Studium przypadku

autor: Igor Czudy 145198

W studium zostały użyte następujące technologie:

- pandas - obsługa tabel
- numpy - obsługa wielowymiarowych wektorów
- matplotlib oraz seaborn - wizualizacja
- sklearn - algorytmy ML

```
import numpy as np
import pandas as pd

df = pd.read_csv("data.csv")
df.shape
```

```
(4600, 18)
```

```
df.columns
```

```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'street', 'city',
       'statezip', 'country'],
      dtype='object')
```

```
df.head()
```

```
                  date      price  bedrooms  bathrooms  sqft_living
sqft_lot
0  2014-05-02 00:00:00   313000.0       3.0       1.50         1340
7912  \
1  2014-05-02 00:00:00  2384000.0       5.0       2.50         3650
9050
2  2014-05-02 00:00:00   342000.0       3.0       2.00         1930
11947
3  2014-05-02 00:00:00   420000.0       3.0       2.25         2000
8030
4  2014-05-02 00:00:00   550000.0       4.0       2.50         1940
10500

   floors  waterfront  view  condition  sqft_above  sqft_basement
yr_built
0     1.5           0     0          3        1340              0
1955  \
1     2.0           0     4          5        3370            280
1921
```

```
2     1.0           0     0          4        1930              0
1966
3     1.0           0     0          4        1000           1000
1963
4     1.0           0     0          4        1140            800
1976

   yr_renovated                    street       city  statezip country

0          2005      18810 Densmore Ave N  Shoreline  WA 98133     USA

1             0           709 W Blaine St    Seattle  WA 98119     USA

2             0  26206-26214 143rd Ave SE       Kent  WA 98042     USA

3             0           857 170th Pl NE   Bellevue  WA 98008     USA

4          1992         9105 170th Ave NE    Redmond  WA 98052     USA
```

Wstęp

Zakup mieszkania jest jedną z najważniejszych decyzji finansowych podejmowaną przez przeciętnego człowieka. Dlatego narzędzie, które informowałoby użytkownika czy mieszkanie, które ma on zamiar kupić, jest w dobrej cenie, bez wątpienia jest przydatne. Celem tego projektu jest przewidzenie ceny mieszkania na podstawie 17 cech.

Opis zbioru danych

Zbiór zawiera 4600 obserwacji będących przeszłymi tranzakcjami zakupu nieruchomości w USA. Dostępny jest na następującej stronie: https://www.kaggle.com/datasets/shree1992/housedata?resource=download Zbór składa sie z 18 cech, takich jak:

- price - cena mieszkania, zmienna przewidywana
- date - data sprzedaży mieszkania
- bedrooms - liczba sypialni w mieszkaniu
- bathrooms - liczba łazienek w mieszkaniu
- sqft_living - liczma stóp kwadratowych powierzchni mieszkalnej
- sqft_lot - liczba stóp kwadratowych działki
- floors - liczba pięter mieszkania
- waterfront - zmienna informująca czy mieszkanie znajduje się przy wodzie
- view - atrakcyjność widoku z okna w skali od 1 do 4, gdzie 4 to najlepszy widok, a 1 najgorszy
- condition - stan mieszkania oceniany w skali od 1 do 5, gdzie 5 to najlepszy stan, a 1 najgorszy
- sqft_above - liczba stóp kwadratowych nadziemnej powierzchni
- sqft_basement - liczba stóp kwadratowych piwnicy
- yr_built - rok budowy

- yr_renovated - rok remontu
- street - nazwa ulicy na której znajduje się nieruchmość
- city - nazwa miasta w którym znajduje się nieruchomość
- statezip - zmienna informująca o lakalizacji nieruchmości będąca połączeniem stanu oraz kodu pocztowego
- country - nazwa kraju, którym znajduje się mieszkanie.

Usunięcie przykładów, które jako cenę sprzedaży posiadają 0.

```
df = df[df['price'] != 0]

max_price_row = df[df['price'] == df['price'].max()]
min_price_row = df[df['price'] == df['price'].min()]
new_df = pd.concat([min_price_row, max_price_row])
new_df
```

```
                     date       price  bedrooms  bathrooms
sqft_living
4351  2014-05-06 00:00:00      7800.0       2.0        1.0
780  \
4350  2014-07-03 00:00:00  26590000.0       3.0        2.0
1180

      sqft_lot  floors  waterfront  view  condition  sqft_above
4351     16344     1.0           0     0          1         780  \
4350      7793     1.0           0     0          4        1180

      sqft_basement  yr_built  yr_renovated             street
city
4351              0      1942             0    4229 S 144th St
Tukwila  \
4350              0      1992             0  12005 SE 219th Ct
Kent

      statezip country
4351  WA 98168     USA
4350  WA 98031     USA
```

Przykładowe dane

Powyżej zostały zaprezentowane dwa przykłady ze zbioru danych. Autor zdecydował się na zaprezentowaniu najdorższej oraz najtańszej nieruchmości. Najtańsza nieruchomość kosztuje 7800, posiada dwa pokoje, jedną łazienkę, powierzchnię 780, co jest wartością zdecydowanie poniżej średniej, która wynosi 2130 . Warto zwrócić również uwagę, że budynek ten jest stary, został wybudowany 1942 i od tego czasu nie został odnowiony, co z pewnością ma wpływ na jego niską cenę. Najdroższa nieruchomość kosztuje 26,59 mln. Posiada ona 3 pokoje i dwie łazienki. Jest ona również dużo nowsza, niż opisana powyżej najtańsza nieruchomość. Wstępne intucje podpowiadałyby, że najdroższa nieruchomość powinna mieć wysoką wartość w kategori

"view" oraz wartośc 1 na cesze "waterfront", jednak w tym przypadku tak nie jest. Z pewnością jednak na tak wysoki koszt ma wpływ stan nieruchomości oceniony na 4/5.

```
import matplotlib.pyplot as plt
df['date'] = pd.to_datetime(df['date'])
```

Rozkład wartości

Poniżej znajduje się tabela zawierająca podstawowe statystyki dla każdej z cech oraz ich wykresy z rozkładami wartości. Warto zwrócić uwagę, że wszystie dane pochodźą z dwóch miesięcy roku 2014. Oznacza to, że dane te nie nadają się do wskazania długoterminowego trendu cen nieruchomości.

Średnia cena nieruchomości wynosi 557 905. Liczba ta może stanowić baseline do sprawdzenia jakości wytrenowanych modelów w pożniejszym czasie.

Dodatkowo widoczne jest, że w danych znajdują się przykłady z wartościami równymi zero na cechach bedrooms, bathrooms, yr_renovated. Najprawdpodobniej oznacza to, że w danych nie mamy informacji o tych cechach, a zamiast 0, powinny się tam znajdować wartości Null.

Dodatkowo, przed obliczeniem poniższej tebeli, zostały usunięte wszystkie przykłady, których cena sprzedaży wynosiła 0. Były to przykłady, które nie zawierały pełnej informacji niezbędnej do wytrenowania modeli.

Cechami, które wydają się autorowi być kluczowe w przewidzeniu wartości "prices" wydają się być: "sqrt_living", której średnia wynosi 4551, "condition", ze średnią 3.44, "yr_building" - średnia 1970 oraz "bedrooms" - średnia 3.39.

Dodatkowo warto spojrzeć na cechę bathrooms. Co ciekawe wartośc ta nie jest wartością dyskretną, a więc nie może ona oznaczać liczby łazienek. Zbiór danych nie dostarcza szczegółowych informacji dlaczego jest to wartość ciągła, a nie dyskretna. Najprawdopodboniej oznacza to, że cecha "bathrooms" jest bardziej złożona i uwzględnia różne rodzaje łazienek, co może prowadzić do liczby łazienek w formie ułamkowej. Na przykład, łazienka z prysznicem i wanną może być liczbą 1.5, a wartość 0.5 to łazienka z toaletą i umywalką, ale bez prysznica lub wanny.

