Milan Rent Price Prediction Report

Introduction

This task focuses on predicting rent prices using predominantly textual features, evaluated via MAE. MAE, compared to MSE, penalizes large errors linearly rather than quadratically, emphasizing robustness to outliers and requiring precise median-level predictions rather than mean-focused fits. Consequently, minimizing MAE demands models that maintain consistent accuracy across the entire distribution.

Given the expected high dimensionality induced by extensive text variables—where naive dummy encoding risks severe sparsity and overfitting—shrinkage methods such as LASSO and dimensionality reduction techniques like PCA are worthy trying to regularize and compress feature space. Moreover, model stacking is employed to combine diverse base learners, enhancing predictive stability and bias-variance trade-off. Moreover, tree-based ensemble models are leveraged to exploit their inherent capacity to handle sparse, high-dimensional data and capture complex nonlinearities, effectively turning the curse of dimensionality into a predictive advantage.

Data Preprocessing and Feature Engineering

The raw dataset contains substantial missing values and high-cardinality textual features, requiring careful preprocessing to ensure robust model performance. Moreover, I experiment some feature engineering to capture nonlinear relationships.

Data Preprocessing

Missing values are concentrated in availability, other_features, conditions, floor, energy_efficiency_class, and condominium_fee. For textual variables with missing entries, I imputed the value "unknown" instead of the mode. This choice reflects the insight that missingness may be informative, rather than random—absence of information, may signal a disadvantage. For the numerical variable condominium_fee, missing values were imputed with the median within each geographic zone. This group-

wise median imputation preserves local distributional characteristics and mitigates bias caused by heterogeneous regional fees.

The availability feature exhibits a large number of categorical values, raising concerns about the curse of dimensionality if handled naively. To address this, I analyzed the temporal distribution of availability dates and observed a high concentration around 2024. Consequently, I discretized availability dates into buckets. This approach reduces feature sparsity and dimensionality while preserving temporal patterns relevant for rent prediction. But this approach is not applied to other textual features since they lack the continuous nature of dates.

Target Distribution and Transformation

Consistent with real estate economic theory, rent prices exhibit a right-skewed distribution, which can adversely affect model training and predictive accuracy. The computed skewness (2.45) and kurtosis (7.43) confirm strong departure from normality. Applying a logarithmic transformation (log1p) effectively normalizes the distribution, as evidenced by substantially improved skewness and kurtosis metrics. This transformation stabilizes variance and enhances model robustness.

At last, I got 242 cleaned features.

Methodology

Baseline

Establishing a baseline is crucial to benchmark model improvements and to understand the predictive capacity of straightforward approaches. I selected Lasso as my baseline due to its effectiveness in high-dimensional settings. Lasso performs automatic feature subset selection via L1 regularization, thereby reducing model variance and mitigating overfitting risks inherent in large feature spaces.

I didn’t choose Ridge because, unlike Ridge which shrinks all coefficients uniformly, Lasso better handles sparsity by driving irrelevant features’ coefficients to

zero—an important advantage given the abundance of potentially noisy categorical text features. However, Lasso's linearity imposes a strong assumption, which may not capture the true nonlinear relationships present in rent pricing, likely resulting in relatively higher MAE compared to more flexible models.

Gradient Boosting

To overcome the linear limitations of Lasso and better capture complex nonlinear interactions in high-dimensional data, I employed Gradient Boosting with cross-validation, which iteratively build an ensemble of decision trees, each correcting errors from previous ones, enabling flexible modeling of intricate patterns.

This method naturally handles mixed data types and is robust to irrelevant features, making it well-suited for datasets with numerous categorical and text-derived variables. Additionally, GBMs are less prone to overfitting through regularization parameters like tree depth and learning rate, allowing improved predictive accuracy compared to linear models in our rent price prediction task.

Stacking

To boost predictive accuracy, I employed stacking by combining three gradient boosting frameworks: XGBoost, LightGBM, and CatBoost. Each implements gradient boosting with unique algorithmic optimizations and regularization techniques, allowing the ensemble to capture diverse aspects of the data.

XGBoost utilizes a second-order Taylor approximation for the loss function and incorporates advanced regularization (L1 and L2), providing robustness against overfitting and effective handling of sparse data.

LightGBM introduces histogram-based decision tree learning and leaf-wise tree growth, offering faster training and better accuracy on large datasets by focusing on splits that maximize loss reduction.

