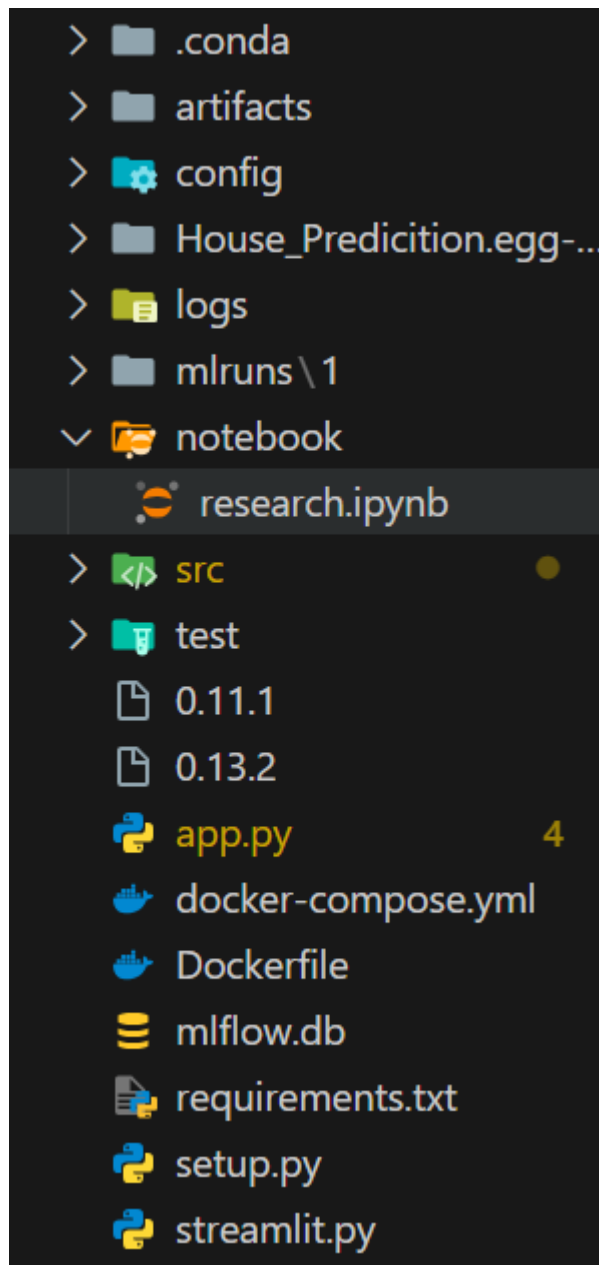


Nama Mentee: Erlangga Dwi Atha

1. Modular Code



Sebagaimana ketentuan yang ada pada tugas Assignment Day 26, saya membuat beberapa folder yang berisi beberapa file yang merupakan pecahan dari kode yang telah dibuat pada [notebook/research.ipynb](#).

- a. Folder artifacts berisikan file-file berupa: best_model.pkl, train.csv

- b. Folder config berisikan file-file berupa: config.yaml (berguna untuk setiap konfigurasi yang ada juga untuk menghindari *hardcode*)
- c. Folder src berisikan beberapa folder:
 - 1) api : Folder untuk menyimpan file app.py untuk implementasi API
 - 2) data: Folder untuk menyimpan pecahan kode dari notebook/research.ipynb berupa data_loader.py dan data_processor.py
 - 3) models: Folder untuk menyimpan pecahan kode dari notebook/research.ipynb berupa model.py dan trainer.py
 - 4) utils: Folder untuk menyimpan alat-alat yang dapat membantu seperti logger.py (membantu untuk mengetahui setiap alur dari pipeline) dan config.py (membantu untuk menghindari *hardcode*)
- d. Folder test berisikan file-file untuk mengetes setiap kode yang sudah dipecah, seperti data_loader.py dan lain-lain.

