

Miłosz Łopatto, nr 305898

Franciszek Sakowski, nr 309439

## IUM projekt - etap 2

### Spis treści

1. Zadanie biznesowe (domenowe) .....	2
2. Zadanie analityczne (zadanie modelowania) .....	2
3. Proces budowy modelu .....	3
3.1. Selekcja atrybutów i feature engineering.....	3
3.1.1. Heatmap z korelacjami Pearsona .....	3
3.1.2. Rozkład danych dla różnych <i>delivery_company</i> .....	4
3.1.3. Histogram czasów dostaw .....	5
3.1.4. Analiza trendu (moving average) dla średniego czasu dostawy danego dnia .....	5
3.1.5. Analiza sezonowości dla średniego czasu dostawy danego dnia.....	5
3.1.6. Periodogram.....	6
3.1.7. Badanie zależności ze średnim czasem dostaw z poprzednich dni.....	7
3.1.8. Mutual information score dla <i>time_diff</i> .....	8
3.2. Przygotowanie danych do modelowania.....	9
4. Ostateczny model.....	10
5. Mikroserwis .....	10
6. Mikroserwis - testy .....	11
7. Repozytorium kodu .....	15

## 1. Zadanie biznesowe (domenowe)

zadanie 3, wariant 2

*“Wygląda na to, że nasze firmy kurierskie czasami nie radzą sobie z dostawami. Gdybyśmy wiedzieli, ile taka dostawa dla danego zamówienia potrwa – moglibyśmy przekazywać tę informację klientom.”*

Chcemy móc przewidywać czas dostawy produktu zakupionego w danym momencie.

### 1.1. Biznesowe kryterium sukcesu

Nasza początkowa propozycja:

*Model będzie szacował czas dostawy i przewidywał jej datę i godzinę z dokładnością  $\pm 12$  godzin w 90% przypadków.*

Ostatecznie udało nam się uzyskać trochę gorsze wyniki, ale prawie zawsze jesteśmy w stanie zapewnić dokładność  $\pm 24$  godziny (w 93% przypadków):

```
RidgeCV score = 0.4603548930522081, training time = 0.49544334411621094s
RandomForestRegressor score = 0.5439410349642648, training time = 15.860059976577759s
number of good predictions for RidgeCV = 2244/2464
which is 91.07142857142857% for  $\pm 24$  hours
```

```
number of good predictions for RandomForestRegressor = 2288/2464
which is 92.85714285714286% for  $\pm 24$  hours
```

```
number of good predictions for RidgeCV = 1508/2464
which is 61.201298701298704% for  $\pm 12$  hours
```

```
number of good predictions for RandomForestRegressor = 1711/2464
which is 69.43993506493507% for  $\pm 12$  hours
```

### 1.2. Wymagania niefunkcjonalne:

Chcemy, aby system był wygodny i łatwy w utrzymaniu.

Ponadto, dodajemy mechanizm ułatwiający interpretowalność predykcji polegający na dodaniu szacowanego czasu dostawy do czasu złożenia zamówienia i wypisaniu otrzymanej daty i godziny użytkownikowi.

## 2. Zadanie analityczne (zadanie modelowania)

Postawione zadanie biznesowe rozwiążemy jako zadanie analizy szeregów czasowych.

Pojedynczą obserwacją będzie zatem złożone zamówienie wraz ze wszystkimi danymi go dotyczącymi. Obserwacje uszeregowane są chronologicznie wedle czasu rozpoczęcia dostawy.

Korzystać będziemy z danych dostarczonych nam przez prowadzących w ramach przedmiotu. Planujemy podzielić dostępne dane na zbiory uczący i testowy w następujących proporcjach:

- zbiór uczący - 80% wszystkich danych
- zbiór testowy - 20% wszystkich danych

Zrezygnowaliśmy ze zbioru walidacyjnego, ponieważ:

- nasz pierwszy model (Ridge regression z wbudowaną walidacją krzyżową) dostosowuje sobie parametr Alpha
- nasz drugi model (RandomForestRegressor) liczbę generowanych drzew po paru testach zostawiliśmy na poziomie 100, ponieważ zwiększanie jedynie wydłużało czas potrzebny na wytrenowanie modelu, a nie dawało lepszych rezultatów. Poza tym próbowaliśmy także przy pomocy *RandomizedSearchCV* oraz *GridSearchCV* ustawiać parametry takie jak *max\_depth*, *max\_features*, *min\_samples\_leaf* oraz *min\_samples\_split*, ale ostatecznie z tego zrezygnowaliśmy.