CatBoost is designed to handle categorical features natively through ordered

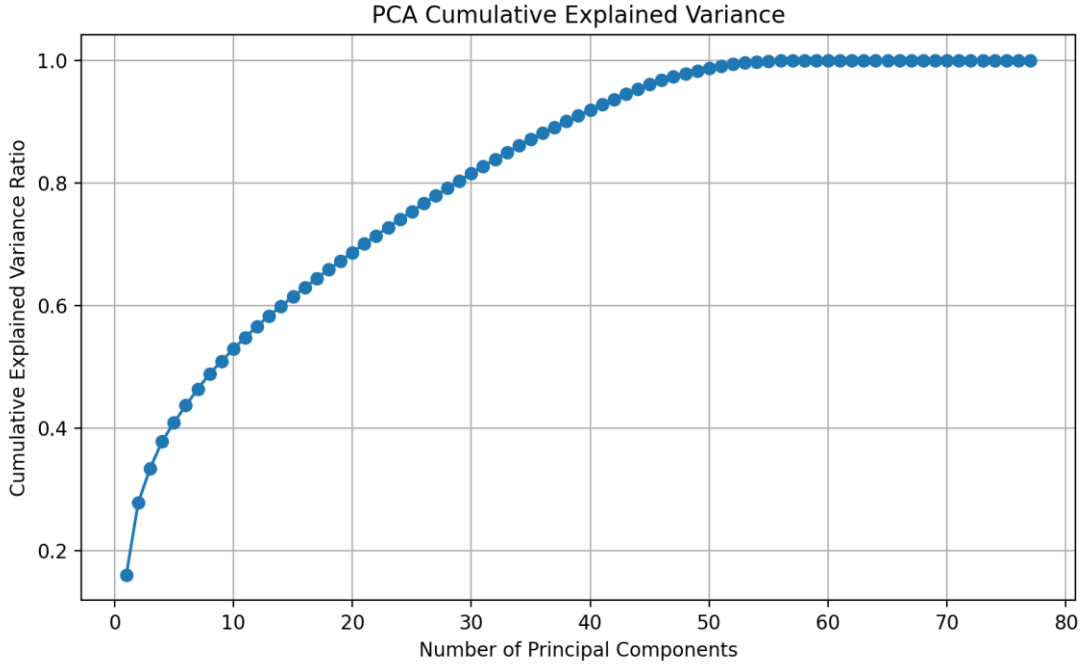
boosting and uses symmetric trees to reduce prediction shift, which enhances performance especially in datasets with many categorical/text features.

Stacking these models on ridge leverages their complementary strengths: XGBoost's regularization, LightGBM's speed and accuracy in large data, and CatBoost's effective categorical handling. The meta-model integrates these diverse predictions, resulting in improved generalization and reduced bias-variance tradeoff, which is critical in the high-dimensional and text-heavy context of rent price prediction.

Each model's hyperparameters were tuned and evaluated using 5-fold cross-validation with MAE as the performance metric, allowing assessment of both predictive accuracy and the variance of cross-validation scores. The reason why I chose 5-fold rather than LOOCV is that I prefer lower variance in high-dimension circumstances. This rigorous validation ensures stability and robustness before stacking.

Shrinkage Trial

Given the high-dimensional feature space and concerns about overfitting, I explored shrinkage techniques beyond regularization with ridge stacking and tree-based methods. Specifically, I experimented with Principal Component Analysis (PCA) for dimensionality reduction.



