MLDockFlow
Experiment - Monitor - Deploy

An integrated pipeline for experiment tracking, model stacking, and deployable ML workflow.

Tác giả: ConQ999 Team

Dự án **MLDockFlow** đã trải qua một hành trình phát triển từ một notebook Jupyter đơn giản với các mô hình hồi quy tuyến tính cơ bản đến một hệ thống học máy hoàn chỉnh với pipeline tự động, tích hợp MLflow cho experiment tracking, và triển khai bằng Docker Compose.

Hệ thống triển khai quy trình MLOps đầy đủ, từ thí nghiệm mô hình đến monitoring và deployment:

> Trong Module 5, dự án **Sales Prediction**, tập trung vào việc dự đoán giá nhà dựa trên bộ dữ liệu *House Prices – Advanced Regression Techniques*. Trên nền tảng này, dự án được mở rộng và phát triển thành **MLDockFlow**, hướng đến việc tự động hoá toàn bộ vòng đời mô hình học máy — từ thí nghiệm, huấn luyện, theo dõi đến triển khai.
>
> Trong giai đoạn đầu, ta tiến hành thực nghiệm và so sánh các mô hình hồi quy đã học (Linear, Ridge, Lasso, Polynomial), đồng thời áp dụng *stacking pipeline* để tăng khả năng khái quát hóa. Giai đoạn tiếp theo tập trung tích hợp **MLflow** nhằm quản lý, ghi nhận và giám sát chi tiết các phiên bản thí nghiệm thay vì chỉ theo dõi log đầu ra thông thường. Cuối cùng, dự án được đóng gói và triển khai dưới dạng dịch vụ web bằng **Docker Compose**, cho phép tái sử dụng và mở rộng dễ dàng.
>
> Với ba trụ cột chính — *experiment, monitor, deploy* — **MLDockFlow** minh họa cách một project học thuật có thể được nâng cấp thành hệ thống machine learning có tính tái lập, giám sát và sẵn sàng triển khai trong môi trường thực tế.

Repository của dự án: Toàn bộ mã nguồn, cấu trúc dự án, và các tài liệu liên quan có thể được truy cập tại repository GitHub: https://github.com/sonvt8/AIO2025_Project5.1_HousesPricing/tree/main. Repository này chứa đầy đủ pipeline từ preprocessing, feature engineering, model training với MLflow tracking, đến deployment với Docker Compose. Bạn đọc có thể clone repository về để thực nghiệm, tùy chỉnh, hoặc triển khai trên môi trường của riêng mình. Phần thực nghiệm tham khảo từ https://github.com/HoangVo-Prog/HousePrices_AdvancedRegressionTechniques.git.

Giới thiệu

Blog này trình bày toàn bộ quá trình phát triển và mở rộng dự án **MLDockFlow**, xuất phát từ đề tài gốc *Sales Prediction* trong Module 5 của chương trình AIO2025. Dự án ban đầu tập trung vào việc dự đoán giá nhà thông qua các mô hình hồi quy tuyến tính và phi tuyến, nhưng trong phiên bản mở rộng này, ta đã tích hợp các công cụ và kỹ thuật hiện đại để tiến tới một quy trình học máy hoàn chỉnh, tự động và có khả năng triển khai thực tế.

Nội dung blog được tổ chức thành các phần chính sau:

1. **So sánh dự án gốc và phiên bản nâng cấp:** Phân tích chi tiết các nâng cấp từ dự án gốc (Project 5.1) lên MLDockFlow, bao gồm cấu trúc code, preprocessing, models, experiment tracking và deployment. Phần này giúp người đọc hiểu rõ sự tiến hóa của dự án.
2. **Mục tiêu của dự án:** Giới thiệu mục đích của việc mở rộng project ban đầu thành hệ thống học máy có thể theo dõi, quản lý và triển khai tự động.
3. **Tiến trình phát triển:** Tóm tắt quy trình ta đã thực hiện — từ giai đoạn nghiên cứu, xây dựng pipeline, thực nghiệm mô hình cho đến triển khai và kiểm thử.
4. **Tính năng cải tiến và tối ưu hóa:** Trình bày các cải tiến kỹ thuật bao gồm stacking pipeline, tự động hóa huấn luyện và kiểm soát mô hình.
5. **Phân tích kết quả và các lựa chọn:** So sánh hiệu năng giữa các mô hình, thảo luận các quyết định liên quan đến lựa chọn siêu tham số, độ chính xác và khả năng tổng quát hóa.
6. **Hệ thống MLFLOW:** Mô tả cách ta sử dụng MLflow để theo dõi, ghi log, và quản lý các phiên bản thí nghiệm, giúp quy trình tái lập và minh bạch hơn.
7. **Docker Integrations:** Giải thích cách đóng gói mô hình bằng Docker Compose, triển khai ứng dụng web và quản lý môi trường thực thi.
8. **Future Works:** Đề xuất các hướng phát triển trong tương lai, bao gồm mở rộng pipeline cho các bài toán khác, cải thiện hiệu năng mô hình và tăng mức độ tự động hóa.
9. **Tổng kết & Bài học rút ra:** Nhìn lại toàn bộ dự án, nêu ra những bài học thực tế về teamwork, kỹ năng kỹ thuật và tư duy triển khai hệ thống học máy hoàn chỉnh.

Giá trị nhận được sau khi đọc Blog:

- Hiểu quy trình phát triển một dự án học máy từ giai đoạn nghiên cứu đến triển khai thực tế với MLOps.
- Nắm bắt cách tích hợp *MLflow* và *Docker* để hình thành một pipeline MLOps đơn giản mà hiệu quả.
- Học được cách thiết kế *stacking pipeline* để kết hợp sức mạnh của nhiều mô hình.
- Biết cách ghi log có tổ chức và quản lý thí nghiệm với MLflow Tracking Server.
- Hiểu rõ cách containerize và triển khai mô hình học máy với Docker Compose.
- Phát triển tư duy hệ thống hóa trong các dự án AI hiện đại — nền tảng quan trọng cho mọi kỹ sư dữ liệu và học máy.

Mục lục

1. So sánh dự án gốc và phiên bản nâng cấp

Dự án **MLDockFlow** được phát triển từ đề tài gốc *Project 5.1 - House Price Prediction* trong Module 5 của chương trình AIO2025. Trong phần này, ta sẽ cùng phân tích các nâng cấp và cải tiến so với phiên bản ban đầu một cách có hệ thống.

1.1 Tổng quan so sánh

Dự án gốc là một notebook Jupyter đơn giản. Nó tập trung vào việc thử nghiệm các mô hình hồi quy tuyến tính cơ bản. Dự án nâng cấp đã chuyển đổi thành một hệ thống học máy hoàn chỉnh. Hệ thống này có pipeline tự động, tích hợp MLOps tools và có khả năng triển khai thực tế.

Tiếp theo, ta sẽ xem xét các khía cạnh khác nhau của sự thay đổi này. Ta sẽ đi qua từng phần một cách chi tiết.

1.2 Phân tích so sánh chi tiết

Để hiểu rõ sự khác biệt, ta cần so sánh từng khía cạnh của dự án. Dưới đây là các bảng so sánh chi tiết.

1.2.1 Cấu trúc dự án

Trước tiên, ta xem xét sự khác biệt về cấu trúc dự án. Dự án gốc chỉ là một file notebook duy nhất. Dự án nâng cấp đã được tổ chức lại thành một hệ thống module hóa.

Bảng 1: So sánh cấu trúc dự án gốc và nâng cấp

Tiêu chí	Dự án gốc	Dự án nâng cấp
Cấu trúc	Notebook Jupyter đơn lẻ	Module hóa với src/, deployments/, notebooks/
Code organization	Tất cả code trong một file	Tách thành modules: processing/, training/, api/, frontend/
Configuration	Hard-coded parameters	File JSON config best_model_config.json
Dependency management	Không rõ ràng	requirements.txt với version pinning

Quan sát Bảng 1, ta có thể thấy rằng dự án nâng cấp có cấu trúc rõ ràng và dễ bảo trì hơn nhiều. Các modules được tách biệt giúp ta dễ dàng thay đổi từng phần mà không ảnh hưởng đến phần khác.

1.2.2 Xử lý dữ liệu và Feature Engineering

Tiếp theo, ta xem xét sự khác biệt về cách xử lý dữ liệu. Dự án gốc chỉ thực hiện các bước tiền xử lý cơ bản. Dự án nâng cấp áp dụng nhiều kỹ thuật phức tạp hơn.

