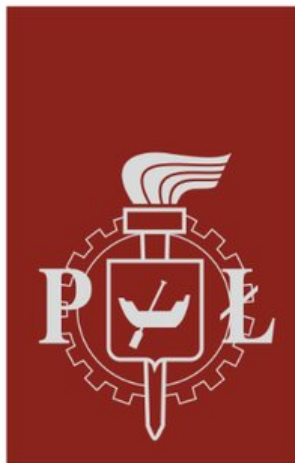


# Politechnika Łódzka

Wydział Elektrotechniki Elektroniki Informatyki i Automatyki



sem, zimowy, r ak. 2024/2025

## **Sprawozdanie z projektu BigData „Predykcja cen samochodów używanych”**



Mateusz Grzybek 240678

Kamil Młynarczyk 240757

19 grudnia 2024

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>3</b>
1.1	Założenia projektowe . . . . .	3
1.2	Komponenty . . . . .	3
1.3	Konteneryzacja . . . . .	3
1.4	Sposób uruchamiania . . . . .	4
1.5	Dane . . . . .	5
<b>2</b>	<b>Diagramy</b>	<b>6</b>
2.1	Diagram przypadków użycia . . . . .	6
2.2	Diagram sekwencji zdarzeń . . . . .	7
<b>3</b>	<b>Aplikacja kliencka</b>	<b>8</b>
3.1	Opis . . . . .	8
3.2	Technologie . . . . .	8
3.3	Widoki aplikacji . . . . .	9
3.3.1	Strona . . . . .	9
3.3.2	Okno z ceną . . . . .	10
3.3.3	Okno z błędem . . . . .	10
<b>4</b>	<b>Komponent pośredniczący</b>	<b>11</b>
4.1	Opis . . . . .	11
4.2	Technologie . . . . .	11
<b>5</b>	<b>Komponent komunikacyjny</b>	<b>12</b>
5.1	Opis . . . . .	12
5.2	Technologie . . . . .	12
<b>6</b>	<b>Przygotowanie danych</b>	<b>13</b>
6.1	Opis . . . . .	13
6.2	Wizualizacja danych . . . . .	13
6.3	Puste pola . . . . .	15
6.4	Zestaw danych z wartościami odstającymi . . . . .	16
6.5	Wykrywanie i usuwanie wartości odstających za pomocą metody IQR . . . . .	16
6.6	Zestaw danych bez wartości odstających . . . . .	17
6.7	Wyodrębnienie znaczących informacji o silniku . . . . .	17
6.8	Faktoryzacja danych . . . . .	18
<b>7</b>	<b>Serwis predykcyjny</b>	<b>20</b>
7.1	Opis . . . . .	20
7.2	Technologie . . . . .	20
7.3	Wybór modelu . . . . .	20
7.4	Testowanie modeli . . . . .	21

7.4.1	Las Losowy . . . . .	21
7.4.2	Drzewo decyzyjne . . . . .	22
7.4.3	Podsumowanie . . . . .	23

# Rozdział 1

## Wstęp

### 1.1 Założenia projektowe

Celem projektu jest zaimplementowanie aplikacji webowej pozwalającej użytkownikowi na predykcję ceny używanego samochodu na podstawie dostarczonego przez niego zestawu cech. Tematyka projektu daje możliwość wykorzystania różnorodnych technologii z dziedziny uczenia maszynowego, rozwoju aplikacji webowych, komunikacji pomiędzy serwisami, architektury oprogramowania oraz gromadzenia i przetwarzania danych. W celu zrealizowania przewidywanych funkcjonalności, aplikacja została podzielona na cztery komponenty, każdy z nich odpowiedzialny za realizację innego aspektu aplikacji.

### 1.2 Komponenty

- Aplikacja kliencka — Interfejs graficzny użytkownika.
- Pośrednik — Komponent pośredniczący w komunikacji pomiędzy aplikacją kliencką i serwisem predykcyjnym
- Komponent komunikacyjny — Komponent zawierający szyny danych, które są wykorzystywane do dostarczania i odbierania informacji od serwisu predykcyjnego
- Serwis predykcyjny — Komponent dokonujący predykcji na podstawie dostarczonych danych, z wykorzystaniem nauczonego modelu.

### 1.3 Konteneryzacja

Wszystkie komponenty zostały skonteneryzowane za pomocą narzędzi **Docker**<sup>1</sup> i **Docker Compose**<sup>2</sup>, co pozwala na uruchomienie projektu bez konieczności dodatkowej konfiguracji. **Obrazy**<sup>3</sup> **kontenerów**<sup>4</sup> dla aplikacji klienckiej, pośrednika oraz serwisu predykcyjnego zostały zdefiniowane za pomocą plików **Dockerfile**<sup>5</sup>, natomiast dla komponentu komunikacyjnego wykorzystano gotowe obrazy Apache Kafka i Zookeeper z rejestru Docker.io.

---

<sup>1</sup>Narzędzie do tworzenia, uruchamiania i zarządzania aplikacjami w izolowanych środowiskach zwanych kontenerami.

<sup>2</sup>Narzędzie usprawniające zarządzanie wieloma kontenerami jednocześnie.

<sup>3</sup>Gotowy do uruchomienia szablon do tworzenia kontenerów, zawierający system plików, aplikację i jej zależności.

<sup>4</sup>Lekkie, izolowane środowisko uruchomieniowe, które zawiera aplikację wraz z jej zależnościami.

<sup>5</sup>Plik tekstowy zawierający zestaw instrukcji do zbudowania obrazu Docker.

## 1.4 Sposób uruchamiania

1. Zainstalować Docker i Docker Compose.
2. Aplikacja kliencka — Otworzyć katalog `frontend` i wewnątrz niego uruchomić skrypt `run.sh` lub uruchomić ręcznie komendy w terminalu. Wyłączenia kontenera można dokonać skryptem `clean.sh`.

- Skrypt `run.sh`:

```
#!/bin/bash

# Builds docker image from local Dockerfile
# and sets image name to "frontend-image"
docker build -t frontend-image .

# Creates and runs container with name "frontend" from
# frontend-image
# in detached mode and container-host port mapping to
# 9091
docker run --name frontend -d -p 9091:9091 frontend-image
```

- Skrypt `clean.sh`:

```
#!/bin/bash

# Stops and removes the "frontend" container
docker stop frontend
docker rm frontend
```

3. Pozostałe komponenty — Otworzyć katalog projektu i uruchomić komendę `docker compose up`. Do wyłączenia kontenerów należy użyć komendy `docker compose down`.

```
# Run containers
docker compose up

# Stop containers
docker compose down
```

## 1.5 Dane

Do utworzenia modelu predykcyjnego wykorzystany został zestaw danych “Used Car Price Prediction Dataset”<sup>6</sup> z platformy Kaggle.

Cechy zbioru danych:

- **brand** — Marka lub nazwa firmy produkującej samochody.
- **mode** — Model pojazdu.
- **model\_year** — Rok produkcji pojazdu.
- **milage** — Przebieg samochodu w milach.
- **fuel\_type** — Rodzaj paliwa wykorzystywanego przez samochód.
- **engine** — Typ silnika.
- **transmission** — Typ skrzyni biegów.
- **ext\_col** — Kolor zewnętrzny pojazdu.
- **int\_col** — Kolor wnętrza pojazdu.
- **accident** — Historia wypadków
- **clean\_title** — Czy pojazd posiada czysty tytuł własności.
- **price** — Cena samochodu według sprzedającego.

