Vestiaire

Where machine learning meets luxury: decoding the future of second-hand fashion.

Continuing...

Vestiaire Collective is a global platform for buying and selling pre-owned luxury and designer fashion.

- Revenue Streams: Commission-Based Model (once sold, the platform deducts a commission (e.g., 10-40% before paying the seller.)
- Market Capacity: The global secondhand luxury market value stood at US$34.39 billion in 2023, and is expected to reach US$60.55 billion by 2029.
- Problems:$0.84 billion GMV in 2023, significantly lower than the competitor's $1.83 billion; low conversion rate per listing

Existing analysis

- 3 of the most important features found out to be discount rate, product like count, & buyers fees rate
- LightGBM model with “product sold” as target. Accuracy 83%
- Causal model suggested top sellers and regular sellers did better with advanced badges signalling their success.

Sentiment Analysis

Product descriptions act as interaction spaces for sellers to market their products through words. Sentiment analysis suggests:

- 60% descriptions express positive sentiments. Example: “Magnificent boiled wool coat. I bought it in the late 2000s...”
- 10% descriptions express negative sentiments. Example: “HARD TO FIND ERMANNO SCERVINO Navy Puffer Jacket with Lace and Fur Accents 44...”
- 30% descriptions are neutral.

Causal Analysis Pt. 2

Does a seller's commissions reflect in the product description sentiment?

Model	ATE	95% CI	Interpretation
S-Learner (LR)	–0.0226	[–0.0226, –0.0226]	A 2.26 percentage-point drop in compound sentiment when you move from low to high price-diff.
S-Learner (XGB)	–0.0310	[–0.4206, 0.3099]	Point estimate is slightly negative, but the CI is so wide that it overlaps zero by a lot—you can't rule out no effect (or even a positive effect).
S-Learner (MLP)	–0.0210	[–0.2370, 0.2001]	Same story as XGB: small negative point estimate, but uncertainty is so large that it's not distinguishable from zero.

Effect:

- All ATEs hover around zero (<1 percentage point).
- Flexible-model CIs are very wide, crossing zero by substantial margins.
- **Conclusion:** There is no convincing causal effect of description sentiment on the likelihood of sale in your data.

Causal Analysis Pt. 2

Does a seller's commissions reflect in the product description sentiment?

Model	ATE	95% CI	Interpretation
S-Learner (LR)	–0.0004	[–0.0004, –0.0004]	A negligible 0.04 percentage-point decrease in sold-rate for "high" sentiment.
S-Learner (XGB)	0.0014	[–0.0331, 0.0368]	Essentially zero, and CI again overlaps zero heavily.
S-Learner (MLP)	0.0077	[–0.0077, 0.0231]	A tiny positive point estimate, but again indistinguishable from zero.

Effect:

- All ATEs hover around zero (<1 percentage point).
- Flexible-model CIs are very wide, crossing zero by substantial margins.
- **Conclusion:** There is no convincing causal effect of description sentiment on the likelihood of sale in your data.

AutoML Setup and Results

Workspace and Data Preparation

Connected to Azure ML workspace: vestiaire_ml_workspace
Loaded 899,281 rows from Azure Blob Storage (cleaned_data.parquet)
Applied feature engineering:
margin_rate, price_to_earning_ratio, price_per_like, seller_activity_ratio
Selected 23 features and sampled 10,000 rows for faster processing

AutoML Configuration and Execution

Task: Classification
Primary metric: AUC_weighted
Submitted experiment: price_elasticity_automl
AutoML automatically:
Performed data quality checks
Detected class imbalance
Trained multiple models (LightGBM, XGBoost, RandomForest, etc.)

Iteration	Pipeline	AUC	Notes
0	MaxAbsScaler LightGBM	0.8242	Good starting point
4	StandardScalerWrapper LightGBM	0.8313	First major improvement
12	StandardScalerWrapper XGBoostClassifier	0.8335	Slight performance boost
24	StandardScalerWrapper LightGBM	0.8372	Best model found
28	MaxAbsScaler LightGBM	0.8315	Consistently strong performance