2. MLflow Integration

MLflow saya atur dan aplikasikan pada file src/models/trainer.py sehingga berguna untuk men-*tracking* setiap proses pelatihan dan pengujian yang dilakukan. Kita bisa mengetahui setiap matriks pada percobaan tertentu. Ini kode pengaplikasiannya:

```
class ModelTrainer:
    """Handles training, evaluation, and selection of the best
    model."""

    def __init__(self, X, y, full_pipeline):
        """
        Initialize ModelTrainer.

        Args:
            X (pd.DataFrame): Processed features.
            y (pd.Series): Target variable.
            pipeline (Pipeline): Preprocessing pipeline.
        """
        try:
            self.X = X
            self.y = y
            self.pipeline = full_pipeline
            self.models = ModelBuilder().get_models()
            self.best_model = None
```

```

        self.experiment_name =
config.get('mlflow.experiment_name', 'house_price_experiment')
        self.artifact_path =
config.get('mlflow.artifact_path', 'model')
        self.tracking_uri =
config.get('mlflow.tracking_uri', 'sqlite:///mlflow.db')
        self._setup_mlflow()
        logger.info("ModelTrainer initialized.")

    except Exception as e:
        logger.error(f"Error during initialization: {e}")
        raise

def _setup_mlflow(self):
    """Set up MLflow tracking."""
    try:
        mlflow.set_tracking_uri(self.tracking_uri)
        # Check if the experiment already exists, if not,
create it
        existing_experiment =
mlflow.get_experiment_by_name(self.experiment_name)
        if existing_experiment is None:
            mlflow.create_experiment(self.experiment_name)
        mlflow.set_experiment(self.experiment_name)
        logger.info("MLflow setup completed.")

    except Exception as e:
        logger.error(f"Error setting up MLflow: {e}")
        raise

    def train_test_split(self, test_size: float = 0.2,
random_state: int = 42):
        """
        Split the data into training and testing sets.

        Args:
            test_size (float): Proportion of data to use for
testing.
            random_state (int): Random seed for
reproducibility.

```

```

        Returns:
            Tuple: Splitted training and testing datasets.
        """
        try:
            logger.info("Splitting data into train and test
sets...")
            return train_test_split(self.X, self.y,
test_size=test_size, random_state=random_state)

        except Exception as e:
            logger.error(f"Error during train-test split:
{e}")
            raise

    def train_and_evaluate(self):
        """
        Train models and evaluate them. Save the best model to
'artifacts/best_model.pkl'.
        """
        try:
            logger.info("Starting training and evaluation...")
            X_train, X_test, y_train, y_test =
self.train_test_split()

            # Convert integer columns to float64
            X_train = X_train.astype({col: 'float64' for col
in X_train.select_dtypes(include=['int']).columns})
            X_test = X_test.astype({col: 'float64' for col in
X_test.select_dtypes(include=['int']).columns})

            best_score = float("inf") # Initialize with a
high value for RMSE
            for model_name, details in self.models.items():
                pipeline = self.pipeline
                pipeline.steps.append(('model',
details['model']))
                logger.info(f"Training model: {model_name}")

```

```

# Integrate preprocessing pipeline with the
model

grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=details['params'],
    cv=2,
    scoring="neg_mean_squared_error",
    n_jobs=-1
)

with mlflow.start_run(run_name=model_name,
nested=True):

    # Log parameters
    mlflow.log_params(details['params'])

    # Fit and evaluate model
    grid_search.fit(X_train, y_train)
    y_pred =
grid_search.best_estimator_.predict(X_test)

    # Compute metrics
    rmse = np.sqrt(mean_squared_error(y_test,
y_pred))

    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    logger.info(f"Model: {model_name} - RMSE:
{rmse:.4f}, R²: {r2:.4f}, MAE: {mae:.4f}")

    # Log metrics
    mlflow.log_metrics({
        "rmse": rmse,
        "mae": mae,
        "r2": r2
    })

    # Get selected features after SelectKBest
    try:
        pipeline.steps.pop() # Remove the
model step temporarily

