Ty: -co to jest to skalowanie, które jest przy XGB a nie ma przy Random Forescie?

-czy ten model na XGB nie jest przetrenowany?

-opisz krótko co dzieje się w trakcie modelowania

ChatGPT:

### 1. \*\*Co to jest skalowanie przy XGB (a nie ma przy Random Forescie)?\*\*

\*\*Skalowanie\*\* to przekształcenie cech liczbowych, żeby były porównywalne między sobą – najczęściej do wspólnej skali (np. średnia 0, odchylenie 1 – tzw. standaryzacja/StandardScaler).

- W XGB pipeline jest to krok:

```python

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

```

- Dla \*\*Random Forest\*\* nie stosujesz skalowania – oryginalne wartości cech wchodzą do modelu.

\*\*Dlaczego?\*\*

- Drzewa decyzyjne (a więc RandomForest i XGBoost) co do zasady \*\*nie wymagają\*\* skalowania cech – są odporne na różnice rozmiarów/zakresu cech.

- Ale:

- W praktyce \*\*XGBoost\*\* (jako booster) może czasem szybciej zbiegać i być nieco bardziej stabilny przy danych zeskalowanych – zwłaszcza przy bardzo różnych zakresach wartości.

- Dla innych modeli (regresja liniowa, sieci neuronowe) \*\*skalowanie jest obowiązkowe!\*\*

- Ty stosujesz skalowanie w pipeline XGB, bo…

- ...w Twoim kodzie pipeline jest wspólny pod porównanie (dla XGB i LinearRegression),

- ...prawdopodobnie próbowałeś też inne modele, gdzie skaler jest niezbędny.

- \*\*Nie jest to wymóg, ale nie przeszkadza XGB.\*\*

\*\*Podsumowanie:\*\*

Skalowanie jest dla bezpieczeństwa i uniwersalności pipeline’u. Dla Random Forest nie ma różnicy, dla XGB też nie powinno zaszkodzić i czasem może nawet pomóc. Bezpieczniej zostawić.

---

### 2. \*\*Czy XGB nie jest przetrenowany?\*\*

#### Co sugeruje przetrenowanie?

- \*\*R² na test secie = 0,997\*\*

- \*\*RMSE = 20 770 PLN\*\* przy średniej cenie ~784 000 PLN

- \*\*MAPE ≈ 1,4%\*\* → bardzo niskie błędy!

#### Czy to możliwe?

- \*\*Zbiór jest bardzo duży\*\* (195 tys. rekordów!).

- \*\*Feature’y\*\* są silnie skorelowane z targetem (“city\_loo”, “squareMeters”).

- Ceny w obrębie pojedynczego miasta są prawdopodobnie przewidywalne bez wielkiej złożoności (bo “city\_loo” już wrzuca adres mocno blisko wartości).

#### Skąd pewność że nie jest przetrenowany?

- \*\*Model był testowany na osobnym, dużym zbiorze testowym\*\* (38 tys. rekordów!)

- \*\*Wyniki walidacji i testu są blisko siebie\*\*:

W Optunie: \*Najlepsze na walidacji\* RMSE ≈ 20 015, \*na teście\* 20 770 PLN.

- \*\*To nie są liczby „idealne”\*\* (R² < 1!), więc model nie zapamiętał idealnie danych.

- \*\*Najważniejszy powód\*\*: silna, prosta zależność (średnia miasta + metraż) sama w sobie pozwala na bardzo wysoką skuteczność.

#### Gdyby był przetrenowany:

- Miałby super niskie błędy na train/val, a złe na test.

- Mogłyby się pojawić duże błędy dla nowych miast, nietypowych lokalizacji, bardzo rzadkich przypadków.

#### \*\*W Twoim przypadku:\*\*

- \*\*Nie wydaje się przetrenowany\*\* – model bazuje na bardzo silnych, stabilnych cechach (lokalizacja, metraż) oraz dużej próbce.

- Wynik jest zaskakująco dobry, bo problem jest przewidywalny.