AutoML Results – Best Model

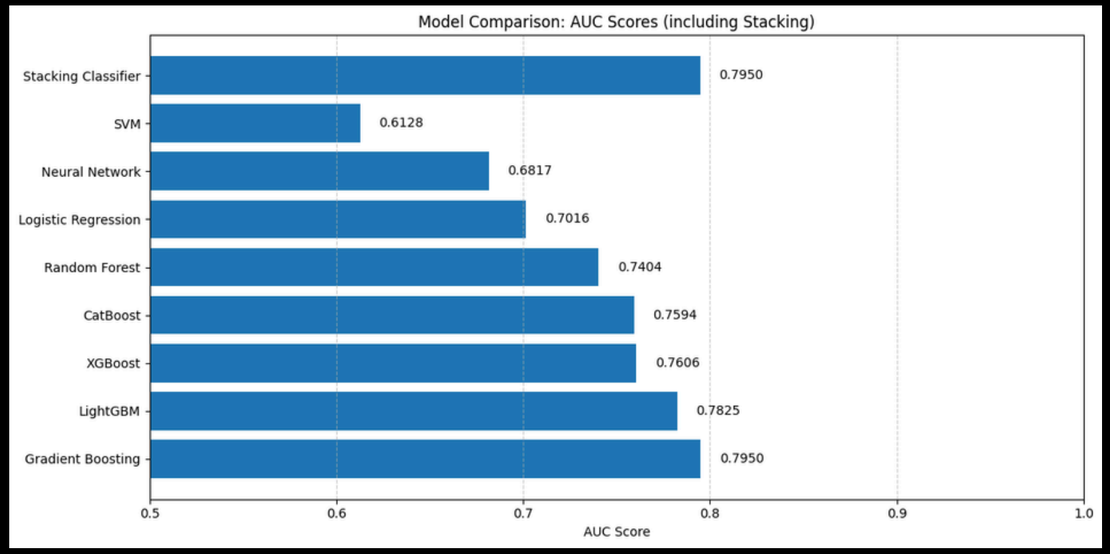
Metric	Value
Accuracy	0.9847
Weighted AUC	0.8372
Macro AUC	0.8372
Micro AUC	0.9949
Weighted Precision	0.9696
Weighted Recall	0.9847
Weighted F1-Score	0.9771
Log Loss	0.0656
Balanced Accuracy	0.5000
Average Precision (Micro)	0.9946
Average Precision (Macro)	0.5792

Best Model Performance Summary

- Achieved high overall accuracy of 98.47% and a very strong weighted AUC of 0.8372.
- The model is excellent at predicting the majority class, shown by the high weighted precision (96.96%) and weighted recall (98.47%).
- Micro-AUC score of 0.9949 confirms almost perfect performance across individual samples.
- Low balanced accuracy (0.50) and low macro recall (0.50) indicate the model struggles with the minority class (products that actually sold).
- Despite the class imbalance, the model maintained a very low log loss (0.0656), meaning predictions are highly confident.

The model is extremely reliable for general predictions. What could be done is replicating the ratio of sold to unsold of the whole dataset (instead of 50:50) to emulate real data a[illegible] have better accuracy.