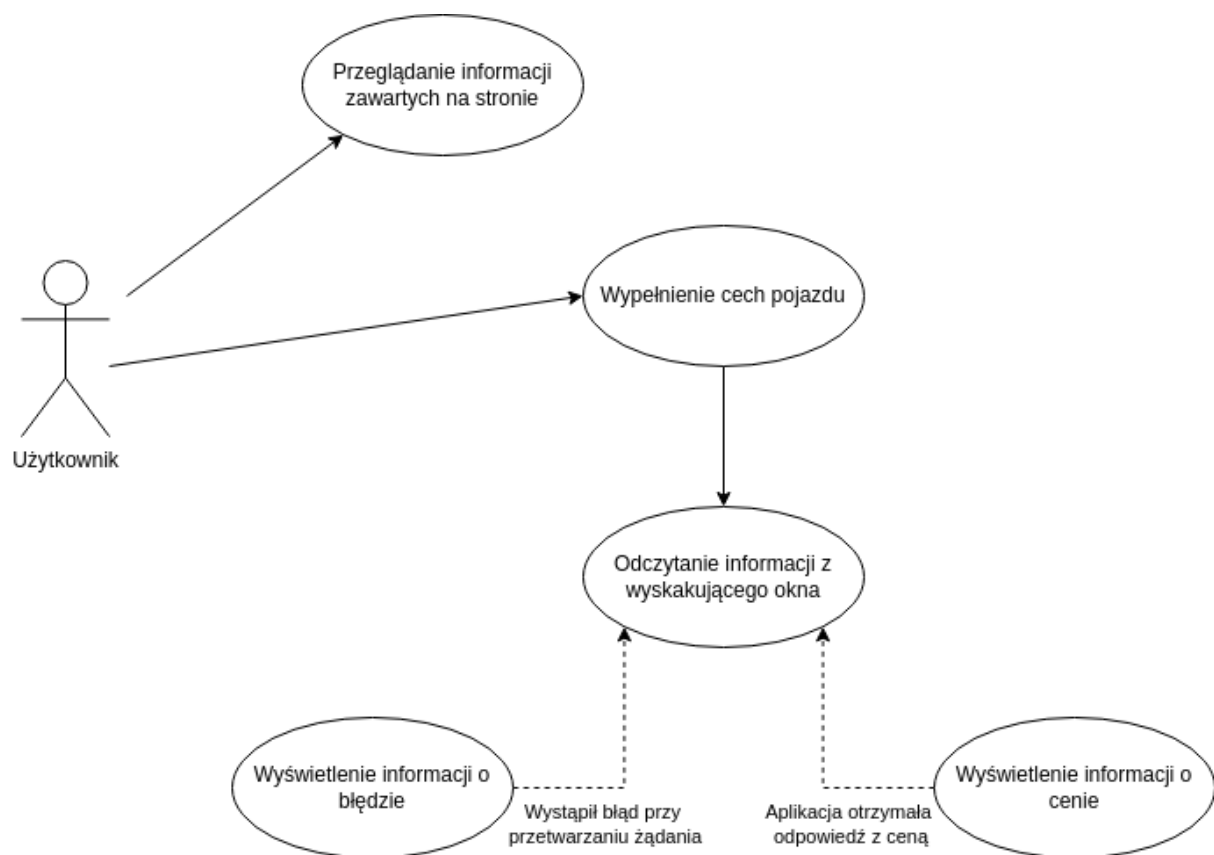
---

<sup>6</sup><https://www.kaggle.com/datasets/taeefnajib/used-car-price-prediction-dataset>

# Rozdział 2

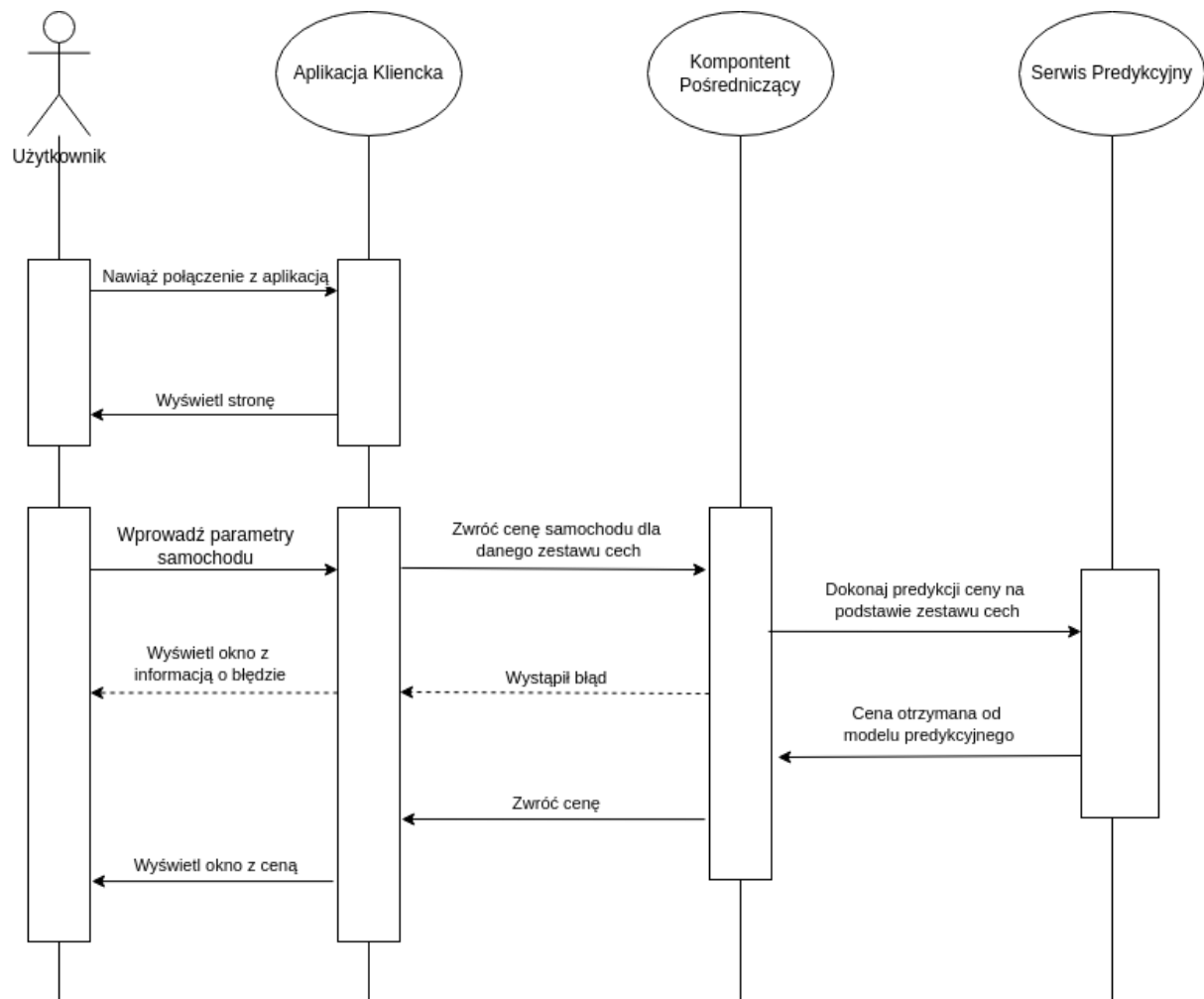
## Diagramy

### 2.1 Diagram przypadków użycia



Rysunek 2.1: Przebieg interakcji użytkownika z aplikacją

## 2.2 Diagram sekwencji zdarzeń



Rysunek 2.2: Przebieg operacji komponentów i działań użytkownika podczas procesu predykcji ceny samochodu



# Rozdział 3

## Aplikacja kliencka

### 3.1 Opis

Aplikacja kliencka stanowi pojedynczą stronę dostępną za pośrednictwem przeglądarki, udostępnianą pod adresem **localhost**<sup>1</sup>, na porcie **9091**. Strona zawiera informacje związane z aplikacją oraz pola do wprowadzania wartości, na podstawie których następnie dokonywana jest predykcja ceny samochodu. Aplikacja łączy się z komponentem middleware za pośrednictwem protokołu **HTTP**<sup>2</sup> w architekturze **REST**<sup>3</sup>.