##### Gdzie warto być ostrożnym?

- Gdyby Twoje dane nie miały tak widocznego efektu miasta, lub pojawiły się nietypowe cechy/testy na nowych miastach — wtedy poziom RMSE/R² mógłby spaść.

- Warto testować model także na nowych miastach, nowych czasach (ale na tych danych: nie, nie wygląda na przetrenowanie).

---

### 3. \*\*Co dzieje się w trakcie modelowania (krótko)?\*\*

1. \*\*Przygotowanie danych\*\*:

- Usuwa się outliery.

- Uzupełnia braki (KNNImputer).

- Zmienna "miasto" zamieniana na średnią ceny w danym mieście (“leave-one-out” encoding).

- Cechy binarne zakodowane jako 0/1.

- Dodatkowo (przy XGB) cechy liczbowe są \*skalowane\* (zachowują swój rozkład, ale mają średnią 0, std 1).

2. \*\*Podział na zbiory\*\*:

- Dane są dzielone na TRAIN/VALIDATION/TEST (60/20/20%).

3. \*\*Uczenie modeli\*\*:

- Na train secie trenuje się XGBoost (i porównawczo: Random Forest), szuka się najlepszych parametrów (Optuna).

- Uczenie jest ewaluowane na validation secie.

4. \*\*Ewaluacja\*\*:

- Po wybraniu najlepszych parametrów, model przewiduje ceny na zbiorze testowym.

- Liczone są metryki: RMSE, MAE, R², MAPE etc.

5. \*\*Analiza błędów i interpretacja\*\*:

- Tworzysz wykresy błędów.

- Za pomocą SHAP sprawdzasz, które cechy najbardziej wpływają na decyzje modelu (np. “city\_loo”, “squareMeters”).

6. \*\*Predykcja na nowych danych\*\*:

- Model wycenia pojedyncze mieszkanie z określonymi cechami, po odpowiednim przetworzeniu danych wejściowych.

---

### \*\*Podsumowanie\*\*

- \*\*Skalowanie\*\*: W XGB wykonywane dla uniwersalności, nie jest absolutnie konieczne.

- \*\*Przetrenowanie\*\*: Nie, wygląda na bardzo dobry fit na takiej strukturze danych – test i walidacja/zachowanie modelu nie zdradzają overfittingu.

- \*\*Przebieg modelowania\*\*: Czyszczenie i kodowanie cech, podział danych, trenowanie modeli, tuning parametrów, testy, interpretacja wpływu cech oraz predykcja nowych przykładów.

---

Chcesz więcej szczegółów o modelowaniu albo interpretacji (np. jak XGBoost podejmuje decyzje)?

Ty: powiedz więcej co dzieje się w trakcie uczenia modeli. co oznaczają poszczególne hiperparametry? Czy random forest też jest testowany w optunie?

ChatGPT:

Jasne, rozbijmy to na szczegółowe etapy:

---

## 1. \*\*Co dzieje się w trakcie uczenia modeli?\*\*

### \*\*Krok po kroku:\*\*

#### \*\*a) Przygotowanie i podział danych\*\*

Skrypt dzieli całość danych na:

- Zbiór treningowy (train): do \*\*uczenia modelu\*\*

- Zbiór walidacyjny (val): do \*\*sprawdzania skuteczności\*\* i doboru hiperparametrów

- Zbiór testowy (test): do \*\*ostatecznej oceny\*\*, na nim nie ustawiamy żadnych parametrów!

#### \*\*b) Ustawianie hiperparametrów\*\*

- Model XGBoost jest początkowo uczony na domyślnych parametrach, a potem – przy użyciu biblioteki \*\*Optuna\*\* – szukane są najlepsze kombinacje hiperparametrów (np. głębokość drzewa, liczba drzew itd.), czyli \*\*tuninguje się\*\* model pod kątem jak najniższego błędu na walidacji.

#### \*\*c) Uczenie właściwe (fit)\*\*

- Model "patrzy" na dane wejściowe i prawdziwą cenę mieszkania, a następnie buduje \*\*drzewa decyzyjne\*\* (albo lasy – w RF), które mają za zadanie przewidzieć tę cenę na podstawie cech mieszkania.

