1. Wstęp

Wynik na Kaggle: **89401.11**

Nick: **Miłosz Lauda**

Zajęte miejsce w rankingu: **10**

W projekcie zastosowano imputację metodą iteracyjnego imputatora dostępną w bibliotece scikit.learn. Jako estymator w metodzie (zamiast domyślnej regresji liniowej) zastosowano algorytm lasu losowego (Random Forest Regressor dla zmiennych flat_rooms i flat_area, Random Forest Classifier dla zmiennych przedstawionych w formacie binarnym). Jako główny model predykcji cen nieruchomości wykorzystano model Random Forest Regressor. W projekcie skorzystano z dużych modeli językowych, głównie z dostępnego w Google Colab modelu Gemini, który znacznie przyspieszył pracę wykonując zlecone mu zadania. Dodatkowo także ChatGPT.

Wyniki są w pełni odtwarzalne poprzez ustawienie ziarna dla modelu predykcji.

2. Metodyka

Wstępna analiza obejmowała usunięcie kolumn datowych, ponieważ nie zdecydowano się ostatecznie na zawarcie informacji w nich zawartych do analizy, a także identyfikację i zamianę wartości nieprawidłowych, takich jak ujemne liczby w kolumnach dotyczących powierzchni mieszkań i liczby pokoi. Dane zostały przygotowane tak, aby mogły być wykorzystane w modelach predykcyjnych.

Kolejnym krokiem była imputacja brakujących danych. W tym celu zastosowano iteracyjny imputator (IterativeImputer) oparty na lasach losowych, który pozwala na uzupełnianie zarówno cech zero-jedynkowych jak i ciągłych. Dla cech kategorialnych(quarter) użyto kodowania One-Hot (OneHotEncoder), co umożliwiło przekształcenie zmiennych tekstowych na reprezentacje numeryczne.

W modelowaniu wykorzystano regresor Random Forest, który charakteryzuje się odpornością na nadmierne dopasowanie i skutecznością w radzeniu sobie z danymi o dużej liczbie zmiennych. Do oceny modelu skorzystano z walidacji krzyżowej. Wyniki oceniano na podstawie metryk, takich jak średni błąd kwadratowy (MSE) oraz współczynnik determinacji (R-squared).

Z metod, z których nie skorzystano ale sprawdzano i brano pod uwagę było skalowanie zmiennych i usuwanie wartości odstających.

3. Wynik

Ilość prób na Kaggle: **53**

Sama zmiana głównego modelu z regresji liniowej na las losowy poskutkowało poprawieniem wyniku z 220 000 do 94 000. Nie zdecydowano się na skorzystania z strojenia parametrów modelu z uwagi na brak polepszenia MSE, a także na długi czas wykonywania kodu. Natomiast zamiast GridSearch (którego czas wykonywania jest zdecydowanie długi z uwagi na ślepe sprawdzanie wszystkich kombinacji parametrów) i zamiast Random Search, ktróry może pominąć najlepsze parametry polecam bibliotekę *Optuna*. Narzędzie to przeszukuje przestrzeń hiperparametrów w sposób inteligentny, dzieląc ją na zbiór mniej i bardziej obiecujący i zawężając swoje poszukiwania. Optuna .

Zmiana metody imputacji z KNNImputer na IterativeImputer zmniejszyła MSE o 5tys.

Ponadto znaczące poprawienie wyników powstało także wskutek skorzystania z wyrażeń regularnych i wyciągnięcia informacji z tytułu (zmniejszenie MSE o około 3tys.), a także zmiana metody imputacji ze zwykłego IterativeImputer na IterativeImputer korzystającego z estymatorów Random Forest Classifier i Regressor (zmniejszenie MSE o około 1tys.).

4. Podsumowanie

Modele LLM okazały się nieocenione w budowie projektu. Nie chodzi tylko o pisanie kodu, ale także o generowanie pomysłów do poprawienia wyniku, mimo iż nie zawsze wynik się poprawiał.