### 3.2 Technologie

- React — Framework JavaScript do tworzenia interfejsów użytkownika w oparciu o komponenty.
- HTML — Język znaczników do tworzenia struktury strony internetowej.
- CSS — Język stylów wykorzystywany do definiowania wyglądu stron internetowych.
- JavaScript — Język programowania wykorzystywany do tworzenia dynamicznych i interaktywnych elementów stron internetowych.
- Axios — Biblioteka JavaScript służąca do wykonywania zapytań HTTP.
- Vite — Narzędzie do budowania i uruchamiania aplikacji front-endowych.

---

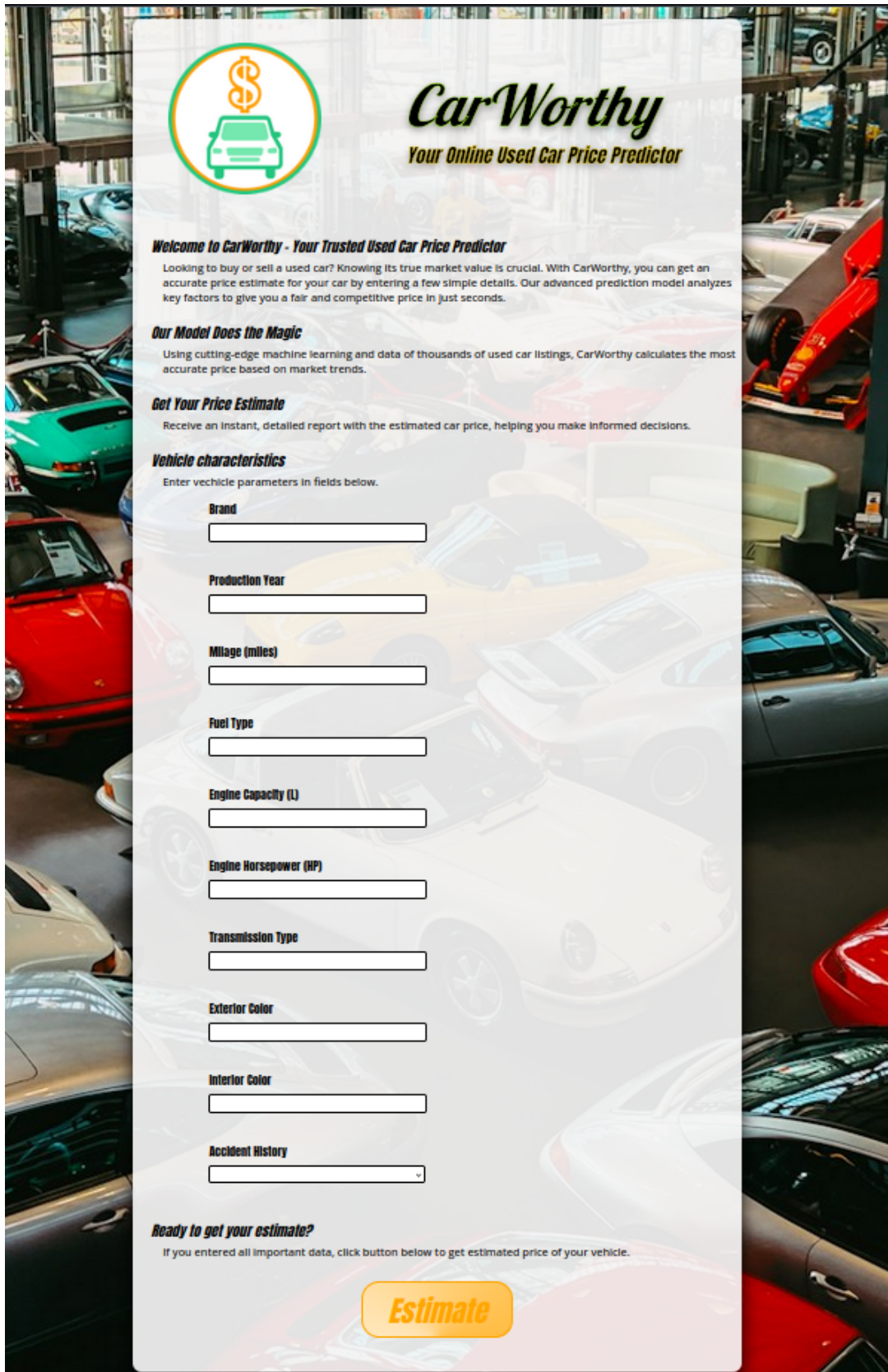
<sup>1</sup>loopback address — adres pętli zwrotnej, który jest wykorzystywany do komunikacji urządzenia z samym sobą.

<sup>2</sup>HyperText Transfer Protocol — protokół komunikacyjny używany do przesyłania danych w sieci.

<sup>3</sup>Representational State Transfer — architektura komunikacji oparta o protokół HTTP definiujący sposoby identyfikacji i manipulacji zasobami za pomocą zapytań HTTP.

## 3.3 Widoki aplikacji

### 3.3.1 Strona



The screenshot displays the CarWorthy website, which is an online used car price predictor. The page features a light gray background with a faint image of a car dealership. At the top left, there is a logo consisting of a green circle with a white car icon and a dollar sign above it. To the right of the logo, the text "CarWorthy" is written in a large, bold, black font, with "Your Online Used Car Price Predictor" in a smaller, italicized font below it.

Below the header, there is a welcome message: "Welcome to CarWorthy - Your Trusted Used Car Price Predictor". This is followed by a paragraph explaining the service: "Looking to buy or sell a used car? Knowing its true market value is crucial. With CarWorthy, you can get an accurate price estimate for your car by entering a few simple details. Our advanced prediction model analyzes key factors to give you a fair and competitive price in just seconds."

Next, there is a section titled "Our Model Does the Magic" with a paragraph: "Using cutting-edge machine learning and data of thousands of used car listings, CarWorthy calculates the most accurate price based on market trends."

Below this, there is a section titled "Get Your Price Estimate" with a paragraph: "Receive an instant, detailed report with the estimated car price, helping you make informed decisions."