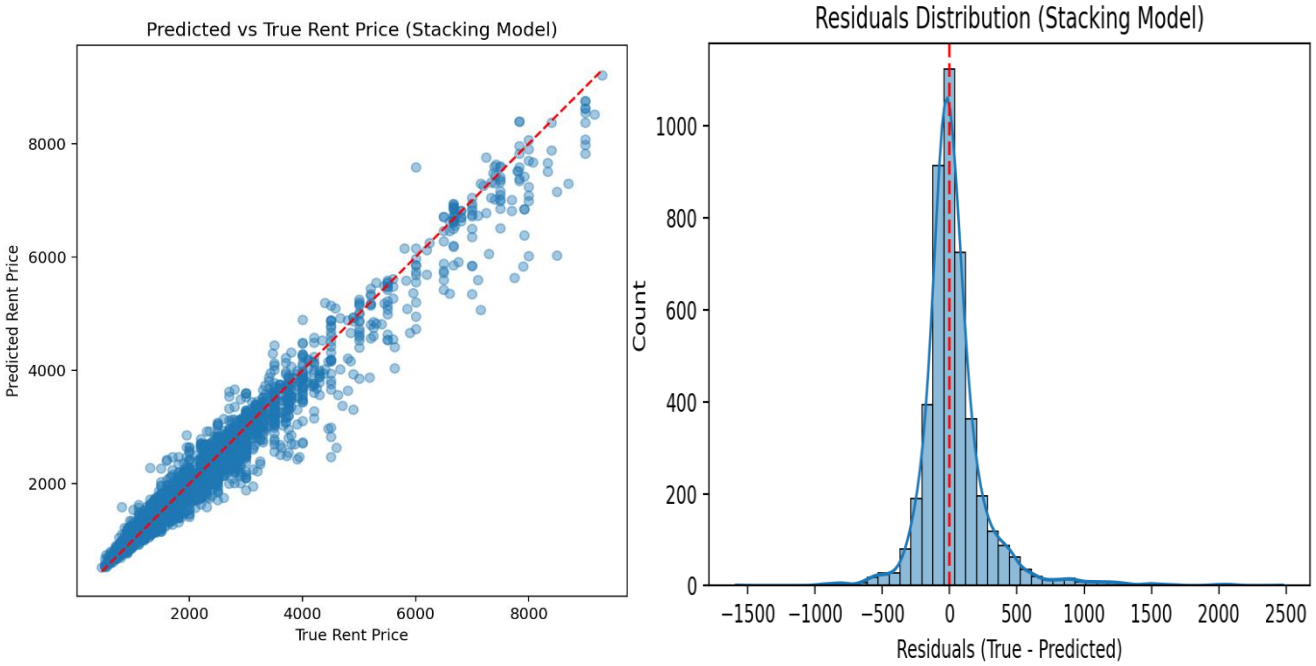
However, PCA's assumption of continuous, normally distributed variables limits its effectiveness with the predominantly categorical and sparse textual features in our dataset. Visualization of the explained variance ratio showed a rapid increase in the first 10 components, a roughly linear increase between components 10 and 50, and a plateau from 50 to 77 components, indicating that a large number of components are needed to capture sufficient variance. This gradual increase diminishes the benefit of dimensionality reduction.

Moreover, with 242 features in total, the dimensionality is moderate rather than extreme. Tree-based models such as gradient boosting inherently mitigate the curse of dimensionality through selective splitting and feature importance weighting. Consistent with this, models incorporating PCA showed no improvement or even slight degradation in prediction accuracy.

Therefore, PCA was not incorporated in the final modeling.

Results

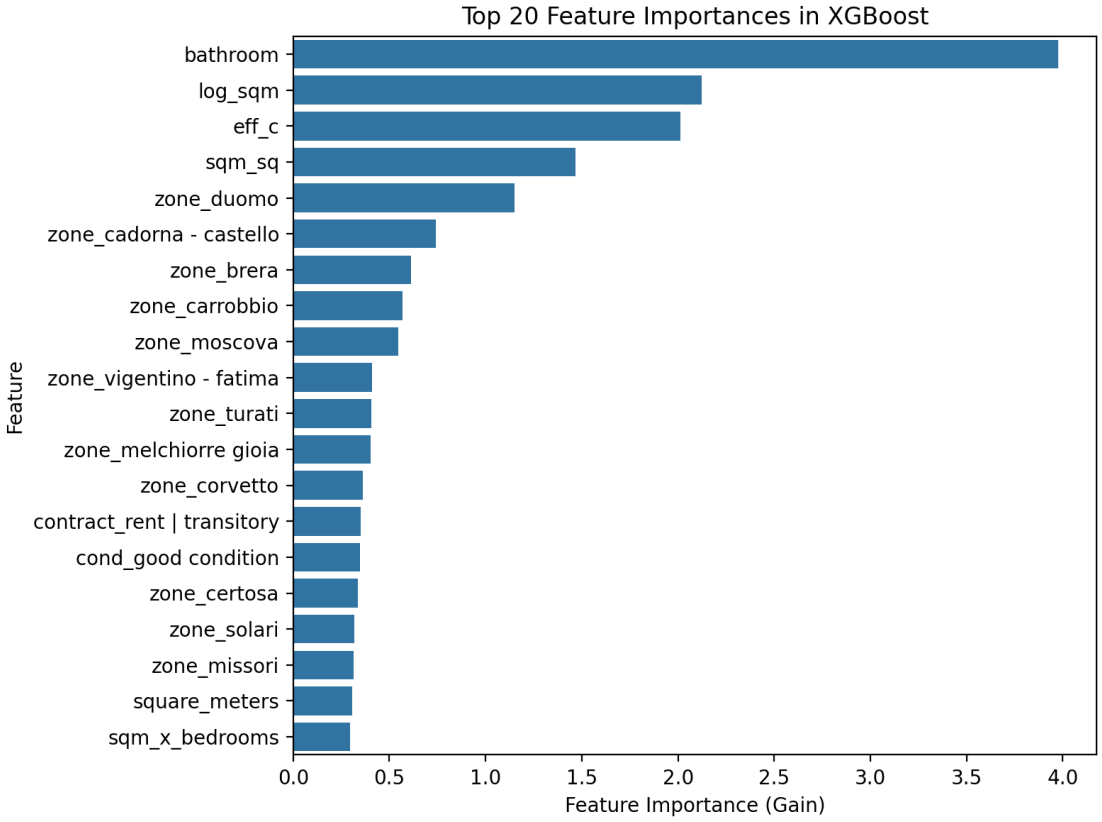
The Stacking model achieves a 5-fold cross-validation MAE of **330.75** (standard deviation: **13.46**), demonstrating strong and stable predictive performance. The predicted versus actual rent plot shows close alignment across the primary value range



(EUR 2,000–8,000), with minor deviations at the upper tail, suggesting limited underfitting or overfitting at extremes.