AutoML Results – Comparison with Baseline Models

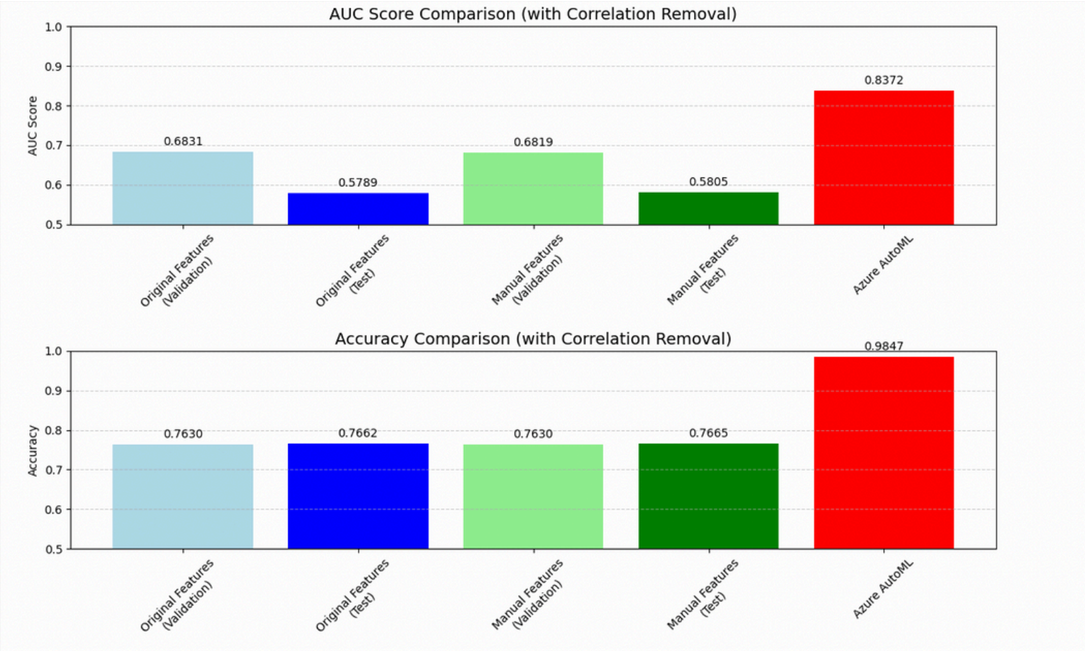


Best AUC was achieved by Gradient Boosting and Stacking Classifier (both 0.7950).
LightGBM also performed very competitively (AUC 0.7825).
Traditional models like Logistic Regression and SVM performed poorly, especially failing to capture the minority class (sold items).
Minority class recall was very low across all models (~3-6%), confirming that class imbalance severely affected model sensitivity.
Stacking Classifier did not outperform Gradient Boosting individually – ensembling added little in this baseline setup.
Baseline vs AutoML:
Best baseline AUC ≈ 0.7950
Best AutoML AUC ≈ 0.8372

Model	AUC Score	Recall (Minority Class)	Comment
Gradient Boosting	0.7950	6%	Best standalone model
Stacking Classifier	0.7950	6%	Same as Gradient Boosting
LightGBM	0.7825	6%	Strong, but slightly lower AUC
XGBoost	0.7606	6%	Good performance
CatBoost	0.7594	3%	Good, lower recall
Random Forest	0.7404	3%	Decent baseline
Logistic Regression	0.7016	0%	Failed on minority class
Neural Network	0.6817	3%	Struggled on minority
SVM	0.6128	0%	Weakest performer

AutoML improved AUC by ~5% over manual baselines.

AutoML Results – Feature Engineering Comparison



Model	AUC	Accuracy (Test)
Original Features (Validation)	0.6831	-
Original Features (Test)	0.5789	0.7662
Manual Features (Validation)	0.6819	-
Manual Features (Test)	0.5805	0.7665
Azure AutoML Best Model	0.8372	0.9847

Performed correlation removal (dropped 4 highly correlated features).
Train/Validation/Test split: 60% / 20% / 20%.

Manual feature engineering marginally improved performance over original features (+0.2% AUC).
Both manual and original features struggled to predict minority class despite preprocessing.
AutoML dramatically outperformed manual models (+44% AUC improvement over manual).
Accuracy for AutoML also significantly higher (~98.5% vs ~76%).

Manual features alone did not bridge the performance gap.
Azure AutoML's automated feature engineering and modeling pipeline was crucial to achieving top performance.

AutoML Ensemble Experiment

Data Setup:
Data Loaded: 899,281 rows (36 columns)
Sample Used: 15,000 rows (Balanced dataset → 10,000 unsold, 5,000 sold)
After Feature Engineering: 44 features created
After Correlation Removal: 41 features retained

AutoML Configuration:
Task: Classification
Metric: AUC_weighted
Cross-validation: 5 folds
Ensembles: Stacking and Voting enabled after 15 iterations
Max concurrent iterations: 4

Category	Result
Number of child runs	27 models evaluated
Best model type	StackEnsemble
Best model AUC (Validation)	1.0000 (perfect - suspicious)
Best individual model	ExtremeRandomTrees (AUC: ~0.9988)
Ensemble vs Individual Gain	+0.12% AUC improvement
Data Guardrails Check	Passed (Balanced classes, no missing values, no high cardinality)

All models achieved unrealistically perfect AUC scores (close to 1.0), indicating overfitting rather than true generalization.

StackEnsemble and VotingEnsemble showed only marginal gains over individual strong models.

Simple individual models like ExtremeRandomTrees already performed extremely well without needing stacking.

The additional complexity of ensembling appears unnecessary for this dataset.

