**New York Housing Market – Raport**

*Autorzy: Paweł Florek, Krysztof Adamczyk*

1. Dane
2. EDA
3. Preprocessing danych
4. Modele z optymalizacją Hiperparametrów i kroswalidacja
5. AutoML
6. Max Voting i Stacking
7. Explainable AI
8. Przeznaczenie projektu
9. **Dane**

Dane, których użyliśmy pochodziły ze strony <https://www.kaggle.com/datasets/nelgiriyewithana/new-york-housing-market> i dotyczyły cen mieszkań w Nowym Yorku. W ramce znajdowały się kolumny:

* BROKERTITLE: Title of broker
* TYPE: type of house
* PRICE: price of the house
* BEDS: number of bedrooms
* BATH: number of bathrooms
* PROPERTYSQFT: square footage of the property
* ADDRESS: full address of the house
* STATE: state of the house
* MAIN\_ADDRESS: main address information
* ADMINISTRATIVE\_AREA\_LEVEL\_2: Administrative area 2 level information
* LOCALITY: Locality information
* SUBLOCALITY: Sublocality information
* STREET\_NAME: street name
* LONG\_NAME: long name of house
* FORMATTED\_ADDRESS: formatted\_address
* LATITUDE: latitude of the house
* LONGITUDE: longitude of the house

a ogónie ramka miała rozmiar: (4801, 17).

Dokonaliśmy również podziału zbioru na learning\_data i validation\_data w stosunku 70:30. Ostatecznie ramka na ktrej przeprowadzaliśmy analizę danych i preprocessing liczyła 3360 wierszy i 17 w.w. kolumn

1. **EDA**

Przed rozpoczęciem pracy z przygotowaniem danych i testowaniem modeli dokonaliśmy Eksploracyjnej Analizy Danych. Dokonaliśmy tego w kilku krokach:

* **Wstępne informacje**

Na starcie rozważyliśmy z jakimi danymi mamy do czynienia, zatem wywołaliśmy metode .info() na ramce danych, otrzymaliśmy:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Jak widać mamy aż 11 zmiennych kategorycznych i 6 numerycznych z czego 4 to inty i 2 to floaty.

* **NULL values**

Z .info() łatwo zauważyć, że w żadnej kolumnie nie mamy brakujących wartości, stąd odpada nam preprocessing związany z null\_values.

* **Duplikaty**

Następnie sprawdziliśmy czy w ramce danych są wiersze duplikujące się, które mogłyby negatywnie wpływać na naszą ramkę.Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Jak widać w ramce było 97 wierszy duplikujących się, które usuwamy, gdyż są zbędne dla dalszych rozważań.

* **Zmienne numeryczne**

Rozważając zależności między zmiennymi numerycznymi, zaczęliśmy od zbadania ich metodą .describe():

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Jak łatwo zauważyć w kolumnach nie związanych z położeniem domu, mamy do czynienia z outlierami, co łatwiej jest widoczne na poniższych wykresach:

Obraz zawierający tekst, diagram, Wykres, linia

Opis wygenerowany automatycznie

Obraz zawierający diagram, linia, tekst, zrzut ekranu

Opis wygenerowany automatycznie

Jak widać w BEDS, BATH, PROPERTYSQFT i PRICE mamy outliery, którymi zajęliśmy się później w procesie preprocessingu. Chcąc zbadać, czy mamy do czynienia z jakimiś konkretnymi rozkładami zbadaliśmy, jak zachowują się zmienne numeryczne po przeskalowaniu logarytmicznym:

Obraz zawierający tekst, diagram, Wykres, zrzut ekranu

Opis wygenerowany automatycznie

I sprawdzając gęstość rozkładów:

Obraz zawierający diagram, Wykres, tekst, linia

Opis wygenerowany automatycznie

Jak widać PRICE i PROPERTYSQFT po zlogarytmowaniu, mają rozkład, który możemy przyjąć za podobny do normalnego, czyli będziemy używać tutaj zasady 3 sigm. Rozkłady pozostałych zmiennych, są już cięższe do zidentyfikowania. Na koniec analizy sprawdziliśmy korelacje zmiennych numerycznych:

Obraz zawierający tekst, zrzut ekranu, kwadrat, diagram

Opis wygenerowany automatycznie

Jak widać dane przed usunięciem outlierów są zle skorelowane ze zmienną PRICE. Bliska korelacja następuje za to między BEDS i BATHS. Ostatnia część analizy, bardziej poglądowa, obejmowała stworzenie mapy Nowego Yorku z cenami mieszkań:

Obraz zawierający mapa, tekst

Opis wygenerowany automatycznie

* **Zmienne kategoryczne**

Sprawdziliśmy też jakie mamy zależności w zmiennych kategorycznych. Rozpoczęliśmy od sprawdzenia, ile mamy wartości unikalnych w każdej kolumnie:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Jak widać w kolumnach: ADDRESS, MAIN\_ADDRESS, LONG\_NAME I FORMATTED\_ADDRESS, większość obserwacji jest unikalna, czyli nie niesie dla nas przydatnych informacji w modelowaniu, zatem usuwamy je.

Natępnie przeanalizowaliśmy jaki mamy rozkład wartości, a także rozkąłd wartości w zależności od różnych miar ceny, takich jak: mean, median, std, max, min, quantiles. Dokłądne rozkłady są dostępne na githubie pod linkiem: , ze względu na obszerność tych danych.

1. **Preprocessing danych**

Po tej wstępnej analizie danych zabraliśmy się do oczyszczania i przygotowaniu danych do modeli. Naszą prace rozbiliśmy na kilka podpunktów:

* **Outliery**

Jak już było widać podczas EDY w kolumnach BEDS, BATH, PROPERTYSQFT i PRICE mamy outliery. Wiedząc, że PRICE po zlogarytmowaniu ma rozkład podobny do normalnego, dodaliśmy nową kolumnę Price\_log i zasadą 3 sigm usunęliśmy odstające obserwacje. W ten sposób rozmiar naszej ramki zmalał do 3324 wierszy. Podobnie zadziałaliśmy na PROPERTYQFT i zredukowaliśmy ramkę danych do 3314 wierszy.

W stosunku do BEDS i BATH usunęliśmy wartości odstające poza quantile(0.95). Z dołu dane zostawiliśmy, gdyż tam outlierów nie było i wpisywały się one w rozkład.

Po usunięciu outlierów pozostało nam 3038 wierszy, gdzie rozkłady zmiennych wyglądają następująco:

Obraz zawierający diagram, zrzut ekranu, Prostokąt, linia

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, diagram, Prostokąt

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

Jak widać problemy z wartościami odstającymi zostały zażegnane. Sprawdzmy jeszcze korlelacje zmiennych:

Obraz zawierający tekst, zrzut ekranu, kwadrat, numer

Opis wygenerowany automatycznie

Jak widać korelacja nam się również poprawiła ogólnie, a np. BATH ma już silną korelacje z Price\_log, w porównaniu np. do BEDS.

* **Groupping**

Następnie przeszliśmy do obróbki zmiennych kategorycznych, a mianowicie do STATE. Wcześniej ta kolumna miała przykładowy format: 233 E 70th ST Apt 6RNew York, NY 10021. Tak długa informacja była nam zbędna, jednakże na końcu jej znajdował się kod pocztowy, który w skali miasta może być już dobrym klasyfikatorem, jeśli chodzi o położenie. Utworzyliśmy zatem nową kolumnę ZIPCODE, zawierającą kody pocztowe mieszkań.

Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznie

Jak widać nie mamy nulli i mamy 175 unikalnych obserwacji, co za tym idzie, jest to dobry klasyfikator położenia.