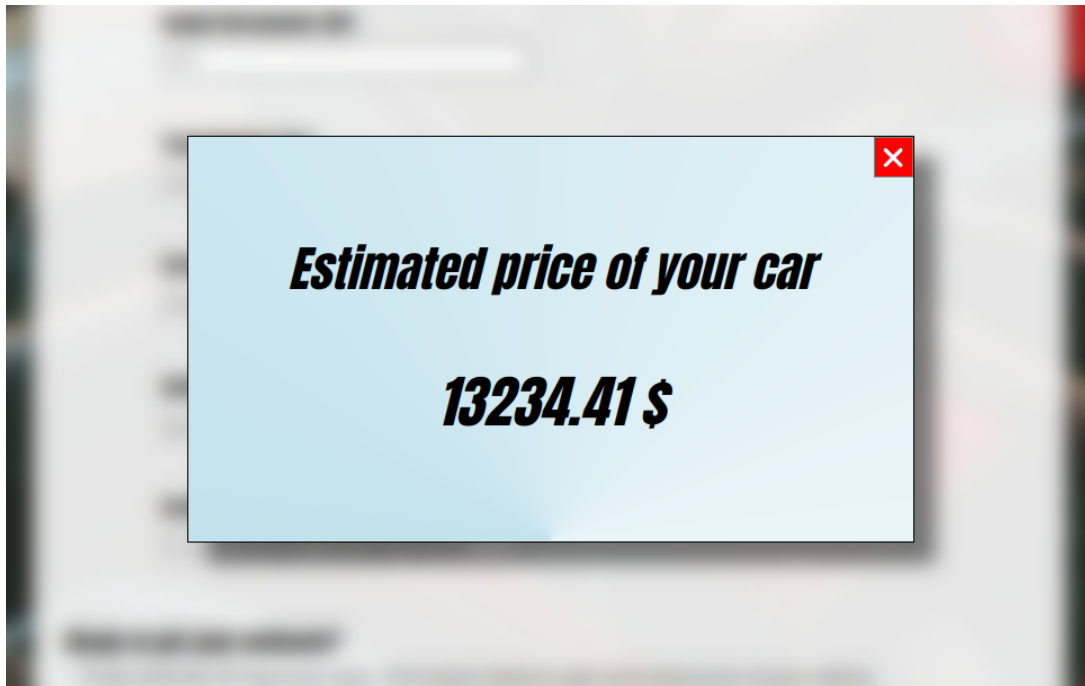
The main section is titled "Vehicle characteristics" and includes the instruction: "Enter vehicle parameters in fields below." This section contains ten input fields, each with a label and a text box:

- Brand
- Production Year
- Milage (miles)
- Fuel Type
- Engine Capacity (L)
- Engine Horsepower (HP)
- Transmission Type
- Exterior Color
- Interior Color
- Accident History

At the bottom of the form, there is a section titled "Ready to get your estimate?" with the instruction: "If you entered all important data, click button below to get estimated price of your vehicle." Below this instruction is a large, orange, rounded rectangular button with the text "Estimate" in white.

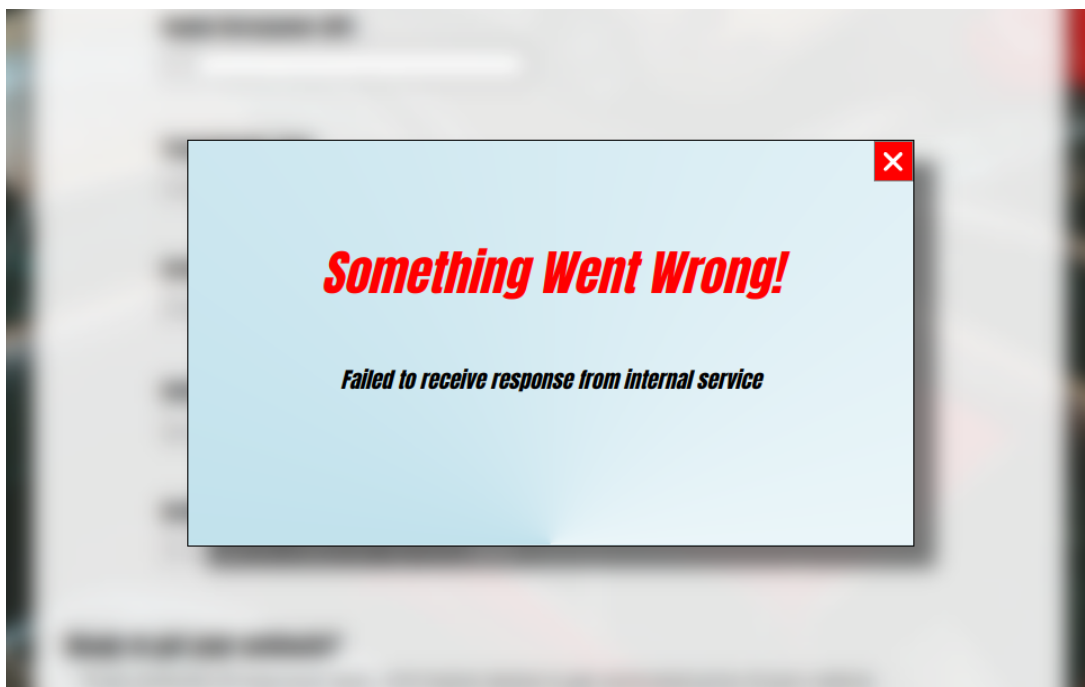
Rysunek 3.1: Widok strony

### 3.3.2 Okno z ceną



Rysunek 3.2: Widok okna z ceną

### 3.3.3 Okno z błędem



Rysunek 3.3: Widok okna z błędem

# Rozdział 4

## Komponent pośredniczący

### 4.1 Opis

Komponent pośredniczący pełni rolę pośrednika pomiędzy aplikacją kliencką i serwisem predykcyjnym. Otrzymywane od **frontendu**<sup>1</sup> dane w formie **JSON**<sup>2</sup> są w tym komponencie przetwarzane na wiadomości w formacie odpowiadającym wejściu modelu, z uwzględnieniem procesu **kodowania liczbowego**<sup>3</sup> pól. Otrzymane w tym procesie wiadomości zapisywane są na **temat**<sup>4</sup> wejściowy Kafki. Pośrednik jest również odpowiedzialny za odczytywanie danych z tematu wyjściowego i przekazywanie uzyskanych z nich informacji do klienta.

### 4.2 Technologie

- Java — Obiektowy język programowania.
- SpringBoot — Framework dla języka Java nastawiony na wytwarzanie aplikacji webowych i mikroservisów
- Gradle — Narzędzie do automatyzacji budowania projektów.

---

<sup>1</sup>Część aplikacji, z którą użytkownik wchodzi w bezpośrednią interakcję, w tym wszystko co widzi oraz elementy wizualne i interaktywne.

<sup>2</sup>JavaScript Object Notation — format danych zapewniający kompaktowe rozmiary i jest czytelny dla ludzi i maszyn.

<sup>3</sup>Technika zamiany wartości danych tekstowych na wartości liczbowe, poprzez przypisanie unikalnej liczby każdej unikalnej wartości tekstowej.

<sup>4</sup>Podstawowy komponent Apache Kafka służący do kategoryzacji napływających wiadomości.

# Rozdział 5

## Komponent komunikacyjny

### 5.1 Opis

Komponent komunikacyjny odpowiedzialny jest za transport danych pomiędzy komponentem pośredniczącym i serwisem predykcyjnym. Wykorzystuje w tym celu skonteneryzowany **broker**<sup>1</sup> wiadomości Apache Kafka wraz z dwoma tematami: input oraz output, wykorzystywanych odpowiednio do gromadzenia danych odczytywanych przez serwis predykcyjny i gromadzenia danych odczytywanych przez pośrednika. Do zarządzania brokerem wykorzystywany jest Apache Zookeeper.

### 5.2 Technologie

- Apache Kafka — Platforma przetwarzania danych w czasie rzeczywistym.
- Apache Zookeeper — Usługa koordynacyjna systemów rozproszonych.

---

<sup>1</sup>Serwer Apache Kafka zawierający dane należące do tematów i partycji, na które może być podzielony temat.

