



**Politechnika
Śląska**

Systemy sztucznej inteligencji

Kierunek: Informatyka

Członkowie zespołu:

Michał Bober

Bartosz Bugla

Kamil Grabowski

Spis treści

1	Wprowadzenie	2
1.1	Opis problemu	2
1.2	Instrukcja obsługi	2
1.3	Użyte biblioteki	2
2	Realizacja zadania	3
2.1	Analiza danych	3
2.2	Przygotowanie danych	7
2.3	Nauka modelu	9
3	Wybór Parametrów i ocena modelu	12
3.1	Metryki oceny modelu	12
3.2	Wybór Parametrów	13
4	Test innych modeli	16
4.1	Drzewo decyzyjne	16
4.2	Random Forest	16
4.3	Regresja Liniowa	16
5	Wnioski	17

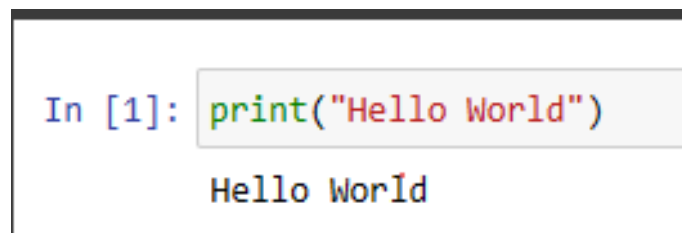
1 Wprowadzenie

1.1 Opis problemu

Celem zadania jest utworzenie modelu, który będzie w stanie przewidywać ceny mieszkań. Zbiór danych jakiego użyjemy do analizy danych i nauki modelu to zbiór "House Sales in King County, USA" ze strony Kaggle.com. Wybraliśmy go ze względu na sporą ilość rekordów ($> 20\,000$) oraz stosunkowo dokładny opis mieszkań 18 kolumn.

1.2 Instrukcja obsługi

Aby uruchomić nasz model, będzie potrzebne IDE, które umożliwi uruchamianie plików typu jupyter notebook np. Jupyter Notebook. Zalecane jest również użycie środowiska Anaconda by zapewnić wszystkie potrzebne biblioteki oraz poprawne działanie naszego notebooka. Notebook zawiera komórki, które możemy uruchomić za pomocą shift+enter. Poniżej uruchomionej komórki pokaże się wartość, zwrócona przez wykonanie kodu.

A screenshot of a Jupyter Notebook interface. It shows a single code cell with the prompt 'In [1]:' followed by the code 'print("Hello World")'. Below the code, the output 'Hello World' is displayed. The code is color-coded: 'In [1]:' is blue, 'print' is green, and the string is red. The output is also color-coded, with 'Hello' in blue and 'World' in red.

```
In [1]: print("Hello World")  
Hello World
```

Rysunek 1: Przykład uruchomionej komórki w notebooku

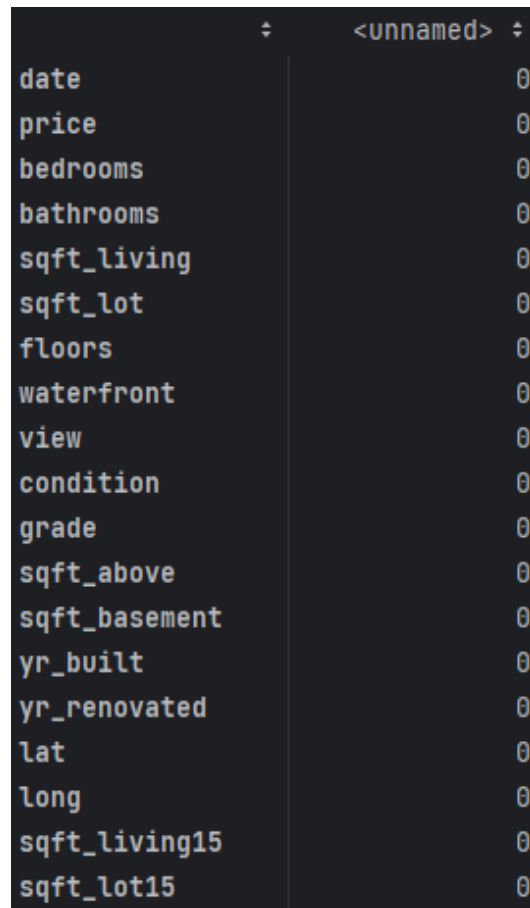
1.3 Użyte biblioteki

- numpy - obliczenia numeryczne
- pandas - manipulacja danymi
- matplotlib - wizualizacja danych
- seaborn - wizualizacja statystyczna
- scikit-learn - biblioteka do uczenia maszynowego