Jednym z pomysłów, które przyniosło nieznaczną poprawę wyniku jest stworzenie zmiennych kawalerka, ponieważ 25% obserwacji w obu zbiorach to są kawalerki, a także stworzenie zmiennej luksus na podstawie specyficznych słów w tytule. Dużą nadzieję wiązałem z wykorzystaniem zmiennej flat_deposit, która dla zbioru treningowego w 25% obserwacji była równa zmiennej price, ale ostatecznie nie przyniosło to zadowalających rezultatów.

5. Kod

Załadowanie potrzebnych pakietów

```
# Import biblioteki pandas do pracy z ramkami danych
import pandas as pd

# Import biblioteki numpy do obliczeń numerycznych
import numpy as np

# Import biblioteki re do pracy z wyrażeniami regularnymi
import re
```

```
# Import klasy OneHotEncoder do kodowania zmiennych kategorialnych na zmienne numeryczne
from sklearn.preprocessing import OneHotEncoder

# Import funkcji cross_validate do walidacji krzyżowej modelu
from sklearn.model_selection import cross_validate

# Import klasy KFold do podziału danych na k-foldy w walidacji krzyżowej
from sklearn.model_selection import KFold

# Włączenie eksperymentalnej funkcji IterativeImputer (uzupełnianie brakujących danych)
from sklearn.experimental import enable_iterative_imputer

# Import klasy IterativeImputer do uzupełniania brakujących danych iteracyjnie
from sklearn.impute import IterativeImputer

# Import klasy RandomForestClassifier do klasyfikacji (uzupełnianie brakujących danych (binarnych))
from sklearn.ensemble import RandomForestClassifier

# Import klasy RandomForestRegressor do regresji (uzupełnianie brakujących danych numerycznych(ciągłych) i budowa modelu)
from sklearn.ensemble import RandomForestRegressor

# Import funkcji mean_squared_error i r2_score do oceny modelu regresji
from sklearn.metrics import mean_squared_error, r2_score
```

Import danych

```
train_df = pd.read_csv('pzn-rent-train.csv')
test_df = pd.read_csv('pzn-rent-test.csv')
```

Przygotowanie danych

Złączenie obu zbiorów danych

```
rent_df = pd.concat([train_df, test_df], ignore_index=True)
```

Usunięcie kolumn zawierających datę

```
rent_df.drop(['date_activ','date_modif','date_expire'], axis=1, inplace=True)
```

Zastąpienie wartości ujemnych w kolumnach flat_area i flat_rooms brakującymi wartościami

```
rent_df.loc[rent_df['flat_area'] < 0, 'flat_area'] = np.nan

rent_df.loc[rent_df['flat_rooms'] < 0, 'flat_rooms'] = np.nan
```