The residual distribution is approximately symmetric and centered near zero, indicating no substantial systematic bias. This supports the model's ability to generalize across diverse property types and price ranges.

XGBoost feature importance highlights the number of bathrooms (bathroom), log-transformed square footage (log_sqm), and energy efficiency (eff_c) as the most influential predictors. These findings emphasize the explanatory power of both structural characteristics and energy-related attributes in capturing rent variation. The model's ability to incorporate non-linear interactions and hierarchical patterns through tree-based learning proves effective, as evidenced by both low prediction errors and well-behaved residuals.



While overall performance is robust, slight miscalibration at extreme values suggests that further refinement—such as quantile-aware loss functions or post-model calibration—may improve accuracy for high-end properties.

Interpretations

Feature importance analysis from the XGBoost model reveals key structural and spatial determinants of rental pricing:

1. Bathrooms Trump Size

Bathroom count is the top predictor, outperforming even square footage. This suggests renters may prioritize convenience and modern amenities over raw space—perhaps reflecting preferences for renovated units or better utility layouts.

2. Size Still Matters, but Smartly

Size-related features like log_sqm and sqm_sq remain highly predictive, with log_sqm offering better fit due to diminishing returns. In contrast, sqm_x_bedrooms ranks low, implying that bedroom count adds little explanatory power unless paired with spatial context.

3. Location Signals Urban Premiums

Central areas (Duomo, Brera, Cadorna) rank higher than peripheral ones, reaffirming urban premium pricing. Rising importance of Moscova and Turati may indicate ongoing gentrification or commercial spillover effects.

4. Contract Type and Condition Matter

Short-term leases (contract_rent | transitory) and well-maintained units (cond_good) positively influence rent, highlighting market demand for flexibility and livability.

Conclusion:

The model's ranking suggests a shift in urban rental economics: functionality (bathrooms) and location now outweigh traditional metrics like size. Combined with strong predictive performance (MAE ~330), these insights provide both empirical robustness and actionable direction.

Appendix Code

```
import pandas as pd

import numpy as np

import re

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import TfidfVectorizer

from scipy.stats import skew, kurtosis

from sklearn.preprocessing import StandardScaler, PowerTransformer, OneHotEncoder

from sklearn.impute import SimpleImputer

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.feature_selection import VarianceThreshold

from sklearn.model_selection import KFold

from xgboost import XGBRegressor

from sklearn.model_selection import KFold, GridSearchCV

from sklearn.metrics import mean_absolute_error

from sklearn.preprocessing import StandardScaler

from xgboost import XGBRegressor

import lightgbm as lgb
```

```
from catboost import CatBoostRegressor

from sklearn.model_selection import GridSearchCV, KFold

from sklearn.linear_model import Ridge

from sklearn.ensemble import StackingRegressor

from sklearn.linear_model import LassoCV, Lasso

from scipy.stats import zscore


train = pd.read_csv("train.csv")

print(train.isnull().sum())

test = pd.read_csv("test.csv")


y = train['y']

print(f"Skewness: {skew(y):.4f}")

print(f"Kurtosis: {kurtosis(y):.4f}")

y_log = np.log1p(train['y'])

print(f"Skewness: {skew(y_log):.4f}")

print(f"Kurtosis: {kurtosis(y_log):.4f}")


df = train.drop(columns=['y','w'])


## check missing values

print(train.isnull().sum())
```

```
## check availability high-frequency
df["availability"] = df["availability"].fillna("unknown").str.lower().str.strip()
df["is_available_now"] = df["availability"].apply(lambda x: 1 if x == "available" else 0)
df["available_date_raw"] = df["availability"].str.extract(r'available from ([\d]{1,2}/[\d]{1,2}/[\d]{4})')
df["available_date"] = pd.to_datetime(df["available_date_raw"], format="%d/%m/%Y", errors="coerce")

plt.figure(figsize=(10, 4))
sns.histplot(df["available_date"].dropna(), bins=50, kde=False)
plt.title("Distribution of Available From Dates")
plt.xlabel("Available Date")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

## check desprition
description = df["description"].fillna("").astype(str).str.lower()
def tokenize(text):
    text = re.sub(r"[^\w\s]", " ", text)
    return text.split()
```

```
token_set = set()
for desc in description:
    tokens = tokenize(desc)
    token_set.update(tokens)
token_list = sorted(token_set)
print(token_list)
print(f"共发现 {len(token_list)} 个唯一词项。")

## checj other_features
other_features = df["other_features"].fillna("").astype(str).str.lower()
def tokenize(text):
    text = re.sub(r"[^\w\s]", " ", text)
    return text.split()
token_set = set()
for desc in other_features:
    tokens = tokenize(desc)
    token_set.update(tokens)
token_list = sorted(token_set)
print(token_list)
print(f"共发现 {len(token_list)} 个唯一词项。")

def preprocess(train, test):
```

```
train['dataset'] = 'train'

test['dataset'] = 'test'

combined = pd.concat([train, test], axis=0)

combined = pd.concat([train, test], keys=["train", "test"])


# 统一处理字符串列，保证是 Series，填充、转小写、strip

for col in ["contract_type", "availability", "conditions", "floor", "elevator", "energy_efficiency_class"]:

                                                combined[col] = combined[col].astype(str).fillna("unknown").str.lower().str.strip()


# ========== 1. Contract Type ==========

    contract_dummies = pd.get_dummies(combined["contract_type"], prefix="contract", drop_first=True)

    combined = pd.concat([combined.drop(columns=["contract_type"]), contract_dummies], axis=1)


# ========== 2. Availability ==========

combined["is_available_now"] = combined["availability"].apply(lambda x: 1 if x == "available" else 0)

                                    combined["available_date_raw"] = combined["availability"].str.extract(r'available from ([\d]{1,2}/[\d]{1,2}/[\d]{4})')

                                        combined["available_date"] = pd.to_datetime(combined["available_date_raw"], format="%d/%m/%Y",
```

