Machine Learning - 20596

Final Challenge Report

Professor: Dr. Durante Daniele
Student: Negri Giacomo, 3155287

1. Executive summary

This report outlines the end to end workflow used to predict monthly rental prices in Milan and highlights the main insights obtained. It is structured into four sections: executive summary, methodology, reproducibility of the code, and interpretation and data insights.

Building on this structure, the modeling began with an extensive feature engineering, which turned semi-structured text into numeric fields, creating dummy variables, encoding categorical features, and applying a response transformation, a stacked ensemble of XGBoost and LightGBM gradient-boosted regressors was fitted. These algorithms were chosen for their strength in capturing non-linear interactions and underliying patterns across different data and regressor.

Model selection focused on minimizing mean absolute error and was rigorously validated with 10-fold cross-validation on the hold-out test set. The final model achieved a MAE of €297.41 on the limited unseen tested data for rental prices across Milan's neighborhoods.

2. Methodology

This section explains the methodological choices behind the selected approaches and introduce some techniques that will be seen in the detailed code section that follows.

Data Preparation and Feature Engineering Key transformation steps included:

- **String to numeric**: fields in `description` and `other_features` contained semi-structured text that encoded crucial insights, such as room, bathroom counts, optic-fiber, availability, and others. Therefore, extracting these cues and casting them to numeric was important as to allow tree-based and baseline models to extract information otherwise invisible to them. This represented also an important feature expansion step, given the modest initial feature count.

- **Dummy variable creation**: one hot encoding was applied only to categorical attributes with few levels and clear relevance, such as energy class, contract type, top features, and macro-zones. This avoided future models from making ordinal assumptions and allowed to isolate each category's effect, simplifying later interpretation.

- **Zone encoding**: with 132 unique `zone` values, one hot encoding would have produced a highly sparse matrix. By consequence executing target encoding, or replacing each label with the mean rent of its

zone delivered improved MAE, even beating median encoding, while at the same time it kept model's complexity in check.

- **Log transformation**: because `y` and `condominium_fees` were highly skewed, a log transformation reduced asymmetry, making distributions easier for models to learn and leverage.

Model selection Multiple models: linear, Ridge, Lasso, logistic, Random Forest were tested through train/validation split. They furnish the benchmark on which XGBoost and LightGBM were chosen, because top performers by a clear margin.

Gradient-boosted trees These models achieved superior performance for several reasons. First, they capture non-linear interactions by splitting on feature thresholds, these models uncovers complex relationships which others can't. Second, they handle high-dimensionality in the dataset and dummy variables with relative ease, avoiding the curse of dimensionality. Third, unlike others, they offer L1/L2 regularization and shrinkage, which is an effective method against over-fitting and redundant variables. Fourth, boosting iteratively reduces residuals by building trees sequentially. Fifth, flexible feature sampling is available at the tree, level, or node through `colsample_bytree/_bylevel/_bynode`. Sixth, rows sampling without replacement is possible through `subsample`.

XGBoost and LightGBM While they are closely related algorithms that also differ in several nuances. XGBoost makes use of the Hessian, its second order gradient, so that each split follows a Newton ("exact greedy") update, computing the optimal threshold and extracting the last bit of performance through reduction of RSS. LightGBM, by contrast, relies on an histogram-based method where every feature's continuous range is bucketed into k bins. Only $k-1$ of these are examined, and it the choice is optimises with first order gradients. Thus, while XGBoost prioritises pure performance, LightGBM achieves faster optimisation. Moreover, tree construction strategies diverge: XGBoost grows depth-wise, expanding all branches to the same depth before proceeding, whereas LightGBM adopts a leaf-wise approach, always splitting the leaf that promises the greatest loss reduction. The depth-wise tecniques moderate the effect of extreme leaves and keeps memory use steadier. On the other hand, the leaf-wise approach pushes deeper, often discovering richer interactions. In short, XGBoost can give more precise local improvements while LightGBM can uncover deeper interactions more efficiently. More details are available at the following references: [2] for XGBoost and [3] for LightGBM. In this work, it was used `tree_method='hist'` in XGBRegressor. This approximation brings XGBoost closer to LightGBM's histogram-based strategy, offering substantial speedups for large datasets without a major loss in accuracy. Therefore, as LightGBM, also XGBoost discretize features values into bins before finding the right split, which reduced computation time and memory usage, given also the constraint to a local running.

Stacking rationale Given what was said above, stacking these models felt natural as they operate in a complementary fashion and avoids significant risk of combinatory disruption.

Meta-Learner with `ElasticNetCV` After a simple weighted average of the two base models resulted in adequate but unsatisfactory results, a meta-learner was tried and selected. A meta-learner with Elastic Net offers three key advantages. First, shrinkage encourages parsimony through L1 regularisation, which works like a Lasso. Second, it remains stable under multicollinearity: because the base learners L2 component smooths the weights of the two models. This is especially relevant when two models are closely related, such is the case for XGBoost and LightGBM. This should reduce variance. Third, there is a built-in cross-validation that test the model on held out folds, ensuring reliability of results. Therefore, unlike a normal weighted average, which rescales all coefficients uniformly, the meta-learner adjusts weights through cross-validation to minimise error. This makes the Elastic Net meta-learner a robust way to boost performance while avoiding over-fitting. For more details about it, consult source [5].

3. Reproducibility of the code

Below it is outline the steps that were implemented to achive the model. Every reference to "See appendix" points to the full extended code.

