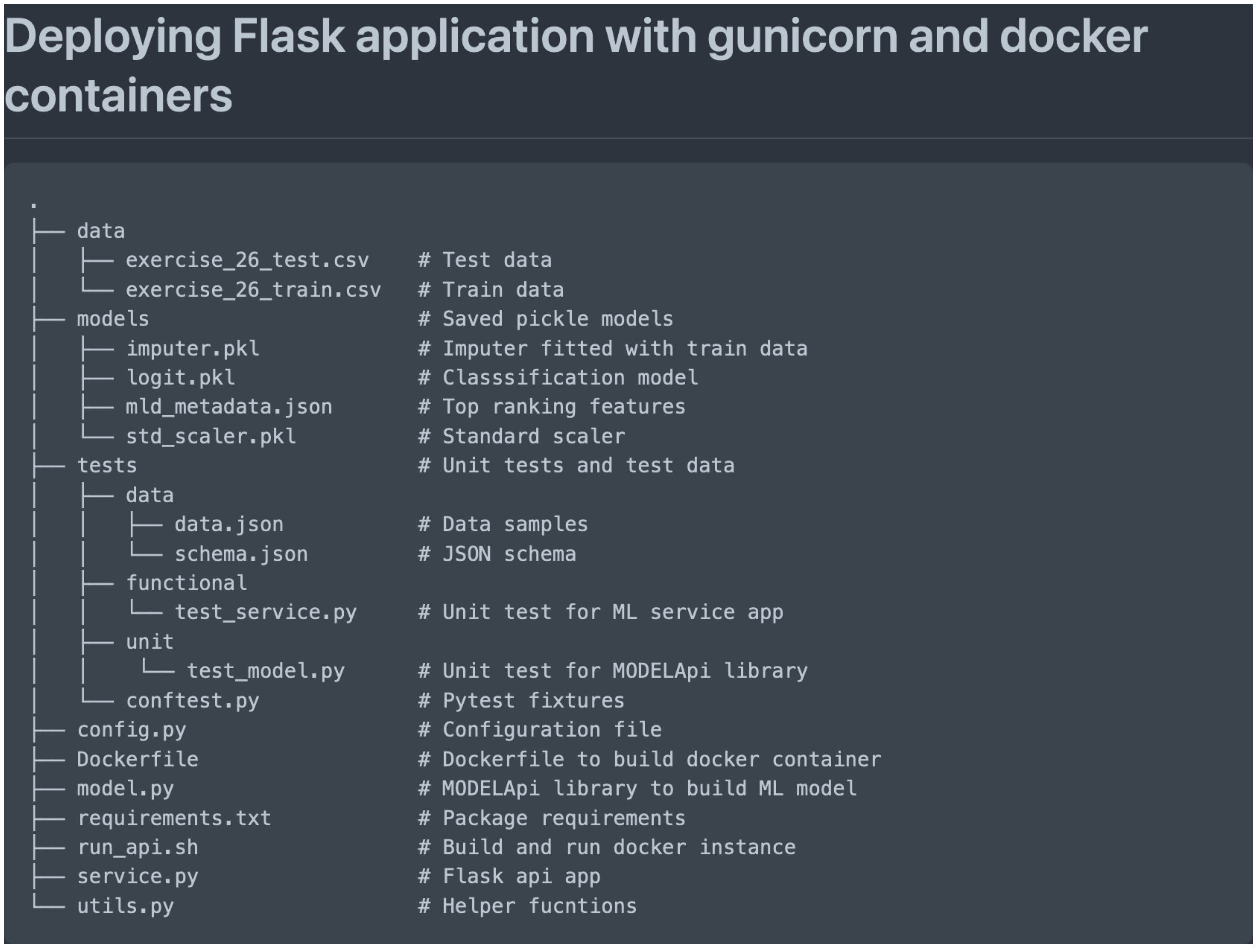

Deploying Flask application with gunicorn and docker containers

```
.
├── data
│   ├── exercise_26_test.csv    # Test data
│   └── exercise_26_train.csv   # Train data
├── models                      # Saved pickle models
│   ├── imputer.pkl             # Imputer fitted with train data
│   ├── logit.pkl               # Classsification model
│   ├── mld_metadata.json       # Top ranking features
│   └── std_scaler.pkl          # Standard scaler
├── tests                       # Unit tests and test data
│   ├── data
│   │   ├── data.json           # Data samples
│   │   └── schema.json         # JSON schema
│   ├── functional
│   │   └── test_service.py     # Unit test for ML service app
│   ├── unit
│   │    └── test_model.py      # Unit test for MODELApi library
│   └── conftest.py             # Pytest fixtures
├── config.py                   # Configuration file
├── Dockerfile                  # Dockerfile to build docker container
├── model.py                    # MODELApi library to build ML model
├── requirements.txt            # Package requirements
├── run_api.sh                  # Build and run docker instance
├── service.py                  # Flask api app
└── utils.py                    # Helper fucntions
```

Project description:

- requirement.txt file allows to install all dependencies for current project.
- model.py allows to build and save Logit model.
- service.py is a Flask API app with two API routes ("/" , GET) and ("/predict" , POST). First route checks the state of the service, the second post route returns predictions with class probability. A POST route includes expects_json decorator that validates the schema of received JSON message and return corresponding error messages when it fails.
- config.py contains all necessary configurations for Flask app and gunicorn web server.
- Dockerfile runs commands to assemble an image.
- run_api.sh is a bash script that builds Docker image, deploys the Flask app to the container and launches it.
- Models directory contains pickle files of the model, imputer, scaler and additional metadata for data preprocessing.

- Test directory contains sample of json data, json schema and pytest files divided into unit and functional tests. (Test can be run with: “python -m pytest”).
- Conftest.py contains pytest fixtures that allows to re-use code.

Deployment optimization for enterprise production and scalability.

For deployment optimization, we can use Ansible which is configuration management and application-deployment tool. All configuration steps can be written in yaml file (playbooks) .

To support scalability, in this project I have decided to use gunicorn (wsgi server) instead of running Flask directly. By running the app directly, we can get a single synchronous process at a time. Gunicorn allows to manage multiple of async processes (workers), where each of which manage its own concurrency.

For enterprise production, docker containers should be managed with Kubernetes clusters or Docker Swarm, which includes auto-scaling based on demand (vertically or horizontally), load balancer and make application fault tolerant.

Nginx web server can be added in front of Kubernetes that would give an access to log file, caching and static files.