Następnie sprawdziliśmy, ile mamy pojedynczych obserwacji w każdej zmiennej kategorycznej:

* Liczba ZIPCODE występujących jednokrotnie: 8
* Liczba SUBLOCALITY występujących jednokrotnie: 3
* Liczba BROKERTITLE występujących jednokrotnie: 438
* Liczba TYPE występujących jednokrotnie: 2

Jak widać BROKERTITLE można zbić w jedną kategorie, tym bardziej, że każdy BROKER jest zapisany raz, więc wszystkie obserwacje jednokrotne zbijamy w jedną obserwacje: ‘Others’. Otrzymaliśmy w ten sposób 372 unikalne obserwacje, co jest już znośne dla modelu.

* **Kategoryzacja cen**

Również musieliśmy dokonać kategoryzacji cen, aby model miał zawężoną predykcje. Rozważyliśmy w tym celu histogram rozkładu Price\_log:

Obraz zawierający Wykres, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

Wybierając odpowiedni podział chcieliśmy wybrać takie kategorie, które będą miały wystarczająco dużo obserwacji, a także miejsca podziałów nie będą w miejscach, gdzie mamy bardzo dużą ilość obserwacji. Zminimalizuje to problem z wartościami bliskimi granicom. Wybór padł na podział na 3 kategorie: Cheap (Price\_log <= 13), Mid (13 < Price\_log <=14.5) i Expensive (Price\_log > 14.5)

* **Encoding zmiennych kategorycznych**

Następnie zencodowaliśmy zmienne kategoryczne. Z racji dużej ilości unikalnych wartości użycie LabelEncodera czy OneHotEncodera byłoby bardzo nieefektywne, zatem użyliśmy lepszego do tego TargetEncodera, który został zastosowany na kolumnach: BROKERTITLE, ZIPCODE, TYPE i SUBLOCALITY. Te bowiem zmienne kategoryczne, ostatecznie używamy w modelu.

* **Skalowanie zmiennych numerycznych**

Na koniec preprocessingu przeskalowaliśmy zmienne kategoryczne StandardScallerem, aby model był bardziej wydajny. Zostało to zastosowane na kolumnach: BEDS, BATH, SQFT\_LOG, LATITUDE i LONGITUDE. Te kolumny będą użyte do trenowania naszego modelu

* **Podział na zbiór treningowy i testowy**

Oczywiście przed modelowaniem, podzieliliśmy nasz zbiór na zbiór treningowy i testowy w stosunku 8:2. Naszą zmienną docelową jest PRICE\_CATEGORY, które zostało zmapowane: Cheap = 0, Mid = 1, Expensive = 2.

1. **Modele z z optymalizacją hiperparametrów i krosswalidacją**

Po przygotowaniu danych rozpoczęliśmy modelowanie. Główną skalą, do której maksymalizacji dążymy jest jak najwyższe accuracy. Ustaliliśmy ta, gdyż nasz problem obejmuje poprawną predykcje ceny, a nie skuteczność TP, TN, FP czy FN. Chcemy osiągnąć jak najwyższe accuracy dla każdej wartości PRICE\_CATEGORY. Użyliśmy do tego modeli:

* Logistic Regression
* Random Forest Classifier
* Decision Tree Classfier
* Gradient Boosting Classifier
* SVC
* KNeighbors Classifier
* XGBoost

Postanowiliśmy znaleźć jak najlepsze parametry, które pozwolą nam na osiągniecie odpowiedniego accuracy. Użyliśmy do tego metod: GridSearchCV(), RandomizedSearchCV(), które stosując metode krosswalidacji pozwalają na znalezienie najbardziej optymalnych hiperparametrów dla danego modelu i naszych danych. Parametry, które otrzymaliśmy:

* Logistic Regression: Best: 0.802715 using {'max\_iter': 100000}
* Decision Tree Classifier: Best: 0.806001 using {'ccp\_alpha': 0, 'criterion': 'gini', 'max\_depth': 6, 'max\_features': 'sqrt'}
* SVC: Best: 0.824947 using {'C': 100, 'degree': 1, 'gamma': 0.01}
* KNeighbors Classiefier: Best: 0.829078 using {'metric': 'cosine', 'n\_neighbors': 7, 'weights': 'uniform'}
* Random Forest Classifier: Best: 0.835249 using {'n\_estimators': 100, 'max\_features': 'log2', 'max\_depth': 5, 'criterion': 'log\_loss'}
* Gradient Boosting Classifier: Best: 0.857494 using {'n\_estimators': 500, 'max\_features': 'sqrt', 'max\_depth': 6, 'loss': 'log\_loss', 'learning\_rate': 0.05, 'criterion': 'friedman\_mse'}