Model w mikroserwisie trenujemy na wszystkich danych.

Danymi wejściowymi naszego modelu będzie zatem zbiór uczący (zmodyfikowany przekształceniami opisanymi w kolejnych rozdziałach). Model nie posiada żadnych innych parametrów wejściowych - wystarczą same dane.

Do oceny rozwiązania korzystać będziemy z prostego wskaźnika określającego procent prawidłowo przewidzianych czasów dostaw na zbiorze testowym - tzn. mieszczących się w określonych widełkach czasowych względem faktycznego czasu dostawy (proponujemy ustalenie tych widełek na +/- 12 godziny).

Funkcją celu będzie dla nas maksymalizacja wartości  $R^2$  score (współczynnik determinacji  $R^2$ )

Zmienną celu w naszym przypadku będzie dodatkowa kolumna będąca różnicą czasu rozpoczęcia i zakończenia dostawy (w sekundach).

## 2.1. Analityczne kryterium sukcesu

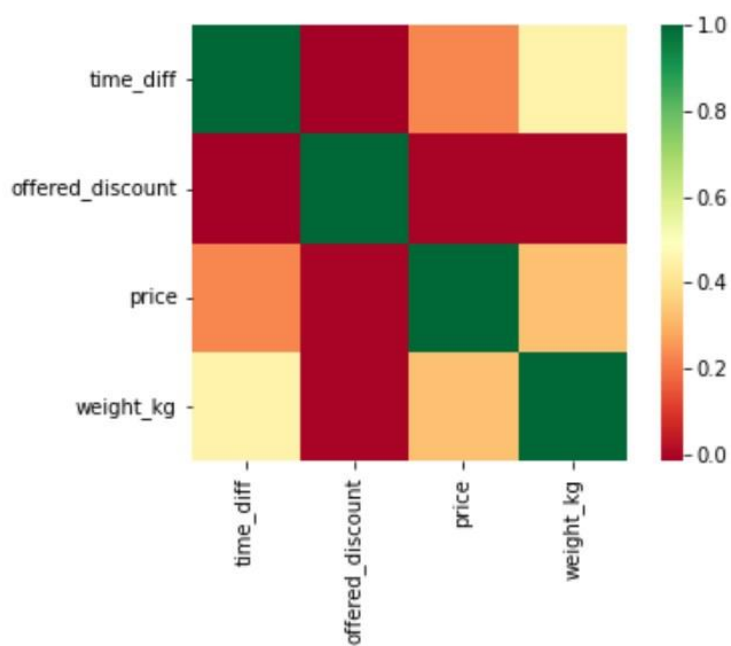
Analitycznym kryterium sukcesu będzie dla nas takie samo jak biznesowe kryterium sukcesu, czyli:

Model będzie szacował czas dostawy i przewidywał jej datę i godzinę z dokładnością +/- 12 godzin (lub +/- 24 godziny) w 90% przypadków.