Root cause may include:
Small and relatively easy-to-separate dataset
Feature engineering capturing too much target signal
Lack of true unseen data variability in sample

Testing

Seller Analysis

Unit Testing

```python
import sys
import os
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname("__file__"), '../src')))

import seller_analysis_notebook_script as sa
import pandas as pd

def test_no_missing_values():
    df = sa.df
    assert df.isnull().sum().sum() == 0, "There are missing values remaining."

def test_irrelevant_columns_dropped():
    df = sa.df
    dropped_cols = [
        'product_keywords', 'product_id', 'product_name',
        'product_description', 'brand_id', 'brand_url', 'seller_username'
    ]
    for col in dropped_cols:
        assert col not in df.columns, f"Column {col} was not dropped."

print("Unit tests loaded. Run with pytest or call functions manually.")
```

Data Tests

```python
import seller_analysis_notebook_script as sa
import pandas as pd

def test_no_duplicate_rows():
    df = sa.df
    assert df.duplicated().sum() == 0, "Duplicate rows found."

def test_column_types_expected():
    df = sa.df
    expected_types = {
        'has_cross_border_fees': 'bool',
        'buyers_fees': 'float64',
    }
    for col, expected_dtype in expected_types.items():
        assert str(df[col].dtype) == expected_dtype, f"{col} is not {expected_dtype}"

print("Data tests loaded. Run with pytest or manually.")
```

Model Validation Tests

```python
# Example:
# After training a model, check if the evaluation metrics are acceptable.

def test_model_auc_above_threshold(model_auc):
    assert model_auc > 0.85, "Model AUC is too low."

print("Model validation test ready (use when model is added).")
```

Load Test

```python
import seller_analysis_notebook_script as sa
import pandas as pd

def test_load_big_data():
    df_big = pd.concat([sa.df]*10, ignore_index=True)
    assert df_big.shape[0] == 10 * sa.df.shape[0], "Data not scaled properly."

print("Load test ready.")
```

Data Skew Test

```python
import seller_analysis_notebook_script as sa
import pandas as pd

def test_data_skew():
    original_mean = sa.df['buyers_fees'].mean()
    new_data = pd.DataFrame({'buyers_fees': [2.0, 2.5, 3.0]})
    new_mean = new_data['buyers_fees'].mean()
    assert abs(original_mean - new_mean) < 5.0, "Buyers fees distribution shifted significantly."

print("Data skew test ready.")
```

Integration Test

```python
import seller_analysis_notebook_script as sa

def test_full_pipeline_runs():
    df = sa.df
    assert df.shape[0] > 0, "DataFrame is empty."
    assert isinstance(sa.num_col, list)
    assert isinstance(sa.cat_col, list)
    assert isinstance(sa.bol_col, list)

print("Integration test ready.")
```

Model Performance Test

```python
def test_training_time(model, X_train, y_train):
    start = time.time()
    model.fit(X_train, y_train)
    end = time.time()
    assert end - start < 60, "Model training took too long."

print("Model performance test ready (use when model is added).")
```

MLflow

- **Setup**:
  - Just set the tracking URI to tell MLflow where to save logs and artifacts.
- **Wrap Your Code:**
  - Use mlflow.start_run() to track everything inside a training run.
- **Log What You Want:**
  - Explicitly log parameters, metrics, and artifacts
- **MLflow handles the rest:**
  - MLflow handles organizing and saving your experiment results automatically.

Set up MLflow:

```
mlflow.set_tracking_uri("file:../mlruns")
mlflow.set_experiment("Vestiaire_Model_Comparison")
if mlflow.active_run():
    mlflow.end_run()
```

Wrap code

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

Specify what to log:

```
mlflow.log_params(model.get_params())
        mlflow.log_metrics({
            "roc_auc": roc_auc,
            "accuracy": accuracy,
            "precision": precision,
            "recall": recall,
            "f1_score": f1,
            "log_loss": logloss
        })
```

Specify what to log:

```
poetry run mlflow ui
INFO:waitress:Serving on http://127.0.0.1:5000
```

MLflow

Logs Metrics:

Metrics (18)

