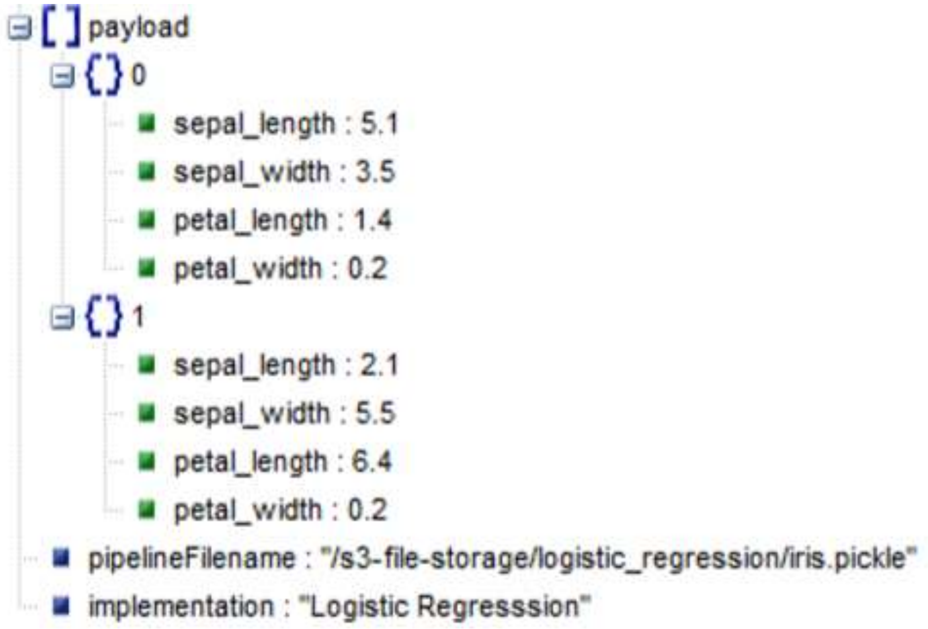

HTTP Client Requests
app.py
/invocations
PredictionResolver.py
Parse Input Dict
Resolve DeserializationPredictor
Return Output Dict
Predictor Factory
DefaultDeserializerPredictor.py
(All scikit-learn algorithm, custom algorithms)
KerasDeserializerPredictor.py
(Lock thread to resolve, h5 model loading)
PipelinePredictionContext
pipeline_load()
pipeline_predictLabels()
pipeline_probabilityDistribution()
getLeafNumber()

Fig 10: Modules involved for inferencing predictions

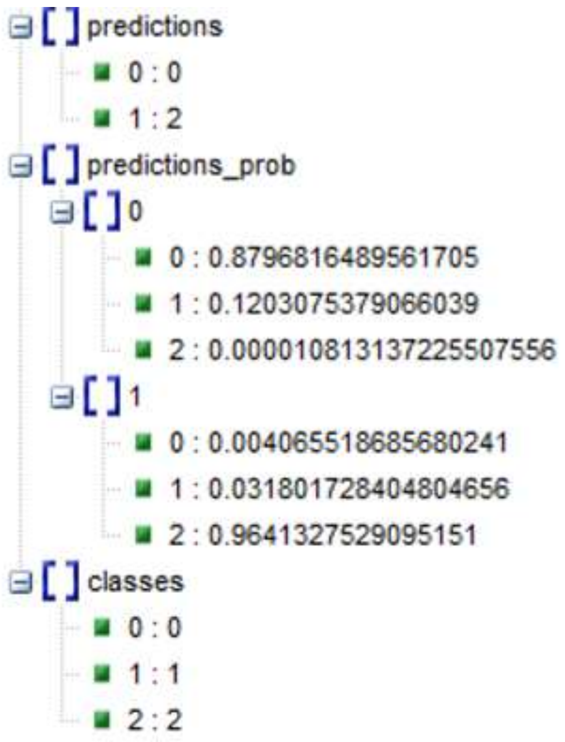
I/O Format of the Inference Endpoint

Input Format:
{'payload': [{'sepal_length': 5.1, 'sepal_width': 3.5, 'petal_length': 1.4, 'petal_width': 0.2}, {'sepal_length': 2.1, 'sepal_width': 5.5, 'petal_length': 6.4, 'petal_width': 0.2}], 'pipelineFilename': '/s3-file-storage/logistic_regression/iris.pickle', 'implementation': 'Logistic Regresssion’}
Output Format:
{"predictions": [0, 2], "predictions_prob": [[0.8796816489561705, 0.1203075379066039, 1.0813137225507556e-05], [0.004065518685680241, 0.031801728404804656, 0.9641327529095151]], "classes": [0, 1, 2]}



```
[ ] payload
  { } 0
    sepal_length : 5.1
    sepal_width : 3.5
    petal_length : 1.4
    petal_width : 0.2
  { } 1
    sepal_length : 2.1
    sepal_width : 5.5
    petal_length : 6.4
    petal_width : 0.2
  pipelineFilename : "/s3-file-storage/logistic_regression/iris.pickle"
  implementation : "Logistic Regresssion"
```

Fig 11: Input Format



```
[ ] predictions
  0 : 0
  1 : 2
[ ] predictions_prob
  [ ] 0
    0 : 0.8796816489561705
    1 : 0.1203075379066039
    2 : 0.000010813137225507556
  [ ] 1
    0 : 0.004065518685680241
    1 : 0.031801728404804656
    2 : 0.9641327529095151
[ ] classes
  0 : 0
  1 : 1
  2 : 2
```

Fig 12: Output Format

Packaging and Running the application

```
FROM python:3.6
WORKDIR /ml_/prediction_engine/
COPY requirementwebservices.txt /ml_webservice/prediction_engine/
RUN pip install -r requirements.txt  --index-url https://pypi.org/simple/  --
extra-index-url http://192.168.0.104:8081/repository/$env/simple --trusted-host
192.168.0.104
COPY prediction_engine /ml_webservice/prediction_engine/
EXPOSE 8080
ENTRYPOINT python3.6 ./app.py
```

Fig 13: Dockerfile for building image of real_time_prediction_engine

```
Inside Root of the project:
#Build the Image
> docker-compose build
#Run the Image
> docker-compose up
```

Fig 14: Running the application

```
version: '3'

services:
    inference-engine:
        build: ./
        image: python-prediction-engine
        container_name: predictionServer
        environment:
        - DEPLOY_ENV=developement
        - HOST=0.0.0.0
        - PORT=8080
        ports:
        - 8080:8080
        volumes:
        - E:/ml_resources/storage/:/ml_webservice/storage/
        - E:/ml_resources/logs/:/ml_webservice/logs/
        - E:/ml_resources/credentials.properties:/ml_webservice/prediction_engine/configs/credentials.properties
```

Fig 15: Docker-compose for building image of real_time_prediction_engine

Questions?

Realtime ML Inference Endpoint is a synchronous mechanism for processing predictions.
For Asynchronous pipelines for processing computationally intensive data we have to adapt to worker based executions.
An MLaaS is implemented for both synchronous and asynchronous pipelines using the same architecture.
Code can be found on:
https://github.com/chandimsett/MLEngine/tree/master/real_time_prediction_engine
A compiled version and in-depth work can be found in the book **Data Sciences For Enterprises: Deployment and Beyond**
https://www.amazon.in/Data-Science-Enterprises-Deployment-beyond/dp/9352673352