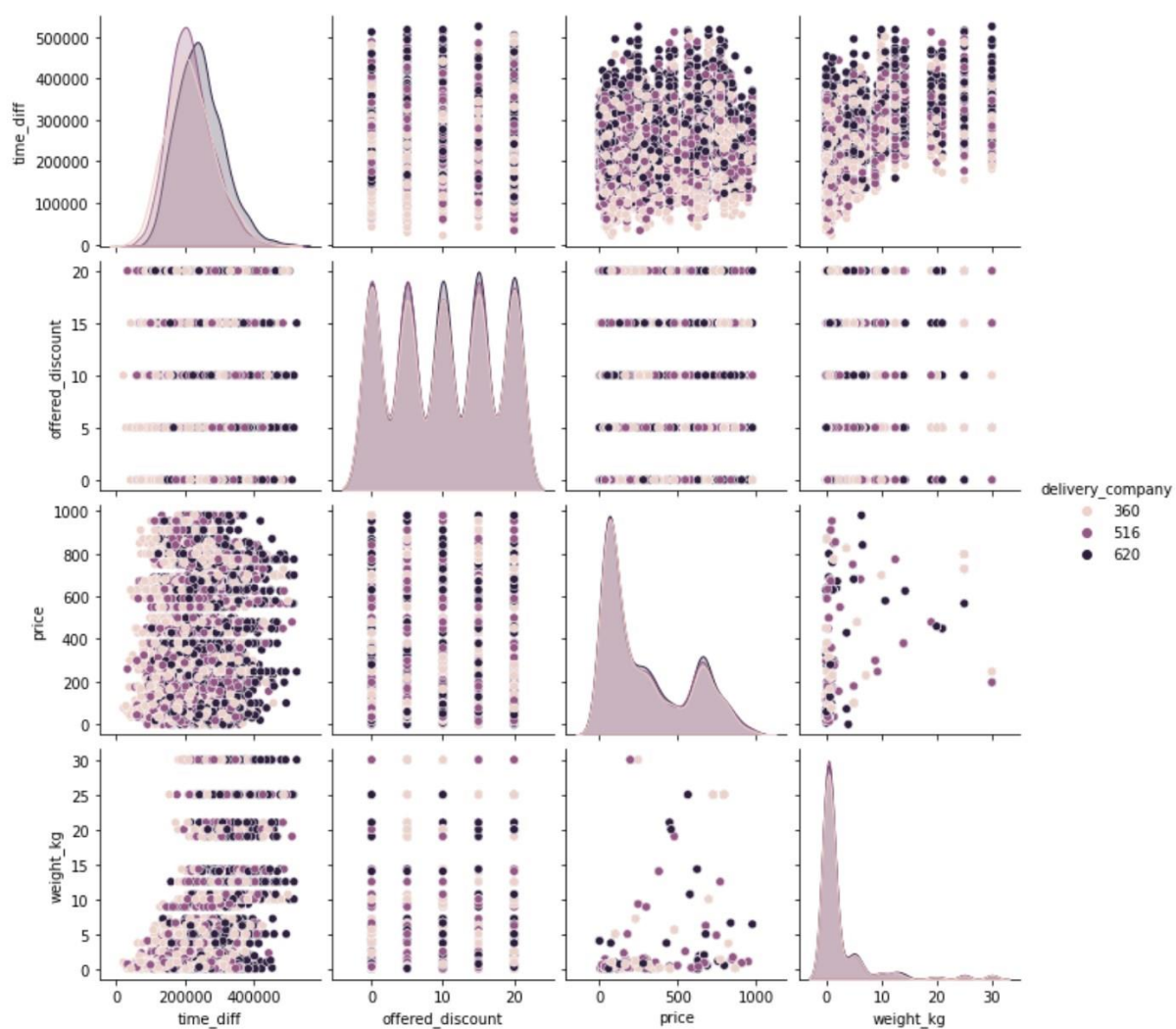
## 3. Proces budowy modelu

### 3.1. Selekcja atrybutów i feature engineering

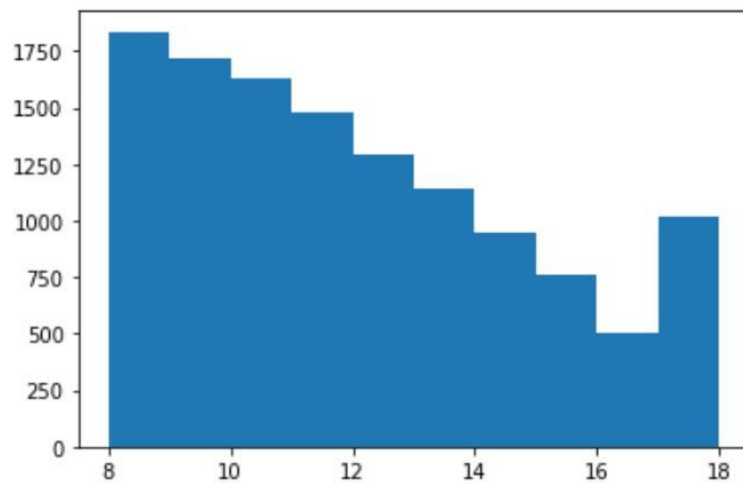
#### 3.1.1. Heatmap z korelacjami Pearsona