Użycie wyrażen regularnych

```
def extract_info(ad_title):
    """
    Wyodrębnia informacje o pokojach, powierzchni, dzielnicy,
kawalerce, balkonie i garażu
    z tytułu ogłoszenia.
    """
    price = None
    area = None
    rooms = None
    furnished = None
    students = None
    balcony = None
    garage = None
    garden = None
    air_conditioning = None
    internet = None
    quarter = None
    kawalerka = None
    luksus = None

    # Extract price
    match = re.search(r'(\d+)\s*zł|zl', ad_title, re.IGNORECASE)
    if match:
        # Check if match.group(1) is not None before converting to int
        if match.group(1) is not None:
            price = int(match.group(1))
        else:
            price = None  # Or any other default value you prefer

    # Extract area
    match = re.search(r'(\d+(?:,\d+)?)\s*(?:m2|m²|metr|mkw|m kw)',
ad_title, re.IGNORECASE)
    if match:
        area = float(match.group(1).replace(',', '.'))

    # Extract number of rooms
    match = re.search(r'(\d+)\s*pok', ad_title, re.IGNORECASE)
    if match:
        rooms = int(match.group(1))
```

```
    # Check for furnished
    match = re.search(r'\b(?!nie)(umebl|wyposaż)\w*\b', ad_title,
re.IGNORECASE)
    if match:
        furnished = 1

    # Check for students
    match = re.search(r'studen', ad_title, re.IGNORECASE)
    if match:
        students = 1

    # Check for balcony
    match =  re.search(r'balkon', ad_title, re.IGNORECASE)
    if match:
        balcony = 1

    # Check for garage
    match = re.search(r'garaż|garaz', ad_title, re.IGNORECASE)
    if match:
        garage = 1

    # Check for garden
    match = re.search(r'ogród', ad_title, re.IGNORECASE)
    if match:
        garden = 1

    # Check for air conditioning
    match = re.search(r'klimatyzacj', ad_title, re.IGNORECASE)
    if match:
        air_conditioning = 1

    # Check for internet
    match = re.search(r'internet', ad_title, re.IGNORECASE)
    if match:
        internet = 1

    # Extract quarter
    match = re.search(r'(Centrum|Chartowo|Chwaliszewo|Dębiec|Franowo|
Górczyn|Grunwald|Jeżyce|Junikowo|Ławica|Łazarz|Malta|Naramowice|Nowe
Miasto|Ogrody|Ostrów Tumski|Piątkowo|Podolany|Polanka|Rataje|Sołacz|
Stare Miasto|Starołęka|Stary Rynek|Strzeszyn|Śródka|Wilczak|Wilda|
Winiary|Winogrady|Wola|Zawady|Żegrze)', ad_title, re.IGNORECASE)
    if match:
        quarter = match.group(1).capitalize()

    # Check for kawalerka
    match  = re.search(r'kawal', ad_title, re.IGNORECASE)
    if match:
        kawalerka = 1
    else:
```

```
        kawalerka = 0

    # Check for luksus
    match = re.search(r'(kancelaria|reprezentac|hotel|secesyj)',
ad_title, re.IGNORECASE)
    if match:
        luksus = 1
    else:
        luksus = 0

    return price, area, rooms, furnished, students, balcony, garage,
garden, air_conditioning, internet, quarter, kawalerka, luksus


def process_dataframe(df):

    """
    Przetwarza DataFrame, aby wypełnić brakujące wartości w różnych
kolumnach
    za pomocą funkcji extract_info.
    """

    for index, row in df.iterrows():
        price, area, rooms, furnished, students, balcony, garage,
garden, air_conditioning, internet, quarter, kawalerka, luksus =
extract_info(row['ad_title'])
        df.loc[index, 'kawalerka'] = kawalerka
        df.loc[index, 'luksus'] = luksus

        if pd.isna(row['price']) or pd.isna(row['flat_area']) or
pd.isna(row['flat_rooms']) or pd.isna(row['flat_for_students']) or
pd.isna(row['flat_balcony']) or pd.isna(row['flat_garage']) or
pd.isna(row['flat_garden']) or pd.isna(row['flat_air_cond']) or
pd.isna(row['flat_internet']):
            if rooms is not None:
                df.loc[index, 'flat_rooms'] = rooms
            if area is not None:
                df.loc[index, 'flat_area'] = area
            if furnished is not None:
                df.loc[index, 'flat_furnished'] = furnished
            if quarter is not None:
                df.loc[index, 'quarter'] = quarter
            if balcony is not None:
                df.loc[index, 'flat_balcony'] = balcony
            if garage is not None:
                df.loc[index, 'flat_garage'] = garage
            if garden is not None:
                df.loc[index, 'flat_garden'] = garden
            if air_conditioning is not None:
                df.loc[index, 'flat_air_cond'] = air_conditioning
```

```
            if internet is not None:
                df.loc[index, 'flat_internet'] = internet
            if students is not None:
                df.loc[index, 'flat_for_students'] = students
            if price is not None and pd.isna(row['price']) and 700 <=
price <= 4000:
                df.loc[index, 'price'] = price

    return df

rent_df = process_dataframe(rent_df)
```

```
<ipython-input-29-a9957d5cb152>:124: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise an error in a future
version of pandas. Value '1' has dtype incompatible with bool, please
explicitly cast to a compatible dtype first.
  df.loc[index, 'flat_air_cond'] = air_conditioning
```