Search metrics

Metric	Value
CatBoost_test_accuracy	0.98760123802287
CatBoost_test_f1	0.3303303303303303
CatBoost_test_log_loss	0.05570689128011065
CatBoost_test_precision	0.9375
CatBoost_test_recall	0.20048602673147023
CatBoost_test_roc_auc	0.8699588756110836
LightGBM_test_accuracy	0.9855811108845933
LightGBM_test_f1	0.13747228381374724
LightGBM_test_log_loss	0.06121900291843374
LightGBM_test_precision	0.7848101265822784
LightGBM_test_recall	0.07533414337788578
LightGBM_test_roc_auc	0.8605019515298695
XGBoost_test_accuracy	0.9870452397279315
XGBoost_test_f1	0.2816032887975334
XGBoost_test_log_loss	0.05760196124114261
XGBoost_test_precision	0.9133333333333333
XGBoost_test_recall	0.16646415552855406
XGBoost_test_roc_auc	0.8679297318442137

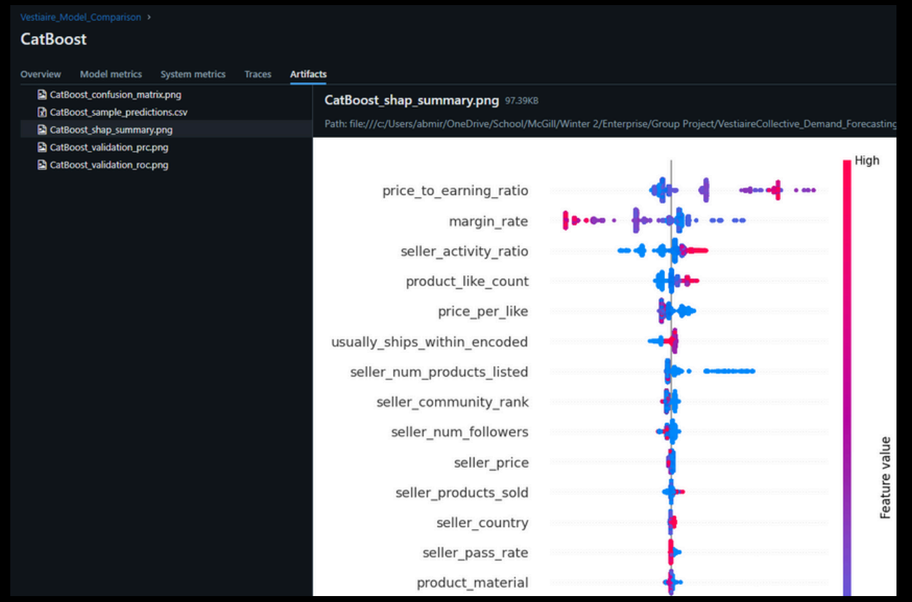
Logs Parameters:

Parameters (5)

Search parameters

Parameter	Value
depth	3
iterations	30
random_state	42
task_type	CPU
verbose	0

Logs Artifacts:



Optuna

- **Bayesian Hyperparameter Tuning with Optuna:**
  - Each Optuna trial is run as a nested MLflow run, sampling hyperparameters (e.g. n_estimators, max_depth, learning_rate) and logging them automatically.
- **Nested MLflow Runs for Trials:**
  - Under a parent run named “Hyperparameter_Tuning,” each trial_i logs its own parameters and validation ROC-AUC metric, making it easy to compare trials in the MLflow UI.
- **Automatic Best-Param Capture:**
  - After all trials, we extract study.best_params and log them (plus the best validation score) in the parent run for transparency and reproducibility.
- **Seamless Transition to Final Training:**
  - We re-initialize our model with the tuned hyperparameters and continue with our usual MLflow-wrapped train/validate/test runs, so everything—tuning and final evaluation—lives in one coherent MLflow experiment.

From Training to Serving

MLflow Experiment Tracking

◯ Logged and compared models using MLflow
◯ Selected best model by ROC AUC and saved with MLflow

MLflow Experiment Tracking

◯ Deployed the best model in a FastAPI app
◯ Containerized using Dockerfile_predicting with model config + best_run_id.txt
◯ Ensures consistent behavior across environments