### 3.1.2. Rozkład danych dla różnych *delivery\_company*

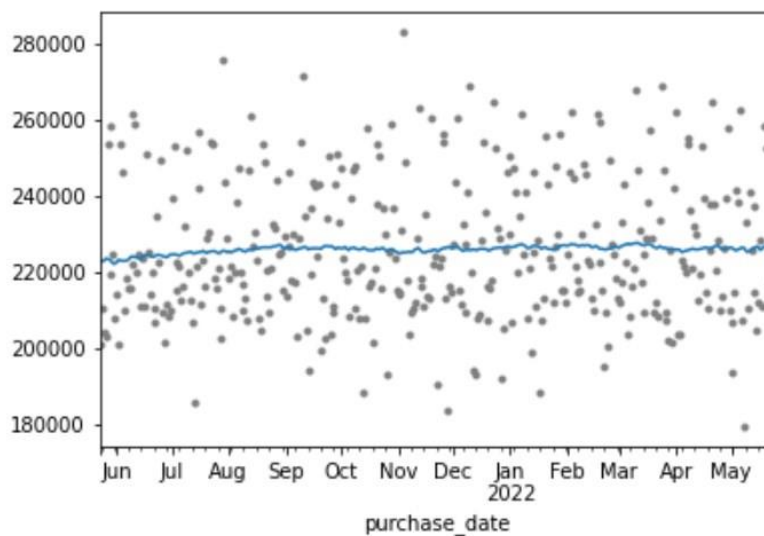


### 3.1.3. Histogram czasów dostaw



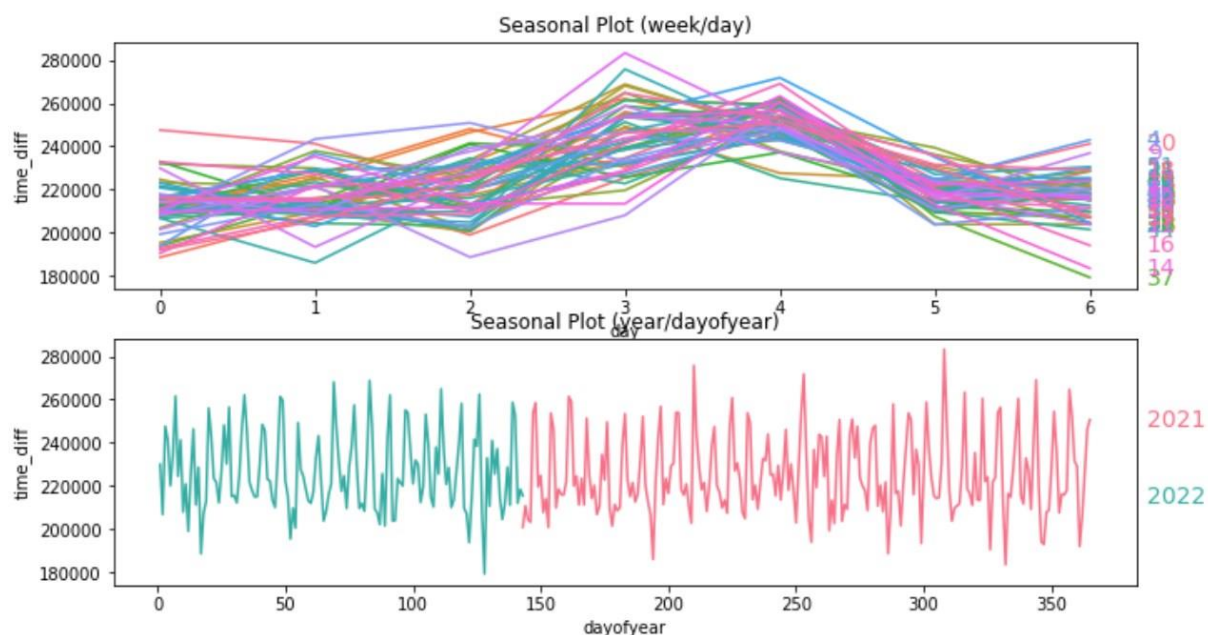
Wcześniej ponad połowa zamówień była dostarczana między 8:00 a 9:00.

### 3.1.4. Analiza trendu (moving average) dla średniego czasu dostawy danego dnia



Można raczej powiedzieć, że trend jest mniej więcej stały.