Widoczne jest również, że bardzo mało mieszkań posiada widok na zbiornik wodny.

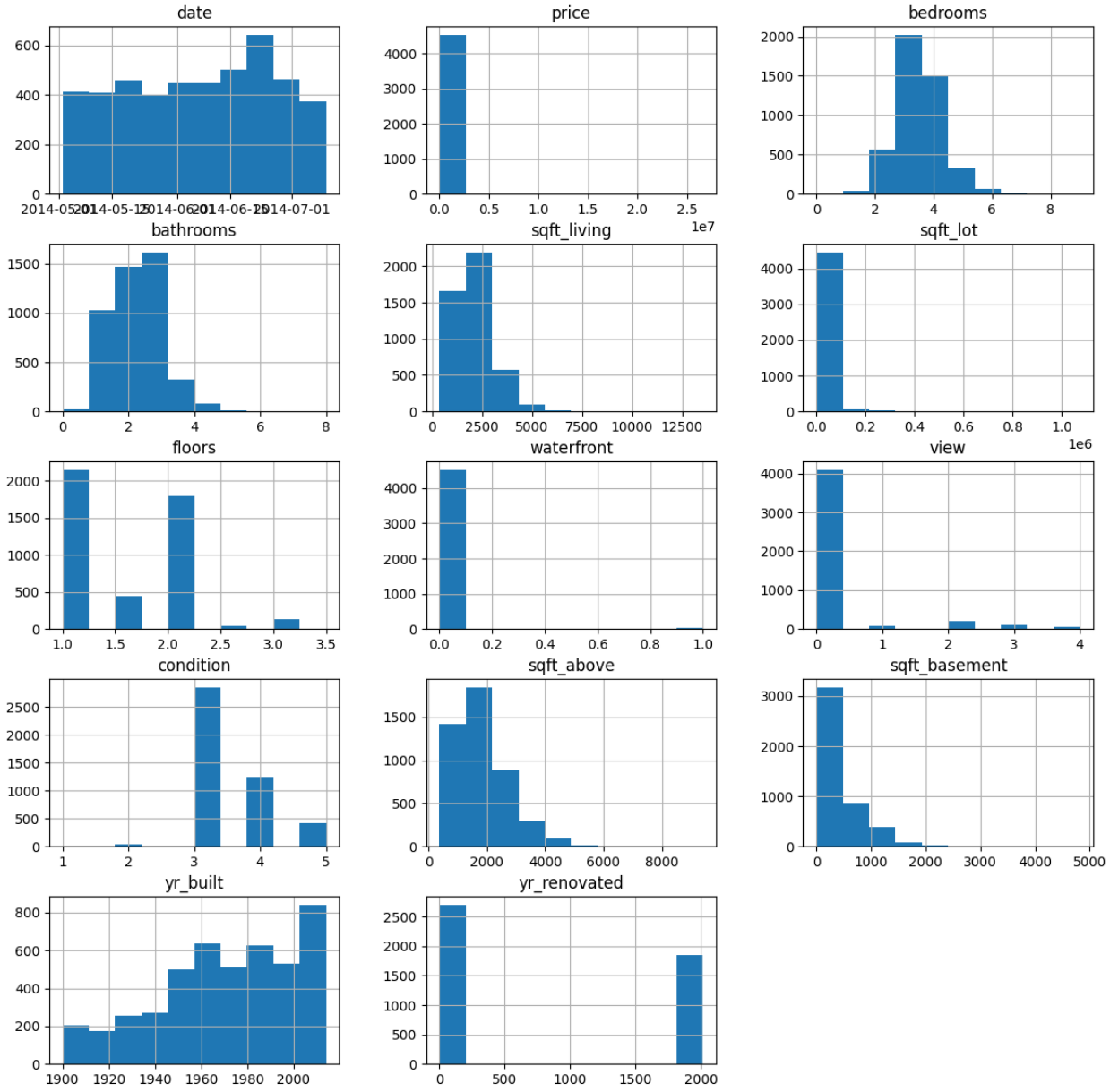
```
df.describe()
```

```
                                date         price     bedrooms
bathrooms
count                           4551  4.551000e+03  4551.000000
4551.000000  \
mean   2014-06-07 03:59:12.537903872  5.579059e+05     3.394639
2.155021
min              2014-05-02 00:00:00  7.800000e+03     0.000000
0.000000
25%              2014-05-21 00:00:00  3.262643e+05     3.000000
1.750000
```

```
50%                  2014-06-09 00:00:00  4.650000e+05     3.000000
2.250000
75%                  2014-06-24 00:00:00  6.575000e+05     4.000000
2.500000
max                  2014-07-10 00:00:00  2.659000e+07     9.000000
8.000000
std                                  NaN  5.639299e+05     0.904595
0.776351

        sqft_living      sqft_lot       floors   waterfront
view
count   4551.000000  4.551000e+03  4551.000000  4551.000000
4551.000000  \
mean    2132.372226  1.483528e+04     1.512195     0.006592
0.234674
min      370.000000  6.380000e+02     1.000000     0.000000
0.000000
25%     1460.000000  5.000000e+03     1.000000     0.000000
0.000000
50%     1970.000000  7.680000e+03     1.500000     0.000000
0.000000
75%     2610.000000  1.097800e+04     2.000000     0.000000
0.000000
max    13540.000000  1.074218e+06     3.500000     1.000000
4.000000
std      955.949708  3.596408e+04     0.538531     0.080932
0.765373

         condition   sqft_above  sqft_basement     yr_built
yr_renovated
count  4551.000000  4551.000000    4551.000000  4551.000000
4551.000000
mean      3.449352  1822.221710     310.150516  1970.795649
808.564052
min       1.000000   370.000000       0.000000  1900.000000
0.000000
25%       3.000000  1190.000000       0.000000  1951.000000
0.000000
50%       3.000000  1590.000000       0.000000  1976.000000
0.000000
75%       4.000000  2300.000000     600.000000  1997.000000
1999.000000
max       5.000000  9410.000000    4820.000000  2014.000000
2014.000000
std       0.675160   854.452888     461.987629    29.760073
979.421487
```

```
ax = df.hist(figsize=(14, 14), layout=(5, 3))
plt.title('Rozkład wartości poszczególnych cech')
```

```
Text(0.5, 1.0, 'Rozkład wartości poszczególnych cech')
```