Parsing the `description` column Each entry is first split on commas, producing a list of components that is then converted to structured data with `extract_structured_info`. Regex patterns retrieve `total_rooms`, `bedrooms`, `other_rooms`, `bathrooms`, `kitchen_type`, and `accessible_bathroom`. All are numeric except `kitchen_type`, a categorical string taking the values `open`, `nook`, `semi`, `ette`, and `diner`. When `total_rooms` is absent, it is imputed as the sum of the other room counts.

```
for df in dataset:
df['description_list']=df['description'].str.split(', ', regex=False)
```

```
def extract_structured_info(desc_list):
\end{verbatim}
\vspace{-0.5\baselineskip}
See appendix~\ref{app:extractinfo}.
 \vspace{-0.5\baselineskip}
\begin{verbatim}
for df in dataset:
  df[['total_rooms', 'bedrooms', 'other_rooms', 'bathrooms',
  'kitchen_type', 'accessible_bathroom']] = df['description_list']
  .apply(extract_structured_info)
for df in dataset:
  df['total_rooms'] = df['total_rooms'].fillna(
    df[['bedrooms', 'other_rooms', 'bathrooms']].sum(axis=1))
```

Other features, macro, and top features The unstructured `other features` field is first split into lists and one-hot encoded with multi-label binarization. Items are then rolled up into nine macro groups: `furnishing`, `safety`, `tv_internet`, `outdoor_space`, `concierge`, `storage`, `windows`, `exposure`, and `luxury`, through the use of a mapping dictionary. Finally, binary flags for the most frequent individual features capture the top amenities reported in the listing, highlighting valuable add-on.

```
for df in dataset:
  df['other_features_list'] = df['other_features'].str.split(' | ', regex=False)
for df in dataset:
  df['other_features_list'] = df['other_features_list'].apply(
      lambda x: x if isinstance(x, list) else [])
for df in dataset:
  mlb = MultiLabelBinarizer()
  feature_dummies = pd.DataFrame(mlb.fit_transform(df['other_features_list']),
      columns=mlb.classes_,
      index=df.index)
```

See appendix B.

```
def aggregate_macro_features(feature_list, mapping):
    if not isinstance(feature_list, list):
        return []
    return list({mapping.get(f, None) for f in feature_list if mapping.get(f, '')
        })

for df in dataset:
  # Apply the macro feature mapping to your DataFrame
  df['macro_features'] = df['other_features_list'].apply(
      lambda feats: aggregate_macro_features(feats, macro_feature_map))
for df in dataset:
  for col_name, keywords in feature_groups.items():
      df[col_name] = df['other_features_list'].apply(
          lambda features: int(any(item in features for item in keywords)))
for df in dataset:
  for feature in macro_features_list:
      df[feature] = df['macro_features'].apply(lambda x:
      x.count(feature) if isinstance(x, list) else 0)
```

Modifying energy records and mapping macro-zones In the energy-class column, inconsistent symbols, in particular commas are replaced by the label `g`. The zones are then mapped to broader macro-zones, following the subsequent zoning scheme [7], so spatial effects are retained while reducing sparsity. This mapping will be applied later.

```
for df in dataset:
  df['energy_efficiency_class'] = df['energy_efficiency_class'].replace(',', 'g')
```

See appendix C.

Geospatial coordinates A lookup table links each Milan `zone_key` to its `macro_area`. Using a rate-limited Nominatim geocoder, every zone label (e.g. “Duomo”) is queried as follow “Duomo, Milano, Italy”;

the respective latitude and longitude are stored in two new columns with the same name. These coordinates are later merged with the full dataset on the `zone` and `zone_key` columns.

Cleaning and dummy generation function Empty categorical fields are imputed with specific value for each feature, depending on the column's mode. Specifically for `energy_efficiency_class`, it was assumed that houses in the same zone share a build era and fill with the zone mode; missing `floor` values are set to the overall mode. One hot encoders then produce dummies for `contract_type` and `energy_efficiency_class`. Finally, zone labels format is modified and mapped to the earlier macro-area scheme.

```
def preprocess_text(text):
    text = text.lower()
    text = text.replace(' - ', ' ')
    text = text.replace("'", "")
    translator = str.maketrans('', '', string.punctuation)
    text = text.translate(translator)
    text = text.replace(' ', '_')
    return text

def preprocess_data(df_train, df_test):
    # Combine train and test temporarily for consistent processing
    df_all = pd.concat([df_train, df_test], keys=["train", "test"])

    # 1. Fill missing values
    df_all['availability'] = df_all['availability'].fillna('available')
    df_all['conditions'] = df_all['conditions'].fillna('excellent')
    df_all['condominium_fees'] = df_all['condominium_fees'].fillna(0.00)
    df_all['other_features'] = df_all['other_features'].fillna(' ')

    # Fill 'energy_efficiency_class' by mode within each zone
    zone_energy_mode = df_all.groupby('zone')['energy_efficiency_class']
    .agg(lambda x: x.mode().iloc[0] if not x.mode().empty else None)
    df_all['energy_efficiency_class'] = df_all.apply(
        lambda row: zone_energy_mode[row['zone']] if pd.isna(row['
            energy_efficiency_class'])
        else row['energy_efficiency_class'], axis=1)

    # Replace string "nan" in 'floor' with the most frequent floor
    most_freq_floor = df_all['floor'].replace("nan", pd.NA).mode()[0]
    df_all['floor'] = df_all['floor'].replace("nan", pd.NA).fillna(most_freq_floor
        )

    # 2. One-hot encode 'contract_type'
    df_all = pd.get_dummies(df_all, columns=['contract_type'], drop_first=True)

    # 3. Encode 'energy_efficiency_class' as ordered category
    energy_order = ['g', 'f', 'e', 'd', 'c', 'b', 'a']  # You may adjust this
        order
    df_all['energy_efficiency_class'] = pd.Categorical(
        df_all['energy_efficiency_class'], categories=energy_order, ordered=True).
            codes

    df_all['zone']=df_all['zone'].apply(preprocess_text)
    df_all['macro_zone']=df_all['zone'].map(zone_mapping)
    df_all=pd.get_dummies(df_all,columns=['energy_efficiency_class'],drop_first=
        True)
    df_all=pd.get_dummies(df_all,columns=['macro_zone'],drop_first=True)

    # Separate back into train and test
    df_train_processed = df_all.loc["train"].copy()
    df_test_processed = df_all.loc["test"].copy()
    return df_train_processed, df_test_processed
```