Bảng 2: So sánh xử lý dữ liệu

Tiêu chí	Dự án gốc	Dự án nâng cấp
Preprocessing	• Drop columns có >50% missing • Fillna(”none”) cho categorical • One-hot encoding đơn giản • MinMaxScaler cho numeric	• Custom transformers: OrdinalMapper, TargetEncoderTransformer, RarePooler, MissingnessIndicator, FiniteCleaner, DropAllNaNColumns • Missingness indicators tự động • QuantileTransformer cho numeric • Pipeline có cấu trúc với ColumnTransformer
Feature Engineering	Không có	• 18 domain features: TotalSF, TotalBath, HouseAge, LotAreaRatio • Interaction features: IQ_OQ_GrLiv, IQ_OQ_TotalSF • Seasonal encoding: MoSold_sin, MoSold_cos • Log transformation: Ln_TotalSF • Winsorization: LotArea_clip
Reproducibility	Manual steps, khó tái lập	Pipeline tự động, có thể serialize bằng joblib

Quan sát Bảng 2, ta thấy rằng dự án nâng cấp có nhiều tính năng mạnh mẽ hơn. Ta sử dụng custom transformers để xử lý dữ liệu một cách linh hoạt. Ta cũng tạo thêm 18 đặc trưng domain-specific để cải thiện hiệu năng mô hình.

Lưu ý: Pipeline tự động giúp ta dễ dàng tái lập quá trình xử lý dữ liệu. Điều này rất quan trọng khi làm việc với dữ liệu mới hoặc cập nhật mô hình.

1.2.3 Mô hình học máy

Bây giờ, ta xem xét sự khác biệt về mô hình học máy. Dự án gốc chỉ sử dụng các mô hình hồi quy tuyến tính đơn giản. Dự án nâng cấp áp dụng nhiều mô hình nâng cao và kỹ thuật ensemble.

Bảng 3: So sánh mô hình và đánh giá

Tiêu chí	Dự án gốc	Dự án nâng cấp
Models	• Linear Regression (baseline) • Ridge Regression • Lasso Regression • Polynomial Regression	• 8 base models: XGBoost, CatBoost, LGBM, RandomForest, Ridge, Lasso, ElasticNet, SVR • Stacking ensemble với meta-learner (Ridge) • Optuna hyperparameter optimization
Hyperparameter Tuning	Không có hoặc manual	Optuna với 40+ trials, tự động log vào MLflow
Evaluation	Simple train/test split, RMSE và R^2	• 5-fold Cross-Validation • Test set evaluation • Kaggle submission với RMSLE • Metrics tracking qua MLflow
Best Performance	Lasso: Test RMSE = 26362.31, R^2 = 0.9008	XGBoost: Test RMSE = 24608.89, R^2 = 0.9210
Model Persistence	Không có	joblib serialization, MLflow model registry

Quan sát Bảng 3, ta thấy rằng dự án nâng cấp đạt được hiệu năng tốt hơn đáng kể. RMSE giảm từ 26362 xuống 24609, tương đương cải thiện khoảng 7%. Điều này nhờ vào việc sử dụng các mô hình ensemble và tối ưu hóa siêu tham số tự động.

Lưu ý: Tối ưu hóa siêu tham số với Optuna giúp ta tìm ra cấu hình tối ưu một cách tự động. Quá trình này được ghi log vào MLflow để ta dễ dàng theo dõi và so sánh.

1.2.4 Experiment Tracking và Version Control

Tiếp theo, ta xem xét sự khác biệt về quản lý thí nghiệm. Dự án gốc không có hệ thống theo dõi thí nghiệm. Dự án nâng cấp tích hợp MLflow để quản lý mọi thứ một cách tự động.

Bảng 4: So sánh quản lý thí nghiệm

Tiêu chí	Dự án gốc	Dự án nâng cấp
Experiment Tracking	Không có	MLflow với: • Parameters logging • Metrics tracking (CV RMSE, Test RMSE, R^2) • Artifacts storage (model files, configs) • UI dashboard tại localhost:5555
Model Versioning	Không có	MLflow Model Registry, version tagging
Reproducibility	Khó khăn, phụ thuộc vào manual notes	Tự động: code, data, config, seed đều được track
Comparison	Manual comparison giữa notebooks	MLflow UI cho phép so sánh nhiều runs trực quan

Quan sát Bảng 4, ta thấy rằng MLflow mang lại nhiều lợi ích. Ta có thể so sánh các thí nghiệm một cách trực quan. Ta cũng có thể khôi phục lại bất kỳ phiên bản mô hình nào một cách dễ dàng.

1.2.5 Triển khai và Deployment

Bây giờ, ta xem xét sự khác biệt về khả năng triển khai. Dự án gốc chỉ có thể chạy trong notebook. Dự án nâng cấp có thể được triển khai như một dịch vụ web.

Bảng 5: So sánh khả năng triển khai

Tiêu chí	Dự án gốc	Dự án nâng cấp
API	Không có	FastAPI với: • POST /predict (single prediction) • POST /predict/batch (batch inference) • GET /health, GET /model/info • Pydantic models cho validation • Auto-generated OpenAPI docs
Frontend	Không có	Streamlit UI với: • Form input cho house features • Real-time prediction display • Preset configurations • Responsive design
Containerization	Không có	Docker Compose với 3 services: • MLflow tracking server • FastAPI inference API • Streamlit frontend • Network isolation, volume mounting
CLI Interface	Không có	src/api/inference.py cho batch prediction từ CSV
Production Readiness	Chỉ dùng để demo/experiment	Có health checks, error handling, CORS support

Quan sát Bảng 5, ta thấy rằng dự án nâng cấp có đầy đủ các công cụ để triển khai. Ta có thể cung cấp dịch vụ dự đoán qua API hoặc giao diện web. Ta cũng có thể xử lý batch prediction một cách tự động.

Nâng cấp giao diện Streamlit: Dự án gốc hoàn toàn không có giao diện người dùng. Ta chỉ có thể tương tác với mô hình thông qua việc chạy code trong notebook. Dự án nâng cấp đã tích hợp **Streamlit UI**. Giao diện web này thân thiện và dễ sử dụng.

Giao diện Streamlit bao gồm các tính năng sau:

- **Form input trực quan:** Ta có thể nhập các thuộc tính của nhà (diện tích, số phòng, năm xây dựng, v.v.) thông qua các trường input được thiết kế rõ ràng.
- **Real-time prediction:** Kết quả dự đoán được hiển thị ngay lập tức. Ta chỉ cần nhập thông

tin và nhấn nút dự đoán.

- **Preset configurations:** Ta cung cấp các cấu hình mẫu. Điều này giúp ta thử nghiệm nhanh với các loại nhà phổ biến.
- **Responsive design:** Giao diện tự động điều chỉnh để hiển thị tốt trên các thiết bị khác nhau. Từ desktop đến tablet, giao diện đều hoạt động tốt.
- **Kết nối với API:** Giao diện giao tiếp với FastAPI backend thông qua biến môi trường API_URL. Điều này cho phép ta dễ dàng thay đổi endpoint khi cần.

Điều này giúp ta không cần kiến thức về lập trình để sử dụng mô hình. Ta chỉ cần truy cập vào giao diện web và nhập thông tin nhà. Ta sẽ nhận được dự đoán giá ngay lập tức.

1.2.6 Workflow và Tự động hóa

Cuối cùng, ta xem xét sự khác biệt về workflow và tự động hóa. Dự án gốc đòi hỏi nhiều thao tác thủ công. Dự án nâng cấp tự động hóa hầu hết các bước.

Bảng 6: So sánh workflow

Tiêu chí	Dự án gốc	Dự án nâng cấp
Training Process	Manual execution từng cell	Single command: python train.py
Pipeline Automation	Không có	End-to-end pipeline: • Data loading • Feature engineering • Model training • Evaluation • Model saving • MLflow logging
Testing	Manual testing trong notebook	test_api.py cho API endpoints
Documentation	Markdown cells trong notebook	README.md chi tiết, API docs tự động

Quan sát Bảng 6, ta thấy rằng dự án nâng cấp giúp ta tiết kiệm nhiều thời gian. Ta chỉ cần chạy một lệnh để hoàn tất toàn bộ pipeline. Điều này giúp ta tập trung vào việc cải thiện mô hình thay vì thực hiện các thao tác lặp lại.