errors="coerce")

def bucket_availability(*date*): # 画图后发现高频特征集中在 2024 年

if pd.isnull(date):

return "unknown"

elif date <= pd.Timestamp("2024-03-01"):

return "early"

elif date <= pd.Timestamp("2024-06-01"):

return "spring_2024"

elif date <= pd.Timestamp("2024-09-01"):

return "summer_2024"

elif date <= pd.Timestamp("2024-12-31"):

return "fall_2024"

else:

return "future"

combined["availability_bucket"] = combined["available_date"].apply(bucket_availability)

availability_dummies = pd.get_dummies(combined["availability_bucket"], *prefix*="avail", *drop_first*=True)

combined = pd.concat([combined.drop(*columns*=["availability", "available_date_raw", "available_date", "availability_bucket"]), availability_dummies], *axis*=1)

```
# ========== 3. Description: Extract Structured Info ==========
kitchen_words = {'kitchen', 'kitchenette', 'diner', 'nook'}
feature_words = {'habitable', 'open', 'semi', 'disabled'}
stopwords = {'for', 'or', 'people', 'other', 'others', 'more', 'suitable', 'room', 'rooms'}

# 所有可能的特征词汇表（要 dummy 的）
all_features = sorted(kitchen_words.union(feature_words))
parsed_data = []

for desc in combined['description'].astype(str).str.lower():
    tokens = re.sub(r"[^\w\s]", " ", desc).split()

    total_room = 0
    bedroom = 0
    bathroom = 0
    dummies = {f'is_{w}': 0 for w in all_features}

    i = 0
    while i < len(tokens):
        token = tokens[i]
```

```
# 首词 + 括号，作为总房间数
if i == 0 and token.isdigit() and '(' in desc:
        total_room = int(token)
        i += 1
        continue


# 数字 + 类型
if token.isdigit() and i + 1 < len(tokens):
        next_token = tokens[i + 1]
        if next_token in {'bedroom', 'bedrooms'}:
                bedroom += int(token)
                i += 2
                continue
       elif next_token in {'bathroom', 'bathrooms'}:
                bathroom += int(token)
                i += 2
                continue
        else:
                i += 1
                continue
# 单词特征（忽略 stopwords）
elif token not in stopwords:
```

```
                if token in all_features:
                    dummies[f'is_{token}'] = 1
            i += 1

        # fallback：若 total_room 仍为 0，使用 bedroom + bathroom 合计
        if total_room == 0:
            total_room = bedroom + bathroom

        parsed_data.append({
            'total_room': total_room,
            'bedroom': bedroom,
            'bathroom': bathroom,
            **dummies
        })

    result_df = pd.DataFrame(parsed_data, index=combined.index)
    combined = pd.concat([combined.drop(columns = ['description']), result_df], axis=1)
    print("Columns after description parse:", combined.columns)

    # other_features
    # 处理 other_features 列
                                        other_feats =
```

```
combined['other_features'].fillna('').str.lower().str.replace(r'[^\w\s]', ' ', regex=True).str.split()


        structure_features = {
            'attic', 'balcony', 'balconies', 'cellar', 'closet', 'court', 'fireplace', 'frames',
            'garden', 'pool', 'tavern', 'tennis', 'terrace', 'window'
        }
        material_features = {'metal', 'wood', 'glass', 'pvc', 'pvcdouble', 'pvcexposure'}
        security_features = {'alarm', 'concierge', 'entryphone', 'reception', 'reception1', 'security', 'gate', 'video', 'door'}
        media_features = {'fiber', 'optic', 'tv', 'satellite', 'dish'}
        orientation_features = {'east', 'west', 'south', 'north', 'exposure', 'internal', 'external'}
        furniture_features = {'furnished', 'partially', 'only', 'single', 'double', 'triple', 'full', 'half'}
        bath_features = {'hydromassage'}
        system_features = {'centralized', 'electric', 'system'}
        stopwords = {'in', 'and', 'day', 'with'}

        # 全部合法特征
        all_features = (