```

```

        pipeline.fit(X_train, y_train) # Fit
pipeline to apply SelectKBest
        selected_features =
pipeline.named_steps['select_k_best'].get_support(indices=True
)
        selected_feature_names =
np.array(X_train.columns)[selected_features]
        input_example =
X_train[selected_feature_names].head(1)
        logger.info(f"Selected features for
{model_name}: {selected_feature_names}")
        except Exception as e:
            logger.warning(f"Could not retrieve
selected features for {model_name}: {e}")

        # Update the best model if current model
is better

        if rmse < best_score:
            best_score = rmse
            self.best_model =
grid_search.best_estimator_
            logger.info(f"New best model:
{model_name} with RMSE: {rmse:.4f}")

        # Ensure input example contains all
necessary columns, even if some are missing
        input_example =
self._ensure_input_example(X_train, input_example)

        # Log model
        mlflow.sklearn.log_model(
            sk_model=self.best_model,
            artifact_path=self.artifact_path,
            registered_model_name=f"{self.experiment_name}_{model_name}",
            input_example=input_example
        )

        # Save the best model locally
        if self.best_model:

```

```

        joblib.dump(self.best_model,
os.path.join(config.get('artifacts'), "best_model.pkl"))
        logger.info("Best model saved to disk.")

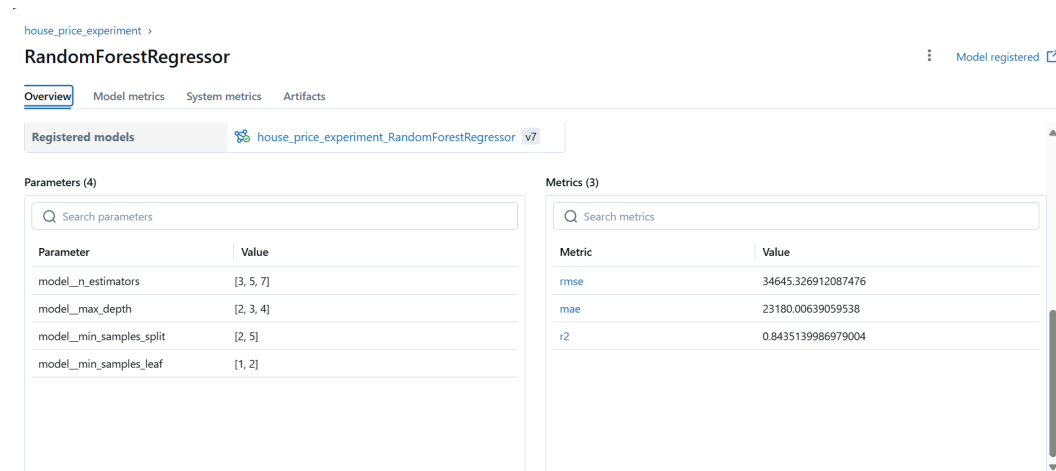
    except Exception as e:
        logger.error(f"Error during training and
evaluation: {e}")
        raise

```

Dapat dilihat bahwa sebelum pengaplikasiannya, kita perlu mensetup beberapa hal sebelum melakukan *tracking*. Setelahnya kita bisa melihat hasil *tracking-an* kita pada local dengan menjalankan file `mlflow.db` di terminal kita dengan perintah:

[`mlflow ui --backend-store-uri sqlite:///mlflow.db`](#)

Lebih dan kurang hasilnya akan terlihat seperti ini:



house_price_experiment > RandomForestRegressor Model registered

Overview | Model metrics | System metrics | Artifacts

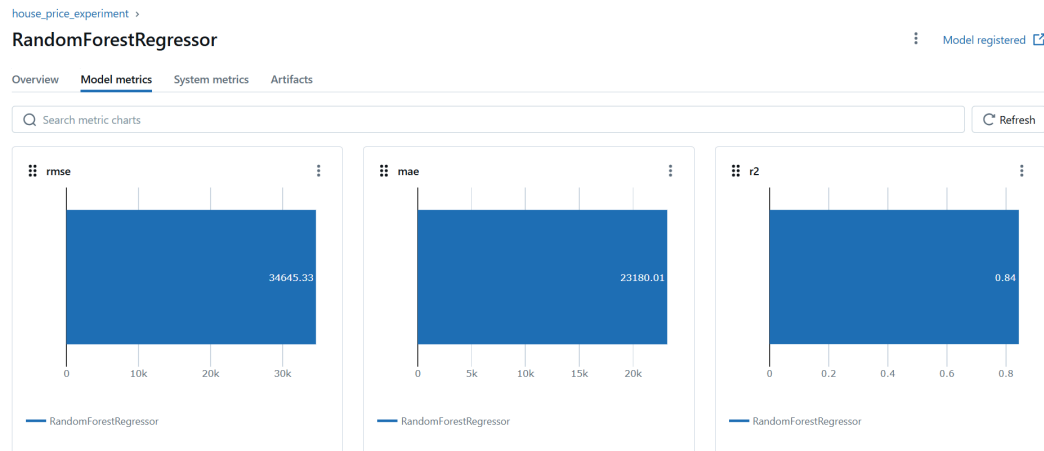
Registered models house_price_experiment_RandomForestRegressor v7