Encoding floor and deriving new columns Textual floor labels are mapped to integers, while `elevator` is dummizied. Two new flags, `high_floor_no_elevator` and `have_condo_fees`, are created to mark units on floor 3 or above without elevator and those rentals that include condominium fees. These

attributes might deter potential tenants and be indicators.

```
clean_train['have_condo_fees']=np.where(clean_train['condominium_fees']>0,1,0)
clean_test['have_condo_fees']=np.where(clean_test['condominium_fees']>0,1,0)
clean_train, clean_test=preprocess_data(df_train, df_test)
floor_map = {
    'Semi-basement': -1,
    'Mezzanine': 0.5,
    'Ground floor': 0,
    '1': 1, '2': 2, '3': 3, '4': 4,
    '5': 5, '6': 6, '7': 7, '8': 8, '9': 9
}

for df in clean_df:
  df['floor'] = df['floor'].map(floor_map).astype(float)
  df['elevator']= df['elevator'].map({'yes':1,'no':0})
  df['high_floor_no_elevator'] = ((df['floor'] >= 3) & (df['elevator'] == 0)).
      astype(int)
  \end{verbatim}

\textbf{Filtering columns for model preparation.}
The DataFrame is trimmed to mantain only those in the \texttt{columns\_to\_keep}
    list, ensuring the model consider only numerical features. See appendix~\ref{
    app:columnskeep}.
  \begin{verbatim}
filtered_df=clean_train[columns_to_keep]
filtered_test=clean_test[columns_to_keep]
  \end{verbatim}
\textbf{Logarithmic transformation.}
Both \texttt{rent\_price} ($y$) and \texttt{condominium\_fees} are log-transformed
    .

  \begin{verbatim}
filtered_df.loc[:,'y'] = np.log1p(filtered_df['y'])
filtered_df.loc[:, 'condominium_fees'] = np.log1p(filtered_df['condominium_fees'])
filtered_test.loc[:, 'condominium_fees'] = np.log1p(filtered_test['
    condominium_fees'])
```

Final model An exhaustive GridSearch was tuned the estimator over a wide hyper-parameter grid: `l1/l2` regularisation, `boosting_type`, `n`, `learning_rate`, `subsample`, `colsample_bytree`, `num_leaves`, `min_child_samples`, `reg_alpha`, and `reg_lambda`, to find the optimal configuration. `subsample` and `colsample_bytree` were also tested but did not yield additional performance gains by differentiating from the usual 0.8. The following are the optimal model parameters.The following are the optimal model.

```
xgb_model = xgb.XGBRegressor(
    objective='reg:squarederror', random_state=42, tree_method='hist',
    enable_categorical=False, n_jobs=-1,
    # --- tuning parameters
    n_estimators=4600, learning_rate=0.0085, max_depth=8, min_child_weight=1,
    subsample=0.8, colsample_bytree=0.8, reg_alpha=0.01, reg_lambda=0.0)

lgb_model = lgb.LGBMRegressor(
    boosting_type='gbdt', objective='regression', random_state=42,
    verbose=-1, n_jobs=-1,
    # --- tuning parameters
    n_estimators=4600, learning_rate=0.008, num_leaves=63, min_child_samples=1,
    subsample=0.8, colsample_bytree=0.8, reg_alpha=0.0, reg_lambda=0.01)
```

Normalization search This step investigates whether applying `l1` and `l2` normalization as an overlay can further improve predictions beyond GridSearch results. Using a stacking regressor that combines XGBoost and LightGBM, performances were evaluated via a meta-learner with 10-fold cross-validation. Interestingly, the optimal normalization weights for both `l1` and `l2` are near zero. Notably, a simple weighted average of XGBoost and LightGBM underperforms compared to the slightly normalized elastic net stack.

```
xgb_model = xgb.XGBRegressor(
    objective='reg:squarederror', random_state=42, tree_method='hist',
```

```
    enable_categorical=False, n_jobs=-1,
    # --- tuning parameters
    n_estimators=4600, learning_rate=0.0085, max_depth=8, min_child_weight=1,
    subsample=0.8, colsample_bytree=0.8, reg_alpha=0.01, reg_lambda=0.0)

lgb_model = lgb.LGBMRegressor(
    boosting_type='gbdt', objective='regression', random_state=42,
    verbose=-1, n_jobs=-1,
    # --- tuning parameters
    n_estimators=4600, learning_rate=0.008, num_leaves=63, min_child_samples=1,
    subsample=0.8, colsample_bytree=0.8, reg_alpha=0.0, reg_lambda=0.01)
meta_learner = ElasticNetCV(
    l1_ratio=[.01],
    alphas=[0.00046415888833612782],
    cv=10,
    random_state=42)
stack = StackingRegressor(
    estimators=[
        ('xgb',  xgb_model),
        ('lgb',  lgb_model)],
    final_estimator=meta_learner,
    cv=KFold(n_splits=10, shuffle=True, random_state=42),
    n_jobs=-1,
    passthrough=False)
```

Prediction. The model is fitted and predictions computed, then transformed back to monetary values by exponentiating the outputs.

```
stack.fit(X_train, y_train)
y_pred_test = stack.predict(filtered_test)
y_pred_test_normal = np.expm1(y_pred_test)
```

The rest of the code, along with the notebooks concerning exploratory data analysis (EDA) and results analysis, is available in the GitHub repository at the following reference [4].