            structure_features | material_features | security_features | media_features |
            orientation_features | furniture_features | bath_features | system_features
```

```
)

# 初始化空 DataFrame 用于哑变量
for feat in sorted(all_features):
    combined[f'feat_{feat}'] = 0

# 遍历行
for idx, tokens in other_feats.items():
    for token in tokens:
        if token in all_features:
            combined.at[idx, f'feat_{token}'] = 1

# 删除原始列
combined.drop(columns=['other_features'], inplace=True)

# ========== 6. Floor, Elevator, Condition ==========
    condition_dummies = pd.get_dummies(combined["conditions"], prefix="cond", drop_first=True)
    combined = pd.concat([combined.drop(columns=["conditions"]), condition_dummies], axis=1)

combined["floor"] = combined["floor"].replace({
    "ground floor": "ground",
```

```
        "semi-basement": "basement",
        "mezzanine": "mezzanine"
    }).fillna("unknown")
    floor_dummies = pd.get_dummies(combined["floor"], prefix="floor", drop_first=True)
    combined = pd.concat([combined.drop(columns=["floor"]), floor_dummies], axis=1)

    elevator_dummies = pd.get_dummies(combined["elevator"], prefix="elevator", drop_first=True)
    combined = pd.concat([combined.drop(columns=["elevator"]), elevator_dummies], axis=1)

    # ========== 7. Energy Efficiency ==========
    combined["energy_efficiency_class"] = combined["energy_efficiency_class"].replace({",": np.nan, "nan": np.nan}).fillna("unknown")
    eff_dummies = pd.get_dummies(combined["energy_efficiency_class"], prefix="eff", drop_first=True)
    combined = pd.concat([combined.drop(columns=["energy_efficiency_class"]), eff_dummies], axis=1)

    ## condominium fees
    combined['condominium_fees'] =
```

```
combined.groupby('zone')['condominium_fees'].transform(
        lambda x: x.fillna(x.median())
    )
                                    combined['condominium_fees']                =
combined['condominium_fees'].fillna(combined['condominium_fees'].median())


    ## zone    length = 132
    '''
    min_samples = int(0.01 * len(combined))

    zone_counts = combined['zone'].value_counts()
    combined['zone_group'] = combined['zone'].where(
                            combined['zone'].isin(zone_counts[zone_counts       >=
min_samples].index),
    '       other'
    )

    n_categories = combined['zone_group'].nunique()
    print(f" {n_categories}") # 40
   '''
    zone_dummies = pd.get_dummies(
        combined['zone'],
        prefix='zone',
```

```
        drop_first=True,
    )

    combined = pd.concat([combined, zone_dummies], axis=1)
    combined = combined.drop(['zone'], axis=1)

    # ========== 9. Feature Engineering ==========
    combined["log_sqm"] = np.log1p(combined["square_meters"])
    combined["sqm_sq"] = combined["square_meters"] ** 2
    combined["sqrt_sqm"] = np.sqrt(combined["square_meters"])
    combined["log_condo"] = np.log1p(combined["condominium_fees"])
    combined["log_rooms"] = np.log1p(combined["total_room"])
    combined["sqm_x_rooms"] = combined["square_meters"] * combined["total_room"]
    combined["sqm_x_fees"] = combined["square_meters"] * combined["condominium_fees"]
    combined["room_density"] = combined["bedroom"] / (combined["total_room"] + 1e-3)

    combined["rooms_x_fees"] = combined["total_room"] * combined["condominium_fees"]
    combined["sqm_x_bedrooms"] = combined["square_meters"] * combined["bedroom"]
```

```
    '''
    # ========== 10. Outlier and Leverage Filtering ==========
    numeric_cols = combined.select_dtypes(include=[np.number]).columns.tolist()
    z_scores = combined[numeric_cols].apply(zscore)
    combined = combined[(np.abs(z_scores) < 5).all(axis=1)]    # remove outliers

    # ========== 11. Scale/transform ==========
    numeric_cols = combined.select_dtypes(include=[np.number]).columns.tolist()