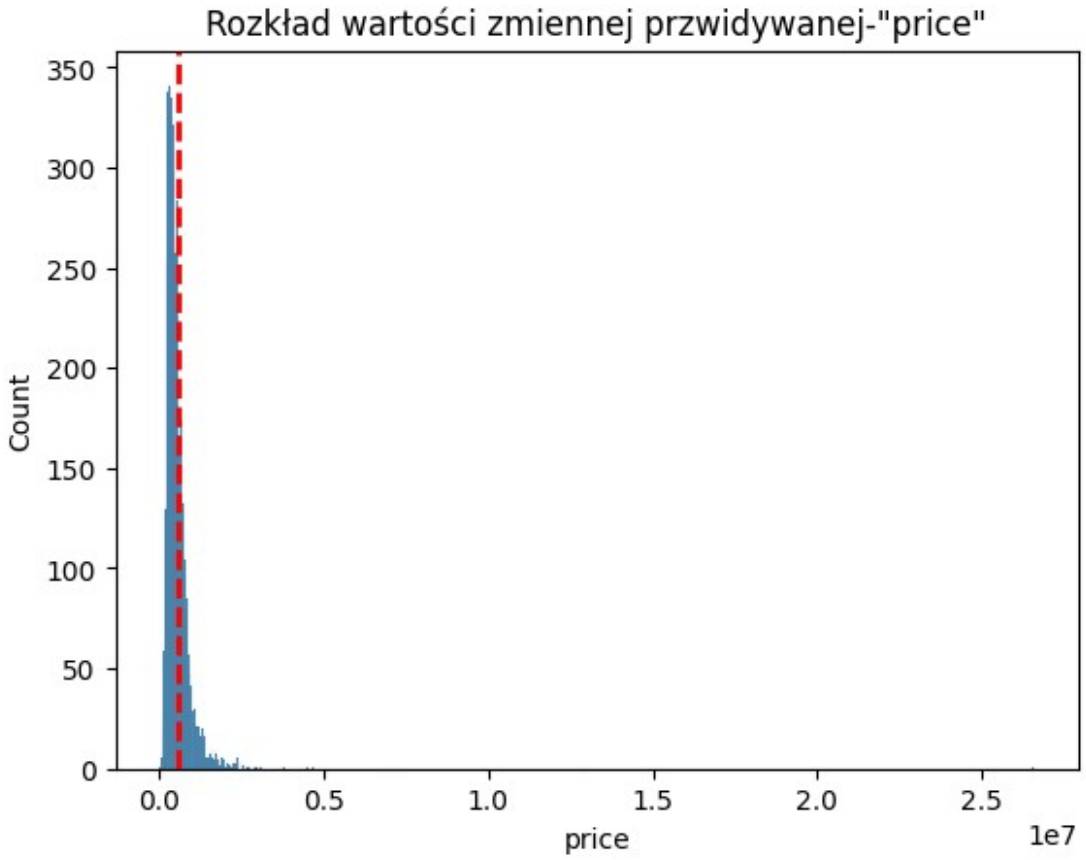


Z powodu istniejących wartości odstających, a więc też dużej rozbieżności wartości, standardowy wykres rozkładu wartości zmiennej price jest mało czytelny. Z tego powodu w na następnym wykresie cecha ta jest przedstawiona na skali logarytmicznej. Dzięki temu widoczne jest, że jej rozkład wartości zlogarytmowanych przypomina rozkład normalny.

```
import seaborn as sns

sns.histplot(df["price"])
plt.axvline(x=df['price'].mean(), color="red", linestyle='--',
```

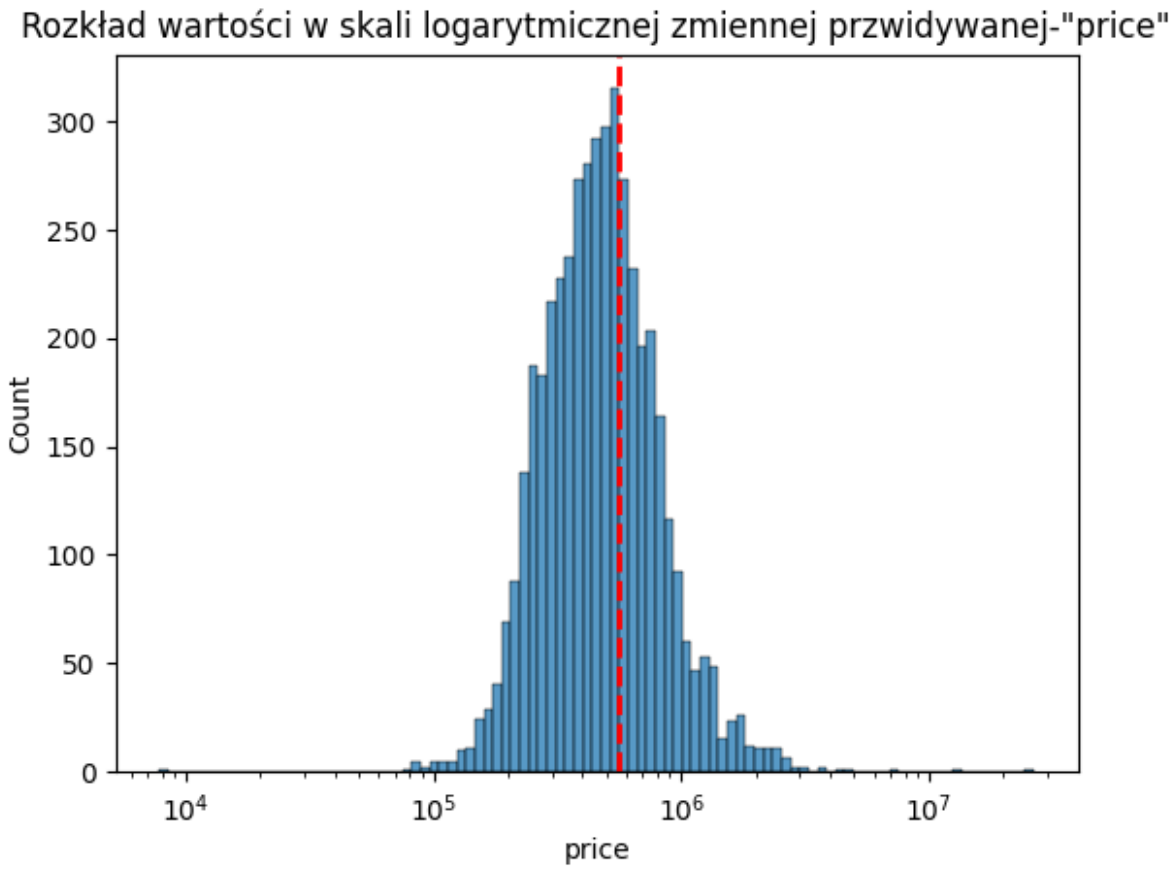
```
linewidth=2)
plt.title('Rozkład wartości zmiennej przwidywanej-"price"')
```

```
Text(0.5, 1.0, 'Rozkład wartości zmiennej przwidywanej-"price"')
```




```
sns.histplot(df["price"], log_scale=True)
plt.axvline(x=df['price'].mean(), color="red", linestyle='--',
linewidth=2)
plt.title('Rozkład wartości w skali logarytmicznej zmiennej
przewidywanej-"price"')
```

```
Text(0.5, 1.0, 'Rozkład wartości w skali logarytmicznej zmiennej
przwidywanej-"price"')
```