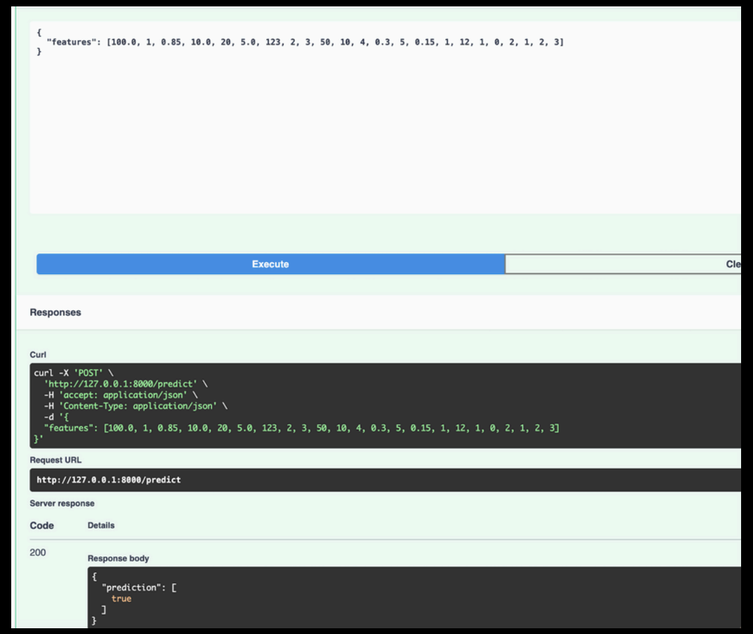
Local API Testing (Swagger UI)

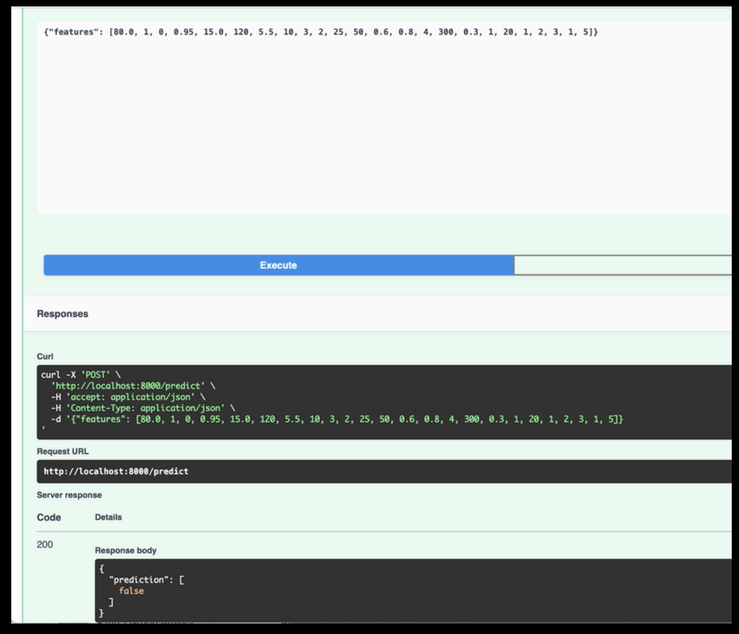
◯ Ran live prediction tests from the container.
◯ Verified model output via Swagger interface. (http://localhost:8000/docs)

```python
# Load the best model
mlflow.set_tracking_uri("file:/app/mlruns")
# Load best run id
with open("best_run_id.txt", "r") as f:
    best_run_id = f.read().strip()

model = mlflow.pyfunc.load_model(
    model_uri=f"runs:/{best_run_id}/model"
)

# FastAPI app
app = FastAPI()
```

```python
@app.get("/ping")  chloe
def ping():
    return {"status": "ok"}

# Input schema
class InputData(BaseModel):  1 usage  chloe
    features: list

# Prediction route
@app.post("/predict")  chloe
def predict(input_data: InputData):
    preds = model.predict([input_data.features])
    return {"prediction": preds.tolist()}
```

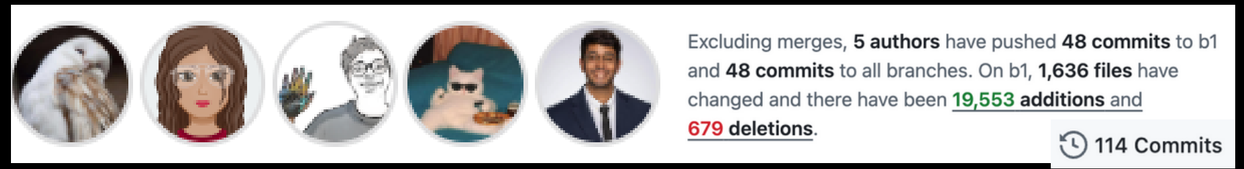
predicting-service e6ab8528e103 fastapi-mlflow-app 8000:800





Repository Organization & Clarity

We Teamed



Excluding merges, **5 authors** have pushed **48 commits** to b1 and **48 commits** to all branches. On b1, **1,636 files** have changed and there have been **19,553 additions** and **679 deletions**.