    exclude_cols = [col for col in numeric_cols if combined[col].nunique() <= 2]
    scale_cols = [col for col in numeric_cols if col not in exclude_cols]

    for col in scale_cols:
        combined[col] = np.log1p(combined[col].clip(lower=0))
    '''

    # ========== 12. Final Split ==========
    df = combined[combined['dataset'] == 'train'].copy()
    test_processed = combined[combined['dataset'] == 'test'].copy()
    combined.drop(columns=['dataset'], inplace=True)

    return df, test_processed
```

```
df, test = preprocess(df,test)

print(df.columns.tolist())

df = df.drop(columns=['dataset'])

test = test.drop(columns=['dataset'])


# baseline: lasso

X = df

lasso_cv = LassoCV(cv=5, random_state=42).fit(X, y_log)

best_alpha = lasso_cv.alpha_

print("Best alpha from LassoCV:", best_alpha)

cv = KFold(n_splits=5, shuffle=True, random_state=42)


mae_scores = []

y_true = np.expm1(y_log)


for train_idx, val_idx in cv.split(X):

    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]

    y_train_log, y_val_log = y_log.iloc[train_idx], y_log.iloc[val_idx]


    model = Lasso(alpha=lasso_cv.alpha_, max_iter=10000)

    model.fit(X_train, y_train_log)
```

```
        y_pred_log = model.predict(X_val)
        y_pred_real = np.expm1(y_pred_log)
        y_val_real = np.expm1(y_val_log)

        mae = mean_absolute_error(y_val_real, y_pred_real)
        mae_scores.append(mae)

print(f"Mean MAE (original scale): {np.mean(mae_scores):.4f}")
print(f"Std of MAE: {np.std(mae_scores):.4f}")

# gradien boosting
X = df
y_true = np.expm1(y_log)
# print(X)
xgb_params = {
        'n_estimators': [1000, 2000, 3000],
        'max_depth': [3, 5, 7, 9, 11],
        'learning_rate': [0.001, 0.01, 0.05],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0]
}
```

```
xgb_cv = GridSearchCV(
    estimator=XGBRegressor(random_state=42, objective='reg:squarederror'),
    param_grid=xgb_params,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)

xgb_cv.fit(X, y_log)
print("Best XGBoost params:", xgb_cv.best_params_)

cv = KFold(n_splits=5, shuffle=True, random_state=42)
mae_scores = []

for train_idx, val_idx in cv.split(X):
    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_train_log, y_val_log = y_log.iloc[train_idx], y_log.iloc[val_idx]

    model = XGBRegressor(
        **xgb_cv.best_params_,
```

```
        random_state=42,
        objective='reg:squarederror'
    )

    model.fit(X_train, y_train_log)

    y_pred_log = model.predict(X_val)
    y_pred_real = np.expm1(y_pred_log)
    y_val_real = np.expm1(y_val_log)

    mae = mean_absolute_error(y_val_real, y_pred_real)
    mae_scores.append(mae)

print("\nXGBoost Performance:")
print(f"Mean MAE (original scale): {np.mean(mae_scores):.4f}")
print(f"Std of MAE: {np.std(mae_scores):.4f}")

# stack
## lightBGM
lgb_params = {
    'n_estimators': [300, 500, 1000],
    'max_depth': [3, 5, 7],
```

```
    'learning_rate': [0.005, 0.01, 0.05],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

lgb_cv = GridSearchCV(
    estimator=lgb.LGBMRegressor(random_state=42),
    param_grid=lgb_params,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)
lgb_cv.fit(X, y_log)
print("Best LightGBM params:", lgb_cv.best_params_)
## catboost
cat_params = {
    'iterations': [300, 500, 1000],
    'depth': [3, 5, 7],
    'learning_rate': [0.005, 0.01, 0.05]
}
```

```
cat_cv = GridSearchCV(
    estimator=CatBoostRegressor(random_seed=42, verbose=0),
    param_grid=cat_params,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    verbose=1
)
cat_cv.fit(X, y_log)
print("Best CatBoost params:", cat_cv.best_params_)

# 4. 用最佳参数初始化模型
xgb_best = XGBRegressor(random_state=42, objective='reg:squarederror', **xgb_cv.best_params_)
lgb_best = lgb.LGBMRegressor(random_state=42, **lgb_cv.best_params_)