2 Realizacja zadania

2.1 Analiza danych

Nie brakuje żadnych danych tzn. każdy rekord ma informacje o każdej z cech mieszkania



The image shows a screenshot of a data table with a dark background. The table has two columns. The first column lists 20 features: date, price, bedrooms, bathrooms, sqft_living, sqft_lot, floors, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_built, yr_renovated, lat, long, sqft_living15, and sqft_lot15. The second column, titled '<unnamed>', contains the value '0' for every row. The table is presented in a monospaced font.

	<unnamed>
date	0
price	0
bedrooms	0
bathrooms	0
sqft_living	0
sqft_lot	0
floors	0
waterfront	0
view	0
condition	0
grade	0
sqft_above	0
sqft_basement	0
yr_built	0
yr_renovated	0
lat	0
long	0
sqft_living15	0
sqft_lot15	0

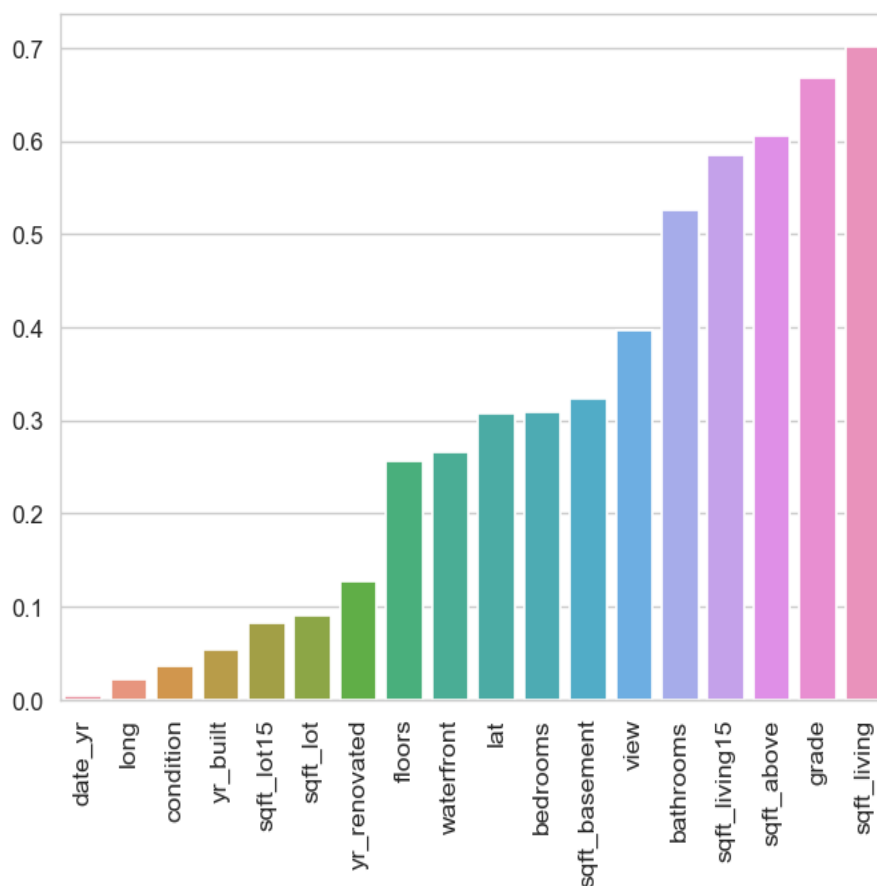
Rysunek 2: Brakujące dane dla konkretnych kolumn

Cechy o największym współczynniku korelacji z kolumną price czyli kolumną, którą chcemy przewidywać to:

- rozmiar mieszkania
- ocena mieszkania według eksperta
- ilość łazienek

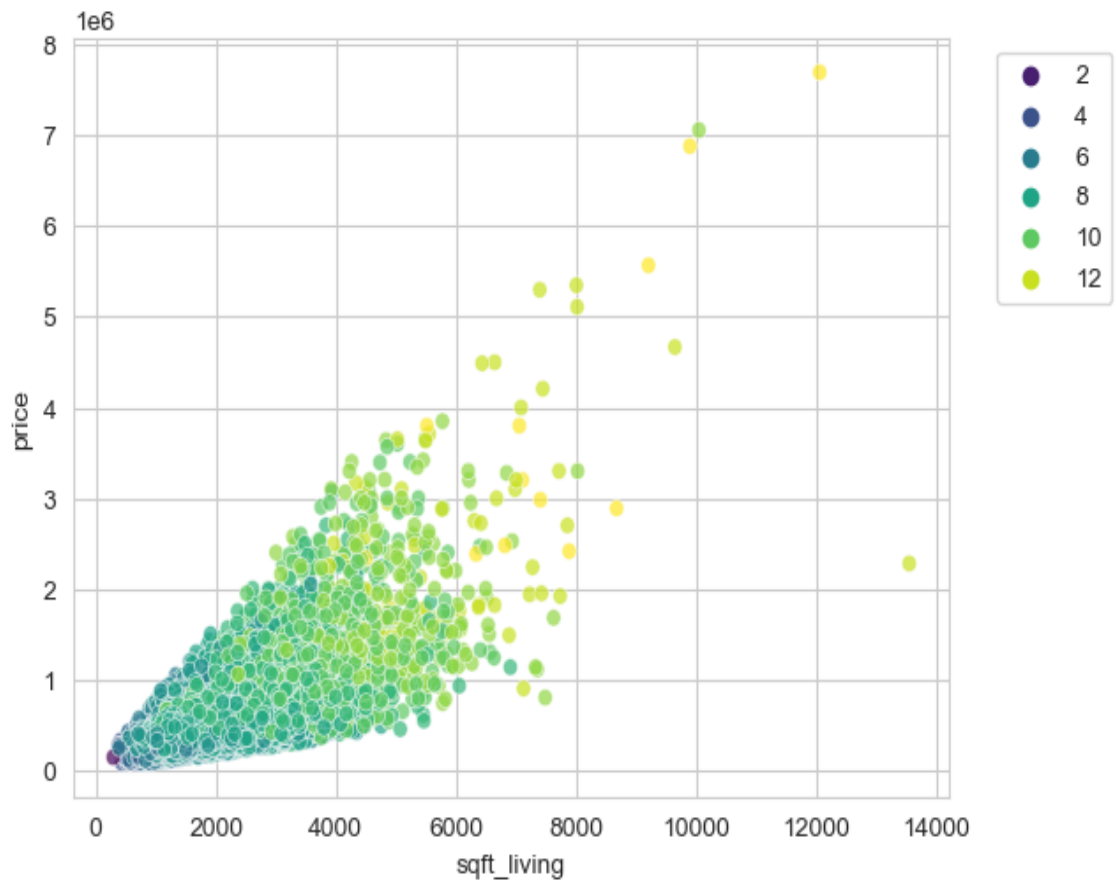
Natomiast cechy, które najpewniej nie wpływają na cenę mieszkania to:

- data zakupu (spowodowane tym że dane są z zakresu tylko dwóch lat 2014,2015)
- długość geograficzna położenia mieszkania
- stan mieszkania (jest to cecha, która powinna wpływać na cenę i dlatego przyjrzymy się jej dokładniej później)