### 3.1.5. Analiza sezonowości dla średniego czasu dostawy danego dnia



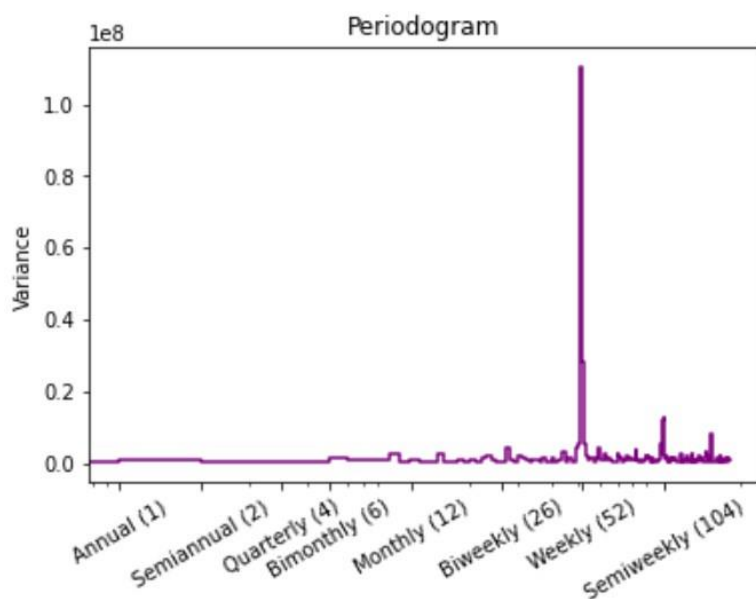
Powyżej widzimy, że dzień tygodnia ma znaczenie - na przykład z czwartego na piąty dzień tygodnia prawie zawsze widać znaczny spadek średniego czasu dostawy.

Natomiast na wykresie przedstawiającym okres roczny ciężko znaleźć jakieś zależności.

Wykryta sezonowość na przestrzeni tygodnia zostanie uwzględniona w modelu za pomocą wskaźników (indicators) poprzez one-hot encoding.

Warto dodać, że na powyższym wykresie widać, że średnia wartość time\_diff dla kolejnych dni mocno skacze góra-dół. Być może są tam jakieś cykle, które uda się wykryć w kolejnym punkcie.

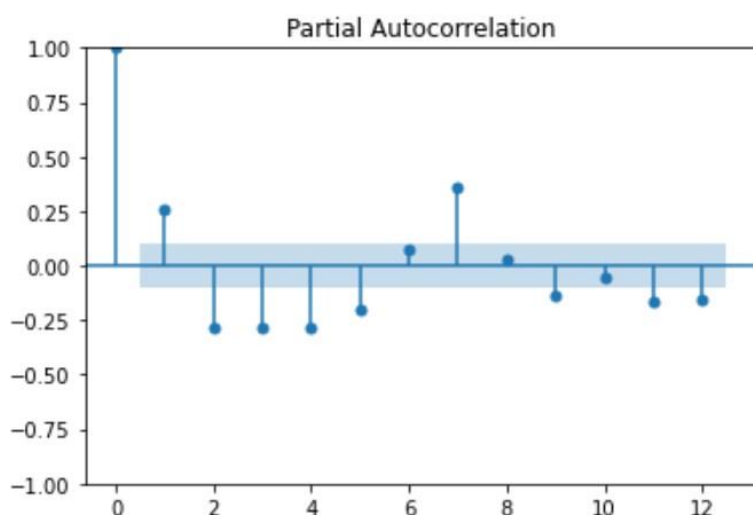
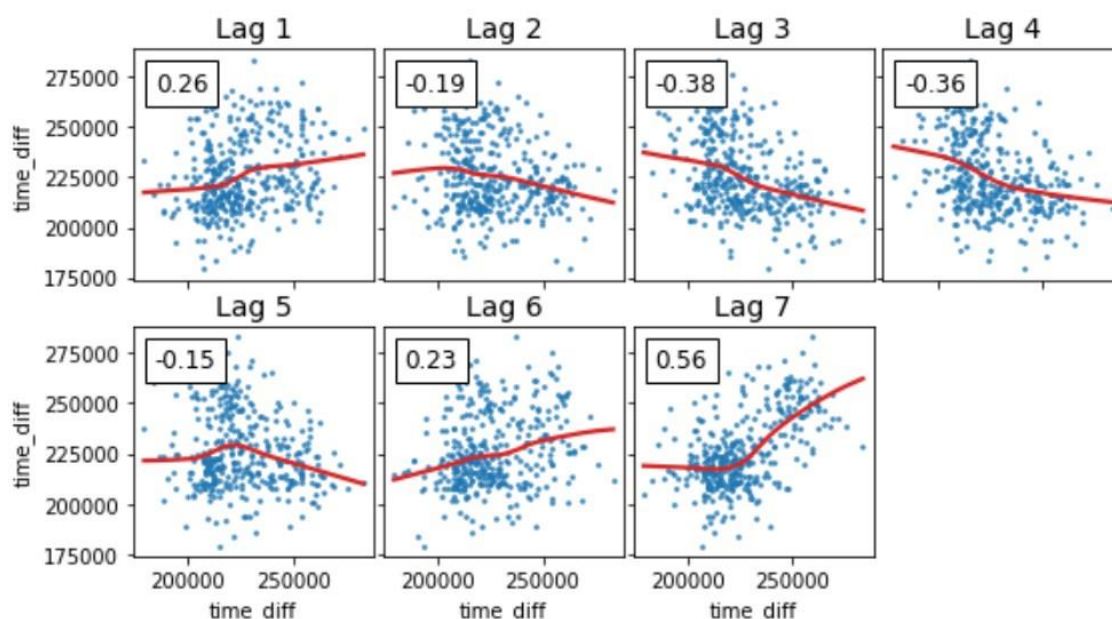
### 3.1.6. Periodogram



