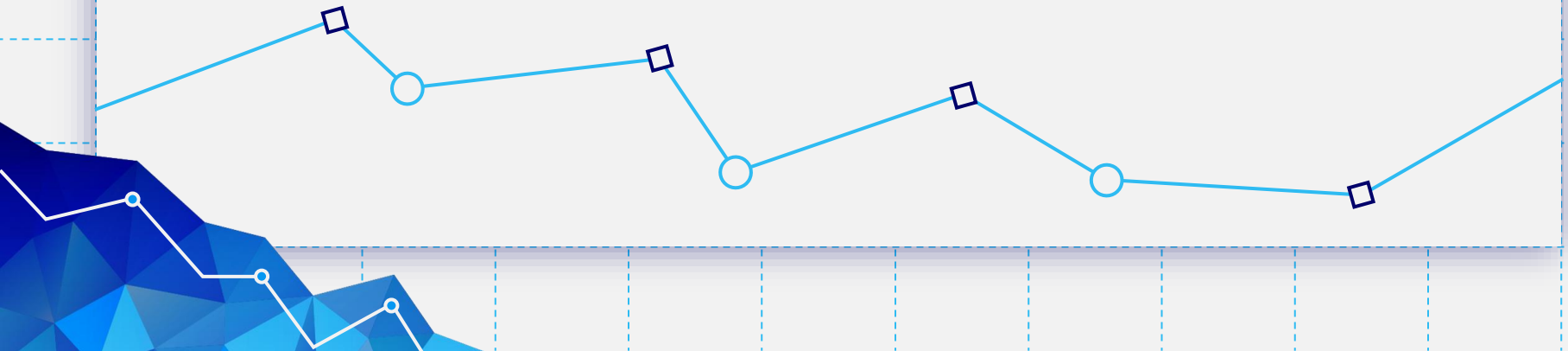
ML project presentation

Giacomo Fantato · **K-16958** · UW WNE 06/2025

Exploratory analysis

Performed on the training set for both classification and regression tasks



Cleaning and splitting the data

Clean missing data: remove rows with NaN values for consistency.

Reduce rare categories: group infrequent values (≤25) in categorical variables into “other".

Drop Low-Variance features: remove variables with little variation to reduce noise.

Encode categories: apply *one-hot encoding* to nominal features for model compatibility.

Visualize target: plot *price_z* and *claim_status* distribution to check class balance.

Save processed data: store both cleaned and encoded datasets for modeling in a pickle file.

Train set: *70/30 for regression, 60/40 for classification **

Train (5 or 10 folders) | Validation

Found best model for both task: →

Make the predictions

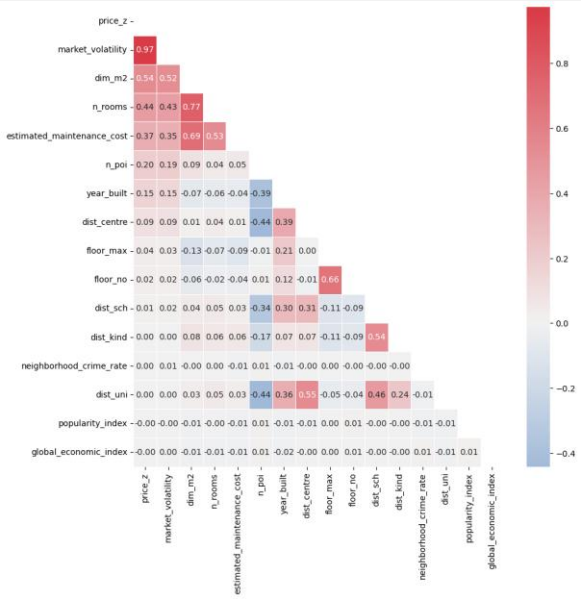
Test

NB: to create a dataset suitable for both regression and classification models, we do not remove non-informative columns (such as *unit_id* or *src_month*) at the beginning.
Instead, these columns will be dropped within each specific model pipeline, making the original dataset more flexible and reusable for various types of analysis.

*‘*random_state=42*’ for reproducibility on CV folder split on the regression | ‘*random_state=123*’ for the CV stratified – classification

Data exploration - *regression*

Now let's look at the behavior of the variables in the **regression** dataset – **quantitative** explanatory vars



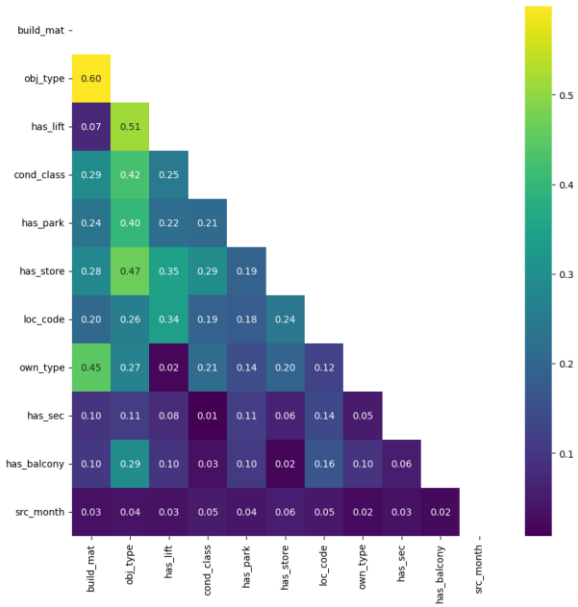
Strong correlation with price: market volatility shows a very high positive correlation with the target (key predictive variable).
Dim_m2, n_rooms, and *estimated_maintenance_cost* also show moderate to strong correlations, suggesting relevance.