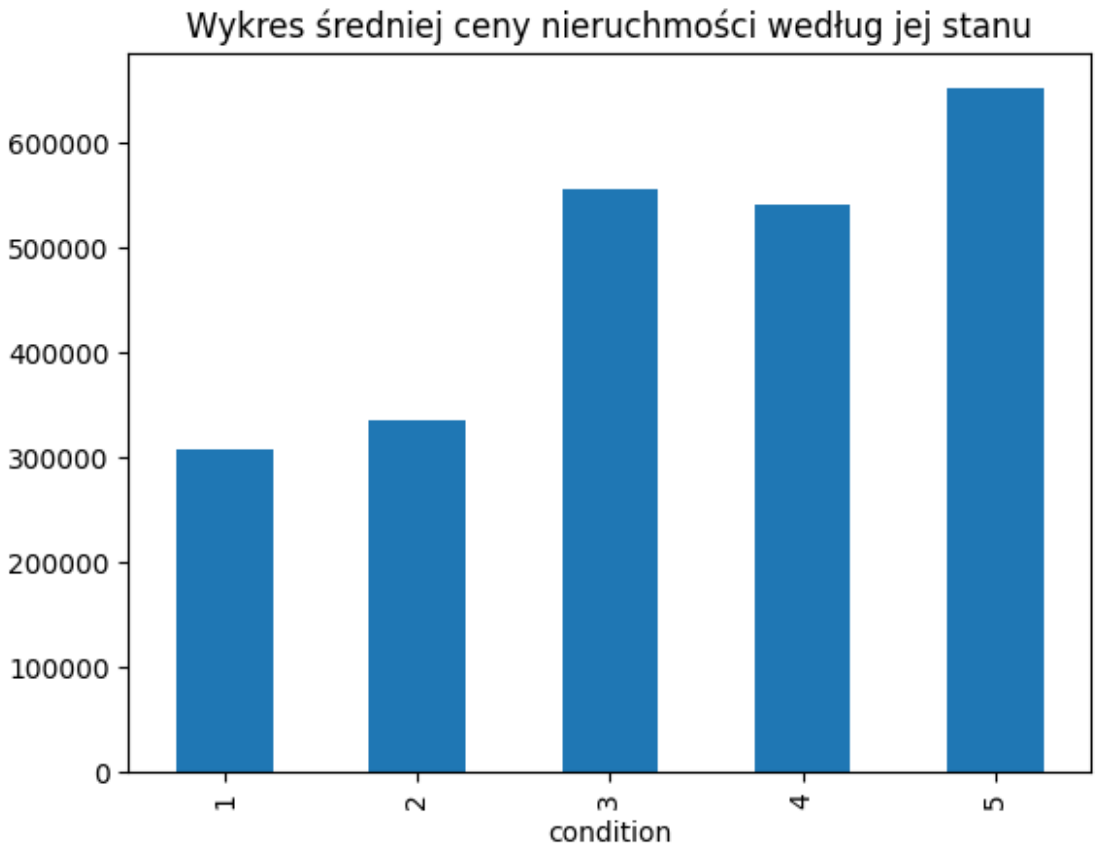


Grupowanie

Wykres przedstawiający średnie ceny nieruchomości w stosunku do jego stanu jest zgodny z intuicją. Naturalne jest, że mieszkania o dobrym stanie są droższe, od mieszkań w słabym stanie. Zaskakujące jest jednak to, że średnia cena nieruchomości przy jego stanie wycenianym na 3 i 4 jest bardzo podobna. Najprawdopodobniej wynika to z faktu, że inne cechy również mają znaczący wpływ na wycenę mieszkania.

```
df.groupby("condition")["price"].mean().plot.bar()
plt.title("Wykres średniej ceny nieruchomości według jej stanu")
```

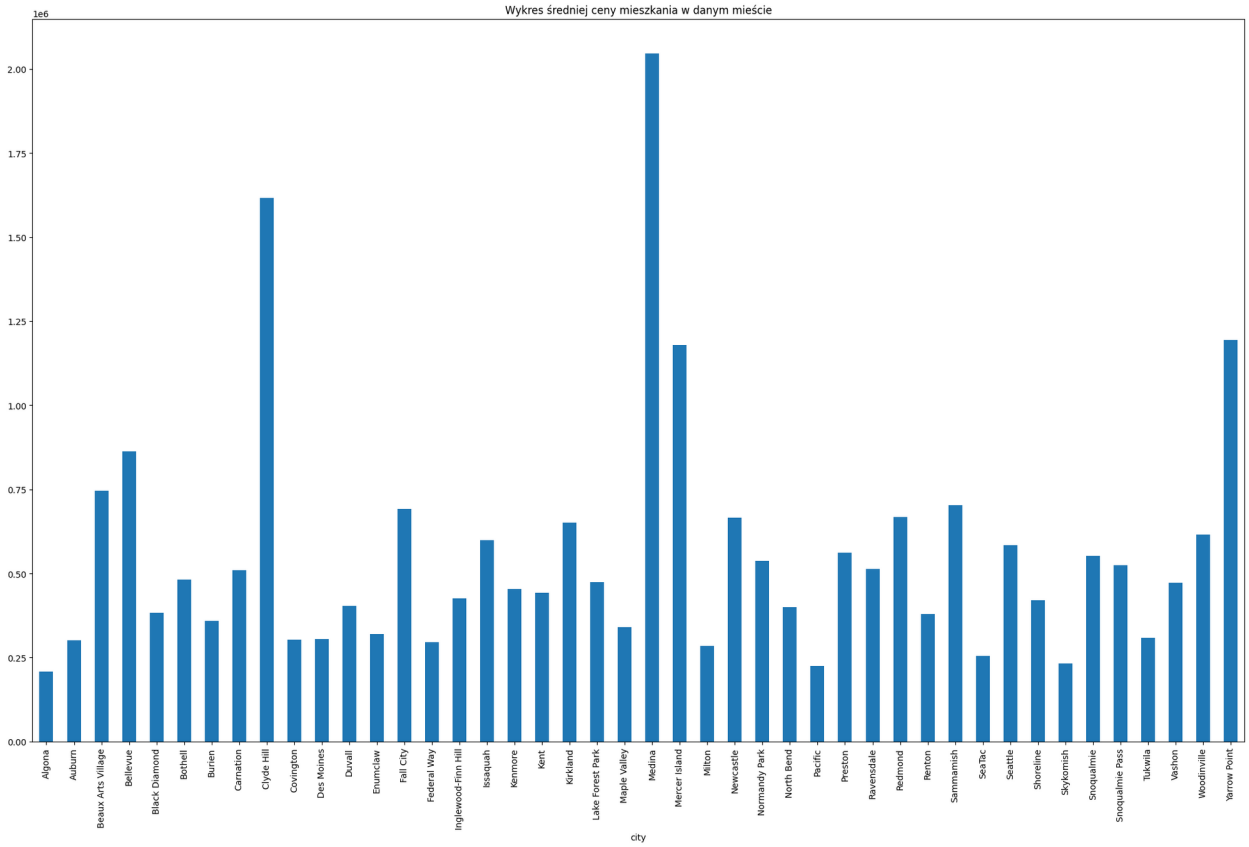
```
Text(0.5, 1.0, 'Wykres średniej ceny nieruchmości według jej stanu')
```



Poniższy wykres prezentuje średnią cene nieruchomości w danym mieście. Mastami z najwyższą średnią cen nieruchomości kolejno są: Medina, Clyde Hill, Mercer Island, oraz Yarrow Point. Z kolei najtańsze miasta to: Algona, Pacyfic, Seatac, Skykomish.

```
plt.figure(figsize=(25,15))
df.groupby("city")["price"].mean().plot.bar(x='city', y='price')
plt.title("Wykres średniej ceny mieszkania w danym mieście")
```

```
Text(0.5, 1.0, 'Wykres średniej ceny mieszkania w danym mieście')
```