Zgodnie z wcześniejszymi spostrzeżeniami na powyższym wykresie widzimy silną zależność tygodniową. Widać także brak silnych zależności w przypadku okresów dłuższych - takich jak miesiąc czy rok.

Nie jest natomiast pewne czy udzielimy to w modelu. Nie mamy pewnego pomysłu jak to zrobić - być może by wymagało to dodatkowego modelu, który przewiduje średni czas dostawy danego dnia. Poza tym zależność tygodniowa prawdopodobnie jest skorelowana z tym, jaki jest dzień tygodnia (poprzedni punkt), co zostanie zamodelowane (punkt 3.1.5.).

### 3.1.7. Badanie zależności ze średnim czasem dostaw z poprzednich dni



Jak widać występują korelacje z czasami dostaw sprzed 3 lub 4 dni. Co ciekawe średni czas dostaw danego dnia jest najsilniej skorelowany z tym co było tydzień wcześniej.

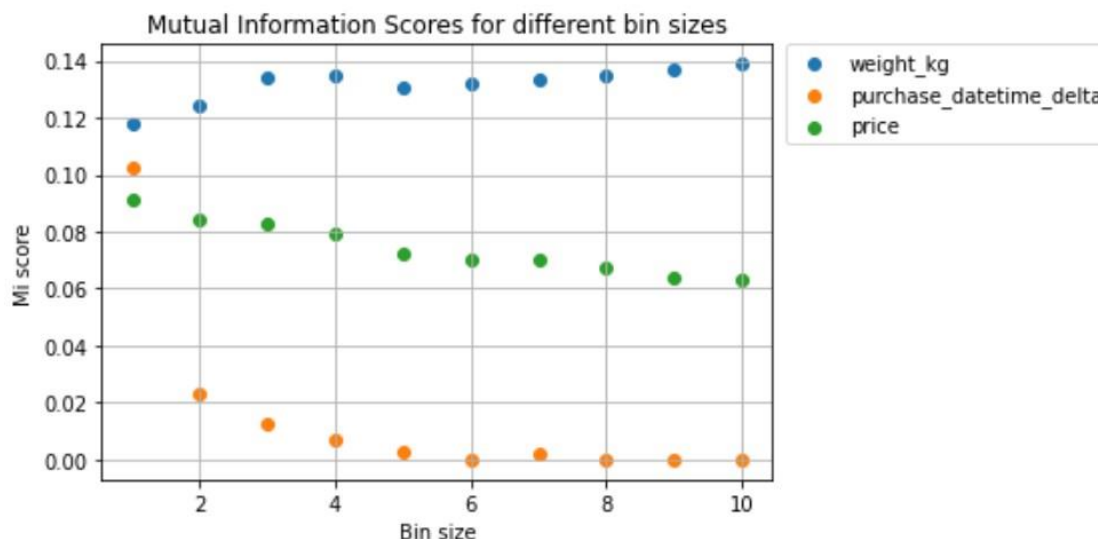
Natomiast korelacje ze średnimi czasami dostaw z poprzednich dni też prawdopodobnie odpuścimy, ponieważ musimy pamiętać o tym, że są to korelacje ze średnim czasem dostawy danego dnia, co musiałoby się pośrednio przenieść na przewidywanie czasu dostawy dla konkretnego zamówienia.