Multicollinearity warning: high inter-correlation between *dim_m2* and *n_rooms*, and between *floor_no* and *floor_max,* may require attention during feature selection or model regularization.

Negative correlations: *year_built* and *distance to center* show negative relationships with *price_z*, indicating newer or centrally located apartments might be valued higher.

Data exploration - *regression*

Now let's look at the behavior of the variables in the **regression** dataset – **qualitative** explanatory vars

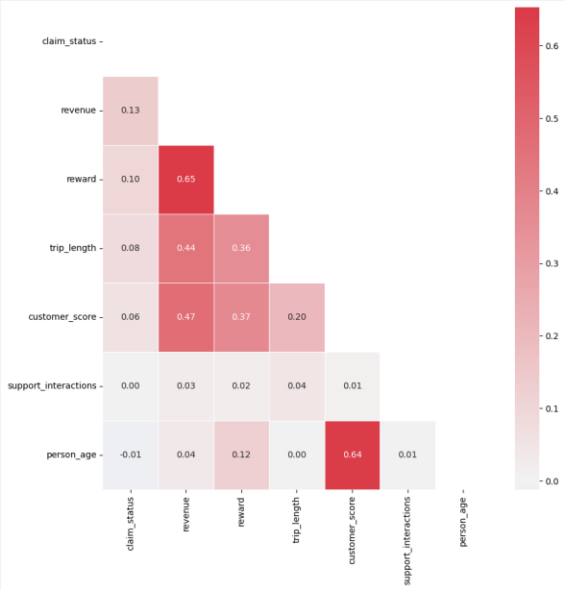


Moderate correlations: *obj_type* and *build_mat* show a notable correlation (0.60), suggesting object type may influence construction materials used. *Has_lift, cond_class*, and *has_store* also correlate moderately with several features, indicating they may capture relevant structural or amenity information.

Related amenities: moderate associations between *has_lift*, *has_park*, and *cond_class* suggest that better-conditioned or higher-quality properties are more likely to include amenities.

Data exploration - *classification*

Now let's look at the behavior of the variables in the **regression** dataset – **quantitative** explanatory vars



No strong correlation found between individual features and **claim_status** (highest: revenue = 0.13).

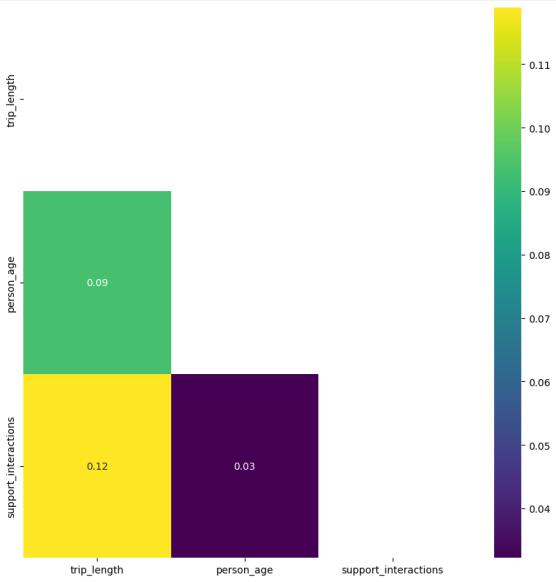
Moderate correlations observed among features:

- Reward and Revenue (0.65) → it suggest that higher premiums are associated with higher revenues.
- *Customer_Score* and *Reward* (0.47)
- *Customer_Score* and *Person_Age* (0.64) → suggest that clients with higher scores tend to be older and receive higher rewards.

Moderate correlations between some independent variables (*reward*, *revenue*, *customer_score*, and *person_age*) suggest **potential multicollinearity**.

Data exploration - *classification*

Now let's look at the behavior of the variables in the **regression** dataset – **qualitative** explanatory vars



Weak correlations across all variable pairs.

Slight positive link between *trip_length* and *support_interactions* (0.12).

Very low correlation between *person_age* and other variables. That means there's no strong linear relationships detected; these features are largely independent.

Regression task

Overview of regression techniques used

Ridge, LASSO, Elastic Net – best model

Elastic Net is the first-best performing model found. The best parameters found due to the pipelining are:

`'model__alpha': 0.00046415888336127773, 'model__l1_ratio': 0.3, 'select__k': 'all'`

with **5 folds** for each of **200 candidates**, total **1000 fits**.