Parameters (4)

Parameter	Value
model__n_estimators	[3, 5, 7]
model__max_depth	[2, 3, 4]
model__min_samples_split	[2, 5]
model__min_samples_leaf	[1, 2]

Metrics (3)

Metric	Value
rmse	34645.326912087476
mae	23180.00639059538
r2	0.8435139986979004



Banyak tidaknya hasil *tracking*-an kita, tergantung pada (sejauh yang saya tahu):

- Banyaknya percobaan
- Banyaknya model
- Banyaknya parameter

3. Logging dan Monitoring

Konsep logging dan monitoring sangat berguna untuk memantau apakah pipeline yang sudah dikonsep berjalan dengan baik atau ada error yang muncul. Ini tidak menyebabkan kode yang kita buat tidak berjalan, hanya saja tetap berjalan dengan munculnya error. Dengan begitu kita bisa mengetahui dimana error yang harus diselesaikan.

Logging harus disetup terlebih dahulu. Saya mengaturnya pada `src/utils/logger.py`

Ini kode pada file tersebut:

```
import sys
import time
import logging
import logging.handlers

from pathlib import Path
from typing import Optional

class CustomLogger:
    """Custom logger configuration"""
```



```

    @staticmethod
    def setup_logger(name: str, log_file: Optional[str] =
None) -> logging.Logger:
        """
        Setup logger with custom configuration

        Args:
            name: Logger name
            log_file: Optional log file path

        Returns:
            logging.Logger: Configured logger instance
        """
        logger = logging.getLogger(name)
        logger.setLevel(logging.INFO)

        # Create formatters
        formatter = logging.Formatter(
            '%(asctime)s | %(name)s | %(levelname)s |
%(message)s',
            datefmt='%Y-%m-%d %H:%M:%S'
        )

        # Create console handler
        console_handler = logging.StreamHandler(sys.stdout)
        console_handler.setFormatter(formatter)
        logger.addHandler(console_handler)

        # Create file handler if log_file is specified
        if log_file:
            # Create logs directory if it doesn't exist
            log_path = Path(log_file)
            log_path.parent.mkdir(parents=True, exist_ok=True)

            # Create rotating file handler
            file_handler =
logging.handlers.TimedRotatingFileHandler(
                log_file,
                when='midnight',

```

```

        interval=1,
        backupCount=7
    )
    file_handler.setFormatter(formatter)
    logger.addHandler(file_handler)

    return logger

# Create default logger
default_logger = CustomLogger.setup_logger(
    'house_prediction',
    log_file='logs/house_prediction.log'
)

```

Setelah saya membuat file logger.py. Selanjutnya adalah pada setiap kode yang ingin kita ketahui apakah berjalan baik atau tidak kita bisa memasangnya dengan

logger.info atau logger.error

Dengan syarat kita harus sudah meng-import file logger.py tersebut dengan cara

From src.utils.logger import default_logger as logger

Setelah itu kita bisa menggunakannya untuk memonitoring setiap kode pada pipeline yang kita bikin.

Contoh penggunaan logger pada file data_loader.py:

```

from src.utils.logger import default_logger as logger
from src.utils.config import config

class DataLoader:
    """Utilities for Loading the data"""

    def __init__(self, data_path:Optional[str] = None):
        """Initialize the class

        Args:
            data_path (Optional[str], optional): A path to
data file. Defaults to None.