1.3 Tổng hợp các nâng cấp chính

Dựa trên phân tích ở trên, ta có thể tổng hợp các nâng cấp thành 5 nhóm chính. Mỗi nhóm đóng góp một phần quan trọng vào sự cải thiện tổng thể.

1. Kiến trúc và Cấu trúc Code Nhóm nâng cấp đầu tiên liên quan đến cấu trúc code. Ta chuyển từ notebook monolith sang kiến trúc module hóa. Các module được tách biệt rõ ràng: processing, training, API, frontend. Ta cũng sử dụng configuration-driven thay vì hard-coded parameters. Điều này giúp ta dễ dàng thay đổi cấu hình mà không cần sửa code.

2. Xử lý dữ liệu và Feature Engineering Nhóm thứ hai tập trung vào việc xử lý dữ liệu. Ta sử dụng custom transformers để xử lý dữ liệu phức tạp hơn. Ta tạo thêm 18 domain-specific features thay vì chỉ dùng raw features. Pipeline có thể serialize và tái sử dụng, giúp ta đảm bảo tính nhất quán. Ta cũng xử lý missing values một cách thông minh hơn với indicators và imputation strategies.

3. Mô hình và Tối ưu hóa Nhóm thứ ba liên quan đến mô hình học máy. Ta mở rộng từ 3 linear models lên 8+ models, bao gồm các ensemble methods. Ta sử dụng Optuna để tối ưu hóa siêu tham số tự động. Ta áp dụng stacking ensemble để kết hợp sức mạnh của nhiều models. Kết quả là hiệu năng được cải thiện đáng kể: RMSE giảm từ 26362 xuống 24609, tương đương cải thiện khoảng 7%.

4. MLOps và Experiment Management Nhóm thứ tư tập trung vào MLOps. Ta tích hợp MLflow để theo dõi thí nghiệm. Ta có model versioning và registry để quản lý các phiên bản mô hình. Ta đảm bảo reproducibility tự động bằng cách track code, data, config và seed. Ta có thể so sánh các thí nghiệm một cách trực quan thông qua MLflow UI.

5. Deployment và Production Readiness Nhóm cuối cùng liên quan đến triển khai. Ta xây dựng RESTful API với FastAPI để cung cấp dịch vụ dự đoán. Ta tạo web UI với Streamlit để người dùng có thể tương tác dễ dàng. Ta sử dụng Docker để containerize toàn bộ hệ thống. Ta thêm health checks và monitoring để đảm bảo hệ thống hoạt động ổn định. Ta cũng có CLI interface để xử lý batch prediction.

1.4 Đánh giá tổng thể

Việc nâng cấp từ dự án gốc lên **MLDockFlow** đã chuyển đổi một prototype đơn giản thành một hệ thống học máy production-ready. Các cải tiến không chỉ về mặt kỹ thuật mà còn về khả năng quản lý, triển khai và mở rộng.

Ưu điểm chính: Các ưu điểm chính của dự án nâng cấp bao gồm:

- **Maintainability:** Cấu trúc code rõ ràng giúp ta dễ bảo trì và mở rộng. Ta có thể thay đổi từng phần mà không ảnh hưởng đến phần khác.
- **Reproducibility:** Ta có thể tái lập mọi thứ nhờ MLflow và Docker. Mọi thứ đều được track và có thể khôi phục lại.
- **Scalability:** Ta có thể dễ dàng thêm models, features, hoặc endpoints mới. Cấu trúc module hóa hỗ trợ việc mở rộng.
- **Usability:** API và UI giúp người dùng không cần code để sử dụng. Họ có thể dự đoán giá nhà một cách dễ dàng.
- **Performance:** Ta đạt được cải thiện đáng kể về độ chính xác và khả năng tổng quát hóa. RMSE giảm khoảng 7% so với dự án gốc.

Điểm cần lưu ý: Bên cạnh các ưu điểm, ta cũng cần lưu ý một số điểm sau:

- Độ phức tạp tăng lên đáng kể. Ta cần hiểu biết về MLOps để vận hành hệ thống hiệu quả.
- Ta cần có infrastructure như Docker và MLflow server để vận hành đầy đủ. Điều này đòi hỏi setup ban đầu.
- Thời gian setup ban đầu lâu hơn so với chạy notebook trực tiếp. Tuy nhiên, điều này được bù đắp bởi tính ổn định và khả năng tái sử dụng.

Nhìn chung, những nâng cấp này là cần thiết và hợp lý cho một dự án muốn chuyển từ giai đoạn nghiên cứu sang triển khai thực tế. Chúng tạo nền tảng vững chắc cho việc phát triển các tính năng MLOps nâng cao hơn trong tương lai. Đây là bước khởi đầu quan trọng để xây dựng các hệ thống machine learning chuyên nghiệp.

2. Experiment

Trong phần này, ta sẽ cùng tìm hiểu chi tiết toàn bộ pipeline huấn luyện và đánh giá mô hình của dự án **MLDockFlow**. Ta sẽ đi qua từng bước: tiền xử lý dữ liệu, lựa chọn mô hình, tinh chỉnh siêu tham số, tổ hợp mô hình, đến đánh giá cuối cùng và kiểm thử thực tế. Mỗi bước được tự động hoá trong một workflow thống nhất, có thể tái lập và giám sát bằng MLflow.

2.1 Data Preprocessing

Ta sử dụng *Ames Housing Dataset* gồm 1460 mẫu và 81 thuộc tính. Dữ liệu được chia thành 80% cho huấn luyện và 20% cho kiểm thử. Pipeline xử lý dữ liệu được thiết kế theo ba tầng:

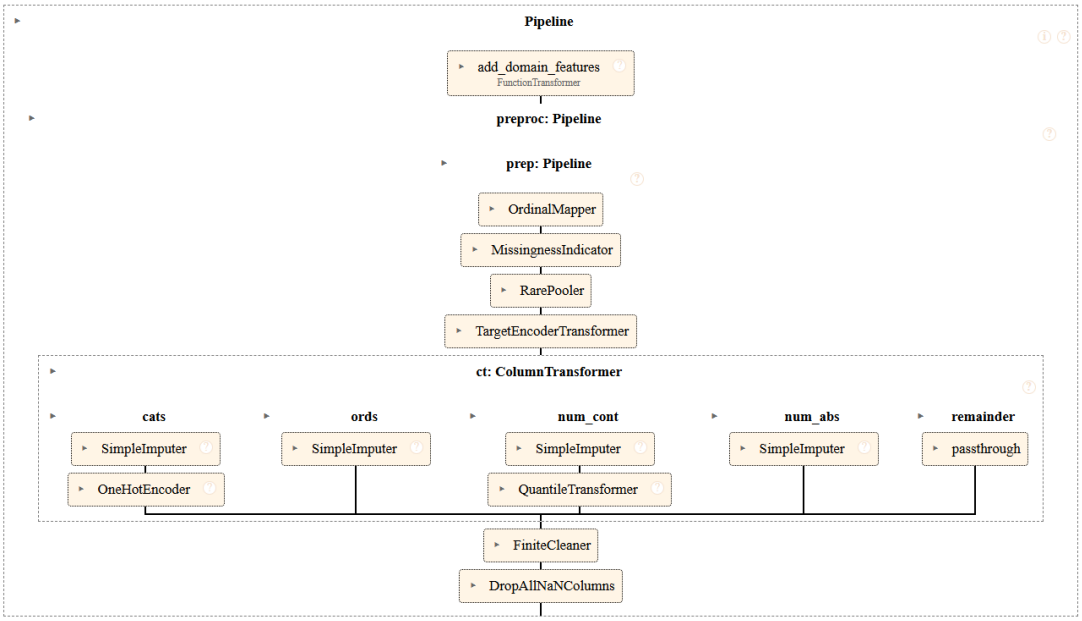
- **Custom Transformers:** Ta xây dựng các bộ biến đổi tuỳ chỉnh gồm *OrdinalMapper*, *TargetEncoderTransformer*, *MissingnessIndicator*, *RarePooler*, *FiniteCleaner*, và *DropAllNaNColumns*.
- **Domain Feature Engineering:** Ta tạo thêm các đặc trưng như *TotalSF*, *TotalBath*, *HouseAge*, *LotAreaRatio*, tương tác *Quality–Area*, *Seasonal Encoding*, và *Neighborhood_BldgType*.
- **Feature Pipeline:** Ta tích hợp toàn bộ bước xử lý bằng *ColumnTransformer* và *Pipeline* gồm One-Hot Encoding, mã hoá thứ bậc, *QuantileTransformer*, và loại bỏ các cột toàn NaN.