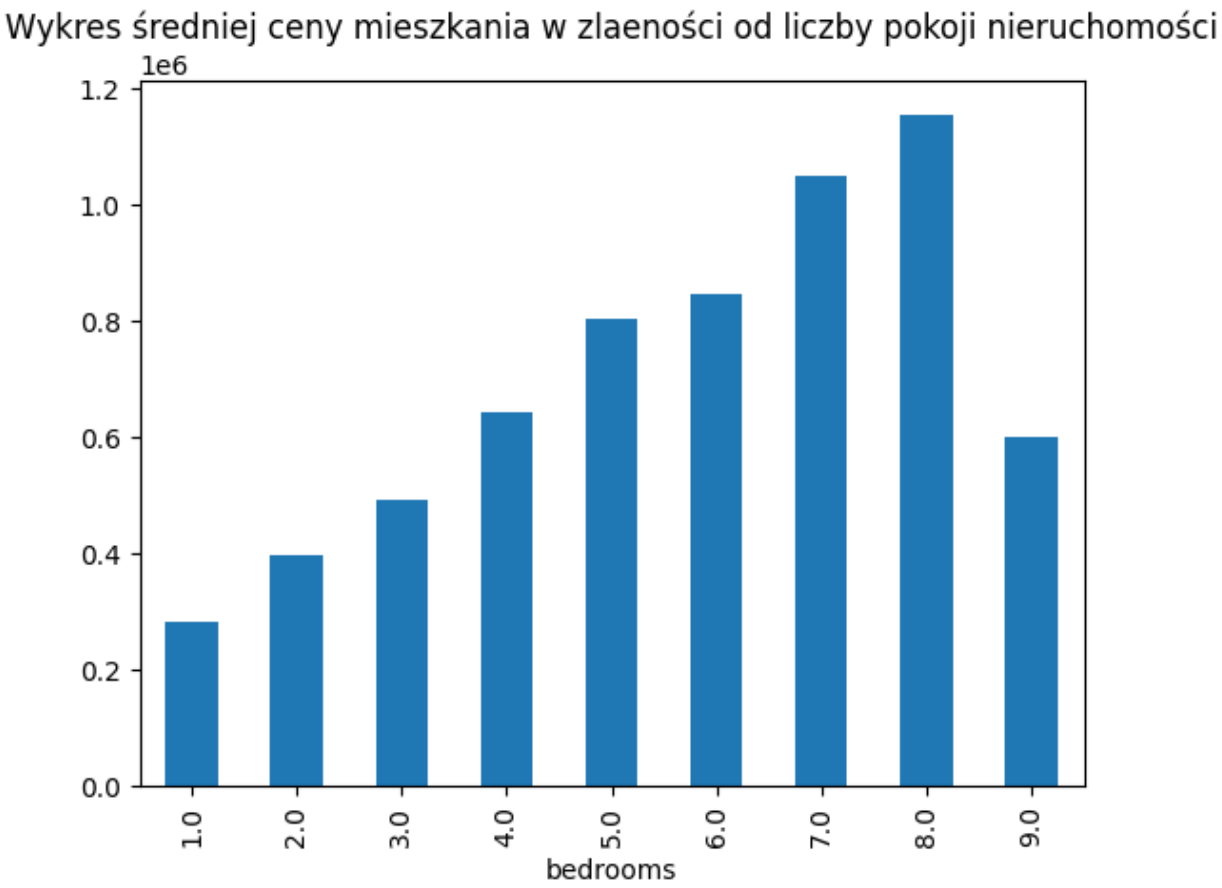
Wszystkie przykłady pochodzą z jedngo kraju - USA, dla tego cecha ta została usunięta.

Dodatkowo W danych znajdują się dwa przykłady, z cechami "bedrooms" i "bathrooms" ustawionymi na 0, i nierealną, outlaierową ceną. Autor zdecydował się na usunięcie tych przypadków z danych.

```
df = df.drop(columns=["country"])

df = df[df["bedrooms"] != 0]
```

Poniżej znajduje się wykres średniej ceny mieszkania w zależności od liczby pokoi nieruchomości. Zgodnie z intuicją czym większa liczba pokoi w mieszkaniu, tym średnia cena za mieszkanie jest większa. Zależność ta pasuje do liczby wartości pokoji od 1 do 8, jednak dla liczby pokoi równej 9, cena jest porównywalna z ceną dla 4 pokojowej nieruchomości.

```
df.groupby("bedrooms")["price"].mean().plot.bar(x='bedrooms',
y='price')
plt.title("Wykres średniej ceny mieszkania w zależności od liczby
pokoi nieruchomości")

Text(0.5, 1.0, 'Wykres średniej ceny mieszkania w zlaeności od liczby
pokoji nieruchomości')
```



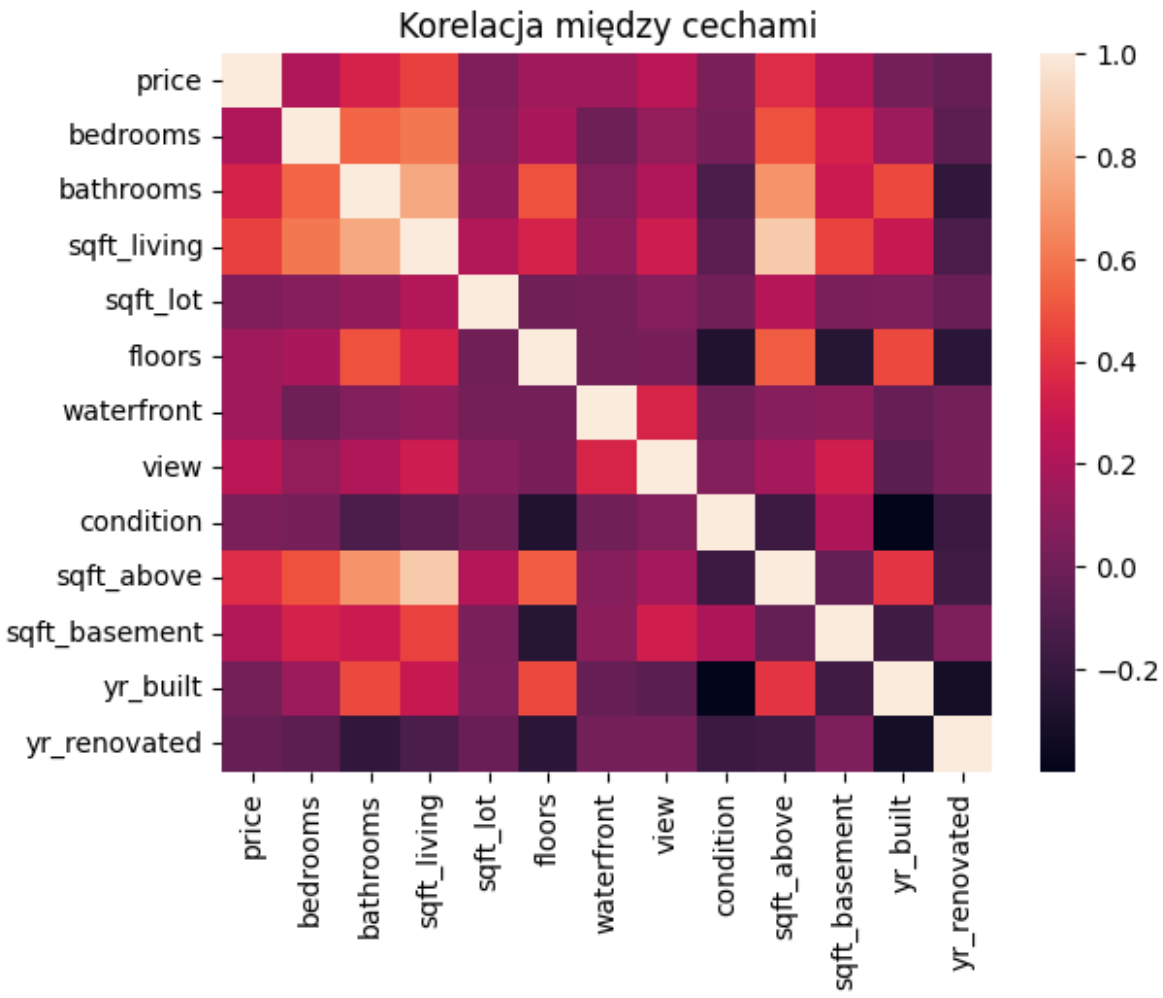
Ponizszy wykres przedstawia korelację pomiędzy cechami. Cechy o najwyższej korelacji to "sqft_living" i "sqft_above". Jest tak z powodu tego, że wielkość części mieszkalnej nieruchomości jest podzbiorem wielkości całkowitej mieszkania. (do wielkości całkowitej liczy się jeszcze strych itd.)