# Rozdział 6

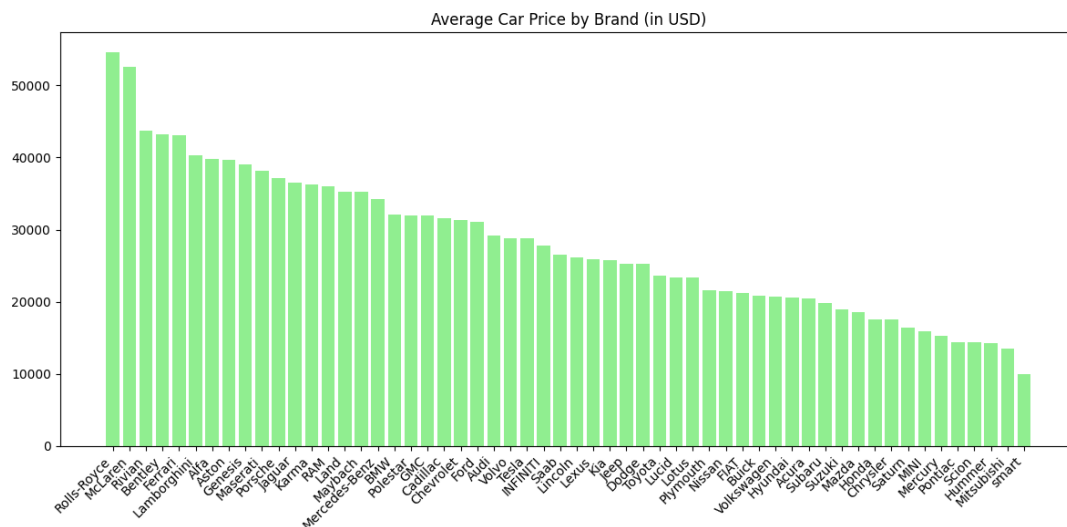
## Przygotowanie danych

### 6.1 Opis

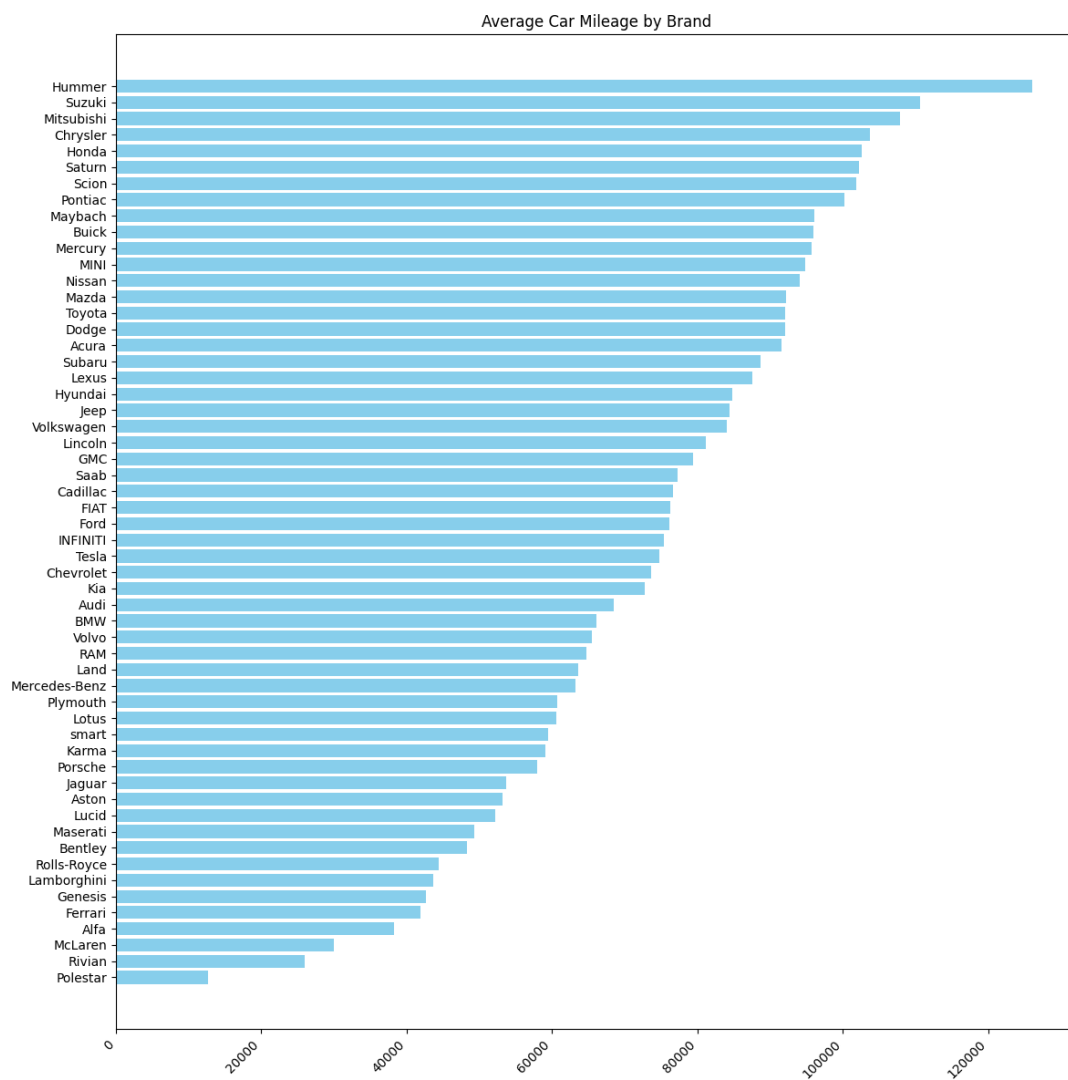
Model jest skuteczny, gdy dane na których się go trenuje są odpowiednio przygotowane. Początkowo należy przeanalizować potencjalne zagrożenia w postaci braków poszczególnych wartości w polach danych oraz wartości odstających, mogących zniekształcić miary statystyczne. Na końcu należy zfaktoryzować, a zatem znumeryzować dane kategoryczne tak aby model mógł się na nich uczyć.

### 6.2 Wizualizacja danych

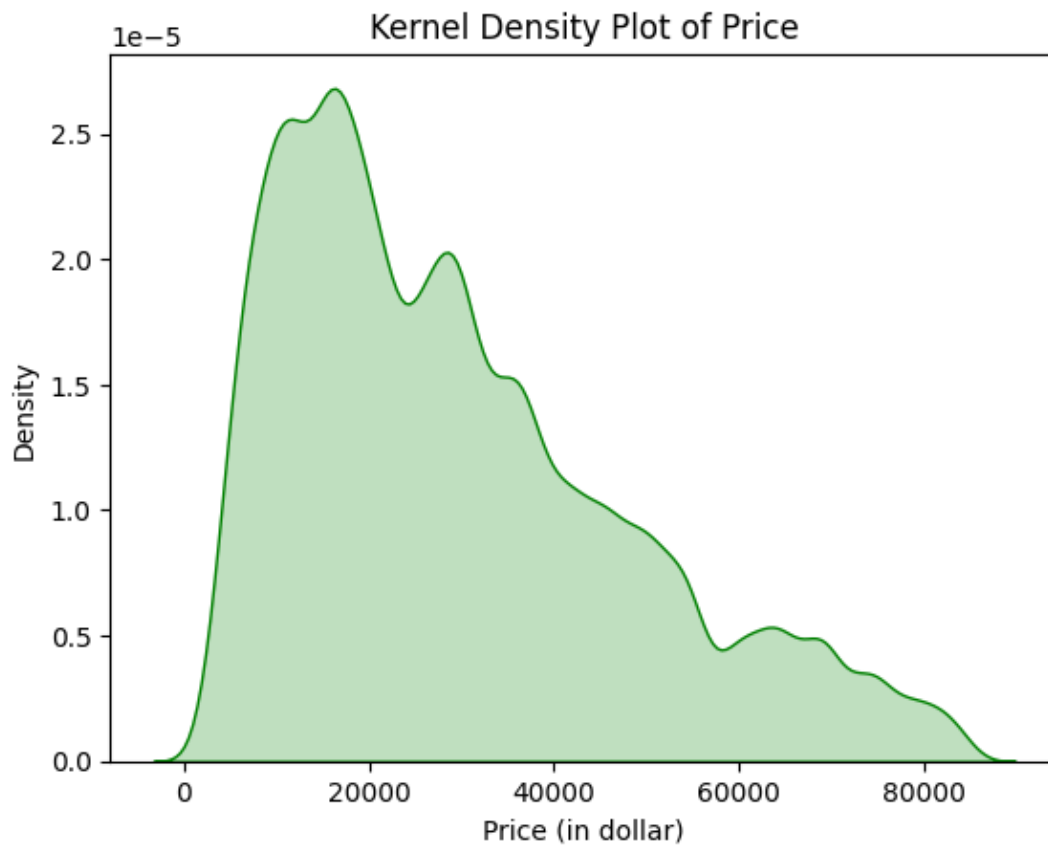
Kilka wykresów pokazujących zbiór danych