Để minh họa, ta xem ví dụ cấu trúc code của pipeline:

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, QuantileTransformer
from src.processing.transformers import (
    OrdinalMapper, TargetEncoderTransformer, RarePooler,
    MissingnessIndicator, FiniteCleaner
)

feature_pipeline = ColumnTransformer([
    ("categorical", OneHotEncoder(handle_unknown="ignore"), cat_cols),
    ("ordinal", OrdinalMapper(), ord_cols),
    ("numeric", QuantileTransformer(), num_cols)
])
```

Lưu ý về thiếu dữ liệu: Mỗi nhánh **Pipeline** có một **Imputer** riêng giúp inference ổn định khi thiếu trường đầu vào.



Hình 1: Sơ đồ tổng quan quy trình xử lý dữ liệu và đặc trưng

Quan sát Hình 1, ta có thể thấy rằng pipeline xử lý dữ liệu được tổ chức thành ba tầng chính: tầng đầu tiên là các custom transformers (OrdinalMapper, TargetEncoderTransformer, MissingnessIndicator, RarePooler, FiniteCleaner, DropAllNaNColumns) để xử lý dữ liệu phân loại và số; tầng thứ hai là domain feature engineering để tạo các đặc trưng mới như TotalSF, TotalBath, HouseAge; tầng thứ ba là feature pipeline sử dụng ColumnTransformer để áp dụng One-Hot Encoding, mã hóa thứ bậc, và QuantileTransformer. Mỗi tầng có thể được thay thế hoặc điều chỉnh độc lập mà không ảnh hưởng đến các tầng khác, giúp pipeline linh hoạt và dễ bảo trì.

Lưu ý về lệch phân phối: Ở biến liên tục, ta áp dụng *QuantileTransformer* hoặc *log1p* để chuẩn hóa phân phối:

```
df["SalePrice_log"] = np.log1p(df["SalePrice"])
```

2.2 Model Selection

Ta huấn luyện **8 mô hình cơ sở** trên cùng pipeline thống nhất để so sánh hiệu năng:

```
# Training 8 base models
python src/training/train_model.py --model xgb
python src/training/train_model.py --model catboost
python src/training/train_model.py --model lgbm
python src/training/train_model.py --model rf
python src/training/train_model.py --model ridge
python src/training/train_model.py --model lasso
python src/training/train_model.py --model elasticnet
python src/training/train_model.py --model svr
```

2.3 Evaluation Protocol

Đánh giá bằng **5-fold Cross-Validation**, log kết quả qua MLflow:

```
from sklearn.model_selection import cross_val_score
import mlflow

scores = cross_val_score(model, X, y, cv=5, scoring="neg_root_mean_squared_error")
mlflow.log_metric("cv_rmse_mean", -scores.mean())
mlflow.log_metric("cv_rmse_std", scores.std())
```

2.4 Hyperparameter Tuning with Optuna

Tối ưu bằng Optuna:

```
import optuna

def objective(trial):
    params = {
        "learning_rate": trial.suggest_float("lr", 0.01, 0.3),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "n_estimators": trial.suggest_int("n_estimators", 100, 600),
    }
    model = xgb.XGBRegressor(**params)
    score = cross_val_score(model, X, y, cv=5,
                            scoring="neg_root_mean_squared_error").mean()
    return -score

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=40)
```

2.5 Stacking Ensemble with Optuna

Cấu trúc Stacking:

```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import Ridge

stack_model = StackingRegressor(
    estimators=[
        ("cat", cat_model),
        ("xgb", xgb_model),
        ("ridge", ridge_model)
    ],
    final_estimator=Ridge(alpha=1.0)
)
```

2.6 Final Evaluation and Real-World Testing

Pipeline cuối được đánh giá trên tập test độc lập:

```
y_pred = stack_model.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
mlflow.log_metrics({"test_rmse": rmse, "test_r2": r2})
```

Bảng 7: Kết quả đánh giá trên tập test độc lập

Model	Test RMSE	Test R^2
Single models		
Ridge (baseline)	27.763,37	0,889.969
Lasso (baseline)	34.402,87	0,831.048
LinearRegression (baseline)	55.032,41	0,567.676
XGB	**24.608,889.79**	**0,921.046.714**
CatBoost	27.138,542.01	0,903.980.555
LGBM	28.937,452.11	0,890.829.137
RandomForest	29.694,111.37	0,885.045.274
Ridge	31.564,712.94	0,870.105.772
ElasticNet	31.820,724.43	0,867.990.164
Lasso	33.069,627.43	0,857.424.544
SVR	88.551,152.4	−0,022.291.149
Ensembles (Stacking)		
CatBoost+RandomForest+Ridge	**27.682,867.28**	**0,900.090.15**
CatBoost+LGBM+Ridge	27.966,034.36	0,898.035.748
CatBoost+RandomForest+LGBM	28.372,076.97	0,895.053.388
CatBoost+XGB+LGBM	28.305,464.5	0,895.545.6
CatBoost+XGB+RandomForest	28.496,588.92	0,894.130.242
CatBoost+XGB+Ridge	28.148,281.75	0,896.702.468
RandomForest+LGBM+Ridge	28.258,559.53	0,895.891.496
XGB+LGBM+Ridge	28.283,320.04	0,895.708.974
XGB+RandomForest+LGBM	28.501,769.42	0,894.091.746
XGB+RandomForest+Ridge	28.266,406.07	0,895.833.672

External Benchmark on Kaggle. Để kiểm chứng khả năng tổng quát hoá ngoài phân phối nội bộ, ta nộp dự đoán lên cuộc thi **House Prices - Advanced Regression Techniques** trên Kaggle. Cuộc thi này sử dụng bộ dữ liệu Ames Housing và khuyến khích các phương pháp hồi quy nâng cao cho dữ liệu tabular. Tập test của Kaggle được ẩn nhãn, người tham gia gửi file Id,SalePrice để hệ thống chấm điểm và xếp hạng trên bảng xếp hạng công khai.

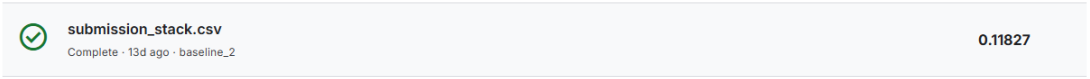
Evaluation metric của Kaggle. Cuộc thi đánh giá bằng **Root Mean Squared Logarithmic Error (RMSLE)** giữa nhãn thật y và dự đoán $\hat{y}$ trên *SalePrice*:

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\big(\log(1+\hat{y}_i) - \log(1+y_i)\big)^2}.$$

Khác với RMSE, RMSLE đo sai số trong không gian log, do đó bớt nhạy với ngoại lệ giá rất cao, đồng thời phản ánh tốt hơn tỉ lệ sai số tương đối. Điều này giải thích vì sao một mô hình tối

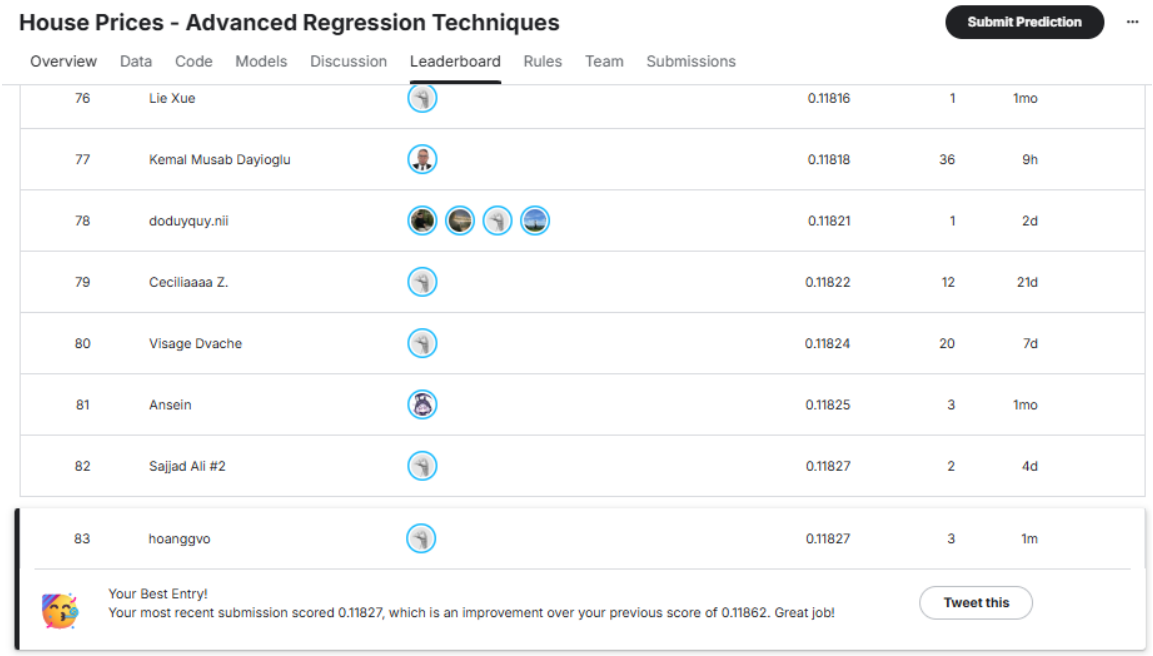
ưu cho RMSE nội bộ có thể cần tinh chỉnh nhẹ để đạt điểm số tốt hơn theo RMSLE của cuộc thi.

Kết quả nộp bài. Với mô hình stacking đã trình bày ở trên, ta nộp file dự đoán submission_stack.csv và đạt điểm **RMSLE = 0.11827** trên leaderboard công khai.



Hình 2: Kết quả nộp bài trên Kaggle với mô hình stacking

Quan sát Hình 2, ta có thể thấy rằng điểm số RMSLE = 0.11827 nằm trong khoảng tốt. Màn hình hiển thị xác nhận submission từ Kaggle website, cho thấy mô hình có khả năng tổng quát hóa tốt trên dữ liệu test của Kaggle.



Hình 3: Vị trí xếp hạng trên public leaderboard của Kaggle

Quan sát Hình 3, ta có thể thấy vị trí xếp hạng của mô hình tại thời điểm nộp bài trên public leaderboard. Điều này xác nhận rằng phương pháp của ta đạt hiệu quả tốt so với các phương pháp khác trong cuộc thi.

Để nộp kết quả lên Kaggle, ta thực hiện các lệnh sau:

```
# Xuất file submission và nộp lên Kaggle
python src/api/inference.py data/test.csv --output submission_stack.csv
kaggle competitions submit -c house-prices-advanced-regression-techniques \
  -f submission_stack.csv -m "MLDockFlow Stacking RMSE=24609"
```

Bàn luận. So với các chỉ số nội bộ dựa trên RMSE và R^2, điểm RMSLE trên Kaggle xác nhận mô hình duy trì sai số tương đối thấp sau biến đổi log. Khoảng điểm $0.11 \sim 0.12$ thường tương ứng với mức dự đoán sát cho biên độ giá trung vị, đặc biệt khi phân phối giá nhà lệch phải. Trong bối cảnh triển khai, ta nên giám sát định kỳ cả RMSE và RMSLE tùy mục tiêu kinh doanh: RMSE cho sai số tuyệt đối tiền tệ, RMSLE cho độ lệch theo tỉ lệ phần trăm.

3. Monitoring

Hệ thống giám sát và theo dõi (*Monitoring*) đóng vai trò cốt lõi trong pipeline của dự án **MLDockFlow**, đảm bảo rằng toàn bộ vòng đời mô hình từ huấn luyện, đánh giá, triển khai đến vận hành đều được ghi nhận, kiểm soát và có thể tái lập. Trong phần này, ta sẽ tìm hiểu cách tích hợp **MLflow** để theo dõi thí nghiệm, tổ chức lưu trữ mô hình, cùng với việc giám sát dịch vụ suy luận qua các endpoint kiểm tra sức khỏe và theo dõi hoạt động.

3.1 MLflow Tracking and Experiment Logging

Để đảm bảo khả năng tái lập thí nghiệm và quản lý lịch sử huấn luyện, ta sử dụng **MLflow Tracking Server** như một trung tâm ghi nhận và quan sát toàn bộ quá trình. Các thành phần chính ta cần biết:

- **Tracking URI:** Nếu không tìm thấy biến môi trường MLFLOW_TRACKING_URI, script src/training/train_model.py sẽ mặc định kết nối đến http://localhost:5555.
- **Experiment Registration:** Mỗi lần huấn luyện tạo ra một *experiment run* mới với ID duy nhất, được tổ chức theo tên mô hình (model_type) để ta dễ dàng so sánh và phân tích.
- **Logged Entities:**
  - **Parameters:** Các siêu tham số của mô hình như xgb_max_depth, xgb_learning_rate, n_estimators, model_type, v.v.
  - **Metrics:** Các chỉ số định lượng gồm cv_rmse_mean, cv_rmse_std, cv_r2_mean, cv_r2_std, test_rmse, test_r2.
  - **Artifacts:** Mô hình huấn luyện model.pkl cùng pipeline suy luận best_pipeline.joblib và feature_pipeline.joblib.

MLflow server được triển khai thông qua tệp deployments/mlflow/docker-compose.yaml, sử dụng image chính thức ghcr.io/mlflow/mlflow. Cấu hình cổng 5555 trên host, với cờ --serve-artifacts để phục vụ trực tiếp các artifact đã lưu. Dữ liệu được lưu tại hai volume chính:

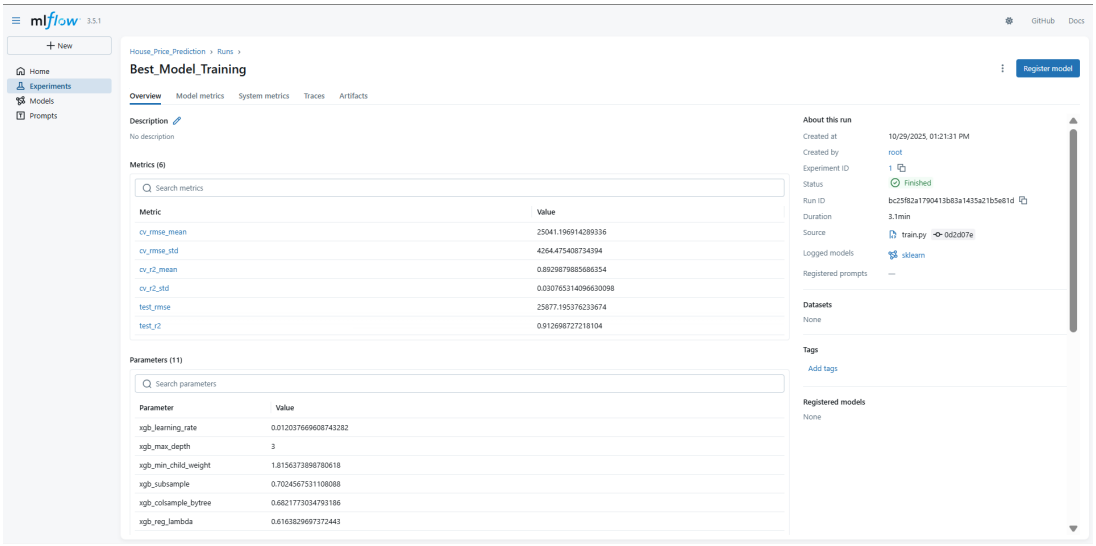
```
./mlflow_db/      # Cơ sở dữ liệu theo dõi metadata
./mlruns/         # Lưu artifacts và logs của từng run
```

Việc theo dõi được thực hiện tự động thông qua các lời gọi sau:

```
mlflow.log_params(params)
mlflow.log_metrics(metrics)
mlflow.log_artifacts(output_dir)
```

Khi sử dụng MLflow, ta nhận được các lợi ích sau:

- Tạo lịch sử trực quan về tất cả lần huấn luyện và tinh chỉnh mô hình.
- Ta có thể truy cập web UI tại http://localhost:5555, nơi hiển thị biểu đồ RMSE, R^2 và so sánh các model.
- Ta có thể khôi phục (restore) pipeline từ bất kỳ phiên bản nào trong tương lai.



Hình 4: Giao diện web của MLflow Tracking Server

Quan sát Hình 4, ta có thể thấy rằng giao diện MLflow hiển thị danh sách các thí nghiệm (experiments) và các runs tương ứng. Mỗi run chứa thông tin chi tiết về parameters, metrics, và artifacts. Giao diện này cho phép ta dễ dàng so sánh kết quả giữa các phiên bản mô hình khác nhau thông qua các biểu đồ và bảng thống kê, từ đó lựa chọn mô hình tốt nhất cho deployment.

3.2 Health Checks và Quan sát Dịch vụ

Dịch vụ suy luận (FastAPI) được triển khai kèm các endpoint kiểm tra trạng thái, giúp ta đảm bảo hệ thống hoạt động ổn định trong môi trường Docker. Các API chính ta có thể sử dụng:

- GET /health: Trả về tình trạng dịch vụ, được cấu hình làm *health check* mặc định trong Dockerfile của API.
- GET /model/info: Cung cấp thông tin mô hình hiện đang nạp, gồm model_name, model_type, version, performance.
- POST /predict: Nhận dữ liệu JSON theo schema HouseFeatures, trả kết quả dự đoán SalePrice và model_version.

Ví dụ lệnh kiểm tra nhanh dịch vụ:

```
curl -s http://localhost:8000/health #{"status":"healthy","model_loaded":true,"version":"1.0.0"}#
curl -s http://localhost:8000/model/info # {"model_name":"XGB","model_type":"Single","version"
↪ :"1.0","performance":{"cv_rmse":25259.41641200907,"cv_r2":0.8921142012886006,"test_
↪ rmse":24608.889785175903,"test_r2":0.921046714298606},"features_count":18}#
```

Trong cấu hình docker-compose.yml, service API có phần:

```yaml
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
  interval: 30s
  timeout: 10s
  retries: 3
```

Điều này đảm bảo container được khởi động lại tự động nếu dịch vụ không phản hồi, giúp ta duy trì tính ổn định của hệ thống.

3.3 Khuyến nghị Theo dõi Vận hành

Mặc dù dự án hiện đã có MLflow cho thí nghiệm và endpoint kiểm tra sức khỏe, ta nên mở rộng theo hướng MLOps chuyên nghiệp hơn để đảm bảo an toàn và khả năng mở rộng. Dưới đây là các khuyến nghị cụ thể:

- **Ghi log nâng cao:** Thu thập thêm thông tin *latency*, *throughput*, và *error rate* của từng endpoint, gửi đến Prometheus Gateway hoặc Grafana Dashboard.
- **Theo dõi Data Drift:** Tính toán thống kê phân phối đầu vào (mean, std, skew) và so sánh với phân phối trong huấn luyện; phát cảnh báo nếu lệch vượt ngưỡng.
- **Audit Requests:** Lưu trữ mẫu request/response đã vô danh hóa (anonymized) để phục vụ kiểm thử hồi quy khi cập nhật mô hình.
- **Model Version Registry:** Duy trì bảng model_registry.csv hoặc module riêng, ghi nhận lịch sử thay đổi, ngày deploy, và người thực hiện.
- **Tự động cảnh báo (alerting):** Thiết lập email/slack webhook gửi cảnh báo khi service không phản hồi, metric sai lệch, hoặc drift tăng cao.

Tổng kết lại, hệ thống **Monitoring** trong **MLDockFlow** không chỉ giúp ta theo dõi hiệu năng mô hình trong giai đoạn huấn luyện mà còn đảm bảo khả năng vận hành ổn định khi triển khai. Việc tích hợp MLflow, FastAPI health checks và hướng mở rộng theo MLOps giúp dự án đạt mức độ chuyên nghiệp tương đương các hệ thống machine learning trong môi trường doanh nghiệp.

4. Deployment

Trong phần này, ta sẽ cùng tìm hiểu chi tiết quá trình triển khai hệ thống **MLDockFlow**. Ta sẽ đi qua từng thành phần, cách đóng gói, và quy trình triển khai từng bước.

4.1 Thành phần hệ thống

Kiến trúc triển khai gồm ba khối chính, được liên kết qua mạng Docker Compose thống nhất. Ta có thể hình dung như sau:

1. **MLflow Tracking Server:** Được khởi chạy bằng tệp deployments/mlflow/docker-compose.yaml. MLflow chạy trên image ghcr.io/mlflow/mlflow, lắng nghe tại cổng 5555. Container này phục vụ giao diện web quản lý thí nghiệm và artifact.

2. **API Suy luận (Inference API):** Viết bằng **FastAPI + Uvicorn**, đóng gói qua Dockerfile tại deployments/api/Dockerfile và khởi động bằng Compose tại deployments/api/docker-compose.yaml. Đây là dịch vụ chính nhận yêu cầu dự đoán từ UI hoặc client bên ngoài.
3. **Streamlit UI Demo:** File giao diện src/frontend/app.py được sử dụng để minh hoạ trực quan việc gọi API. Biến môi trường API_URL cho phép ta cấu hình endpoint suy luận (POST /predict) khi chạy.

Mỗi thành phần có vai trò riêng và hoạt động độc lập, nhưng được kết nối với nhau thông qua mạng Docker để tạo thành một hệ thống hoàn chỉnh.

4.2 Đóng gói API bằng Docker

Dockerfile của API định nghĩa toàn bộ môi trường thực thi và các bước chuẩn bị mô hình. Ta cần lưu ý các điểm sau:

- Sao chép mã nguồn vào thư mục /app bên trong container.
- Mở cổng 8000 cho dịch vụ FastAPI.
- Thêm lệnh **HEALTHCHECK** gọi http://localhost:8000/health để giám sát tình trạng.
- Thiết lập biến môi trường MODEL_PATH trỏ tới /app/src/models/best_pipeline.joblib.

Trong Docker Compose (deployments/api/docker-compose.yaml), service api bật trên cổng host 8000, gắn với volume chỉ đọc. Ta có thể thấy cấu trúc volume như sau:

- src/models: chứa mô hình đã huấn luyện.
- src/configs: chứa tệp cấu hình huấn luyện và tham số.
- data/raw: chứa dữ liệu đầu vào mẫu để kiểm thử nhanh.

Khi container khởi động, ta có thể quan sát log để xác nhận API đã sẵn sàng. Lệnh khởi chạy trong container:

```
uvicorn src.api.main:app --host 0.0.0.0 --port 8000
```

Khi ta build và chạy container, quá trình diễn ra như sau: từ source code, Docker build tạo image, sau đó image được chạy thành container. Healthcheck sẽ tự động kiểm tra trạng thái của API và restart container nếu cần thiết.

4.3 Quy trình triển khai tối thiểu

Các bước triển khai cơ bản cho toàn bộ hệ thống:

1. **Khởi chạy MLflow Tracking:**

```
cd deployments/mlflow
docker compose up -d
```

2. **Huấn luyện mô hình và ghi log vào MLflow:**

```
pip install -r requirements.txt
python train.py
```

3. **Khởi chạy:** Truy cập:

   API: http://localhost:8000 (Docs: http://localhost:8000/docs)
   Frontend: http://localhost:8501
   MLflow: http://localhost:5555

4. **Mở giao diện demo:**

```
python src/api/run_api.py     # chạy API tại 8000
streamlit run src/frontend/app.py

```

5. **Khởi chạy model qua API:**

```
# Single prediction:
curl -X POST "http://localhost:8000/predict" \
  -H "Content-Type: application/json" \
  -d '{"OverallQual": 7, "GrLivArea": 1710, "YearBuilt": 2003}'

# Batch prediction:

curl -X POST "http://localhost:8000/predict/batch" \
  -H "Content-Type: application/json" \
  -d '{"houses": [{...}, {...}]}'

```

6. **Khởi chạy model qua Command Line Interface (CLI):**