Stosunkowo wysoką korelację mają również wyżej wymienione cechy z cechą "bathrooms". Tę korelację również da się wyjaśnić w prosty sposób. Najczęsciej czym nieruchomość jest większa tym ma większą liczbę łazienek.

Mimo wszystko najbardziej interesujące dla nas jest korelacja z cechą którą staramy się przewidzieć. Potencjalnie taka cecha jest ważna w predykcji. W zbiorze danych cechą o najwyższej korelacji z cechą "price" jest "sqft_living". Korelacja ta wynosi około 0.5.

```
sns.heatmap(df.select_dtypes(include=['number']).corr())
plt.title("Korelacja między cechami")
```

```
Text(0.5, 1.0, 'Korelacja między cechami')
```



Preprocesing

W ramach preprocesingu cecha "date" będąca dotychczas typem data została zamieniona na typ int. Dokonano tego licząc liczbę dni od najwcześniejszej daty.

Dodatkowo wszystkie wartości numeryczne zostały przeskalowane za pomocą MinMaxScalera.

Wartości nienumeryczne ('street', 'city', 'statezip') zostały zakodowane za pomocą "one-hot encoding", który tworzy kolumny binarne (0 lub 1) dla każdej unikalnej wartości cechy wskazując, czy dana nazwa jest obecna w obserwacji.

Następnie dokonano podziału danych na zbiór testowy i treningowy w proporcji 70-30.

```
df["date"] = (df['date'] - df['date'].min()).dt.days
df["date"] = pd.to_numeric(df["date"])

num_cols = [col for col in df.columns if df[col].dtype in
['float64','int64','int32']]
```

```
cat_cols = [col for col in df.columns if df[col].dtype not in
['float64','int64','int32']]

from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

df[num_cols] = MinMaxScaler().fit_transform(df[num_cols])

encoder = OneHotEncoder(sparse=False,
handle_unknown='ignore').fit(df[cat_cols])
encoded_cols = list(encoder.get_feature_names_out(cat_cols))
df[encoded_cols] = encoder.transform(df[cat_cols])

df = df.drop(cat_cols, axis=1)

from sklearn.model_selection import train_test_split

X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

Uczenie modeli

Autor zdecydował się na sprawdzenie wyników następujących modeli regresji z domyślnymi parametrami wywołania:

- SVR - model regresji opartym na maszynach wektorów nośnych (SVM). Działa na zasadzie znajdowania hiperpłaszczyzny optymalnego dopasowania, która oddziela dane na dwie strony.
- XGBRegressor - implementacja XGBoost dla problemów regresji. Jest oparty na drzewach decyzyjnych.
- BayesianRidge - opiera się na podejściu bayesowskim do regresji.
- LinearRegression - jeden z najprostszych modeli regresji. Zakłada liniową zależność między cechami, a wartością docelową.
- RandomForestRegressor - tworzy wiele drzew decyzyjnych, a wynik regresji jest uśredniany z wyników wszystkich drzew

```
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error

models = {
    'SVR':SVR(),
    'XGBRegressor':XGBRegressor(),
```

```
    'BayesianRidge':BayesianRidge(),
    'LinearRegression':LinearRegression(),
    'RandomForestRegressor':RandomForestRegressor()
}

model_names = []
model_r2_scores = []
model_maes = []
model_mses = []

for name,model in models.items():
    a = model.fit(X_train,y_train)
    predicted = a.predict(X_test)

    r2 = r2_score(y_test, predicted)
    mae = mean_absolute_error(y_test, predicted)
    mse = mean_squared_error(y_test, predicted)

    model_r2_scores.append(r2)
    model_maes.append(mae)
    model_mses.append(mse)
    model_names.append(name)

df_results = pd.DataFrame({
    'Model': model_names,
    'R-squared': model_r2_scores,
    'MAE': model_maes,
    'MSE': model_mses
}).sort_values(by='MSE', ascending=False)
```

Wykorzystane miary

W zadaniu zdecydowano się na wykorzystanie następujących miar używanych w ocenie modelu regresji:

- Mean Squared Error(MSE) - oblicza średnią wartość kwadratów różnic między rzeczywistymi danymi, a prognozami modelu. Im niższa wartość MSE, tym lepsza jakość modelu. MSE jest szczególnie wrażliwe na wartości odstające.
- Mean Absolute Error (MAE) - MAE mierzy średnią wartość bezwzględnego błędu między rzeczywistymi danymi a prognozami modelu. MAE jest mniej wrażliwe na wartości odstające niż MSE. Im niższa wartość MAE, tym lepsza jakość modelu. MAE jest przydatne, gdy nie chcemy, aby wartości odstające znacząco wpływały na ocenę modelu.
- R-squared (R^2) - mierzy odsetek zmienności w danych, który jest wyjaśniany przez model. Jej wartość mieści się w zakresie od 0 do 1. Im wyższa wartość R-squared, tym lepsza jakość modelu. R-squared jest przydatne, gdy chcemy ocenić, jak dobrze model ogólnie dopasowuje się do danych i ile zmienności jest wyjaśniane.

Poniżej widoczna jest tabela z wynikami oraz wykres miar MSE i MAE dla wszystkich użytych modeli. Wykresy przedstawiające miary ukazują wyniki tylko trzech najlepszych klasyfikatorów, ponieważ słabe wyniki klasyfikatorów regresji liniowej oraz SVR zaburzały czytelność wykresu.

Najlepsze wyniki otrzymały klasyfikatory: XGBRegressor, RandomForest oraz BayesianRidge. Warto zauważyć, że pierwsze dwa wymienione klasyfikatory są oparte na drzewach. Ciekawe jest, że także stosunkowo prosty klasyfikator - BaysianRidge również uzyskał zadowalające wyniki.

Klasyfikatory LinearRegression oraz SVR poradziły sobie bardzo słabo. Słaby wynik regresji liniowej nie jest zaskoczeniem, ponieważ jest ona w stanie przewidywać tylko zależności liniowe. Najprawdopodobniej aby uzyskać lepszy rezultat dla SVR trzeba sprawdzić inne funkcje jądrowe.

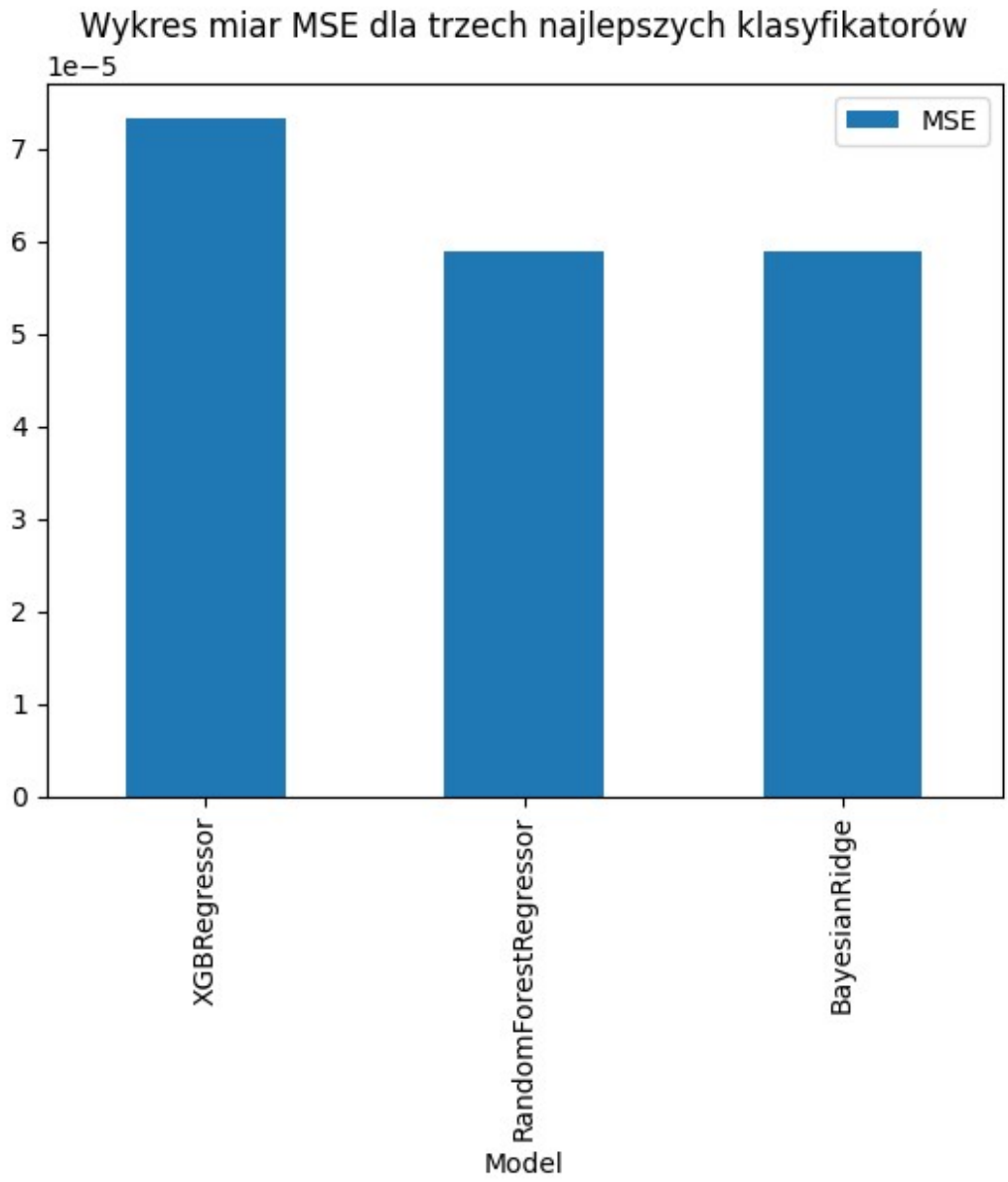
W dalszej części autor zdecydował się na znalezienie najlepszych parametrów dla klasyfikatora RandomForest, ponieważ wypadł on bardzo dobrze w początkowej fazie eksperymentów oraz jest sosunkowo łatwy w wizualizacji.