```

```

"""
    self.data_path = data_path or config.get('data_path')
    logger.info(f"Initialized DataLoader with Path:
{self.data_path}")

```

Ini merupakan contoh tampilan logger saat kita menjalankan pipeline di terminal:

```

2025-01-21 15:40:00 | house_prediction | INFO | Initializing DataProcessor...
2025-01-21 15:40:00 | house_prediction | INFO | DataProcessor initialized.
2025-01-21 15:40:00 | house_prediction | INFO | Starting data processing...
2025-01-21 15:40:00 | house_prediction | INFO | Dropping some columns
2025-01-21 15:40:00 | house_prediction | INFO | Splitting our dataset into X and
y
2025-01-21 15:40:00 | house_prediction | INFO | Creating a processor pipeline
2025-01-21 15:40:00 | house_prediction | INFO | Data processing completed.
2025-01-21 15:40:00 | house_prediction | INFO | Initializing ModelTrainer...
2025-01-21 15:40:00 | house_prediction | INFO | ModelBuilder initialized.
2025-01-21 15:40:00 | house_prediction | INFO | Models and hyperparameters
loaded from config.
2025-01-21 15:40:01 | house_prediction | INFO | MLflow setup completed.
2025-01-21 15:40:01 | house_prediction | INFO | ModelTrainer initialized.
2025-01-21 15:40:01 | house_prediction | INFO | Starting training and
evaluation...
2025-01-21 15:40:01 | house_prediction | INFO | Splitting data into train and test
sets...
2025-01-21 15:40:01 | house_prediction | INFO | Training model:
RandomForestRegressor
2025-01-21 15:40:53 | house_prediction | INFO | Model:
RandomForestRegressor - RMSE: 34645.3269, R²: 0.8435, MAE: 23180.0064

```

4. FastAPI

API biasanya digunakan (setahu saya) untuk menghubungkan antara backend dan frontend. Dalam proyek ini API digunakan untuk model serving. Berikut adalah contoh pembuatan API menggunakan FastAPI:

```

# Initialize FastAPI
app = FastAPI()

# Load the best model
try:

```

```

        model = joblib.load('artifacts/best_model.pkl')
except Exception as e:
    raise RuntimeError("Model could not be loaded!") from e

# Input validation
class HousePriceFeatures(BaseModel):
    """House price features

    Args:
        BaseModel (_type_):
        """
    MSSubClass: float = Field(..., description='The general
zoning classification')
    LotFrontage: float = Field(..., description='Linear feet
of street connected to property')
    LotArea: float = Field(..., description='Lot size in
square feet')
    LotShape: str = Field(..., description='General shape of
property')
    LandContour: str = Field(..., description='Flatness of the
property')
    Condition1: str = Field(..., description='Proximity to
main road or railroad')
    OverallQual: int = Field(..., description='Overall
material and finish quality')
    ExterQual: str = Field(..., description='Exterior material
quality')
    ExterCond: str = Field(..., description='Present condition
of the material on the exterior')
    Foundation: str = Field(..., description='Type of
foundation')

@app.post("/predict")
def predict_house_price(features: HousePriceFeatures):
    # Ubah data input menjadi format yang diterima oleh model
(seperti DataFrame atau array)
    input_data = np.array([[features.MSSubClass,
features.LotFrontage, features.LotArea,
                                features.LotShape,
features.LandContour, features.Condition1,

```

```

        features.OverallQual,
features.ExterQual, features.ExterCond, features.Foundation]])

# Prediksi menggunakan model yang telah diload
prediction = model.predict(input_data)

return {"predicted_price": prediction[0]}