```
python src/api/inference.py data/raw/test_data.csv --output predictions.csv
```

Khi tất cả services đã khởi động, ta có thể kiểm tra trạng thái bằng lệnh docker compose ps để xác nhận cả 3 container (MLflow, API, Streamlit) đều ở trạng thái ”Up”. Ta cũng có thể truy cập giao diện Streamlit để thử nghiệm dự đoán trực tiếp.

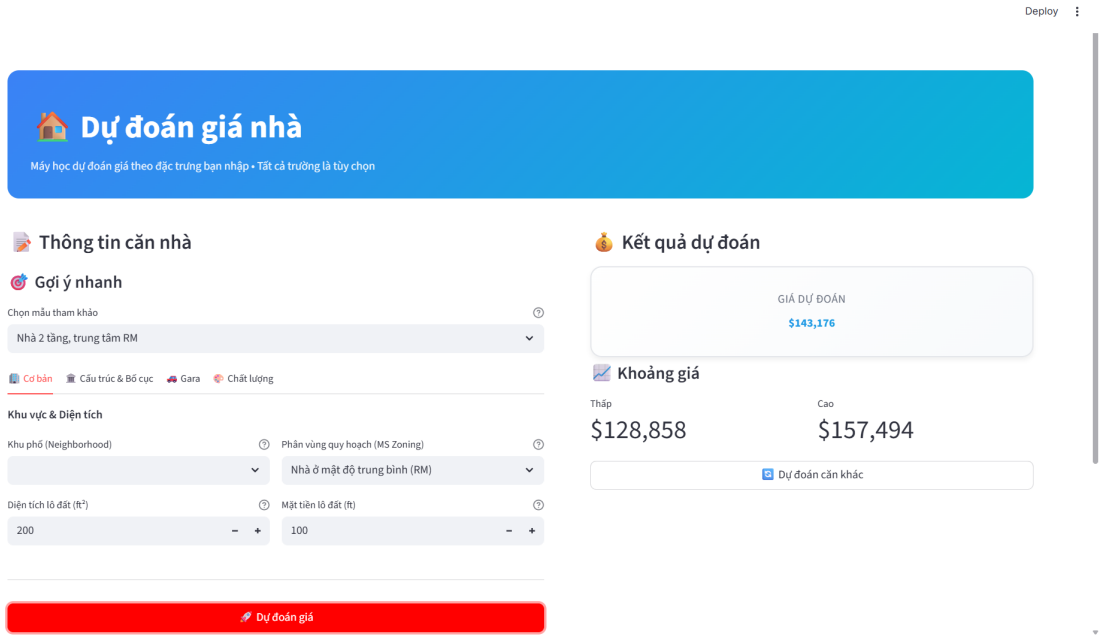
4.4 Giao diện Streamlit - Trải nghiệm Người dùng

Giao diện Streamlit là một trong những nâng cấp quan trọng của dự án. Nó cho phép ta tương tác với mô hình dự đoán giá nhà một cách trực quan và dễ dàng. Trước đây, ta phải chạy code Python trong notebook. Bây giờ, ta chỉ cần mở trình duyệt và nhập thông tin nhà. Ta sẽ nhận được kết quả dự đoán ngay lập tức.

4.4.1 Thiết kế và Tính năng

Giao diện Streamlit được thiết kế với các tính năng chính. Ta sẽ cùng xem xét từng tính năng:

- **Input Form:** Form nhập liệu được tổ chức rõ ràng. Ta có thể nhập các thuộc tính quan trọng của nhà. Ví dụ như diện tích sống (GrLivArea), chất lượng tổng thể (OverallQual), năm xây dựng (YearBuilt), và nhiều thuộc tính khác.
- **Preset Configurations:** Ta cung cấp các cấu hình mẫu để thử nghiệm nhanh. Ví dụ như nhà cao cấp, nhà trung bình, hoặc nhà cơ bản.
- **Real-time Prediction:** Khi ta nhập đầy đủ thông tin và nhấn nút "Predict", giao diện sẽ gọi API backend. Kết quả dự đoán được hiển thị ngay lập tức.
- **Error Handling:** Giao diện xử lý các lỗi một cách thân thiện. Ta sẽ thấy thông báo rõ ràng khi có vấn đề kết nối với API hoặc khi dữ liệu đầu vào không hợp lệ.

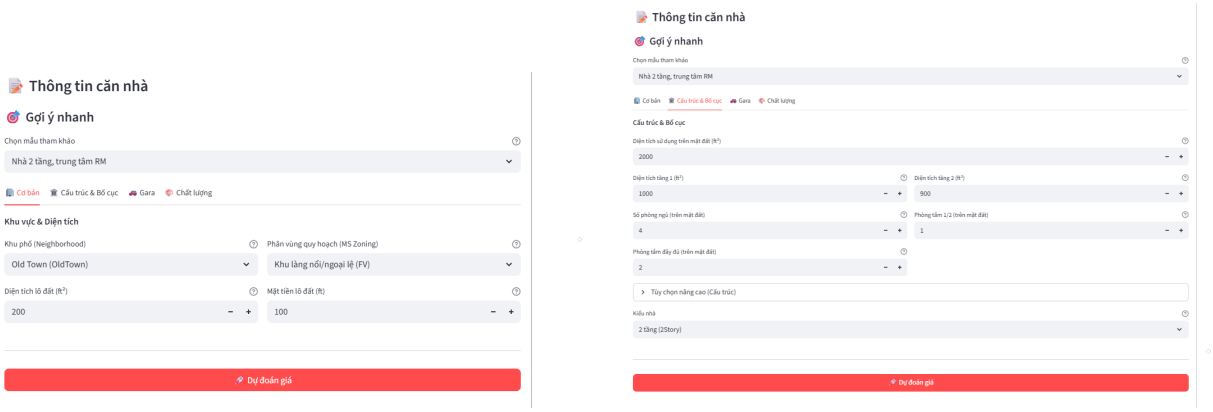


Hình 5: Giao diện Streamlit demo cho dự đoán giá nhà

Quan sát Hình 5, ta có thể thấy giao diện Streamlit hiển thị form nhập liệu. Các trường input được tổ chức rõ ràng. Ta có thể nhập thông tin về nhà như diện tích, số phòng, chất lượng, và các thuộc tính khác. Khi ta nhấn nút dự đoán, kết quả sẽ được hiển thị ngay bên cạnh. Kết quả cho thấy giá nhà dự đoán cùng với các thông tin liên quan.

4.4.2 Các Phương án Thiết kế

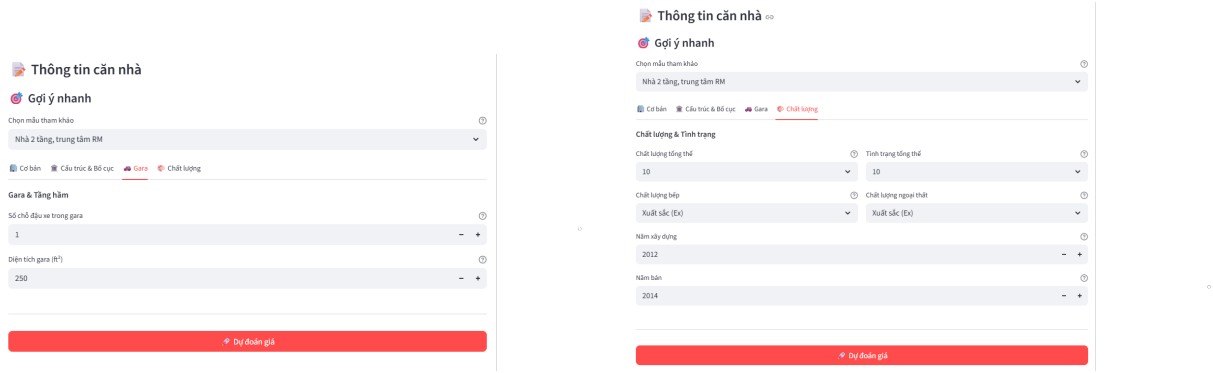
Trong quá trình phát triển, ta đã thử nghiệm nhiều phương án thiết kế khác nhau. Mục tiêu là tối ưu trải nghiệm người dùng. Mỗi phương án có ưu điểm riêng.



(a) Option 1: Input form tập trung (b) Option 2: Interactive dashboard

Hình 6: Các phương án thiết kế giao diện Streamlit

Quan sát Hình 6, ta có thể thấy hai phương án thiết kế khác nhau. Option 1 tập trung vào form nhập liệu đơn giản. Nó giúp ta dễ dàng điền thông tin. Option 2 cung cấp dashboard tương tác với nhiều biểu đồ và visualization. Phương án này phù hợp cho những người muốn xem phân tích chi tiết hơn.



(a) Option 3: Comparison view (b) Option 4: Results visualization

Hình 7: Các phương án thiết kế bổ sung

Quan sát Hình 7, ta có thể thấy Option 3 cung cấp chế độ so sánh. Nó cho phép ta so sánh nhiều cấu hình nhà cùng lúc. Option 4 tập trung vào visualization kết quả. Phương án này có các biểu đồ và thống kê chi tiết về dự đoán giá nhà.

Cuối cùng, ta chọn giao diện chính (như trong Hình 5). Giao diện này cân bằng tốt giữa tính đơn giản và đầy đủ thông tin. Nó phù hợp với đại đa số người dùng.

4.4.3 Tích hợp với API Backend

Giao diện Streamlit giao tiếp với FastAPI backend thông qua HTTP requests. Code chính được viết trong file src/frontend/app.py. Ta có thể xem ví dụ sau:

```
1 import streamlit as st
2 import requests
3 import os
```

```

API_URL = os.getenv("API_URL", "http://localhost:8000")

def predict_house_price(house_features):
    response = requests.post(
        f"{API_URL}/predict",
        json=house_features
    )
    return response.json()

# Streamlit UI code
st.title("House Price Prediction")
# ... form inputs ...
if st.button("Predict"):
    result = predict_house_price(features)
    st.success(f"Predicted Price: ${result['prediction']:,.2f}")
```

Ta sử dụng biến môi trường API_URL để cấu hình endpoint. Điều này giúp ta dễ dàng thay đổi địa chỉ API mà không cần sửa code. Khi chạy trong Docker Compose, ta có thể cấu hình URL này thông qua file docker-compose.yaml.

4.5 Quản lý và Cập nhật Mô hình

Việc cập nhật mô hình cần đảm bảo không gián đoạn dịch vụ:

- **Cập nhật trực tiếp:** Ta thay file best_pipeline.joblib trong volume src/models, sau đó restart container:

```
docker compose restart api
```

  Container sẽ tự động nạp mô hình mới qua MODEL_PATH khi khởi động lại.

- **Cập nhật qua MLflow Model Registry:** Đăng ký và chỉ định version khi load trong API:

```
import mlflow
model = mlflow.pyfunc.load_model("models:/House_Price_Prediction/v3")
```

- **Kết thúc session chạy:** Khi muốn dừng tất cả services và xóa volumes, ta chạy:

```
docker compose down -v
```

  Lệnh này sẽ dừng các container và xóa các volume đã tạo, giúp ta có môi trường sạch cho lần chạy tiếp theo.

5. Kết luận

Dự án **MLDockFlow** đánh dấu một bước tiến quan trọng trong việc hiện thực hóa một pipeline học máy hoàn chỉnh – từ giai đoạn thí nghiệm mô hình, ghi nhận và giám sát bằng MLflow, cho đến triển khai bằng Docker và vận hành thực tế thông qua API và Streamlit UI. Kết quả của dự án không chỉ thể hiện năng lực kỹ thuật trong việc xây dựng mô hình chính xác, mà còn minh

chứng cho khả năng tích hợp công nghệ theo hướng *MLOps*, giúp quy trình trở nên đồng bộ, tự động và có thể mở rộng. Ta sẽ cùng xem xét các hướng phát triển tiếp theo và những bài học rút ra từ dự án này.

Hướng phát triển trong tương lai (Future Works)

Mặc dù pipeline hiện tại đã đạt mức hoàn thiện cao, vẫn còn nhiều hướng phát triển tiềm năng để mở rộng khả năng ứng dụng và nâng tầm hệ thống lên chuẩn MLOps chuyên nghiệp. Các hướng phát triển chính bao gồm:

1. **Mở rộng pipeline cho các bài toán khác:** Cấu trúc hiện tại cho phép ta tổng quát hóa sang các tác vụ khác như phân loại, dự báo chuỗi thời gian hay phát hiện bất thường. Với việc thiết kế mô-đun hóa và cấu hình linh hoạt, ta có thể tái sử dụng toàn bộ pipeline chỉ bằng cách thay đổi các tệp cấu hình JSON hoặc YAML, thay vì viết lại mã nguồn.
2. **Cải thiện hiệu năng mô hình:** Kết hợp **Stacking ensemble** đa tầng với meta-learner phi tuyến (XGBoost hoặc Neural Network), áp dụng **Feature Selection** dựa trên SHAP hoặc RFE, và sử dụng **Optuna** song song trên GPU để tối ưu hóa siêu tham số nhanh hơn.
3. **Tự động hóa quy trình (Automation):** Tích hợp CI/CD với GitHub Actions hoặc Jenkins để tự động build, test và deploy. Bổ sung **auto-retraining** dựa trên theo dõi *data drift* và kích hoạt lại pipeline huấn luyện khi phát hiện sai lệch đáng kể. Sử dụng **Airflow** hoặc **Prefect** để điều phối các tác vụ huấn luyện, kiểm thử và ghi log.
4. **Tăng cường hệ thống giám sát:** Mở rộng MLflow với **Model Registry**, đồng thời tích hợp **Prometheus** và **Grafana** để theo dõi hiệu năng thời gian thực (latency, error rate, drift). Bổ sung bảng điều khiển trung tâm (Dashboard) giúp theo dõi sức khỏe mô hình và các container API.
5. **Chuyển đổi sang hạ tầng Cloud-native:** Đóng gói toàn bộ hệ thống thành các service trên **Kubernetes**, hỗ trợ autoscaling và cập nhật liền mạch. Sử dụng **Helm Charts** để triển khai nhanh trên các nền tảng như AWS, GCP, hoặc Azure.

Bài học và giá trị rút ra

Trong quá trình phát triển, ta không chỉ hoàn thiện kỹ năng kỹ thuật mà còn rèn luyện được tư duy hệ thống và khả năng phối hợp theo phong cách công nghiệp. Một số bài học nổi bật bao gồm:

Bài học kỹ thuật

- Việc thiết kế pipeline có cấu trúc rõ ràng (từ tiền xử lý đến triển khai) giúp ta giảm sai sót, tăng khả năng tái sử dụng và dễ dàng mở rộng.
- Khi sử dụng MLflow, ta nhận thấy lợi thế lớn trong việc quản lý thực nghiệm, cho phép so sánh trực quan giữa các phiên bản mô hình.
- Docker hóa toàn bộ dịch vụ đảm bảo môi trường nhất quán, giúp ta loại bỏ vấn đề ”chạy được trên máy tôi”.
- Tối ưu hóa mô hình bằng Optuna kết hợp Stacking cho thấy ta có thể cải thiện đáng kể hiệu năng mà vẫn giữ pipeline đơn giản, dễ triển khai.

Bài học về teamwork và quản lý dự án

- Sự phân chia rõ vai trò giữa các thành viên (data, model, deployment, monitoring) giúp ta theo dõi tiến độ rõ ràng và giảm chồng chéo công việc.
- Việc ghi log và version hóa mô hình giúp ta dễ dàng phục hồi và tái tạo kết quả, đặc biệt khi thực hiện nhiều thí nghiệm song song.
- Giao tiếp thông qua công cụ quản lý mã nguồn (Git, issues, commit logs) tạo sự minh bạch và giúp mọi người nắm bắt được tiến trình dự án một cách rõ ràng.

Tổng kết chung

Nhìn lại toàn bộ quá trình, **MLDockFlow** không chỉ là một dự án học thuật mà là một mô hình thực nghiệm hoàn chỉnh của một hệ thống Machine Learning trong thực tế. Từ góc nhìn kỹ thuật, dự án đã chứng minh khả năng tích hợp ba yếu tố cốt lõi:

1. **Khoa học dữ liệu** – qua việc xử lý, trích chọn và huấn luyện mô hình hiệu quả.
2. **Kỹ thuật phần mềm** – qua việc đóng gói, triển khai và tự động hóa quy trình bằng Docker và MLflow.
3. **Tư duy hệ thống** – qua khả năng thiết kế, giám sát và mở rộng pipeline.

Tóm lại, dự án này giúp ta hiểu sâu về cách xây dựng một pipeline học máy có thể triển khai được trong thực tế. Đây là bước khởi đầu quan trọng trước khi tiến tới các hệ thống MLOps nâng cao hơn, nơi mà tính tự động, khả năng mở rộng và quản trị mô hình trở thành yếu tố bắt buộc trong mọi hệ thống AI hiện đại.