Imputacja

Usunięcie kolumn, które nie wymagają imputowania

```
rent_df.drop(['ad_title','price','flat_internet','flat_anti_blinds'],
axis=1, inplace=True)
```

Zamiana kategorycznej kolumny quarter na wiele kolumn numerycznych stosując **One Hot Encoding**

```
from sklearn.preprocessing import OneHotEncoder

# Kodowanie typu one-hot
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Przekształcenie kolumny 'quarter' na wiele kolumn numerycznych
encoded_quarter = encoder.fit_transform(rent_df[['quarter']])

# Utworzenie nowego DataFrame z zakodowanymi danymi
encoded_df = pd.DataFrame(encoded_quarter,
columns=encoder.get_feature_names_out(['quarter']))

# Połączenie oryginalnego zbioru danych z przekształconą kolumną
quarter
rent_df = pd.concat([rent_df, encoded_df.drop('quarter_nan',axis=1)],
axis=1)

# Usunięcie oryginalnej kolumny 'quarter'
rent_df = rent_df.drop(['quarter'], axis=1)
```

Utworzenie iteracyjnego imputera. Jako parametr - estymator użyto lasu losowego.

```
# Iteracyjna imputacja

# Utworzenie obiektu IterativeImputer z estymatorem
RandomForestClassifier dla cech kategorycznych
imputer = IterativeImputer(estimator=RandomForestClassifier())

# Utworzenie obiektu IterativeImputer z estymatorem
RandomForestRegressor dla cech numerycznych
imputer_num = IterativeImputer(estimator=RandomForestRegressor())

# Wybór kolumn zaczynających się od 'quarter' za pomocą wyrażenia
listowego
quarter_cols = [col for col in rent_df.columns if
col.startswith('quarter_')]

# Ustawienie kolumn do imputacji
columns_to_impute = quarter_cols + ['flat_balcony']

# Imputacja brakujących wartości
rent_df_imputed =
pd.DataFrame(imputer.fit_transform(rent_df[columns_to_impute]))

# Zastąpienie oryginalnych kolumn w rent_df wartościami imputowanymi
rent_df = rent_df.drop(columns_to_impute, axis=1)
rent_df = pd.concat([rent_df, rent_df_imputed], axis=1)

# Imputacja brakujących wartości w kolumnach 'flat_area', 'flat_rooms'
i 'individual'
rent_df['flat_area'] =
imputer_num.fit_transform(rent_df[['flat_area']])
rent_df['flat_rooms'] =
imputer_num.fit_transform(rent_df[['flat_rooms']])
rent_df['individual'] = imputer.fit_transform(rent_df[['individual']])

# Ustawienie kolumn do imputacji
columns_to_impute =
['flat_garage','flat_dishwasher','flat_furnished','flat_garden']

# Imputacja brakujących wartości
rent_df_imputed =
pd.DataFrame(imputer.fit_transform(rent_df[columns_to_impute]))

# Zastąpienie oryginalnych kolumn w rent_df wartościami imputowanymi
rent_df = rent_df.drop(columns_to_impute, axis=1)
rent_df = pd.concat([rent_df, rent_df_imputed], axis=1)

# Ustawienie kolumn do imputacji
columns_to_impute = ['flat_for_students','flat_closed_area']
```

```
# Imputacja brakujących wartości
rent_df_imputed =
pd.DataFrame(imputer.fit_transform(rent_df[columns_to_impute]))

# Zastąpienie oryginalnych kolumn w rent_df wartościami imputowanymi
rent_df = rent_df.drop(columns_to_impute, axis=1)
rent_df = pd.concat([rent_df, rent_df_imputed], axis=1)

# Połącz kolumnę 'price' z zbioru uczącego do rent_df na podstawie
'id'
rent_df = pd.merge(rent_df, train_df[['id', 'price']], on='id',
how='left')

# Podziel rent_df na zbiory treningowe i testowe na podstawie
obecności brakujących wartości w kolumnie 'price'
poznan_rent_train = rent_df[rent_df['price'].notnull()]
poznan_rent_test = rent_df[rent_df['price'].isnull()]
poznan_rent_test.drop(['price'], axis=1, inplace=True)
```

```
<ipython-input-34-ea48997fc11e>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  poznan_rent_test.drop(['price'], axis=1, inplace=True)
```

```
# Konwersja nazw kolumn na typ tekstowy

poznan_rent_train.columns = poznan_rent_train.columns.astype(str)
poznan_rent_test.columns = poznan_rent_test.columns.astype(str)
```