4. Interpretation and Data Insights

Missing and incomplete data During the implementation of the solution, several peculiarities emerged from the housing listings. To begin with, many entries were missing in the energy_efficiency_class (380), condominimum_fees (197), and availability (645). While energy efficiency seem quite a significant factor, its relevance could be lower for renter, who might see the current housing as a momentary solution, particularly given the high rate of home ownership in Italy (71%)[1]. Moreover, given that most listing fall into the lower category of efficiency the energy efficiency might be a positive factor, but an unlikely driver of price. (Reference: Fig. 5).

Rental contract types Another notable observation concerns rental contract types. Most listings do not specify a contract format, but when they do, they tend to follow the standard Italian 4+4 model (four years with the option to renew for another four). Surprisingly, student-specific contracts were rarely mentioned. These offer significant tax benefits, but they may be underutilized possibly because they require additional bureaucratic effort, and private landlords or agencies might not find the added complexity worthwhile. This is odd given that many rentals are in the città study or close by, locations that are a prime choice for university students. (Reference: Fig. 6).

Floors and buildings' height The floor distribution indicates that most buildings have around three stories, suggesting that taller buildings could potentially command a premium. Although the highest floor recorded is the 9th, it remains within a relatively moderate range. (Reference: Fig. 7).

Geographical Concentration and prices Geographically, listings are significantly concentrated in areas such as Città Studi, Porta Romana, and Navigli. Despite their central location and popularity, these zones exhibit relatively low average prices, possibly reflecting higher population density and a prevalence of smaller or older units. Notably, 80% of all listings are concentrated within just 48% of the zones, indicating a moderate Pareto distribution that is not excessively skewed. Moreover, price levels vary significantly across different areas, reinforcing the idea that location is a determinant in price formation (Reference: Fig. 1).

Notable property features Among the various features, a few stood out. Whether an apartment is furnished,, which might plays a crucial role in rentals and indeed was the most frequently mentioned in

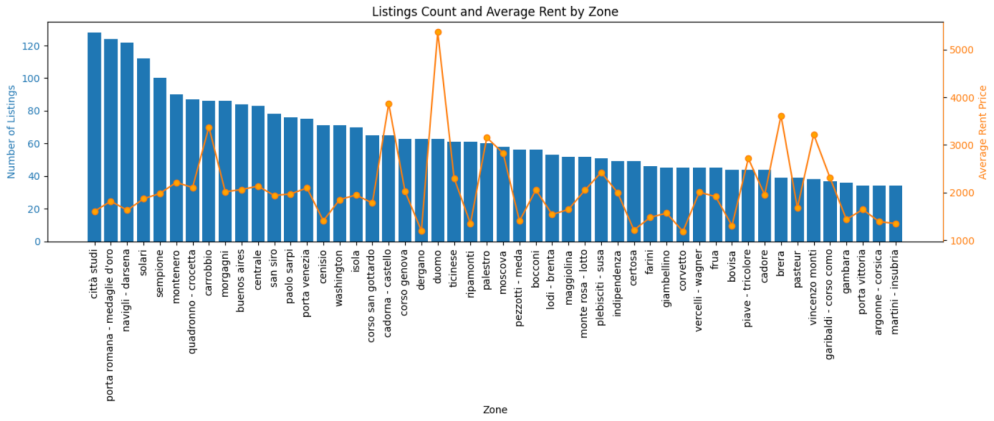

Figure 1: Average price by zone for top 50 by frequency

other_feature. Other notable attributes include the presence of a concierge and various luxury elements like fireplaces, hydromassage tubs, pools, and tennis courts, which, while less common, might influence pricing.

Lastly, both rent prices and condominium fees exhibit left skewed distributions. Rent prices have a mean of €1 983 and a median of €1 550, while condominium fees have a mean of €188 and a median of €150, as shown in the accompanying plots. This indicates a concentration of listings at the lower end of the price range, with a few high-end outliers pulling the mean upwards.(Reference: Fig. 2).

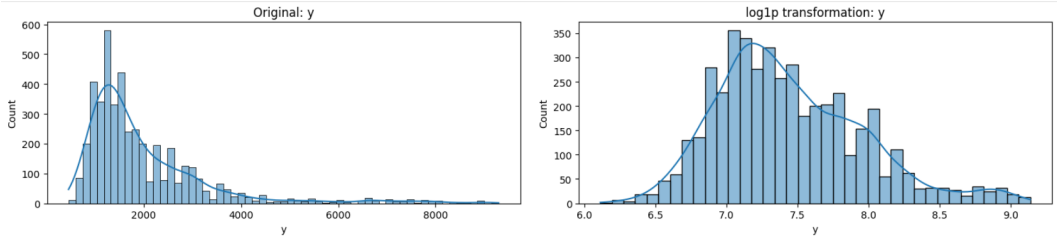

Figure 2: Price distribution: actual (Left) and log-transformed (Right)

LightGBM feature importance analysis In evaluating the model, LightGBM's feature importance analysis reveals those key drivers in the property market. Square meters is, overwhelmingly, the dominant predictor, with approximately 3 times the importance of any other feature. This confirms the intuitive understanding that property size is the primary determinant of value, as represented by the correlated number of bathrooms and total_rooms. Location factors emerge as the second the second for importance, with zone_encoded showing strong predictive power, given also that it is encoded as the mean price. This suggests that specific neighborhoods or districts have a significant influence on pricing beyond what is captured by raw geographical coordinates (longitude and latitude), which appear as moderately important features lower in the ranking. This may be because neighborhoods implicitly reflect factors such as proximity to the center, like the Duomo, or local peculiarities including parks, schools, and other desirable characteristics. Interestingly, the presence of condominium fees also appears to be a distinguishing factor, likely separating listings with no fees from those that include this additional costs. (Reference: Fig. 3).