Rysunek 3: Współczynnik korelacji z kolumną price dla pozostałych kolumn

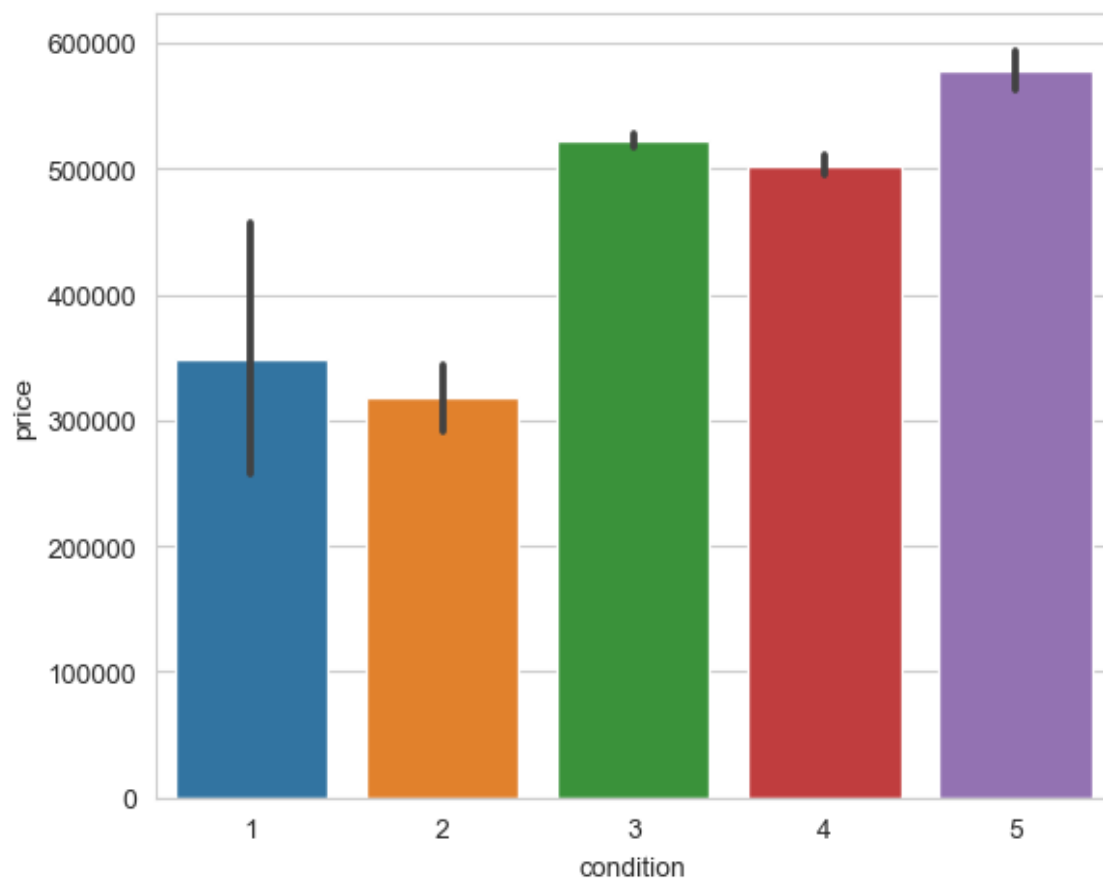
Wraz ze wzrostem ceny wzrasta również rozmiar mieszkania oraz ocena mieszkania



Rysunek 4: Zależność rozmiaru mieszkania od ceny i oceny ogólnej mieszkania

Widać, również że występują rekordy, które zdecydowanie odstają od reszty, co mogłoby zaburzyć ogólną naukę modelu. By tego uniknąć te rekordy zostały usunięte.

Mimo, że współczynnik korelacji kolumn cena i stan mieszkania był bardzo niski to widać, że jeżeli stan jest ≥ 3 to średnia cena takich mieszkań zdecydowanie wzrasta



Rysunek 5: Średnia cena mieszkań według oceny eksperta stanu mieszkania

2.2 Przygotowanie danych

Standaryzacja danych

By wszystkie dane miały taki sam wpływ na odległość w algorytmie knn, znormalizujemy je do przedziału (0,1) by to uzyskać można zastosować następujący wzór na każdej wartości w zbiorze danych.

$$\hat{x} = \frac{x - \mu}{\sigma}$$

gdzie:

- \hat{x} oznacza standaryzowany wynik,
- x to oryginalna wartość,
- μ to średnia wartość dla wszystkich obserwacji,
- σ to odchylenie standardowe dla wszystkich obserwacji.