```

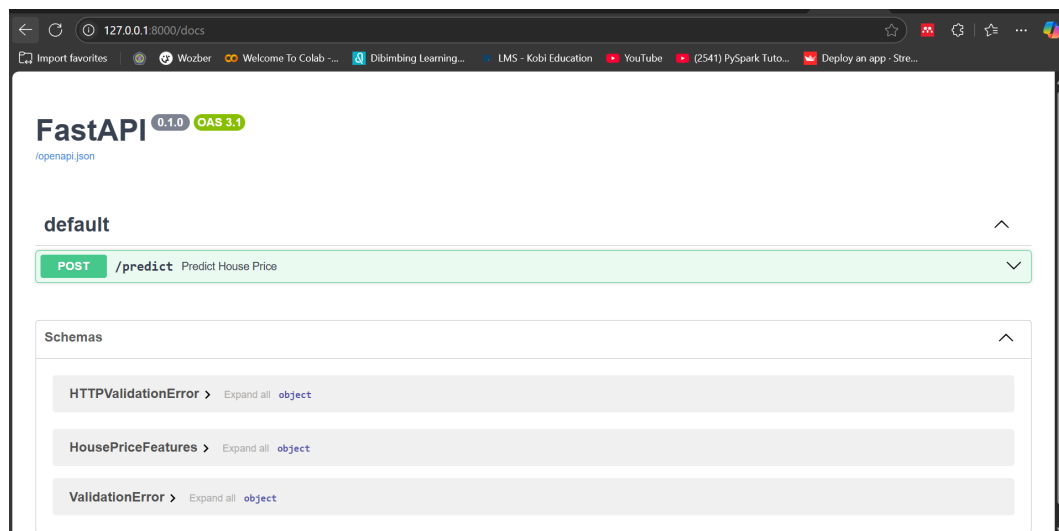
Setelah kita berhasil membuat kode di atas, kita bisa menjalankan API di localhost kita dengan perintah di terminal:

```
uvicorn main:app --reload
```

Setelah itu kita bisa melihat API kita pada localhost dengan menambahkan

/docs

Sehingga nanti akan muncul tampilan seperti ini.



5. Docker

Setelah semua proses sebelumnya kita lalui dan ingin mendeploy sebuah aplikasi. Kita bisa menggunakan Docker sebagai kontainernya. Untuk membuat docker, yang pertama kita lakukan adalah membuat file docker, bisa dengan nama Dockerfile dan kita bisa mengikuti kode seperti di bawah ini:

```
FROM python:3.9-slim
```

```

# Tetapkan direktori kerja di dalam container
WORKDIR /app

# Salin semua file proyek ke dalam container
COPY . .

# Perbarui pip ke versi terbaru untuk menghindari masalah
kompatibilitas
RUN pip install --upgrade pip

# Instal dependensi dari requirements.txt
RUN pip install -r requirements.txt

# (Opsional) Jika menggunakan API, buka port 8000
EXPOSE 8000

```

Setelah itu jangan lupa untuk membuat docker-compose.yaml. Kode nya bisa seperti ini:

```

version: '3'

services:
  ml_app:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./app
    environment:
      - PYTHONUNBUFFERED=1

```

Setelah itu kita login pada docker desktop dan bisa memasukkan perintah pada terminal, yaitu:

`docker build -t <nama akun di docker>/<nama project dan versinya> .`

Contoh:

`docker build -t edatha/house-price-prediction:v1.0 .`

Setelah kita memasukkan kode itu akan muncul seperti ini:

```
[+] Building 0.0s (0/0) docker:des[+] Building 0.0s (0/0) docker:des[+] Building 0.0s (0/0) docker:des[+] Building 0.0s (0/0) docker:des[+] Building 0.0s (0/1) docker:des[+] Building 0.2s (1/2) docker:des[+] Building 0.3s (1/2) docker:des[+] Building 0.5s (1/2) docker:des[+] Building 0.6s (1/2) docker:des[+] Building 0.8s (1/2) docker:des[+] Building 0.9s (1/2) docker:des[+] Building 1.1s (1/2) docker:des[+] Building 1.2s (1/2) docker:des[+] Building 1.3s (1/3) docker:des[+] Building 1.5s (2/3) docker:des[+] Building 1.7s (2/3) docker:des[+] Building 1.8s (2/3) docker:des[+] Building 2.0s (2/3) docker:des[+] Building 2.1s (2/3) docker:des[+] Building 2.3s (2/3) docker:des[+] Building 2.4s (2/3) docker:des[+] Building 2.6s (2/3) docker:des[+] Building 2.7s (2/3) docker:des
```