XGBoost feature importance comparison Evaluating the feature importance from the complementary XGBoost model reveals that energy efficiency, class 4, emerges as the most influential predictor, followed closely by the number of bathrooms and square meters. This likely indicates that this particular energy class provides a good initial split, allowing the model to better differentiate between property types and reduce residuals effectively. Location continues to play a significant role, and most other influential features remain consistent. Notably, the distribution of importance is more balanced in XGBoost compared to LightGBM, which enables the model to capture a different range of nuances in the listings. (Reference: Fig. 3)

Meta-learner results analysis The final blend was:

$$\hat{y} = 0.421 \cdot \text{XGB} + 0.572 \cdot \text{LGB} + 0.055$$

This confirms the complementary nature of the two models, as well as part of their similarity. LightGBM, with the higher weight, captures deeper interactions, while XGBoost contributes stability with more homo-

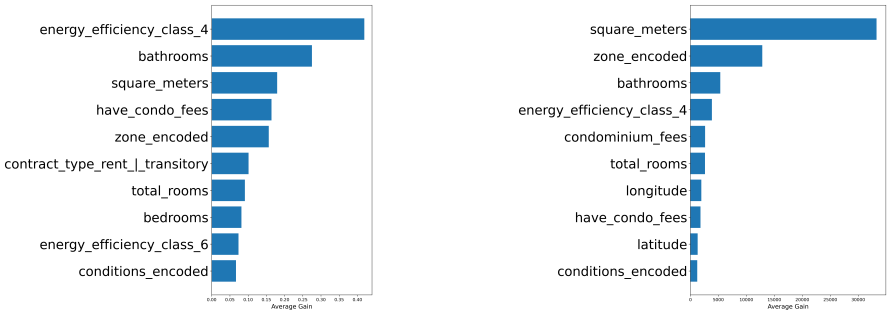

Figure 3: Top-25 feature importances for XGBoost (Left) and LightGBM (Right)

geneous features weights. The ElasticNet meta-learner balances both effectively, using the small intercept as a bias correction.

Permutation Importance of stack Permutation importance highlights `square_meters` and `zone_encoded` as dominant features, consistent with prior results. Notably, `longitude` and `latitude` now are much higher in the rank, underliying stack's ability to capture some unseen patter in location, previously ignored. Moreover, features like `condominium_fees` and `total_rooms` also gain relevance, reflecting specific nuances of the ensemble. For more details about permutation importance and how it compute the expected decrease in MAE, please consult [6]. (Reference: Fig. 4).

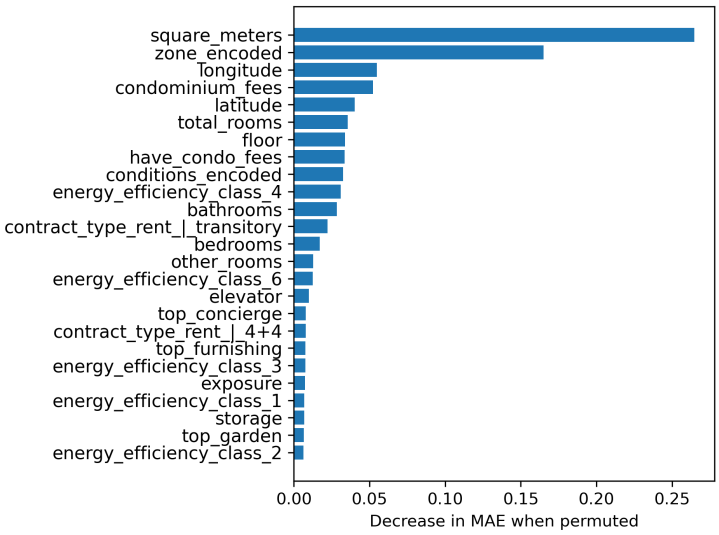

Figure 4: Permutation importance for stack model

Conclusion

This report explored the determinants of rental prices in Milan through a structured approach. Starting from an unrefined and limited dataset, key variables were engineered and transformed to enhance model interpretability and performance. Machine learning tecniques, such as XGBoost, LightGBM, and a stacked ElasticNet meta-learner, were used to capture complex relationships between features and the response variable. Moreover, the analysis highlighted the central role of property size, number of rooms, and location, while also evaluating some of the nuanced that amenities like fees and energy class may have. Besides providing useful insights, the model captured with moderate success specific patterns in Milan's rental pricing, offering the reader a deeper understanding of the market.

References

[1] Censis. *Casa, gli italiani, un popolo di prorietari*. https://www.censis.it/economia/casa-gli-italiani-un-popolo-di-proprietari. Homepage, https://www.censis.it/. 2022.

[2] *DMLC XGBoost*. https://xgboost.readthedocs.io/en/stable/index.html. Parameters https://xgboost.readthedocs.io/en/stable/contrib/ci.html, Boosted trees https://xgboost.readthedocs.io/en/stable/tutorials/model.html, 2014.

[3] *LightGBM*. https://lightgbm.readthedocs.io/en/stable/index.html. Parameters https://lightgbm.readthedocs.io/en/stable/Parameters.html, Features of LightGBM https://lightgbm.readthedocs.io/en/stable/Features.html, 2016.

[4] Negri Giacomo. *GitHub repository*. https://github.com/GiacomoNegri/UNI_Projects/tree/main/Milan_Rental_Prices. 2025.

[5] scikit-learn developers. *ElasticNetCV*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNetCV.html. Scikit-learn library, https://scikit-learn.org/stable/index.html. 2007.

[6] scikit-learn developers. *Permutation feature importance*. https://scikit-learn.org/stable/modules/permutation_importance.html. Scikit-learn library, https://scikit-learn.org/stable/index.html. 2007.

[7] Wikipedia editors. *Municipi di Milano*. https://it.wikipedia.org/wiki/Municipi_di_Milano. Comune di Milano, https://www.comune.milano.it/comune/palazzo-marino. 2016.