Dopasowanie modelu Random Forest do danych treningowych

```
X_train = poznan_rent_train.drop(['price'], axis=1).values

y_train = poznan_rent_train['price'].values

rf_model = RandomForestRegressor(n_estimators = 300, random_state=42)

#Ustawione ziarno w celu odtworzenia jednakowych wyników
#Im wyższy parametr n_estimators tym trafniejsze predykcje, ale
rownież wyższa złożoność obliczeniowa i dłuższy czas trenowania modelu

rf_model.fit(X_train, y_train)
```

```
RandomForestRegressor(n_estimators=300, random_state=42)
```

Utworzenie obiektu KFold do walidacji krzyżowej modelu

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

Ewaluacja modelu

```
scores = cross_validate(rf_model, X_train, y_train, cv=kf,
                        scoring=['neg_mean_squared_error', 'r2'],
                        return_train_score=True)

metrics = pd.DataFrame({
    'Test': [-np.mean(scores['test_neg_mean_squared_error']),
             np.std(scores['test_neg_mean_squared_error']),
             np.mean(scores['test_r2']),
             np.std(scores['test_r2'])],
    'Train': [-np.mean(scores['train_neg_mean_squared_error']),
              np.std(scores['train_neg_mean_squared_error']),
              np.mean(scores['train_r2']),
              np.std(scores['train_r2'])]
}, index=['Average MSE', 'Std Dev of MSE', 'Average R-squared', 'Std
Dev of R-squared'])

print(metrics)
```

```
                              Test         Train
Average MSE           93018.302745  12781.338914
Std Dev of MSE         6161.021811    120.866299
Average R-squared         0.694799      0.958068
Std Dev of R-squared      0.019146      0.000378
```

Obliczenie wartości przewidywanych

```
y_pred = rf_model.predict(poznan_rent_test.values)
```

Stworzenie DataFrame z predykcjami o wymaganej strukturze

```
submission_df = pd.DataFrame({'ID': range(1, len(y_pred) + 1),
'TARGET': y_pred})
submission_df
```


```
{"summary":"{\n  \"name\": \"submission_df\",\n  \"rows\": 4842,\n
\"fields\": [\n    {\n      \"column\": \"ID\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 1397,\n
\"min\": 1,\n        \"max\": 4842,\n        \"num_unique_values\":
4842,\n        \"samples\": [\n          4810,\n          3078,\n
2693\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

\"TARGET\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 442.38146387180655,\n   \"min\": 732.9333333333333,\n
\"max\": 3856.6,\n   \"num_unique_values\": 4790,\n
\"samples\": [\n   1806.2533333333333,\n
962.7333333333333,\n   1166.1433333333334\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"submission_df"}

Eksport wyników do pliku csv

```
submission_df.to_csv('submission.csv', index=False)
```