Test set (mean CV):

RMSE: 101953.8414 | **MAE**: 75167.8712 | **MedAE:** 58191.1299 | **R²:** 0.9532

Validation set:

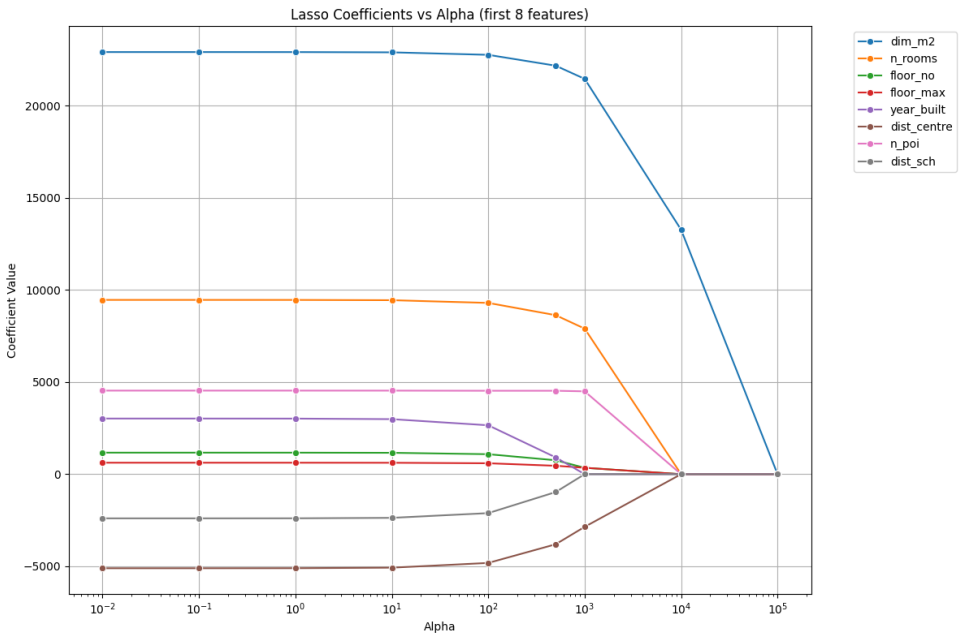
RMSE: 101838.4489 | **MAE**: 75315.6437 | **MedAE**: 57549.2436 | **R2**: 0.9534

It showed excellent generalization, with validation RMSE nearly matching CV RMSE. A low alpha and moderate *l1_ratio* (0.7) yielded the best results, balancing L1 and L2 regularization. In addiction the **RMSE %** (RMSE as percentage of mean z-price) is **12.39%**

The high R^2 (0.95) and low MAPE (9.32%) indicate strong predictive performance, confirming ElasticNet as a reliable and interpretable alternative to SVR. The model **does not overfit**, as shown by minimal RMSE gaps: 0.22% (Train vs Val) and 0.11% (CV vs Val).

'unit_id' and *'src_month'* were excluded due to their lack of predictive relevance.

Ridge, LASSO, Elastic Net – best model



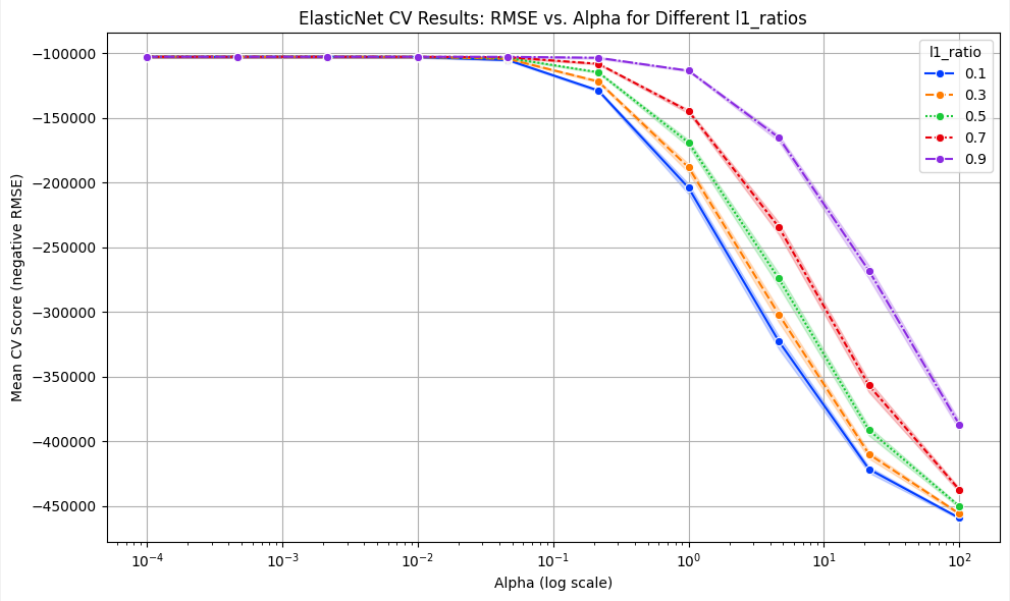
LASSO coefficient analysis reveals that *dim_m2* and *n_rooms* consistently have the largest and most stable coefficients, indicating they are strong and robust predictors of the target variable.

Features like *dist_centre*, *dist_sch*, and others are shrunk to zero when alpha is large. These may be less info or redundant.

Computaional costs (L1): $O(K^2n)$ if $K < n$, $O(k^3)$ if $K \geqslant n$; better SVR*.

** Least Squares Optimization with L1-Norm Regularization, Mark Schmidt, CS542B, Project Report, December 2005*

Ridge, LASSO, Elastic Net – best model



Elastic Net analysis: ElasticNet performance is optimal at low alpha values across all *l1_ratios*.

Models with **stronger L1 regularization** (l1_ratio between 0.7 – 0.9) **slightly outperform others**, suggesting that promoting sparsity is beneficial, as long as the regularization strength remains moderate.