- Dla XGBoost: kolejne drzewa poprawiają to, czego nie udało się przewidzieć poprzednikom.

#### \*\*d) Predykcja i ocena\*\*

- Po nauczeniu modelu na zbiorze treningowym, testuje się przewidywania na zbiorze walidacyjnym i testowym.

- Obliczane są metryki (RMSE, MAE, MAPE, R²…), dzięki temu wiesz jak dobrze model "radzi sobie" poza zbiorem treningowym.

#### \*\*e) Interpretacja efektów\*\*

- Dzięki SHAP, sprawdzasz które cechy miały największy wpływ na przewidywaną cenę.

---

## 2. \*\*Co oznaczają poszczególne hiperparametry XGBoost?\*\*

Oto te, które tunujesz (i co oznaczają):

| Parametr | Opis |

|-----------------------|---------------------------------------------------------------------------------------|

| \*\*n\_estimators\*\* | Liczba drzew (ile "kroków" wykona model, kolejnych dopasowań) |

| \*\*max\_depth\*\* | Maksymalna głębokość pojedynczego drzewa decyzyjnego |

| \*\*learning\_rate\*\* | O ile zmienia się waga nowych drzew, tzn. jak mocno każde nowe drzewo poprawia błąd |

| \*\*subsample\*\* | Jaki procent danych jest losowany do budowy każdego drzewa (często 0.5–1.0) |

| \*\*colsample\_bytree\*\* | Jaki procent cech (kolumn) jest brany do budowy każdego drzewa |

| \*\*min\_child\_weight\*\* | Minimalna "waga" (liczba obserwacji) w "liściu" drzewa (ogranicza nadmierne dzielenie)|

| \*\*gamma\*\* | Minimalne polepszenie wymagane do podziału liścia (większa wartość ogranicza dzielenie)|

| \*\*random\_state\*\* | Losowość (by zawsze otrzymać ten sam wynik) |

\*\*Efekt\*\*:

- Więcej drzew (`n\_estimators`), większa głębokość (`max\_depth`): model może wyłapać bardziej skomplikowane relacje (ale ryzyko nadmiernego dopasowania!).

- Mniejsze `learning\_rate`: model uczy się wolniej, ale dokładniej.

- Ograniczające parametry (`min\_child\_weight`, `gamma`, subsample) zapobiegają przeuczeniu.

---

## 3. \*\*Czy Random Forest jest testowany w Optunie?\*\*

\*\*Nie, w Twoim kodzie Random Forest NIE przechodzi przez tuning Optuna.\*\*

- Random Forest trenujesz raz, z domyślnymi parametrami (`n\_estimators=100` itd.).

- Funkcja `optimize\_xgboost` jest zdefiniowana \*\*tylko dla XGBoost\*\* (tam brane są kolejne losowania hiperparametrów).

\*\*Gdybyś chciał tunować Random Forest:\*\*

- Potrzebujesz utworzyć podobną funkcję dla RF (np. `max\_depth`, `max\_features`, `n\_estimators`) i podać ją do Optuny.

- Przykład takiej optymalizacji można łatwo dodać, ale XGBoost daje realnie lepsze wyniki na takich danych.

---

## 4. \*\*Podsumowanie – kluczowe etapy uczenia modeli:\*\*

1. Dane dzielisz (train/val/test), obrabiasz.

2. Model "uczy się" wyłapywać relacje cech-cena na danych treningowych.

3. Szukasz najlepszych ustawień tylko w XGBoost (Optuna automatycznie zmienia parametry i ocenia na walidacji).

4. Ostatecznie model przewiduje ceny na testowym secie, porównujesz wyniki i interpretujesz cechy.

---

\*\*W Twoim kodzie optymalizujesz hiperparametry tylko dla XGBoost. Random Forest używa domyślnych ustawień i nie jest optymalizowany w Optunie, choć też można by to zrobić.\*\*