A Extract structured info

```
def extract_structured_info(desc_list):
    total_rooms = None
    bedrooms = 0
    bathrooms = 0
    other_rooms = 0
    kitchen_type = 'none'
    accessible_bathroom = 0

    for item in desc_list:
        item = item.strip().lower()

        # Total rooms (standard or 5+ format)
        match_total = re.match(r'(\d+)\s*\(', item) or re.match(r'(\d+)\s+rooms?',
            item)
        match_total_plus = re.match(r'(\d+)\+\s*\(', item)

        if match_total:
            total_rooms = int(match_total.group(1))
        elif match_total_plus:
            total_rooms = int(match_total_plus.group(1))  # interpret "5+" as 5

        # Bedrooms (even inside parentheses)
        match_bed = re.search(r'(\d+)\s+bedrooms?', item)
        if match_bed:
            bedrooms += int(match_bed.group(1))

        # Bathrooms (standard + accessible)
        match_bath = re.match(r'(\d+)\s+bathrooms?', item)
        if match_bath:
            bathrooms += int(match_bath.group(1))
        elif "1 bathroom" in item:
            bathrooms += 1

        if "suitable for disabled" in item:
            accessible_bathroom = 1

        # Other rooms
        match_other = re.search(r'(\d+)\s+others?\)?', item)
        if match_other:
            other_rooms += int(match_other.group(1))
        elif "1 other" in item:
            other_rooms += 1

        # Kitchen type
        if "open kitchen" in item:
            kitchen_type = "open"
        elif "kitchen nook" in item:
            kitchen_type = "nook"
        elif "semi-habitable kitchen" in item:
            kitchen_type = "semi"
        elif "kitchenette" in item:
            kitchen_type = "ette"
        elif "kitchen diner" in item:
            kitchen_type = "diner"

    return pd.Series([total_rooms, bedrooms, other_rooms, bathrooms,
    kitchen_type, accessible_bathroom])
```

B Macro features mapping

```
macro_feature_map = {
    # 1. FURNISHING
    'furnished': 'furnishing',
    'partially furnished': 'furnishing',
    'only kitchen furnished': 'furnishing',
    'kitchen': 'furnishing',

    # 2. SECURITY
    'security door': 'security',
    'electric gate': 'security',
    'alarm system': 'security',
    'video entryphone': 'security',

    # 3. TV / INTERNET
    'centralized tv system': 'tv_internet',
    'single tv system': 'tv_internet',
    'tv system with satellite dish': 'tv_internet',
    'optic fiber': 'tv_internet',

    # 4. OUTDOOR SPACES
    'balcony': 'outdoor_space',
    '3 balconies': 'outdoor_space',
    'terrace': 'outdoor_space',
    'private garden': 'outdoor_space',
    'shared garden': 'outdoor_space',
    'private and shared garden': 'outdoor_space',

    # 5. CONCIERGE
    'full day concierge': 'concierge',
    'half-day concierge': 'concierge',
    'reception': 'concierge',
    'reception1 balcony': 'concierge',

    # 6. STORAGE / ROOMS
    'closet': 'storage',
    'cellar': 'storage',
    'attic': 'storage',
    'tavern': 'storage',

    # 7. WINDOWS
    'window frames in double glass / wood': 'windows',
    'window frames in double glass / pvc': 'windows',
    'window frames in glass / metal': 'windows',
    'window frames in double glass / pvcdouble exposure': 'windows',
    'window frames in double glass / metal': 'windows',
    'window frames in glass / wood': 'windows',
    'window frames in glass / pvc': 'windows',
    'window frames in triple glass / wood': 'windows',
    'window frames in triple glass / pvc': 'windows',
    'window frames in triple glass / metal': 'windows',
    'window frames in glass / pvcdouble exposure': 'windows',
    'window frames in triple glass / pvcdouble exposure': 'windows',
    'window frames in double glass / pvcexposure south': 'windows',
    'window frames in double glass / pvcexposure east': 'windows',
    'window frames in double glass / pvcexposure north': 'windows',
    'window frames in double glass / pvcexposure south, east': 'windows',
    'window frames in double glass / pvcexposure east, west': 'windows',
    'window frames in double glass / pvcexposure north, south': 'windows',
    'window frames in double glass / pvcexposure west': 'windows',
    'window frames in triple glass / pvcexposure south': 'windows',
    'window frames in double glass / pvcexposure south, west': 'windows',
    'window frames in glass / pvcexposure east': 'windows',
```

```
    'window frames in glass / pvcexposure west': 'windows',
    'window frames in double glass / pvcexposure north, west': 'windows',
    'window frames in double glass / pvcexposure south, east, west': 'windows',
    'window frames in glass / pvcexposure north, south': 'windows',
    'window frames in double glass / pvcexposure north, south, east': 'windows',
    'window frames in double glass / pvcexposure north, east, west': 'windows',
    'window frames in glass / pvcexposure south, east, west': 'windows',
    'window frames in glass / pvcexposure north, south, west': 'windows',
    'window frames in glass / pvcexposure north': 'windows',
    'window frames in glass / pvcexposure east, west': 'windows',
    'window frames in glass / pvcexposure south': 'windows',
    'window frames in triple glass / pvcexposure east, west': 'windows',
    'window frames in triple glass / pvcexposure south, east': 'windows',

    # 8. EXPOSURE
    'internal exposure': 'exposure',
    'external exposure': 'exposure',
    'double exposure': 'exposure',
    'exposure south': 'exposure',
    'exposure east': 'exposure',
    'exposure west': 'exposure',
    'exposure north': 'exposure',
    'exposure east, west': 'exposure',
    'exposure north, south': 'exposure',
    'exposure south, east': 'exposure',
    'exposure south, west': 'exposure',
    'exposure north, west': 'exposure',
    'exposure north, east': 'exposure',
    'exposure north, south, east, west': 'exposure',
    'exposure north, south, west': 'exposure',
    'exposure north, south, east': 'exposure',
    'exposure north, east, west': 'exposure',

    # 9. LUXURY / AMENITIES
    'fireplace': 'luxury',
    'hydromassage': 'luxury',
    'pool': 'luxury',
    'tennis court': 'luxury',
}
feature_groups = {
    'top_furnishing': {'closet', 'partially furnished'},
    'top_security': {'security door', 'video entryphone', 'alarm system'},
    'top_tv_system': {'centralized tv system'},
    'top_fiber': {'optic fiber'},
    'top_concierge': {'full day concierge', 'half-day concierge', 'reception'},
    'top_garden': {'shared garden', 'private garden'},
    'top_balcony': {'balcony', 'terrace'},
    'top_kitchen': {'kitchen', 'only kitchen furnished'},
    'luxury':{'fireplace','hydromassage','pool','tennis court'}
}
```