```
df_results
```

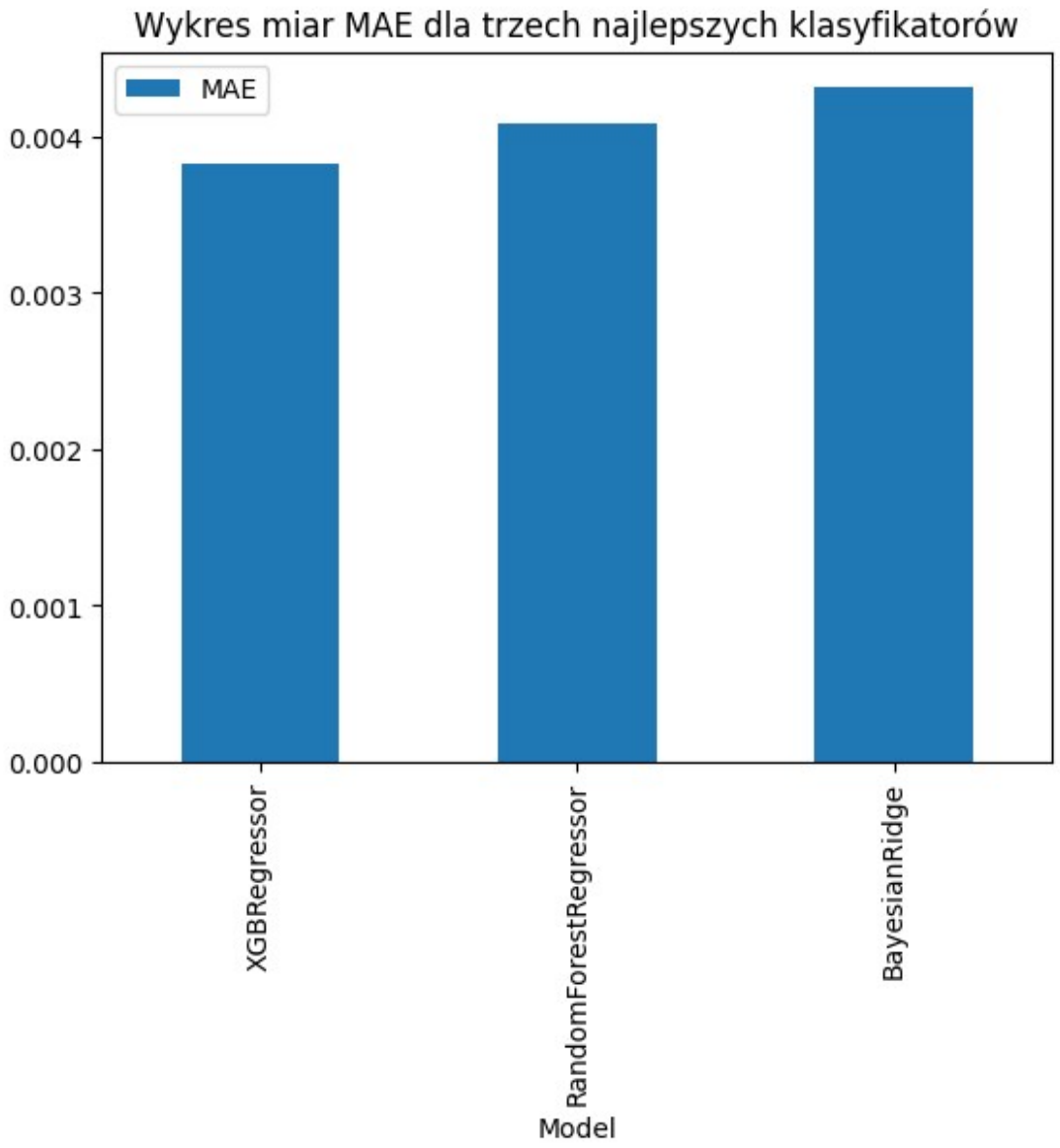
	Model	R-squared	MAE	MSE
3	LinearRegression	-1.646194e+21	1.436639e+08	3.207651e+17
0	SVR	-3.398031e+01	8.130698e-02	6.816002e-03
1	XGBRegressor	6.234376e-01	3.825781e-03	7.337414e-05
4	RandomForestRegressor	6.977069e-01	4.088511e-03	5.890257e-05
2	BayesianRidge	6.981417e-01	4.323528e-03	5.881786e-05

```
df_results[(df_results["Model"] != "LinearRegression") &
(df_results["Model"] != "SVR")].plot.bar(y=["MSE"], x ="Model")
plt.title("Wykres miar MSE dla trzech najlepszych klasyfikatorów")
```