Funkcja do standaryzacji danych, jako argument przyjmuje cały zbiór danych i kolumnę do przewidywania, której nie będziemy brać pod uwagę

```
1 def standardize(df, target):
2     for name in df.drop(target, axis=1).columns:
3         df[name] = (df[name] - df[name].min()) / (df[
4             name].max() - df[name].min())
5     return df
```

Tasowanie danych

w celu uniknięcia wpływu porządku danych na wyniki uczenia oraz poprawy zdolności modelu do generalizacji na nowe dane, zbiór został losowo potasowany. Tasowanie również zapewnia większą różnorodność danych i zmniejsza ryzyko wprowadzenia uprzedzeń, co przyczynia się do lepszej jakości i uczciwości modelu.

Funkcja do przetasowania danych. Jako argument przyjmuje zbiór danych

```
1 def shuffle(data):
2     length = len(data)
3     for x in range(1, length):
4         i = random.randint(0, length-x)
5         data.iloc[i], data.iloc[-1-x] = data.iloc[-1-x],
6             data.iloc[i]
7     return data
```

Podział na zbiór treningowy i testowy

Podział na zbiór treningowy i testowy umożliwia ocenę skuteczności modelu uczenia maszynowego na danych, które nie były wykorzystane w procesie uczenia. Dzięki temu możemy oszacować, jak dobrze nasz model generalizuje się na nowe, nieznane dane, co pomaga nam ocenić jego skuteczność w praktycznych zastosowaniach. Ponadto, podział na te dwa zbiory pozwala nam monitorować i uniknąć problemu nadmiernego dopasowania (overfittingu), czyli sytuacji, gdy model uczy się na pamięć treningowych danych, ale słabo radzi sobie z nowymi danymi.

Funkcja do podziału na zbiór treningowy i testowy. Jako argument przyjmuje zbiór danych i współczynnik podziału od 0 do 1

```
1 def split(df, size):
2     train_size = int(size * len(df))
3     train_df = df.iloc[:train_size]
4     test_df = df.iloc[train_size:]
5     return train_df, test_df
```

2.3 Nauka modelu

Do nauki modelu użyliśmy algorytmu k-najbliższych sąsiadów. Polega on na znajdowaniu k najbliższych sąsiadów nowego rekordu na podstawie podobieństwa (odległości) między cechami, a następnie przewidywaniu w przypadku regresji wartości na podstawie średniej wartości wcześniej wspomnianych k-sąsiadów.

Metryka odległości

Jako metrykę odległości użyliśmy metryki Minkowskiego.

$$d(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

gdzie:

- $d(x, y)$ oznacza odległość między punktami x i y ,
- x, y to punkty w przestrzeni n -wymiarowej,
- x_i, y_i to współrzędne punktów x i y w i -tej wymiarze,
- p to parametr kontrolujący stopień metryki Minkowskiego.

Funkcja do obliczenia odległości między dwoma rekordami. Jako argumenty przyjmuje rekord x i y oraz stopień metryki

```
1 def distance(x, y, m=1):  
2     dist = (sum([pow(abs(a-b), m) for a, b in zip(x, y)])  
3             ) ** (1/m)  
4     return dist
```

Parametry modelu

Model w trakcie uczenia będzie przyjmował 2 parametry, które mogą wpłynąć na jego dokładność. Są to:

- k - parametr k oznacza ilość szukanych sąsiadów, którzy wpłyną przewidzianą wartość
- p - parametr p oznacza stopień metryki Minkowskiego

Model KNN

Pseudo Kod algorytmu K-najbliższych sąsiadów

Algorithm 1 Algorytm K - najbliższych sąsiadów.

Data: Dane wejściowe $X_{train}, y_{train}, X_{test}$, k - k-sąsiadów, p -stopień metryki Minkowskiego