Computational costs: $O(AK^2n)$ | $O(Ak^3)$, where A is the grid size for tuning the parameters, that balances the weights of ridge versus LASSO; SVR still better in performance.

SVR – second model

After evaluating 45 combinations of hyperparameters using GridSearchCV with 10-fold cross-validation (10 folds for each of 45 candidates, 225 total fits), the SVR model with linear kernel emerged as the second-best performing model. Optimal parameters found:

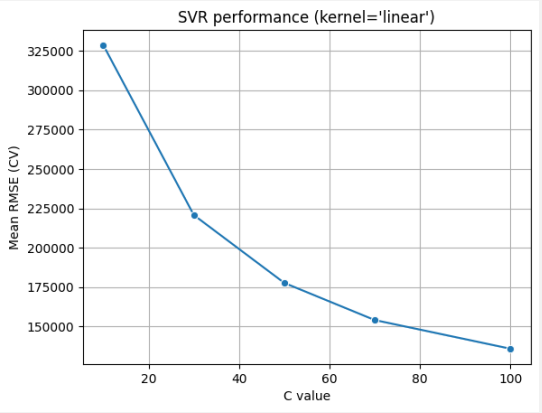
```
'model__C': 100 | 'model__gamma': 0.001 | 'model__kernel': 'linear'
```

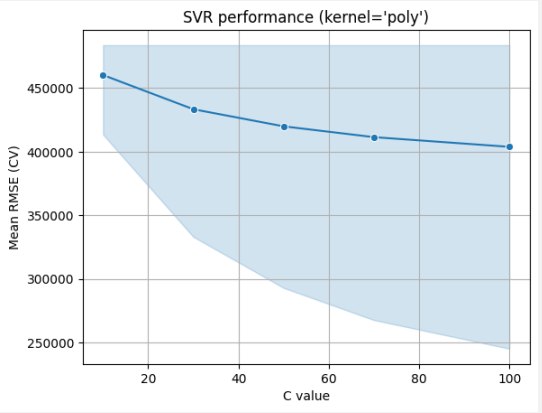
It achieved the *second-lowest* average RMSE across all tested models, demonstrating solid predictive accuracy and consistent performance on both training and validation sets. The model also proved to be stable and generalized well to unseen data.

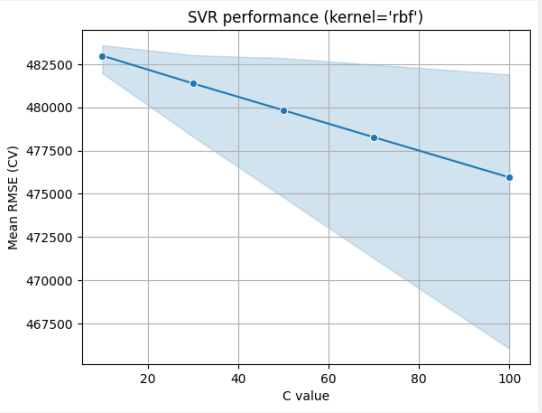
- **Avg CV RMSE**: 131.483 (+22.5%)
- **Avg CV R^2** : 0.9220 (-3.4%)
- **Validation RMSE** 130,084 (+21.6%)
- **Validation R^2** 0.9239 (-3.2%)

Although the ElasticNet model had slightly better overall performance in terms of RMSE and R^2, the SVR model was a close second, and still represents an excellent option due to its robustness and minimal overfitting.

SVR – second model







Best performance
RMSE drops sharply as C increases

Poor performance
even at high C, RMSE remain > 200K$

High RMSE
~ 475K$ across all C values

Hyper parameters optimization: a full *scikit-learn pipeline* was applied for consistent preprocessing and model training. GridSearchCV with 10-fold cross-validation was used to test 45 hyperparameter combinations (in total 450 model fits).
The best parameters are: **C = 100 | gamma = 0.001 | kernel = 'linear'.**

SVR – second model

Introducing an SVR model based on the best-performing configuration, and including **engineered interaction features** (combined and/or individual), did not lead to improvements (it resulted in significant performance degradation).
The added features either **do not contain additional predictive patterns** or **introduce noise** into the model.

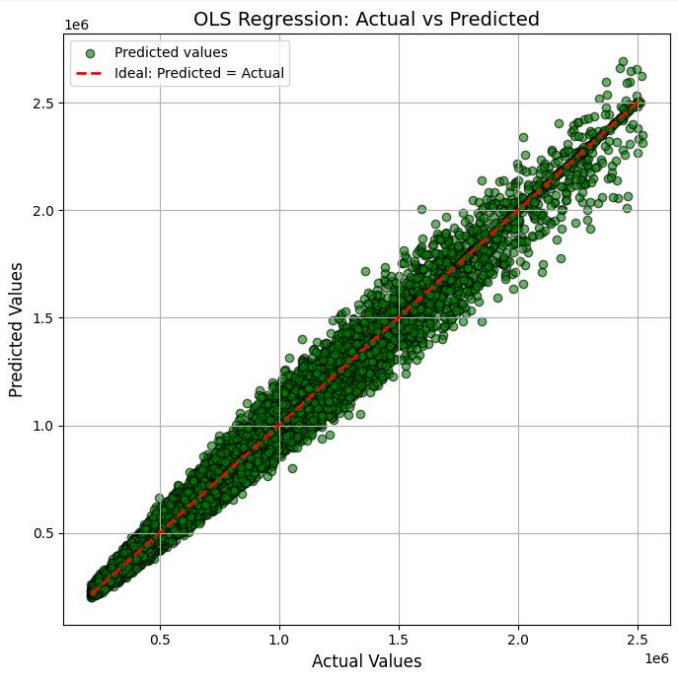
- ***m2_per_room***: living space per room (spatial efficiency) → X['dim_m2'] / (X['n_rooms'])
- ***cost_per_m2***: maintenance cost per square meter → X['estimated_maintenance_cost'] / X['dim_m2']
- ***green_dist_ratio:*** green space access relative to distance from city center → X['green_space_ratio'] / X['dist_centre']