### 3.1.8. Mutual information score dla *time\_diff*

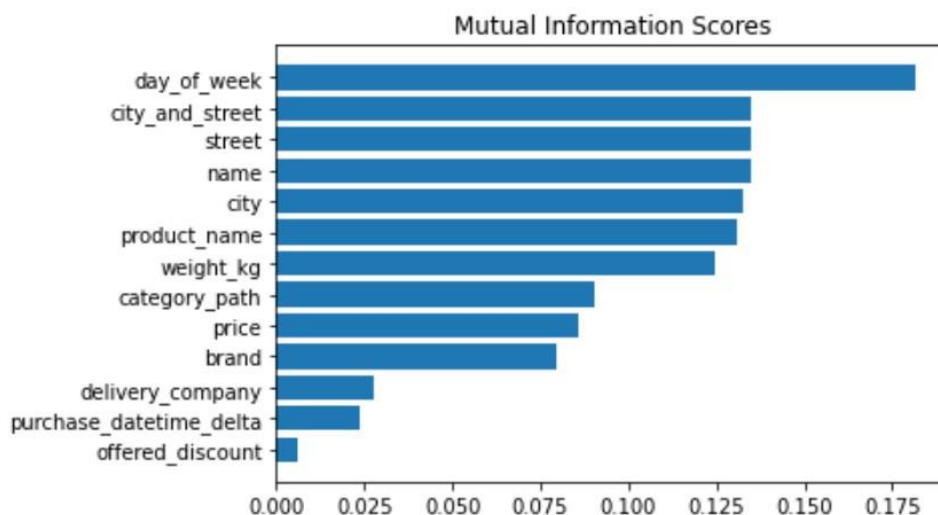
Do wyliczenia mutual information score dla *time\_diff* korzystamy z `sklearn.feature_selection.mutual_info_regression`, ponieważ jest to zmienna ciągła.

Najpierw ustaliliśmy jednak rozmiary kubełków dla zmiennych ciągłych, dla których mutual information score będzie wyliczany:



Jak widać *weight\_kg* najlepiej wypada dla *bin\_size* = 10, a *purchase\_datetime\_delta* oraz *price* najlepiej wypadają dla *bin\_size* = 2.

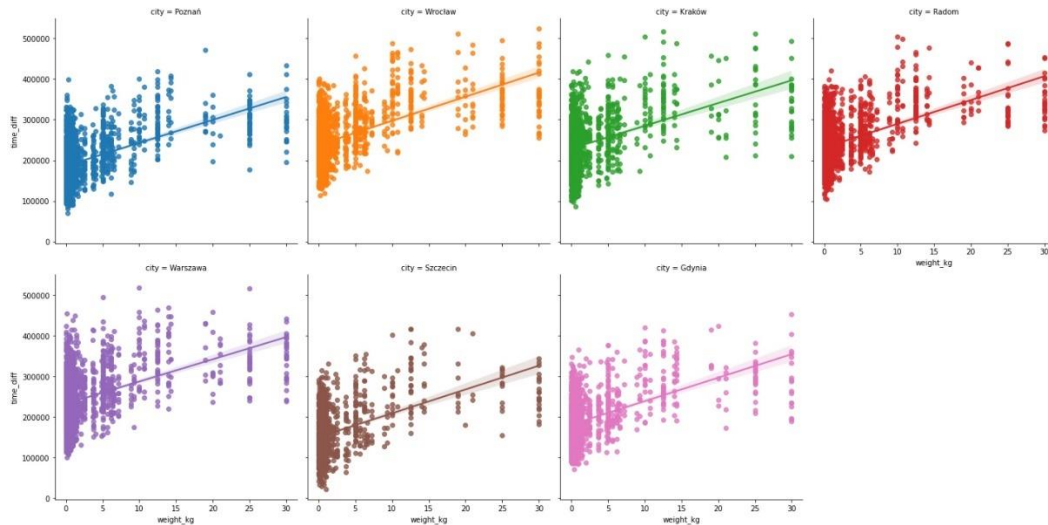
Teraz możemy przedstawić współczynniki informacji wzajemnej dla wszystkich atrybutów jednocześnie:



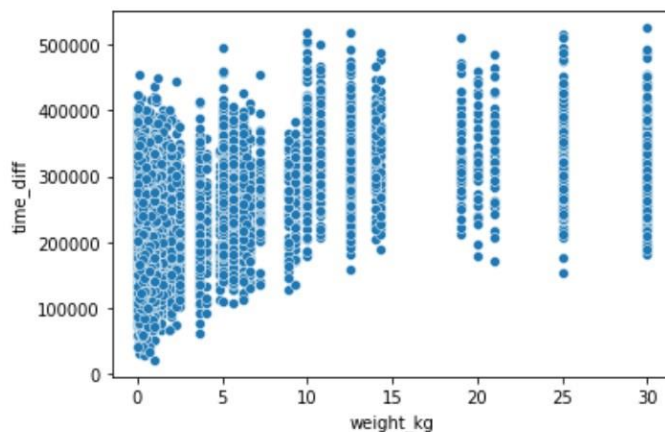
- zgodnie z oczekiwaniami street, name i user\_id są prawdopodobnie nadmiarowe i wystarczy wybrać jeden z tych atrybutów