Rysunek 6.1: Średnia cena pojazdu danej marki



Rysunek 6.2: Średnia ilość przejechanych mil pojazdów danej marki



Rysunek 6.3: Wykres gęstości cen pojazdów, widać że najwięcej jest ich w okolicach 17 tyś. USD

## 6.3 Puste pola

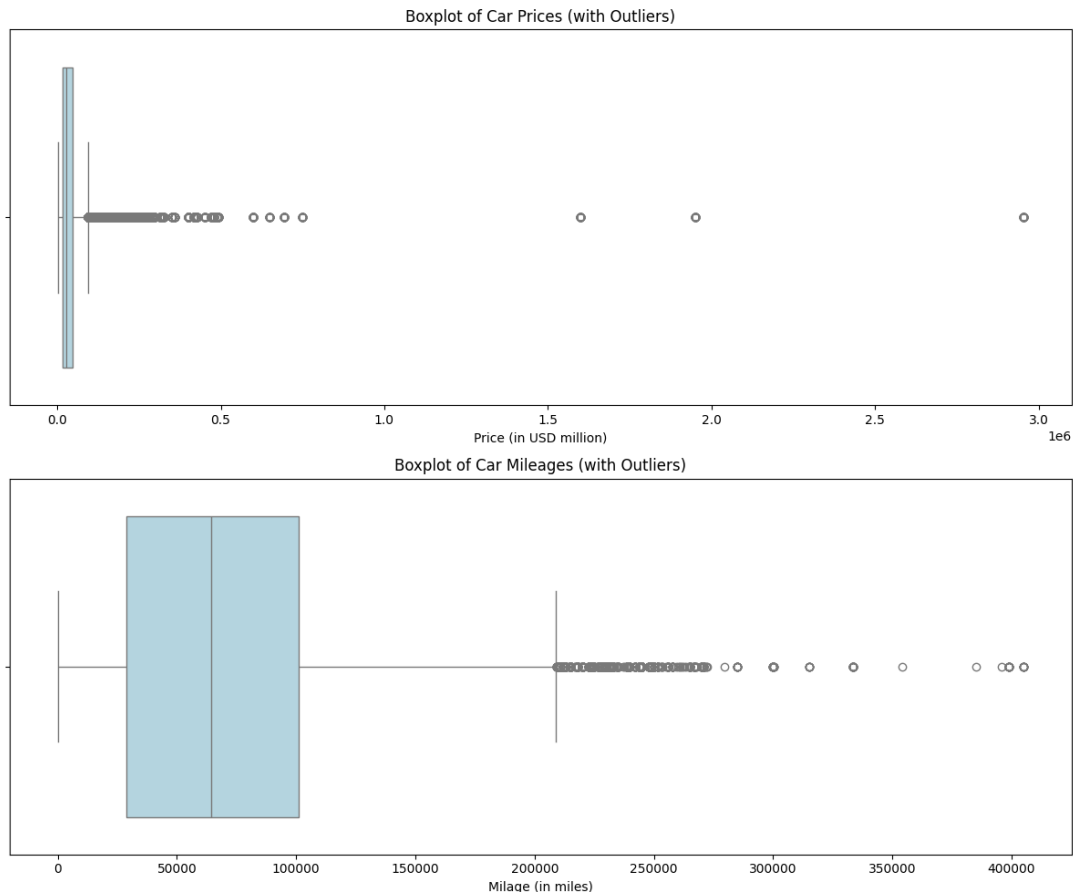
Postanowiliśmy usunąć puste pola zamiast stosować inne metody sztucznego ich wypełniania na przykład średnią, ponieważ zestaw danych jest obszerny i usunięcie próbek z pustymi wartościami nie odbije się na dokładności modelu.

Przed usunięciem wartości odstających	188533
Po usunięciu wartości odstających	162610

Tabela 6.1: Wielkość zbioru danych przed i po usunięciu wartości odstających



## 6.4 Zestaw danych z wartościami odstającymi



Rysunek 6.4: Niektóre wartości są zbyt duże, należy się ich pozbyć

## 6.5 Wykrywanie i usuwanie wartości odstających za pomocą metody IQR

Aby wykryć i usunąć wartości odstające w zbiorze danych na podstawie kolumn `price` oraz `milage`, wykonaliśmy następujące kroki:

### 1. Obliczenie kwartyli

Pierwszy kwartył ( $Q_1$ ) oraz trzeci kwartył ( $Q_3$ ) wyznaczone są dla każdej kolumny. Odpowiadają one 25. i 75. percentylowi:

$$Q_1 = 25. \text{ percentyl}, \quad Q_3 = 75. \text{ percentyl}$$

### 2. Obliczenie rozstępu międzykwartylowego (IQR)

Rozstęp międzykwartylowy (IQR) oblicza się jako:

$$IQR = Q_3 - Q_1$$

### 3. Definiowanie granic wartości odstających

Wartości odstające to te, które znajdują się poza przedziałem:

$$\text{Dolna granica} = Q_1 - 1.5 \cdot IQR, \quad \text{Górna granica} = Q_3 + 1.5 \cdot IQR$$

Dla kolumn `price` oraz `milage` wyznacza się odpowiednio:

$$\text{Dolna granica}_{\text{price}} = Q_{1,\text{price}} - 1.5 \cdot IQR_{\text{price}}, \quad \text{Górna granica}_{\text{price}} = Q_{3,\text{price}} + 1.5 \cdot IQR_{\text{price}}$$