C Macro Zones Mapping

```
zone_mapping = {
    # milano_centro
    'arco_della_pace': 'milano_centro',
    'arena': 'milano_centro',
    'brera': 'milano_centro',
    'borgogna_largo_augusto': 'milano_centro',
    'cadorna_castello': 'milano_centro',
    'carrobbio': 'milano_centro',
    'duomo': 'milano_centro',
    'guastalla': 'milano_centro',
    'lanza': 'milano_centro',
    'missori': 'milano_centro',
    'moscova': 'milano_centro',
    'navigli_darsena': 'milano_centro',
    'palestro': 'milano_centro',
    'paolo_sarpi': 'milano_centro',
    'quadrilatero_della_moda': 'milano_centro',
    'san_vittore': 'milano_centro',
    'san_babila': 'milano_centro',
    'scala_manzoni': 'milano_centro',
    'sempione': 'milano_centro',
    'vincenzo_monti': 'milano_centro',
    'quadronno_crocetta':'milano_centro',
    'turati': 'milano_centro',

    # milano_nord_est
    'buenos_aires': 'milano_nord_est',
    'bignami_ponale': 'milano_nord_est',
    'cascina_dei_pomi': 'milano_nord_est',
    'casoretto': 'milano_nord_est',
    'cimiano': 'milano_nord_est',
    'centrale': 'milano_nord_est',
    'citt _studi': 'milano_nord_est',
    'crescenzago':'milano_nord_est',
    'dezza': 'milano_nord_est',
    'greco_segnano': 'milano_nord_est',
    'gorla': 'milano_nord_est',
    'istria': 'milano_nord_est',
    'indipendenza':'milano_nord_est',
    'lambrate': 'milano_nord_est',
    'maggiolina': 'milano_nord_est',
    'melchiorre_gioia': 'milano_nord_est',
    'morgagni': 'milano_nord_est',
    'ortica': 'milano_nord_est',
    'repubblica': 'milano_nord_est',
    'parco_trotter': 'milano_nord_est',
    'pasteur': 'milano_nord_est',
    'plebisciti_susa': 'milano_nord_est',
    'piave_tricolore': 'milano_nord_est',
    'porta_venezia':'milano_nord_est',
    'precotto': 'milano_nord_est',
    'ponte_nuovo':'milano_nord_est',
    'quartiere_adriano': 'milano_nord_est',
    'quartiere_feltre': 'milano_nord_est',
    'rovereto':  'milano_nord_est',
    'rubattino': 'milano_nord_est',
    'turro': 'milano_nord_est',
    'villa_san_giovanni': 'milano_nord_est',
    'zara': 'milano_nord_est',

    # milano_nord_ovest
    'affori': 'milano_nord_ovest',
```

```
    'bicocca': 'milano_nord_ovest',
    'bovisa': 'milano_nord_ovest',
    'bruzzano': 'milano_nord_ovest',
    'ca_granda': 'milano_nord_ovest',
    'cascina_merlata_musocco': 'milano_nord_ovest',
    'cenisio': 'milano_nord_ovest',
    'certosa': 'milano_nord_ovest',
    'city_life': 'milano_nord_ovest',
    'comasina': 'milano_nord_ovest',
    'dergano': 'milano_nord_ovest',
    'farini': 'milano_nord_ovest',
    'ghisolfa_mac_mahon': 'milano_nord_ovest',
    'gallaratese': 'milano_nord_ovest',
    'garibaldi_corso_como': 'milano_nord_ovest',
    'isola': 'milano_nord_ovest',
    'monte_rosa_lotto': 'milano_nord_ovest',
    'niguarda': 'milano_nord_ovest',
    'porta_nuova': 'milano_nord_ovest',
    'prato_centenaro': 'milano_nord_ovest',
    'qt8': 'milano_nord_ovest',
    'quarto_oggiaro': 'milano_nord_ovest',
    'trenno': 'milano_nord_ovest',
    'san_carlo':'milano_nord_ovest',
    'udine':'milano_nord_ovest',

    # milano_sud_est
    'argonne_corsica': 'milano_sud_est',
    'bologna_sulmona': 'milano_sud_est',
    'bocconi': 'milano_sud_ovest',
    'cadore': 'milano_sud_est',
    'cantalupa_san_paolo': 'milano_sud_est',
    'chiesa_rossa': 'milano_sud_est',
    'corvetto': 'milano_sud_est',
    'famagosta': 'milano_sud_est',
    'gratosoglio': 'milano_sud_est',
    'martini_insubria': 'milano_sud_est',
    'molise_cuoco': 'milano_sud_est',
    'montenero': 'milano_sud_est',
    'lodi_brenta': 'milano_sud_est',
    'pezzotti_meda': 'milano_sud_est',
    'ponte_lambro': 'milano_sud_est',
    'porta_romana_medaglie_doro': 'milano_sud_est',
    'porta_vittoria': 'milano_sud_est',
    'portello_parco_vittoria': 'milano_nord_ovest',
    'quartiere_forlanini': 'milano_sud_est',
    'ripamonti': 'milano_sud_est',
    'rogoredo': 'milano_sud_est',
    'santa_giulia': 'milano_sud_est',
    'vigentino_fatima': 'milano_sud_est',
    'viale_ungheria_mecenate': 'milano_sud_est',

    # milano_sud_ovest
    'amendola_buonarroti': 'milano_sud_ovest',
    'ascanio_sforza': 'milano_sud_ovest',
    'baggio': 'milano_sud_ovest',
    'bagno': 'milano_sud_ovest',
    'barona': 'milano_sud_ovest',
    'bande_nere': 'milano_sud_ovest',
    'bisceglie': 'milano_sud_ovest',
    'cermenate_abbiategrasso': 'milano_sud_ovest',
    'corso_genova': 'milano_sud_ovest',
    'corso_san_gottardo': 'milano_sud_ovest',
    'de_angeli': 'milano_sud_ovest',
    'frua': 'milano_sud_ovest',
    'giambellino': 'milano_sud_ovest',
```

```
    'gambara': 'milano_sud_ovest',
    'inganni': 'milano_sud_ovest',
    'lorenteggio': 'milano_sud_ovest',
    'pagano:': 'milano_sud_ovest',
    'piazza_napoli': 'milano_sud_ovest',
    'piazzale_siena': 'milano_sud_ovest',
    'primaticcio': 'milano_sud_ovest',
    'quarto_cagnino': 'milano_sud_ovest',
    'san_siro': 'milano_sud_ovest',
    'santambrogio':'milano_sud_ovest',
    'solari': 'milano_sud_ovest',
    'tre_castelli_faenza': 'milano_sud_ovest',
    'ticinese': 'milano_sud_ovest',
    'tripoli_soderini': 'milano_sud_ovest',
    'vercelli_wagner': 'milano_sud_ovest',
    'washington': 'milano_sud_ovest'
}
```