1. **AutoML**

Po dobraniu najlepszych hiperparametrów w.w. metodami postanowiliśmy użyć również metody AutoML, a dokładnie TPOT, który automatycznie wybiera jak najlepsze hiperparametry dla modeli używając krosswalidacji, przy okazji optymalizując cały proces. Po wywołaniu otrzymaliśmy:

Obraz zawierający tekst, Czcionka, zrzut ekranu, algebra

Opis wygenerowany automatycznie

Obraz zawierający tekst, Czcionka, zrzut ekranu, biały

Opis wygenerowany automatycznie

Zatem będziemy dalej używać również XGBClassfier o podanych parametrach.

1. **Crosswalidacja i ostateczny wynik**

Przed porównaniem najlepszych modeli, użyliśmy jeszcze dla każdego z nich RepeatedKFolds, który crosswalidacyjnie znajdzie nam najlepsze wyniki dla każdego z modeli o optymalnych hiperparametrach. Ostatecznie otrzymaliśmy wyniki:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Łatwo zauważyć, że najlepsze modele w kontekście skali mean i std dl crosswalidacji to XGBClassifier, RandomForestClassifier i GradientBoostingClassifier. Również pod względem accuracy są to najlepsze modele, w naszym problemie.

1. **Max Voting i Stacking**

Po znalezieniu prostych modeli postanowiliśmy użyć bardziej zaawansowanych modeli, które agregują wyniki otrzymane przez różne modele. Max Voting został użyty z voting = ‘hard’, zatem każdy model ma taką samą wagę. Natomiast Stacking jest na końcu połączony prostym meta-modelem, którym jest Regresja Logistyczna. W estymatorach zagregowanych wszystkich najbardziej optymalnych estymatorów. Otrzymaliśmy:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Zatem widzimy, że najlepsze wyniki otrzymaliśmy dla: XGBClassfier, StackingClassifier i VotingClasiffier, czyli dla bardziej zaawansowanych modeli.

1. **Explainable AI**

Opisz ja nie wiem co

1. **Przeznaczenie projektu**

Projekt i modele otrzymane przez nas mogłyby być użyte przede wszystkim do predykcji cen dla ludzi chcących zakupić dom/mieszkanie szczególnie od prywatnych właścicieli. Znając dane takie jak, ilość sypialni, ilość łazienek, powierzchnia czy położenie mieszkania, jesteśmy w stanie z 88% skutecznością przewidzieć, jaka powinna być jego cena. Pozwoliłoby to ludziom, na sprawdzenie czy cena jaką żąda właściciel nie jest zbyt wygórowana względem statystycznej ceny i wartości. Kupując od firmy/pośrednika, też klient może łatwo sprawdzić w jakiej kategorii powinna zaleźć się cena mieszkania w stosunku do wartości. Pozwoliłoby to na pewno na ochronne klientów przed oszustami czy naciągaczami, w szczególności tych klientów, którzy nie mają dużej wiedzy na temat cen mieszkań. Klienci również używając naszych modeli, byliby w stanie porównywać ceny i filtrować po kliku najważniejszych kategoriach, dzięki czemu mogliby wybrać jak najkorzystniejsze dla siebie oferty. Znajomość jeszcze kilku kategorii, których nie było w naszej ramce danych tj. data wybudowania mieszkania czy data ostatniego remontu etc. Reasumując projekt może mieć duże zastosowanie biznesowe dla klientów, lecz także dla pośredników, którzy mogliby sprawdzać automatycznie, ile powinni zażądać za nieruchomość. Bez wątpienia można by go jeszcze udoskonalić, przy poświęceniu większej ilości czasu.