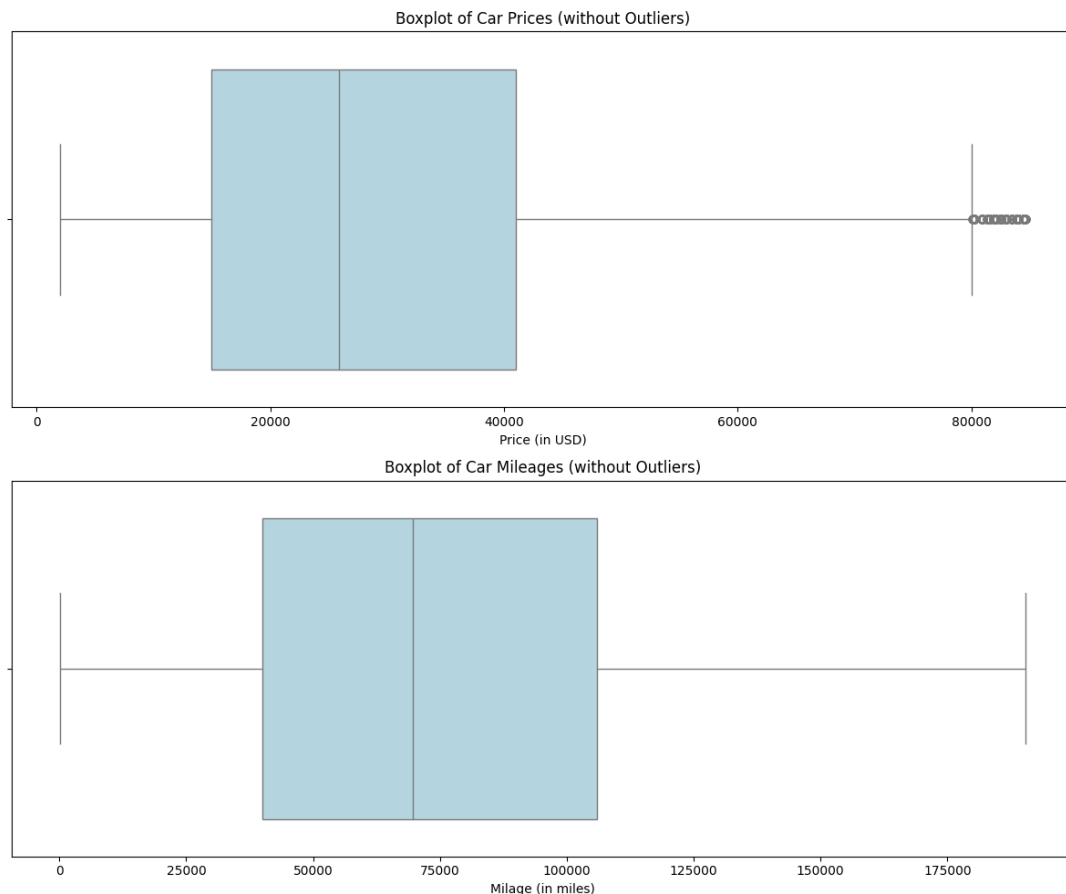
$$\text{Dolna granica}_{\text{milage}} = Q_{1,\text{milage}} - 1.5 \cdot IQR_{\text{milage}}, \quad \text{Górna granica}_{\text{milage}} = Q_{3,\text{milage}} + 1.5 \cdot IQR_{\text{milage}}$$

### 4. Filtrowanie wierszy bez wartości odstających

Zbiór danych jest filtrowany w taki sposób, aby wartości w kolumnach `price` oraz `milage` spełniały następujące warunki:

$$Q_1 - 1.5 \cdot IQR \leq X \leq Q_3 + 1.5 \cdot IQR$$

## 6.6 Zestaw danych bez wartości odstających



Rysunek 6.5: Brak lub mała ilość wartości odstających

## 6.7 Wyodrębnienie znaczących informacji o silniku

W zestawie danych istniała kolumna o nazwie `engine`, w której wartości przedstawiały krótki opis silnika, zawierający takie informacje jak liczba koni mechanicznych, pojemność silnika oraz inne cechy. Przykładowy opis to:

172.0HP 1.6L 4 Cylinder Engine Gasoline Fuel

Z takiego opisu, za pomocą wyrażeń regularnych, wyodrębniliśmy dwie główne informacje: pojemność silnika oraz liczbę koni mechanicznych.

Dla przykładu:

172.0HP 1.6L 4 Cylinder Engine Gasoline Fuel → 1.6, 172.0

## Zastąpienie kolumny **engine** nowymi kolumnami:

Po wyodrębnieniu informacji o pojemności silnika oraz liczbie koni mechanicznych, zastąpiliśmy istniejącą kolumnę **engine** dwiema nowymi kolumnami:

- **engine\_horsepower** – zawierającą moc silnika w koniach mechanicznych (HP).
- **engine\_capacity** – zawierającą pojemność silnika w litrach (L).

## 6.8 Faktoryzacja danych

Modele uczenia maszynowego wymagają danych numerycznych jako wejścia. Wartości tekstowe (kategoryczne) muszą być przekonwertowane na liczby w sposób, który zachowa sens danych, ale jednocześnie nie wprowadzi sztucznej relacji między kategoriami. Do każdej wartości kategorycznej został przypisany numer, następnie dzięki słownikowi zamieniliśmy wszystkie próbki w wektory cech w następujący sposób:

### Dane przed faktoryzacją

Przykład danych wejściowych przed faktoryzacją:

```
{
  "brand": "Toyota",
  "transmission": "Automatic",
  "fuel_type": "Gasoline",
  "ext_col": "Red",
  "int_col": "White",
  "accident": "No accident",
}
```

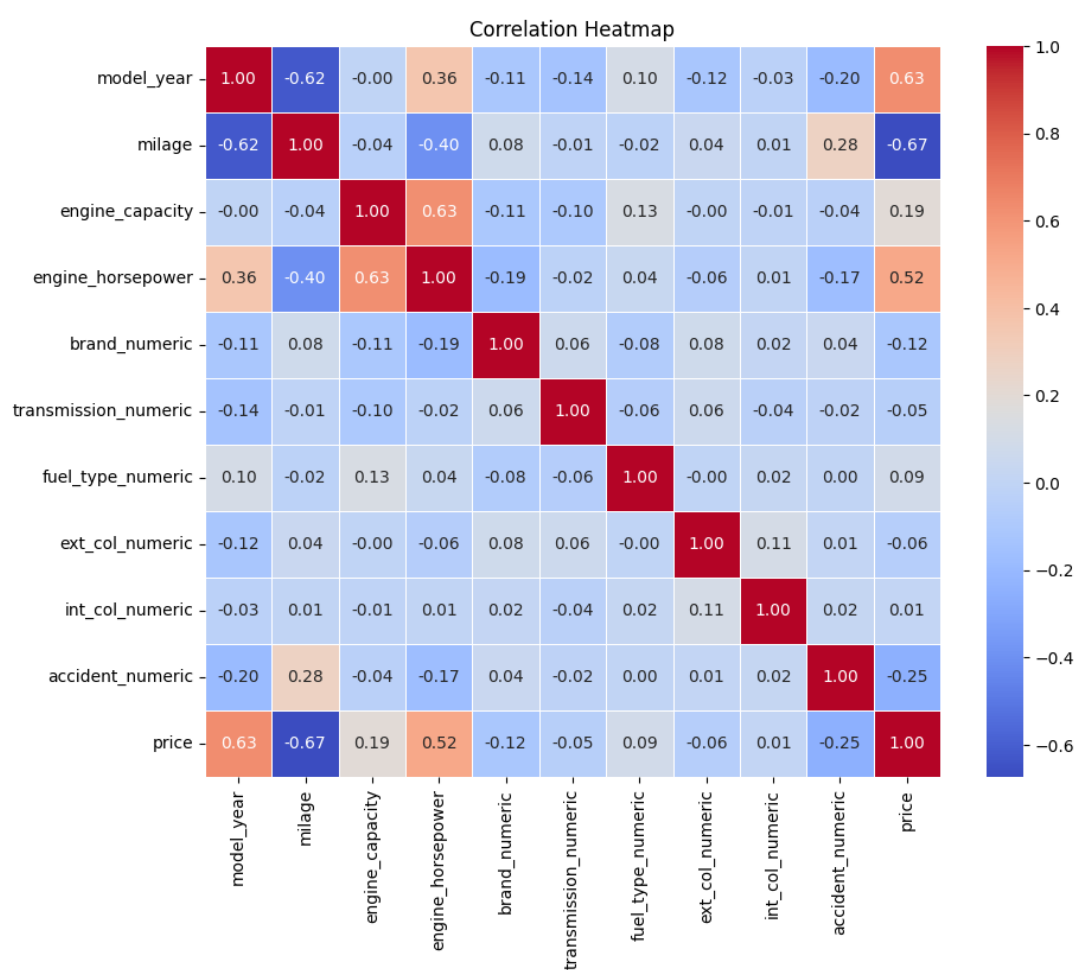
### Dane po faktoryzacji

Po zastosowaniu faktoryzacji 'Label-Encoding' (przypisaniu liczb do kategorii), dane będą wyglądać następująco:

```
{
  "brand": 5,
  "transmission": 0,
  "fuel_type": 0,
  "ext_col": 3,
  "int_col": 5,
  "accident": 0,
}
```