Result: Predykcje dla zbioru testowego

$predykcje = []$

for *rekord in Xtest* **do**

for *train rekord in Xtrain* **do**

 Oblicz odległość między rekord i train rekord według metryki minkowskiego o stopniu p

end

 Wybierz k - rekordów treningowych o najmniejszej odległości

 Oblicz średnią wartość y_{train} tych sąsiadów Dodaj wartość do $predykcje$

end

 zwróć $predykcje$

Kod modelu w języku Python

```
1 class Knn:
2     # konstruktor klasy modelu
3     def __init__(self, k=3, m=1):
4         self.k = k
5         self.m = m
6         self.y = None
7         self.X_train = None
8         self.y_train = None
9         self.train = None
10
11     # przekazanie do modelu zbioru treningowego
12     def fit(self, train, y=None):
13         if y:
14             self.y = y
15         else:
16             self.y = train.columns[-1]
17             self.X_train = train.drop(self.y, axis=1)
18             self.y_train = train[self.y]
19             self.train = train
20
21     # przewidzenie wartosci dla zbioru testowego
22     def predict(self, test):
23         X_test = test.drop(self.y, axis=1)
24         predictions = []
25         for record in X_test.to_numpy():
26             dist = []
27             for train_record in self.X_train.to_numpy():
28                 dist.append(Utils.distance(record,
29                                             train_record, self.m))
30             args = np.argsort(dist)
31             result = np.mean(self.y_train.iloc[args[:5]])
32             predictions.append(result)
33         return predictions
```

3 Wybór Parametrów i ocena modelu

3.1 Metryki oceny modelu

Średni błąd bezwzględny

Obliczanie średniego błędu bezwzględnego (MAE):

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

gdzie:

MAE oznacza średni błąd bezwzględny,

n to liczba przykładów,

y_i to prawdziwa wartość dla i -tego przykładu,

\hat{y}_i to przewidziana wartość dla i -tego przykładu.

Pierwiastek błędu średniokwadratowego

Obliczanie pierwiastka błędu średniokwadratowego (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

gdzie:

RMSE oznacza pierwiastek błędu średniokwadratowego,

n to liczba przykładów,

y_i to prawdziwa wartość dla i -tego przykładu,

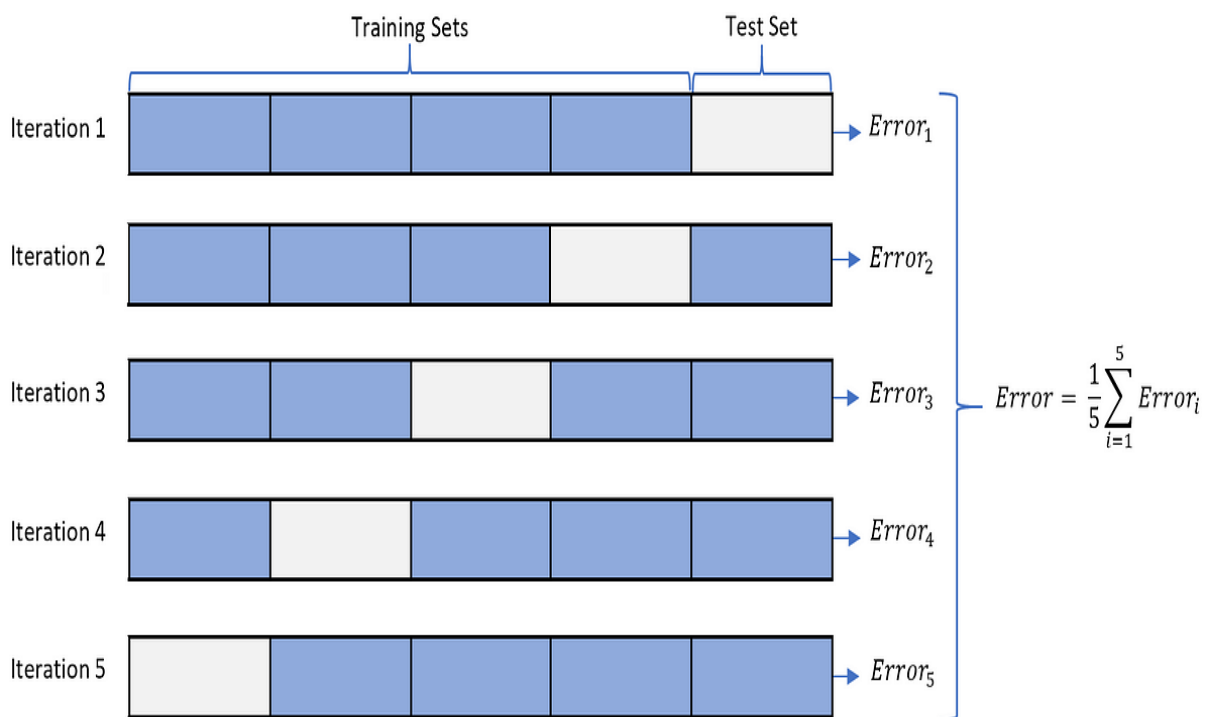
\hat{y}_i to przewidziana wartość dla i -tego przykładu.