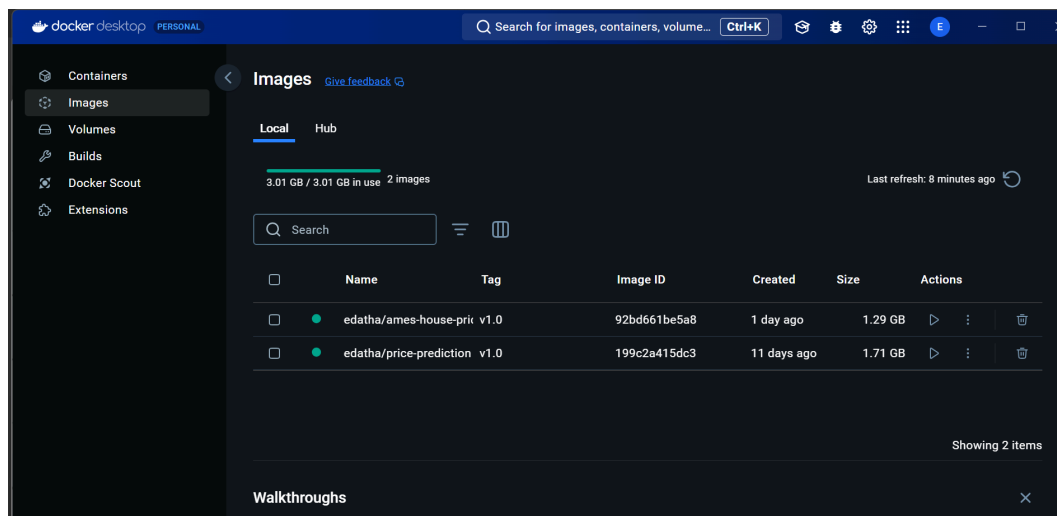
Setelah semuanya selesai kita bisa melakukan run pada docker kita dengan memasukkan perintah di terminal

```
docker run -p 8000:8000 edatha/price-prediction:v1.0
```

Akan muncul info seperti ini

```
INFO: Started server process [1]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 172.17.0.1:55880 - "GET /docs HTTP/1.1" 200 OK
INFO: 172.17.0.1:55880 - "GET /openapi.json HTTP/1.1" 200 OK
```

Kita bisa melihat docker images kita di docker desktop. Tampilannya seperti ini:



Kode lengkapnya dapat dilihat di [edatha/house-price-prediction](https://github.com/edatha/house-price-prediction)

Disclaimer :

Saya masih belum mengubah error yang saya bilang sehari atau 2 hari sebelumnya mas Bayuzen karena ada kegiatan lain yang tidak bisa ditinggal ditambah ada Assignment Day 27. Ya, itu salah satu kelalaian saya.

Error yang saya bilang seperti ini, "Mas, saya dah berusaha mengerjakan Assignment MLOps, hanya saja saya mentok. Di awal pembuatan pipeline saya berusaha untuk menyeleksi fitur yang akan digunakan pada saat di-deploy. Sudah saya masukkan ke pipeline processor-nya untuk handle numerik dan kategorikal serta missing value juga menyeleksi 10 fitur. Saya buat file data_loader, data_preprocessor, model, dan trainer, berjalan dengan baik. MLflow (sudah saya run, saya rasa) berjalan dengan baik juga. Nah, masalahnya di sini, saat saya coba prediksi di API local, malah error yang muncul di terminal "ValueError: X has 10 features, but ColumnTransformer is expecting 73 features as input."

Walaupun begitu, saya rasa, sekali lagi, saya rasa bahwa garis besar tiap soal yang ada pada ketentuan sudah saya laksanakan. Tapi saya tidak menafikan, bisa saja mas Bayuzen mengurangi nilai saya karena adanya error tersebut dan saya tidak boleh tidak menerima itu.

Selanjutnya saya ucapkan terima kasih sudah menjadi mentor pada bootcamp kali ini, mas. Semoga apa yang kami terima, dapat kami manfaatkan nantinya dan menjadi ilmu yang bermanfaat bagi mas Bayuzen.