## Przykładowy wiersz danych po faktoryzacji:

Nazwa zmiennej	Wartość
model_year	2002
milage	143250.0
price	4999
engine_capacity	3.9
engine_horsepower	252.0
brand_numeric	17.0
transmission_numeric	0.0
fuel_type_numeric	0.0
ext_col_numeric	3.0
int_col_numeric	1.0
accident_numeric	1.0



Rysunek 6.6: Macierz korelacji wszystkich cech, cechy korelujące się w największym stopniu z ceną: rok produkcji modelu, przebieg samochodu oraz konie mechaniczne

# Rozdział 7

## Serwis predykcyjny

### 7.1 Opis

Zadaniem serwisu predykcyjnego jest dokonanie predykcji ceny samochodu na podstawie dostarczonego zestawu cech. W tym celu wykorzystuje gotowy, zapisany model przygotowany przy użyciu modułu SparkML. Dane przekazywane do modelu są odczytywane z tematu input za pomocą frameworka Spark. Cena zwrócona przez model zostaje zapisana na temat wyjściowy output.

### 7.2 Technologie

- Python — Język skryptowy.
- Apache Spark — Framework do sprawnego przetwarzania zbiorów danych w pamięci.
- Apache SparkML — Moduł Apache Spark przeznaczony do uczenia maszynowego.

### 7.3 Wybór modelu

Podczas wyboru modelu kierowaliśmy się strukturą danych. Skorzystaliśmy z **Label Encoding**, czyli przypisania unikalnych wartości liczbowych dla każdej kategorii, np.:

"BMW" → 0, "Audi" → 1.

W tym przypadku **modele regresji liniowej** nie są skuteczne, ponieważ takie kodowanie może prowadzić do błędnych założeń o istnieniu relacji liczbowych między kategoriami. Przykładowo:

Różnica między 0 a 1 = Różnica między 1 a 2,

co nie ma sensu w przypadku zmiennych kategorycznych.

**Rozwiązaniem są modele oparte na drzewach decyzyjnych**, takie jak:

- Drzewo decyzyjne,
- Las losowy.

## 7.4 Testowanie modeli

### 7.4.1 Las Losowy

Do testowania modeli użyliśmy metod z biblioteki `pyspark.ml.tuning`. Jednym z kluczowych kroków było zdefiniowanie siatki hiperparametrów za pomocą `ParamGridBuilder`.

**Definiowanie siatki hiperparametrów:**

Listing 7.1: Siatka hiperparametrów dla modelu

```
paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees, [50, 100]) \
    .addGrid(rf.maxDepth, [5, 10]) \
    .addGrid(rf.minInstancesPerNode, [1, 2, 4]) \
    .addGrid(rf.maxBins, [32, 64, 128]) \
    .addGrid(rf.subsamplingRate, [0.5, 0.7, 1.0]) \
    .addGrid(rf.featureSubsetStrategy, ['all', 'sqrt', 'log2']) \
    .build()
```

**Tworzenie obiektu ewaluatora:** Aby obliczyć dokładność modelu według metryki RMSE (Root Mean Squared Error), użyliśmy `RegressionEvaluator`.

Listing 7.2: Ewaluator dla metryki RMSE

```
evaluator_rmse = RegressionEvaluator(
    labelCol="price",
    predictionCol="prediction",
    metricName="rmse"
)
```

**Walidacja krzyżowa:** W celu strojenia hiperparametrów wykorzystaliśmy `CrossValidator`, który przeprowadza walidację krzyżową z podziałem na  $K$ -podzbiorów. Technika ta polega na:

1. Podziale danych na  $K$ -podzbiorów (folds),
2. Trenowaniu modelu  $K$ -razy na  $K - 1$ -podzbiorach,
3. Użyciu pozostałego podzbioru do walidacji,
4. Uśrednieniu wyników dla ostatecznej oceny modelu.

Listing 7.3: `CrossValidator` do walidacji krzyżowej

```
crossval = CrossValidator(
    estimator=rf,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator_rmse,
    numFolds=3 # 3-fold cross-validation
)
```

**Uczenie modelu:** Najlepszy model z najmniejszym RMSE został zapisany do zmiennej `best_model`.

Listing 7.4: Uczenie modelu i wybór najlepszego

```
cv_model = crossval.fit(train_data)
best_model = cv_model.bestModel
```

## Najlepsze znalezione parametry Lasu losowego

Parametr	Wartość
NumTrees	100
MaxDepth	10
MinInstancesPerNode	4
MaxBins	64
SubsamplingRate	0.5
FeatureSubsetStrategy	sqrt
MaxMemoryInMB	256

Czas trenowania: 116 minut

## Metryki ewaluacyjne dla powyższych hiperparametrów

Metryka	Wartość
Root Mean Squared Error (RMSE)	11011.20
Mean Absolute Error (MAE)	7962.28
R-squared ( $R^2$ )	0.6625

### 7.4.2 Drzewo decyzyjne

#### Definiowanie siatki hiperparametrów:

Listing 7.5: Siatka hiperparametrów dla modelu drzewa decyzyjnego

```
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [10, 15]) \
    .addGrid(dt.maxBins, [20, 30, 40]) \
    .addGrid(dt.minInstancesPerNode, [1, 2, 4]) \
    .addGrid(dt.minInfoGain, [0.0, 0.1, 0.2]) \
    .addGrid(dt.maxMemoryInMB, [512, 1024]) \
    .addGrid(dt.cacheNodeIds, [True, False]) \
    .addGrid(dt.checkpointInterval, [10, 20]) \
    .build()
```

Obiekt ewaluatora oraz walidacji krzyżowej są takie same jak w przypadku Lasu losowego

## Najlepsze znalezione parametry Drzewa decyzyjnego

Parametr	Wartość
MaxDepth	10
MaxBins	20
MinInstancesPerNode	4
MinInfoGain	0.0
MaxMemoryInMB	512
CacheNodeIds	True
CheckpointInterval	10

**Metryki ewaluacyjne dla powyższych hiperparametrów**

Metryka	Wartość
Root Mean Squared Error (RMSE)	11318.40
Mean Absolute Error (MAE)	8143.66
R-squared ( $R^2$ )	0.6434

**Czas trenowania:** 66 minut

### 7.4.3 Podsumowanie

Na podstawie miar ewaluacji obu modeli, Las Losowy ma niewielką przewagę nad Drzewem decyzyjnym, jednak trening tego modelu oraz znalezienie najlepszych parametrów zajęło blisko godzinę więcej.