3.2 Wybór Parametrów

By znaleźć optymalne parametry dla których model działa najlepiej użyliśmy walidacji krzyżowej

Walidacja Krzyżowa

Walidacja krzyżowa to technika używana w uczeniu maszynowym do oceny skuteczności modelu i optymalizacji hiperparametrów. Polega na podziale dostępnych danych na część treningową i część testową, a następnie wielokrotnym wykorzystaniu obu tych części w procesie treningu i ewaluacji modelu.



Rysunek 6: Zobrazowanie działania walidacji krzyżowej

Funkcja do wykonania walidacji krzyżowej. Jako argument przyjmuje zbiór treningowy, parametry k do sprawdzenia i parametry stopnia metryki minkowskiego

```
1 def cv_knn(df, ks, ms):
2     length = int(len(df)/5)
3     stats = {}
4     for k in ks:
5         for m in ms:
6             errors = []
7             for c in range(5):
8                 fold_start = c*length
9                 fold_end = (c+1)*length
10                test = df.iloc[fold_start:fold_end]
11                train = df.iloc[:fold_start] + df[
12                    fold_end:]
13                knn_model = Knn(k,m)
14                knn_model.fit(train,y='price')
15                pred = knn_model.predict(test)
16                error = knn_model.accuracy(test, pred)
17                errors.append(error)
18                stats[f'k={k},m={m}'] = np.mean(errors)
19     return stats
```

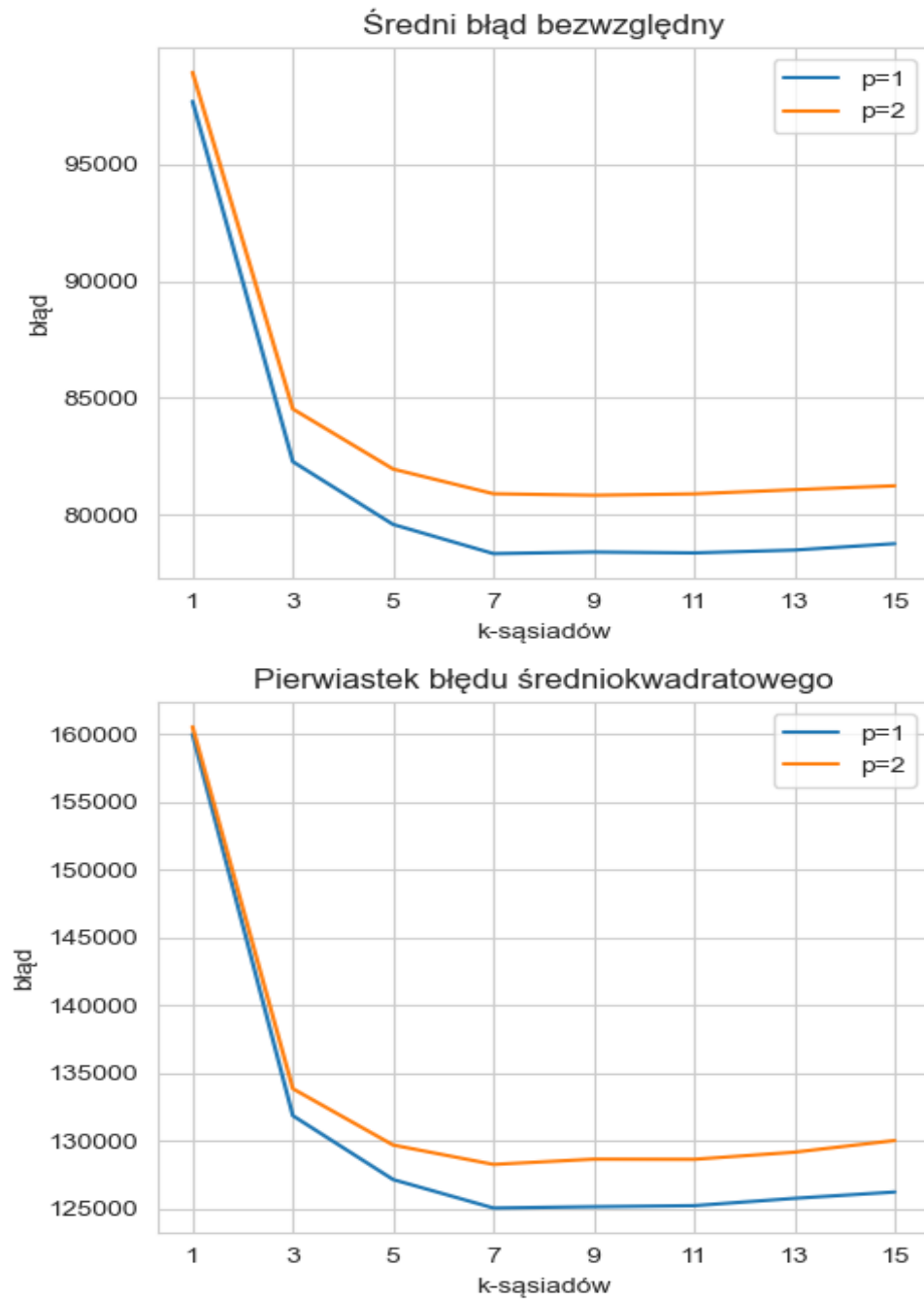
Przeprowadziliśmy walidację krzyżową dla