Applying a **log_transformation** did not improve performance (target distribution is not skewed enough).

No signs of underfitting or overfitting: the model **does not overfit** the training data nor **underfit** the problem. The performance on the validation set is **in line with or better than** cross-validation results, suggesting **good generalization** (less than 10% error).

Computational cost: $O(n^3)$ for the training | $O(sv \times features)$ for the predictions

OLS – worst case scenario



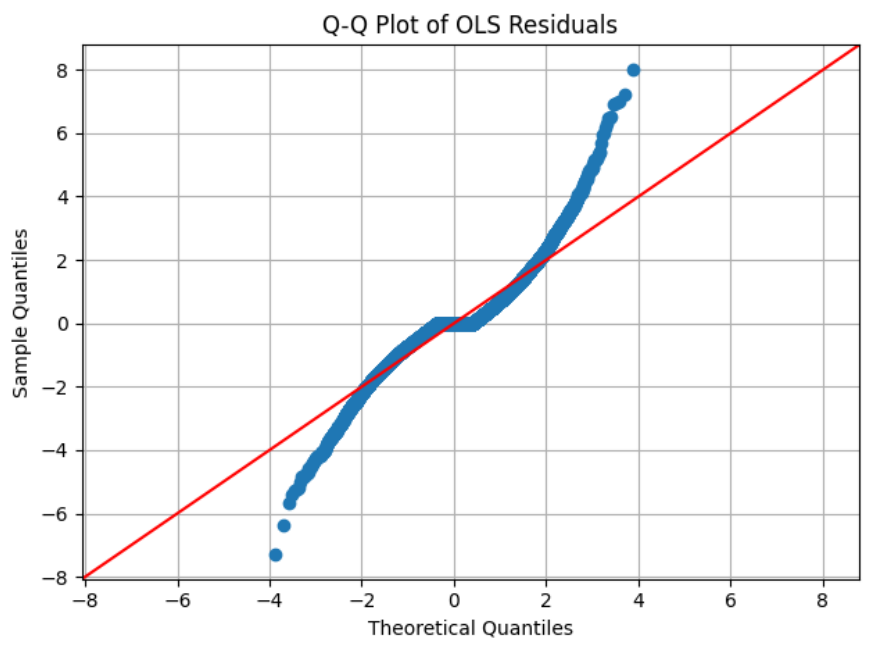
The predicted values are generally aligned along the ideal line, suggesting that the **model captures some linear relationship** between the variables.

However, there is increasing dispersion at higher value ranges, indicating that the model loses precision as the target values grow.

The presence of *systematic deviations suggests that the linear regression model fails to capture the full complexity of the dataset*, likely due to non-linear relationships or unmodeled interactions.

Mean RMSE: 132442.82
The variability across folds is also notable, indicating some instability in the model's performance. Moreover, the average RMSE is the highest recorded among all tested models.

OLS – worst case scenario



The residuals are not normally distributed, which undermines the reliability of estimates such as confidence intervals and significance tests in OLS models.

This further supports the idea that linear regression is not well suited for this dataset and that more flexible, non-linear models may provide a better fit.

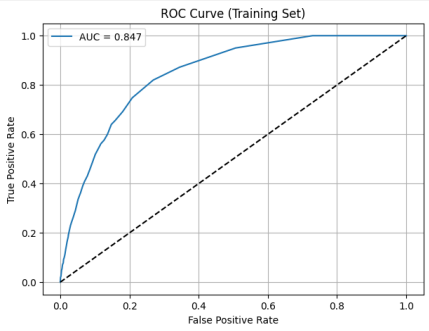
Computational cost: $O(n^2d + d^3)$, n = # of samples, d = # of features; d^3 = inverted matrix.