D Columns to Keep

```
columns_to_keep=['y', 'square_meters', 'floor', 'elevator', '
    high_floor_no_elevator',
   'conditions',
   'total_rooms', 'bedrooms', 'other_rooms','bathrooms', 'kitchen_type', '
       accessible_bathroom',
   'top_furnishing', 'top_security', 'top_tv_system', 'top_fiber', 'top_concierge'
       ,
   'top_garden', 'top_balcony', 'top_kitchen', 'luxury', 'furnishing', 'security',
   'tv_internet', 'outdoor_space', 'concierge', 'storage', 'windows', 'exposure',
   'energy_efficiency_class_1', 'energy_efficiency_class_2', '
       energy_efficiency_class_3',
   'energy_efficiency_class_4', 'energy_efficiency_class_5', '
       energy_efficiency_class_6',
   'condominium_fees', 'zone', 'contract_type_rent | 3+2', 'contract_type_rent |
       4+4',
   'contract_type_rent | 6+6', 'contract_type_rent | open',
   'contract_type_rent | students (6 - 36 months)', 'contract_type_rent |
       transitory',
   'macro_zone_milano_nord_est', 'macro_zone_milano_nord_ovest',
   'macro_zone_milano_sud_est', 'macro_zone_milano_sud_ovest']
```

E Insights graphs

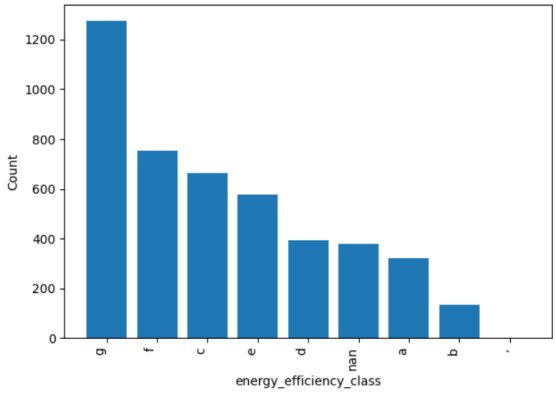

Figure 5: Energy efficiency class

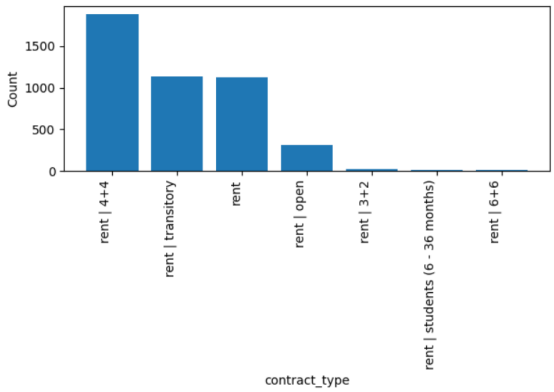

Figure 6: Contracts type distribution

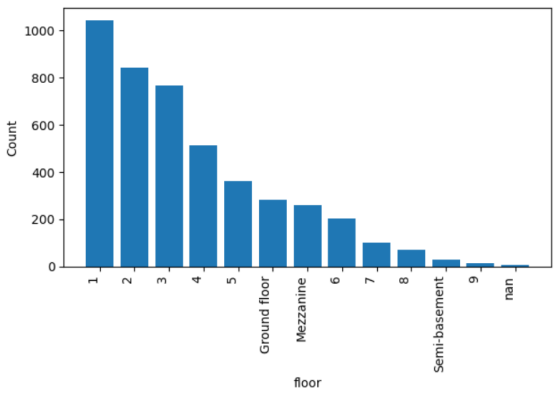

Figure 7: Floors distribution