```
Text(0.5, 1.0, 'Wykres miar MSE dla trzech najlepszych
klasyfikatorów')
```

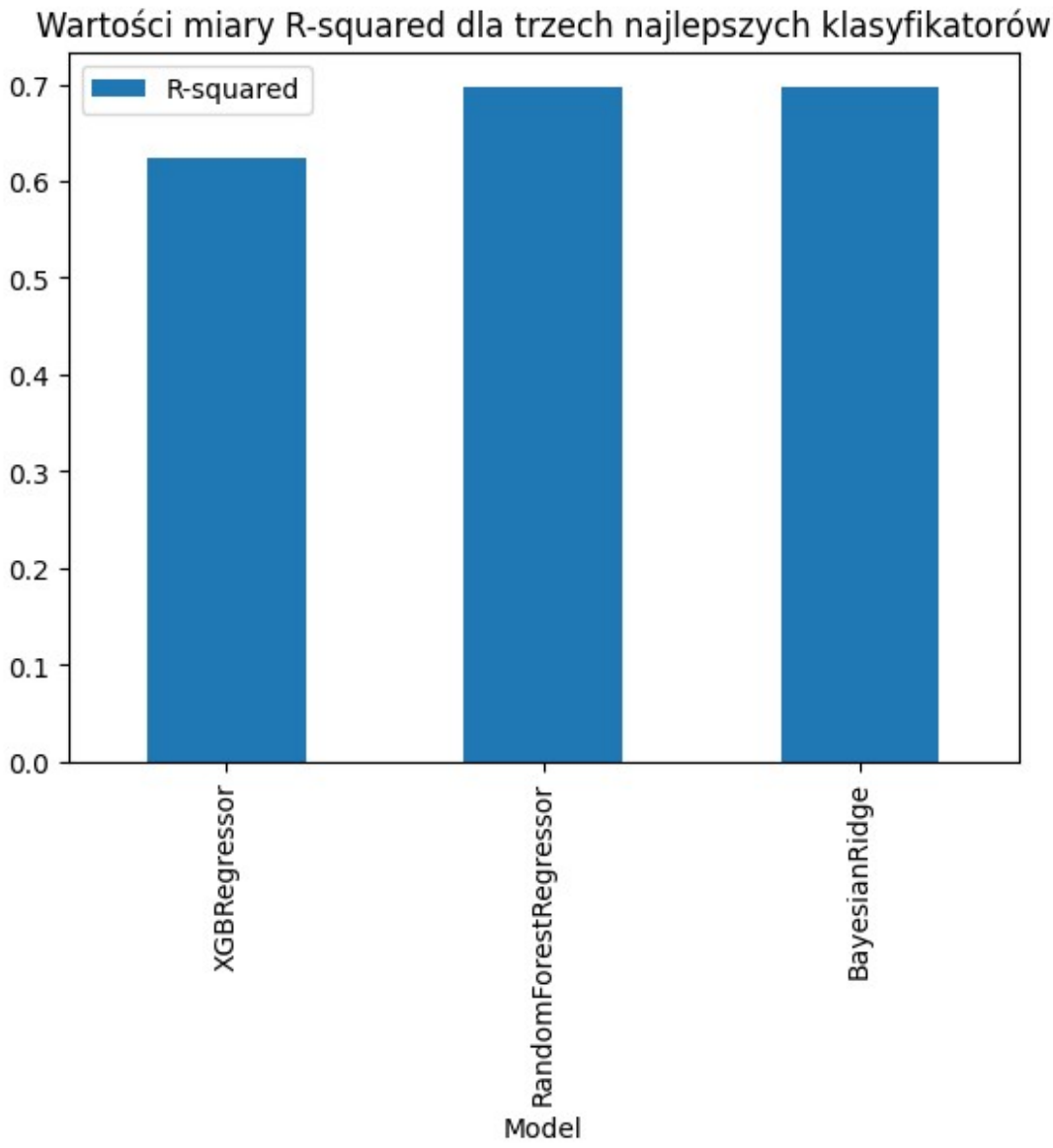



```
df_results[(df_results["Model"] != "LinearRegression") &
(df_results["Model"] != "SVR")].plot.bar(y=["MAE"], x ="Model")
plt.title("Wykres miar MAE dla trzech najlepszych klasyfikatorów")

Text(0.5, 1.0, 'Wykres miar MAE dla trzech najlepszych
klasyfikatorów')
```



Miara R-squared dla klasyfikatora LinearRegression oraz SVR przyjęła ujemne wartości. Oznacza to, że klasyfikatory te są bardzo słabe. Najlepszy wynik na tej mierze uzyskał random forest, uzyskując wartość 0.7. Porównywalnie do niego wypadł BayesianRidge, a nienzacznie niżej znajduje się XGBoost. Wyniki tych trzech klasyfikatorów są zadowalające.

```
df_results[(df_results["Model"] != "LinearRegression") &
(df_results["Model"] != "SVR")].plot.bar(x="Model", y="R-squared")
plt.title("Wartości miary R-squared dla trzech najlepszych
klasyfikatorów")

Text(0.5, 1.0, 'Wartości miary R-squared dla trzech najlepszych
klasyfikatorów')
```



Do poszukiwanie najlepszych parametrów dla klasyfikatora random forest został użyta siatka przeszukiwań z walidacją krzyżową z parametrem podziału ustawionym na pięć częsci. Najlepsze znalelzione parametry są następujące: {'max_depth': None, 'n_estimators': 150}. Uzyskują one wynik MSE równy $4.9 * 10^{-5}$.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
}

rf_regressor = RandomForestRegressor(random_state=42)
```

```
grid_search = GridSearchCV(estimator=rf_regressor,
param_grid=param_grid,
                           scoring='neg_mean_squared_error', cv=5)

grid_search.fit(X, y)

best_params = grid_search.best_params_

best_model = grid_search.best_estimator_

y_pred_val = best_model.predict(X)
mse_val =mean_squared_error(y, y_pred_val)

print("Najlepsze parametry:", best_params)
print("Najlepszy MSE:", mse_val)
```

```
Najlepsze parametry: {'max_depth': None, 'n_estimators': 150}
Najlepszy MSE: 4.9065553300580366e-05
```

```
feature_importances = best_model.feature_importances_
feature_names = X.columns
sorted_indices = np.argsort(feature_importances)[::-1]

n_top_features = 5
top_features_indices = sorted_indices[:n_top_features]
top_feature_names = [feature_names[i] for i in top_features_indices]
top_feature_importances = [feature_importances[i] for i in
top_features_indices]

print("Najważniejsze cechy:")
for feature_name, feature_importance in zip(top_feature_names,
top_feature_importances):
    print(f"{feature_name}: {feature_importance:.4f}")
```

```
Najważniejsze cechy:
street_12005 SE 219th Ct: 0.3309
sqft_living: 0.2767
street_5426 40th Ave W: 0.1338
yr_built: 0.0218
sqft_above: 0.0179
```

Jedną z zalet korzystania z klasyfikatorów drzewiastych jest ich wyjaśnialność. W łatwy sposób można znaleźć najistotniejsze cechy, których używa model. W przypadku powyższego modelu cechy te to: "street_12005 SE 219th Ct", "sqft_living", "street_5426 40th Ave W", "yr_built", "sqft_above". Dodatkowo drzewo można w sprawny sposób przedstawić na wykresie. Ilustruje to poniższy przykład. Aby wykres był czytelny w małej skali autor wytrenował nowy klasyfikator ustawiając parametr max_depth na liczbę 3.

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

```
selected_tree = RandomForestRegressor(max_depth=3,
min_samples_leaf=10).fit(X, y).estimators_[0]

plt.figure(figsize=(15, 10))
plot_tree(selected_tree, filled=True, feature_names=X.columns,
fontsize=10)
plt.title('Uproszczone drzewo regresji z RandomForestRegressor')
plt.show()
```