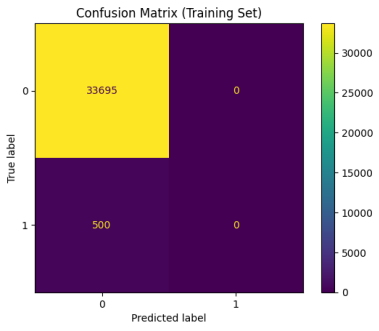
Classification task

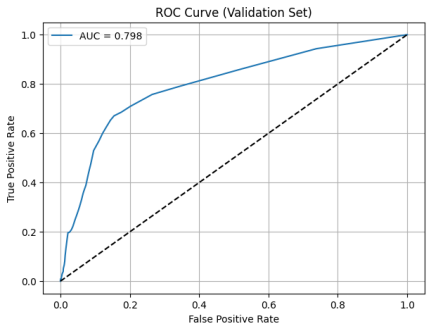
Classification models applied to predict the claim outcomes

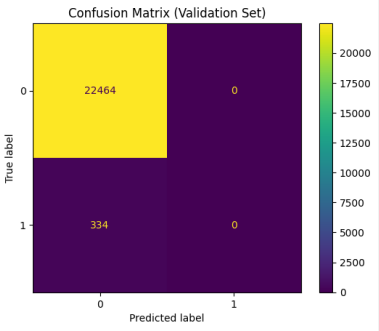
KNN – worst case scenario

Best model hyperparameters: `K = 279 | weights = uniform`









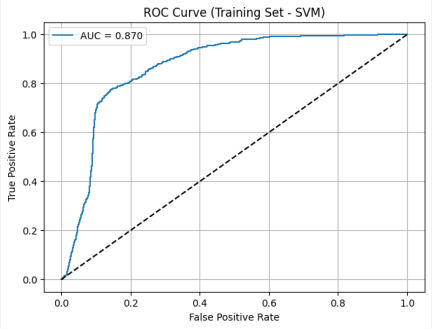
An initial approach involved the use of a K-Nearest Neighbors (KNN) model, despite the presence of a **significant class imbalance in the dataset**.

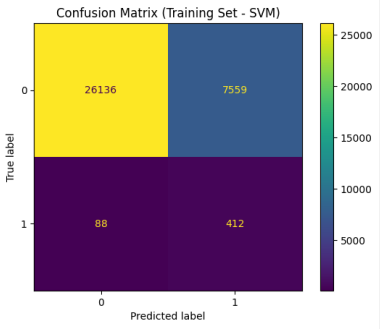
The data is heavily skewed, which is reflected in the confusion matrix above. It's noticeable that during validation, each fold yields a different AUC, while the balanced accuracy consistently remains at 0.5000.

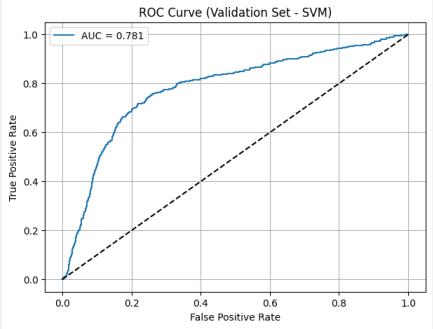
Since **KNN does not handle imbalanced classes** effectively, switching to an algorithm that supports the `class_weight='balanced'` parameter is recommended to better account for this issue.

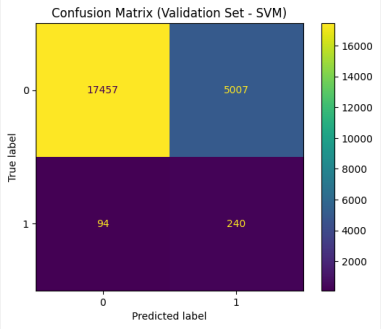
SVM – second best model

Best model SVM parameters: `Kernel: rbf | C: 1.0 | Gamma: scale | Class Weight: balanced | Probability: True`









Cross validation performance (5 folds):

- Avg ROC AUC: 0.7719
- Avg Balanced Accuracy: 0.7320
- Stable performance across folds with noticeable variation in AUC.
- Very high precision for class 0 (1.00), but low for class 1 (0.05) → class imbalance is evident.

Validation set:

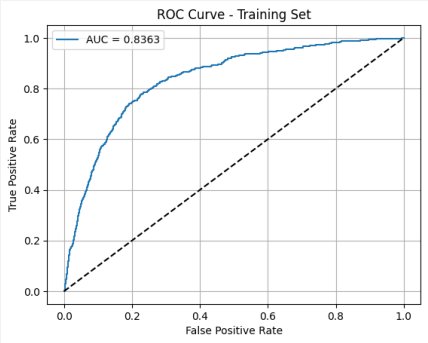
- ROC AUC: 0.7810
- Balanced Accuracy: 0.7478
- Similar trends as training: strong performance on majority class, but weak precision on minority class

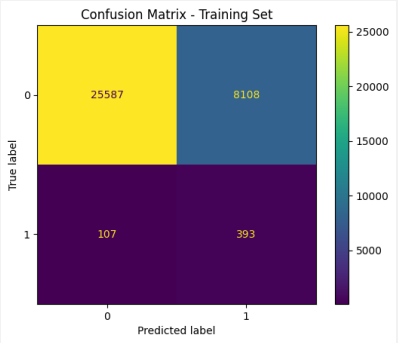
NB: same computational costs of SVR in time and resources

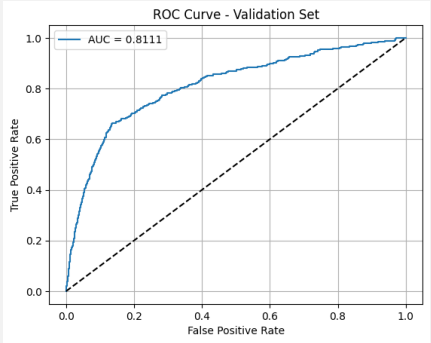
The columns *agent_id_**** and *entity_a_**** were dropped because they increase the risk of overfitting, especially with linear models like logistic regression. Additionally, they introduce sparse features with limited generalizable predictive power.

