# Evaluation Report

For the ImmoEliza KNN Regression model I also have to write an evaluation report covering among others the following:

1) a code snippet of the model instantiation (to see which model it is, what parameters,...)

2) MAE (on training/test), RMSE (on training/test), R2 (on training/test), MAPE (on training/test), sMAPE (on training/test)

3) The list of features you've used and how you got it (to quickly understand if you've done data leakage)

4) Accuracy computing procedure (on a test set? What split %, 80/20, 90/10, 50/50? k-fold cross?)

5) Efficiency (training and inference time). The fastest the best (sustainability).

6) a quick presentation of the final dataset (how many records, did you merge some datasets together? did you scrape again? what cleaning step you've done, scaling, encoding, cross-validation.. No need of visuals, just bullet points)

For 1): This is the code for the model instantiation:

main.py:

"""Assigning variables, splitting test and training set & standardizing features"""

y = df[TARGET].values # Convert to NumPy array for compatibility with k-fold

X = FeatureUtils.select\_features(df, features=FEATURES).values

# Perform k-fold cross-validation

results = ModelUtils.train\_and\_cross\_validate(

X, y, n\_splits=CV\_N\_SPLITS, n\_neighbors=KNN\_N\_NEIGHBORS

)

metrics = results["metrics"]

y\_true = results["y\_true"]

y\_pred = results["y\_pred"]

train\_time = results["train\_time"]

inference\_time = results["inference\_time"]

model\_utils.py:

@classmethod

def train\_and\_cross\_validate(

cls, X: np.ndarray, y: np.ndarray, n\_splits: int = 5, n\_neighbors: int = 5

) -> Dict[str, Any]:

"""

Perform k-fold cross-validation and compute metrics for training and test sets.

Args:

X (np.ndarray): Feature set.

y (np.ndarray): Target variable.

n\_splits (int): Number of folds for cross-validation. Defaults to 5.

n\_neighbors (int): Number of neighbors for KNN. Defaults to 5.

Returns:

Dict[str, Any]: Average metrics and collected predictions.

"""

kf = KFold(n\_splits=n\_splits, shuffle=True, random\_state=42)

metrics\_list = {"train": [], "test": []}

all\_y\_true = [] # Collect true values

all\_y\_pred = [] # Collect predictions

# Track training and inference time

total\_train\_time = 0

total\_inference\_time = 0

for train\_index, test\_index in kf.split(X):

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

# Standardize the data

X\_train\_scaled = cls.\_scaler.fit\_transform(X\_train)

X\_test\_scaled = cls.\_scaler.transform(X\_test)

# Train Model & Measure training time

start\_train\_time = time.time()

cls.\_model = KNeighborsRegressor(

n\_neighbors=n\_neighbors, weights="distance"

)

cls.\_model.fit(X\_train\_scaled, y\_train)

end\_train\_time = time.time()

train\_time = end\_train\_time - start\_train\_time

total\_train\_time += train\_time

# Predict on training and test sets & Measure inference time

start\_inference\_time = time.time()

y\_train\_pred = cls.\_model.predict(X\_train\_scaled)

y\_test\_pred = cls.\_model.predict(X\_test\_scaled)

end\_inference\_time = time.time()

inference\_time = end\_inference\_time - start\_inference\_time

total\_inference\_time += inference\_time

# Collect predictions for the test set

all\_y\_true.extend(y\_test)

all\_y\_pred.extend(y\_test\_pred)

# Evaluate and store metrics for training and test sets

train\_metrics = cls.evaluate(y\_train, y\_train\_pred)

test\_metrics = cls.evaluate(y\_test, y\_test\_pred)

metrics\_list["train"].append(train\_metrics)

metrics\_list["test"].append(test\_metrics)  
  
config.py:  
  
KNN\_N\_NEIGHBORS = 8

CV\_N\_SPLITS = 5

TARGET = "price"

FEATURES = [

    "living\_area",

    "com\_avg\_income",

    "building\_condition\_encoded",

    "subtype\_of\_property\_encoded",

    "latitude",

    "longitude",

    "equipped\_kitchen\_encoded",

    "min\_distance",

    "terrace\_encoded",

]

For 2):

Cross-Validation Metrics (Averaged):

Train Metrics:

Mean Absolute Error (MAE): 4137.7838

Root Mean Squared Error (RMSE): 14208.5938

Mean Squared Error (MSE): 201930810.7503

R Squared: 0.9904

Mean Absolute Percentage Error (MAPE): 1.1406

Symmetric Mean Absolute Percentage Error (sMAPE): 1.1170

Test Metrics:

Mean Absolute Error (MAE): 51453.9462

Root Mean Squared Error (RMSE): 79155.8218

Mean Squared Error (MSE): 6266987938.4740

R Squared: 0.7022

Mean Absolute Percentage Error (MAPE): 16.3763

Symmetric Mean Absolute Percentage Error (sMAPE): 14.7858

For 3):

These are the features I used are: [ "living\_area", "com\_avg\_income", "building\_condition\_encoded", "subtype\_of\_property\_encoded", "latitude",

"longitude", "equipped\_kitchen\_encoded", "min\_distance", "terrace\_encoded"]

Living, area, building condition, subtype of property, equipped kitchen and terrace were scraped from the Immoweb website.   
Building condition was label encoded ordered by quality from highest to lowest. Subtypes were reduced to 5 categories which were label encoded by quality from highest to lowest. Equipped kitchen was label encoded by quality from highest to lowest.   
Terrace was encoded binary (1 = terrace available, 0 = no terrace) since there were several instances were a terrace was indicated but no information on the size. 'com\_avg\_income' is the average taxable income per belgian commune. The source data was downloaded from the Belgian statistical office under: <https://statbel.fgov.be/en/themes/households/taxable-income#figures>The data was mapped into the real estate dataframe based on the reformed NIS codes of communes imported also from a dataset of the Belgian statistical office at: <https://statbel.fgov.be/nl/over-statbel/methodologie/classificaties/geografie>

Longitude and latitude were loaded separately, as these data won‘t change. They were fetched based on the communes scraped from immoweb by using the module Nominatim from geopy.geocoders. The code can be found in ‚coordinates.py‘. Main.py uses the dataset ‚real\_estate\_w\_coordinates.py‘ that already contains latitude and longitude since these data doesn‘t have to be reload.  
‘min\_distance‘ means the distance to the nearest of one of belgian‘s largest 10 cities: Brussels, Antwerp, Ghent, Charleroi, Liège, Anderlecht, Schaarbeek, Bruges, Namur, Leuven, Molenbeek-Saint-Jean, Mons

As found on: <https://www.statista.com/statistics/525853/the-10-largest-cities-and-municipalities-in-belgium/>

The Haversine distance is calculated in main.py based on a dictionary of these ten cities and their coordinated imported from city\_coordinates.json which was generated by ‚coordinates.py‘ using Nominatim.

A correlation matrix was used (using the spearman method as many features don‘t have a normal distribution)to eliminate features highly correlated with each other such as ‚com\_prosp\_index‘ (Prosperity index of communes), which has a correlation with ‚com\_avg\_income‘ of 0.92 or ‚bedroom\_nr‘ which has a correlation with ‚living\_area‘ of 0.78, etc.

The features were selected based on their correlation coefficients (living area: 0.43, com\_avg\_income: 0.32, building\_condition: 0.25, subtype\_of\_property: 0.18, equipped\_kitchen: 0.17, latitude: 0.20, longitude: -0.11, min\_distance: -0.14, terrace: 0.13) as well as testing and comparing model evaluation metrics. Features such as zip\_code or refnis\_code were not used, despite a moderate correlation of -0.16 and -0.29 respectively, although including these features seemingly improved the model evaluation results. They were nevertheless excluded as the selected features already include location data such as latitude, longitude and min\_distance in order to avoid a manual overweighing of location-related features.

For 4):   
k-fold cross was used (see code under ‚For 1)‘ ) with a cross-validation split of 5. Please add some explanation.  
  
For 5):  
 Training Time: 0.1343 seconds

Inference Time: 4.3527 seconds  
  
For 6):  
See explanations under ‚For 3)‘  
  
  
Additionally:  
- main.py calculates and plots distance-based analysis as well as permutation importance. SHAP was abandoned since it is in combination with cross-validation of knn regression computationally too costly (execution time over 30 min)  
- main.py also prints a set of 10 sample predictions and plots predictions vs true values