114 Commits

Merge pull request #1 from pity-piggy/liliana to the b1 to sync	pity-piggy*
toml update	chloe
Testing	Liliana García
updated ci	chloe
updated ci	chloe
CI for unit tests on price elasticity	chloe
poetry adding seller_analysis_v2 dependency in and locked	chloe
poetry adding price elasticity test.py dependency in and locked	chloe
create images, adding dependency management in only,	chloe
create docker file	chloe
create docker file	chloe
Merge branch 'b1' of https://github.com/pity-piggy/VestiaireCc	atharvavyas1
Causal ML analysis	atharvavyas1
Delete project_ml_flow copy/tests/test_data.py	Berly*
preliminary mlflow testing	Mirza Abubacker
Properly add cleaned_data.parquet using Git LFS	BERLY BRIGITH BIJU
Merge branch 'b1' of https://github.com/pity-piggy/VestiaireColl	Mirza Abubacker
New notebook	atharvavyas1
Delete justtesting.txt	Mirza Abubacker
just testing to see if im on the right fork hehe	Mirza Abubacker
Create extra	Berly*
Delete project_ml_flow copy/data/blah	Berly*
Create blah	Berly*
Delete project_ml_flow copy/src/blah	Berly*
Add files via upload	Berly*
Create blah	Berly*
Add files via upload	Berly*
file added	chloe
new cookiecutter with poetry setup	chloe
add file	chloe

Merge remote-tracking branch 'origin/b1' into b1 origin & b1	chloe
file updated	chloe
Add AutoML notebook for Vestiaire Collective project	BERLY BRIGITH BIJU
deletion	chloe
updated	chloe
Merge pull request #2 from pity-piggy/new-feature-model	pity-piggy*
add file	chloe
Remove cleaned_data.parquet from Git tracking	chloe
select best model and add Dockerfile_predicting	chloe
Merge remote-tracking branch 'origin/b1' into b1	chloe
conflict resolve	chloe
Made some format changes to file	atharvavyas1
Merge branch 'b1' of https://github.com/pity-piggy/VestiaireCc	atharvavyas1
Causal Analysis added	atharvavyas1
Create ext	Berly*
Delete project_ml_flow copy/src/price_elasticity/etc	Berly*
Add files via upload	Berly*
Create etc	Berly*
adding jupyter nbconvert	Mirza Abubacker
fully implemented mlflow	Mirza Abubacker
Update extra	Berly*
Create documentation_unittesting	Berly*
Added causalml	atharvavyas1
Merge branch 'b1' of https://github.com/pity-piggy/VestiaireColl	atharvavyas1
Added depenedencies	atharvavyas1
updated	chloe
deleted the jupter version	chloe
cleaned structure in .py ready to go	chloe
running tests on all the tests.py & converting the jupter into .py	chloe
Merge pull request #1 from pity-piggy/liliana to the b1 to sync	pity-piggy*
toml update	chloe

We Organized

- **Repo Structure (Cookiecutter)**
  - Ensures scalability, modularity, and ease of navigation for any new developer.
- **Dependency Management (Poetry)**
  - Environment fully managed with pyproject.toml and poetry.lock for reproducibility.
- **Testing with Pytest**
  - All core logic is tested via scripts in the tests/ folder and integrated into CI for validation.
- **CI/CD Workflow**
  - Every push or PR to main or b1 triggers automated tests via GitHub Actions — ensuring clean and stable code at every step.
- **Docker & Containerization**
  - The best-performing model is containerized with a FastAPI app using Docker

ci_pipeline.yml

Dockerfile_predicting

pyproject.toml

poetry.lock

- auto-ml
- data
- docs
- mlruns
- models
- module
- notebooks
- references
- reports
- src
- tests
- .gitignore
- Dockerfile_predicting
- Dockerfile_training
- LICENSE
- Makefile
- README.md
- __init__.py
- best_run_id.txt
- ci_pipeline.yml
- poetry.lock
- pyproject.toml

Thank You

For Your Attention

APPENDIX

Future Considerations

Model Registration, Deployment, and Monitoring in Azure