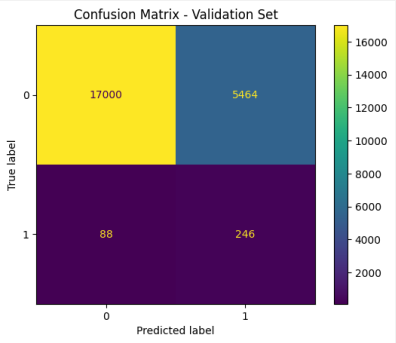
Logistic regression – best model

Best model parameters: `'logreg__C': 1 | 'logreg__penalty': 'l2' | 'logreg__solver': 'liblinear'`









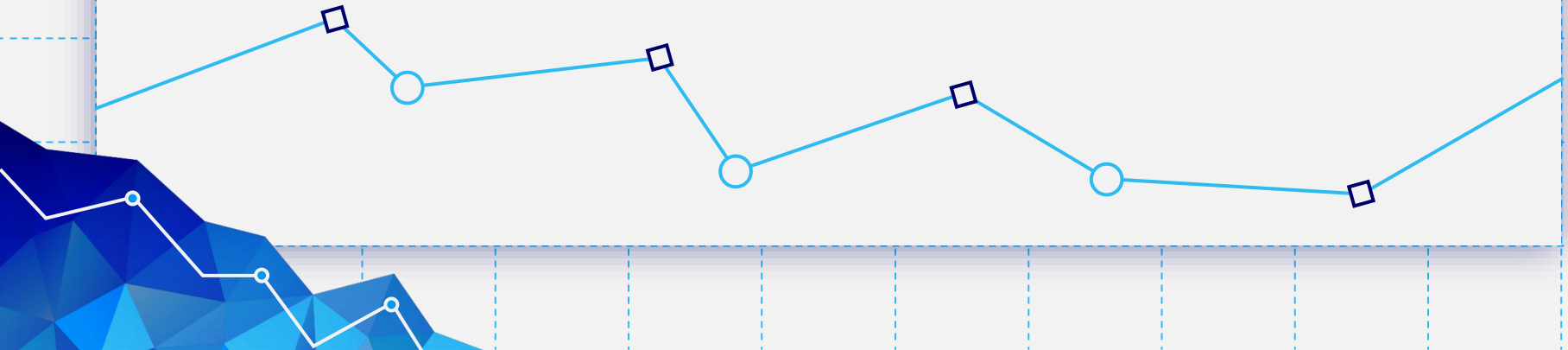
This is the best model so far because it achieves higher overall performance compared to the previous one. It shows improved average ROC AUC (**0.8363** vs. 0.7719) and balanced accuracy (**0.7727** vs. 0.7320) across cross-validation folds. Additionally, it performs better on the **validation set** with a ROC AUC of **0.8111** and balanced accuracy of **0.7466** indicating better handling of the class imbalance.

The ROC curves and confusion matrices show that this model and the SVM perform very similarly, with this model offering a slight improvement.

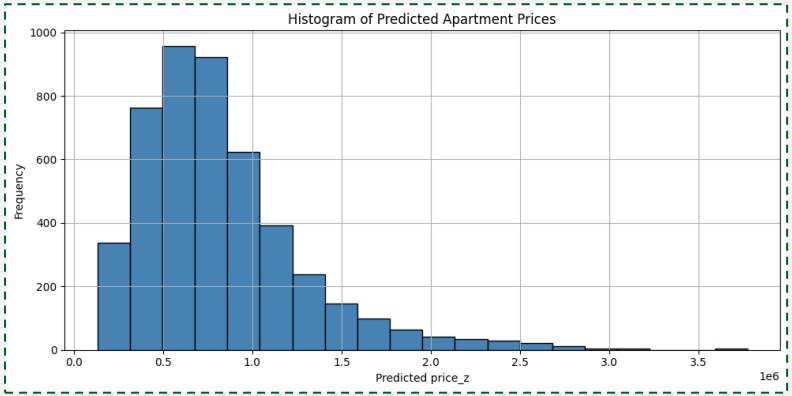
Best computational cost found: O(nd) for the training | O(d) for predictions, n = # samples, d = # features (*source*)

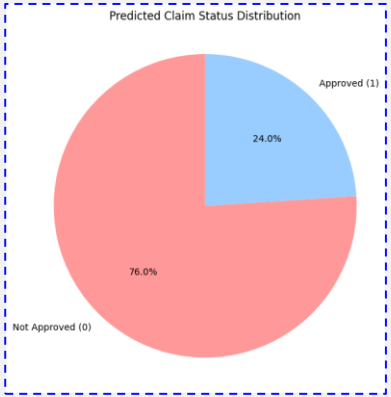
Predictions

Performed only on the best classification and regression models at the end, to avoid *data leakage*



Elastic Net and LR forecast distribution





Elastic net

Distribution of apartment prices, highlighting the most frequent price ranges.