cat_best = CatBoostRegressor(random_seed=42, verbose=0, **cat_cv.best_params_)

# 5. Stacking 模型（最终学习器用岭回归）
stacking_model = StackingRegressor(
    estimators=[('xgb', xgb_best), ('lgb', lgb_best), ('cat', cat_best)],
    final_estimator=Ridge(),
    cv=5,
```

```
    n_jobs=-1,
    passthrough=False
)

# 6. KFold CV 计算 MAE（真实尺度）
cv = KFold(n_splits=5, shuffle=True, random_state=42)
mae_scores = []

for train_idx, val_idx in cv.split(X):
    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_train_log, y_val_log = y_log.iloc[train_idx], y_log.iloc[val_idx]

    stacking_model.fit(X_train, y_train_log)
    y_pred_log = stacking_model.predict(X_val)

    y_pred_real = np.expm1(y_pred_log)
    y_val_real = np.expm1(y_val_log)

    mae = mean_absolute_error(y_val_real, y_pred_real)
    mae_scores.append(mae)

print(f"Stacking Model Mean MAE (original scale): {np.mean(mae_scores):.4f}")
```

```
print(f"Stacking Model Std MAE: {np.std(mae_scores):.4f}")


# generate prediction
'''
train_features = df.drop(columns=['w', 'y', 'log_y']).columns
df_var = selector.fit_transform(df[train_features])
cols_var = train_features[selector.get_support()]
test_var = selector.transform(test[train_features])
test_var = pd.DataFrame(test_var, columns=cols_var, index=test.index)
test_uncorr = test_var.drop(columns=[col for col in dropped_cols if col in test_var.columns])
test_scaled = scaler.transform(test_uncorr)
test_pca = pca_final.transform(test_scaled)
test_pca = pd.DataFrame(test_pca, columns=[f'PC{i+1}' for i in range(n_components)], index=test.index)
'''


test_pred_log = stacking_model.predict(test)
test_pred = np.expm1(test_pred_log)


with open("output.txt", "w") as f:
    for value in test_pred:
        f.write(f"{value:.6f}\n")
```

```
## PCA picture
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# 假设 combined 是包含所有数值特征的 DataFrame，已经做了适当的数值编码和缺失值处理

# 选取数值型特征（如果包含分类需先编码）
numeric_features = X.select_dtypes(include=[np.number]).fillna(0)

# 标准化（PCA 通常建议先标准化）
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(numeric_features)
print(numeric_features)
# 运行 PCA，n_components 设置为全部
pca = PCA(n_components=X_scaled.shape[1])
pca.fit(X_scaled)

# 计算累计解释方差比例
cum_var_exp = np.cumsum(pca.explained_variance_ratio_)
```

```
plt.figure(figsize=(8,5))
plt.plot(range(1, len(cum_var_exp)+1), cum_var_exp, marker='o', linestyle='-')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('PCA Cumulative Explained Variance')
plt.grid(True)
plt.tight_layout()
plt.show()

# 结果图
stacking_model.fit(X, y_log)
y_pred_log = stacking_model.predict(X)
y_pred_real = np.expm1(y_pred_log)
y_real = np.expm1(y_log)

# 1. 预测值 vs 真实值散点图
plt.figure(figsize=(6,6))
plt.scatter(y_real, y_pred_real, alpha=0.4)
plt.plot([y_real.min(), y_real.max()], [y_real.min(), y_real.max()], 'r--')
plt.xlabel("True Rent Price")
plt.ylabel("Predicted Rent Price")
```

```
plt.title("Predicted vs True Rent Price (Stacking Model)")
plt.tight_layout()
plt.show()

# 2. 残差分布图
residuals = y_real - y_pred_real
plt.figure(figsize=(6,4))
sns.histplot(residuals, bins=50, kde=True)
plt.axvline(0, color='red', linestyle='--')
plt.xlabel("Residuals (True - Predicted)")
plt.title("Residuals Distribution (Stacking Model)")
plt.tight_layout()
plt.show()

# 3. XGBoost 特征重要性条形图
xgb_model = stacking_model.named_estimators_['xgb']

# 注意：xgb_model 是 XGBRegressor，使用 get_booster 获取底层 Booster
importance = xgb_model.get_booster().get_score(importance_type='gain')
importance = dict(sorted(importance.items(), key=lambda item: item[1], reverse=True))

plt.figure(figsize=(8,6))
```

```
sns.barplot(x=list(importance.values())[:20], y=list(importance.keys())[:20])

plt.xlabel("Feature Importance (Gain)")

plt.ylabel("Feature")

plt.title("Top 20 Feature Importances in XGBoost")

plt.tight_layout()

plt.show()
```