- $k = [1,3,5,7,9,11,13,15]$
- $p = [1,2]$

Najlepsze wyniki otrzymano dla

- $k = 7$
- $p = 1$

Dokładne wyniki błędów MAE oraz RMSE dla poszczególnych parametrów



Rysunek 7: Błędy po walidacji krzyżowej

4 Test innych modeli

4.1 Drzewo decyzyjne

```
1 from sklearn.tree import DecisionTreeRegressor
2 tree_model = DecisionTreeRegressor()
3 tree_model.fit(X_train, y_train)
4 tree_model.predict(X_test)
5 MAE = mean_absolute_error(y_test, pred)
6 RMSE = np.sqrt(mean_squared_error(y_test, pred))
```

MAE: 89330.41

RMSE: 139102.91

4.2 Random Forest

```
1 from sklearn.ensemble import RandomForestRegressor
2 rand_forest_model = RandomForestRegressor()
3 rand_forest_model.fit(X_train, y_train)
4 rand_forest_model.predict(X_test)
5 MAE = mean_absolute_error(y_test, pred)
6 RMSE = np.sqrt(mean_squared_error(y_test, pred))
```

MAE: 60987.63

RMSE: 99166.18

4.3 Regresja Liniowa

```
1 from sklearn.linear_model import LinearRegression
2 linear_model = LinearRegression()
3 linear_model.fit(X_train, y_train)
4 linear_model.predict(X_test)
5 MAE = mean_absolute_error(y_test, pred)
6 RMSE = np.sqrt(mean_squared_error(y_test, pred))
```

MAE: 105855.72

RMSE: 149260.54

5 Wnioski

model	MAE	RMSE
random_forest	60987.63	99166.18
knn	78325.55	125041.34
drzewo decyzyjne	89330.41	139102.91
regresja liniowa	105855.72	149260.54

Rysunek 8: Wyniki poszczególnych modeli

Jak widać najlepsze wyniki otrzymano dzięki algorytmowi Random Forest. Średnia wartość mieszkań w tym zbiorze wynosiła 512 tysięcy czyli nawet najniższy średni błąd bezwzględny stanowi 12 % średniej ceny. Biorąc to pod uwagę oraz to, że RMSE jest zawsze zdecydowanie większy oznacza to, że nasz model czasami myli się bardzo mocno względem docelowej ceny. Uważamy, że model byłby dobrym sposobem na przybliżenie możliwej ceny mieszkania ale nie jest wystarczająco dobry by stosować go do dokładnego przewidywania ceny za jaką mieszkanie powinno być sprzedane.