Logistic Regression

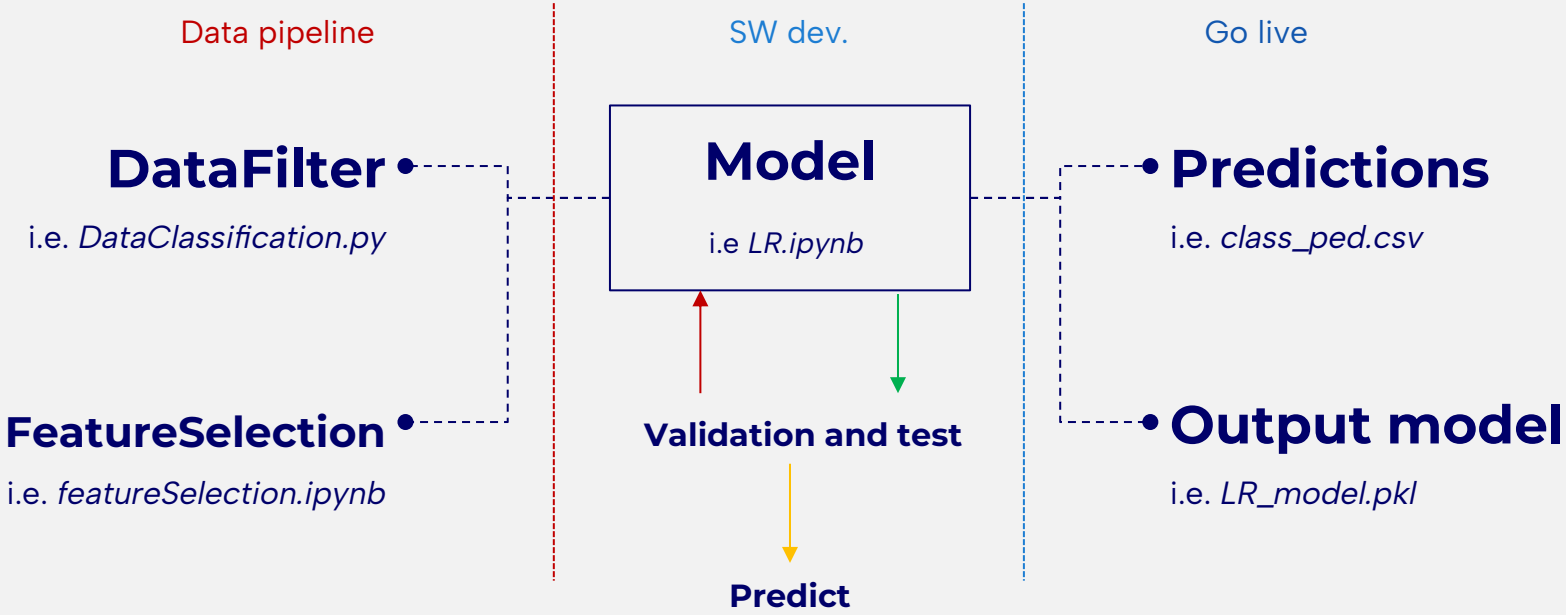
The distribution of predicted classes (0 and 1) is as follows on Test Set: **24% for 1 - 76% for 0 class**.

Test set analyses are unreliable since the construction and sampling methods of the test sets are unknown.

Technical choices and conclusion

Let's look at the technical choices made and the conclusions

Software engineering – UML diagram



Cloud computing – HPC

Each machine learning experiment was structured into modular, class-based components, separating:

- Model logic (e.g., training, validation, evaluation)
- Data loading and preprocessing

This decoupled architecture ensured high flexibility, reusability, and clean integration across different experiments (e.g., ElasticNet, Logistic Regression).
Execution was significantly accelerated using HPC resources on **Cineca's Leonardo Boost Cluster**, exploiting:

- 1× 32-core Intel Xeon Platinum 8358 CPU (2.6 GHz), 512 GB DDR4 RAM, **4× NVIDIA A100** (used 1) 64 GB GPUs, and 2× dual-port HDR (400 Gbps) interconnect.
- Parallelized K-Fold cross-validation

This reduced average training time from ~160 minutes (Intel Core i7, 10th gen laptop) to just ~2 minutes on the HPC cluster, enabling rapid experimentation and scalable model evaluation, using CUDA acceleration.

Writing the projects as Python classes enabled fast deployment on the HPC platform, where commands are executed via shell (SLURM) and files are uploaded through FTP. This structure made the code easy to load and run efficiently on Cineca's system.

Cloud computing – HPC

```
#!/bin/bash
#SBATCH --job-name=svm_job              # Job name
#SBATCH --output=logs/svm_output_%j.log # Output log (%j = job ID)
#SBATCH --error=logs/svm_error_%j.log   # Error log
#SBATCH --partition=gpu                 # Partition name (use the one defined for GPUs)
#SBATCH --gres=gpu:1                    # Request 1 GPU
#SBATCH --cpus-per-task=4               # Number of CPU cores per task
#SBATCH --mem=16G                       # Total memory
#SBATCH --time=01:00:00                 # Max run time (hh:mm:ss)

# Load modules
module load python/3.10
module load cuda/12.2

# Activate your environment
source ~/envs/ml_env/bin/activate

# Run your Python script
python SVM.py
```

Example of model execution on HPC Leonardo using a SLURM batch script for SVM training

`sbatch svm_job.sh`

Thanks!

Do you have any questions?

g.fantato@student.uw.edu.pl

GitHub Project link