- city jest istotnym atrybutem, co widać także na poniższych wykresach:



- product\_name, product\_id i weight\_kg być może także są redundantne. Zostanie to sprawdzone dalej.
- weight\_kg na pewno będzie brane pod uwagę przez wcześniej wykrytą korelację liniową:



- decyzje co do pozostałych atrybutów zostaną podjęte później

Ostatecznie po przeanalizowaniu powyższych wniosków oraz testowaniu modeli dla różnych atrybutów, zdecydowaliśmy się na następujący zestaw atrybutów:

- day\_of\_week*
- city*
- street*
- weight\_kg*
- price*
- delivery\_company*
- purchase\_timestamp\_delta*

### 3.2. Przygotowanie danych do modelowania

Przygotowanie danych realizowane jest przez kod w pliku *TimeDiffDataTransformer.py*.

1. Łączone są 4 tabele wejściowe.
2. Wyliczana jest kolumna *time\_diff*.
3. Tworzona jest interakcja *city\_and\_street*, która łączy kolumny *city* oraz *street*:  
`df['city_and_street'] = df['city'] + ' ' + df['street']`
4. Dodanie kolumny *purchase\_datetime\_delta*, która zawiera informację o tym, ile minut minęło od pierwszego w danych *purchase\_timestamp*:  
`df['purchase_datetime_delta'] = (df['purchase_timestamp'] - df['purchase_timestamp'].min()) / np.timedelta64(1, 'D')`
5. Następnie po selekcji atrybutów usuwane są niepotrzebne kolumny.
6. Dla atrybutów ciągłych realizowany jest one-hot encoding oraz normalizacja minmax.

## 4. Ostateczny model

Początkowo planowaliśmy wykorzystać do tego regresję liniową (dokładniej [Ridge Regression z wbudowaną walidacją krzyżową](#)). Był to nasz model bazowy.

Jako drugi model zdecydowaliśmy się na las losowy ([RandomForestRegressor](#)).

Otrzymaliśmy następujące wyniki:

```
RidgeCV score = 0.4603548930522081, training time = 0.49544334411621094s
RandomForestRegressor score = 0.5439410349642648, training time = 15.860059976577759s
number of good predictions for RidgeCV = 2244/2464
which is 91.07142857142857% for +-24 hours
```

```
number of good predictions for RandomForestRegressor = 2288/2464
which is 92.85714285714286% for +-24 hours
```

```
number of good predictions for RidgeCV = 1508/2464
which is 61.201298701298704% for +-12 hours
```

```
number of good predictions for RandomForestRegressor = 1711/2464
which is 69.43993506493507% for +-12 hours
```

Wnioski są następujące:

1. Jeden model nie jest jednoznacznie lepszy od drugiego, ponieważ:
  - a. Regresja liniowa jest trenowana około 30 razy szybciej od lasu losowego
  - b. Las losowy daje bardziej szczegółowe predykcje (mniej uśrednione), przez co ma lepszy procent dobrych predykcji w zakresie + 12 godzin.

Nasz mikroserwis umożliwia więc realizację testów A/B, co zostało opisane pod koniec następnego rozdziału.

## 5. Mikroserwis

W ramach naszego zadania zaprojektowaliśmy mikroserwis, pozwalający na obsługę zewnętrznych żądań i uzyskiwanie predykcji z wytrenowanych modeli.

Aplikacja została napisana w języku Python we frameworku Flask. Predykcje serwowane są żądaniom typu POST, które zawierają zestaw pól nazwanych tak jak kolumny w omawianych wcześniej tabelach.

Do skorzystania z aplikacji wymagane jest pobranie biblioteki *flask*, a sam mikroserwis uruchomić należy poleceniem:

*flask run*

Po uruchomieniu aplikacja rozpocznie trenowanie modeli na dostępnym zbiorze danych. Takie trenowanie trwa zazwyczaj około minuty. Po upływie tego czasu aplikacja gotowa jest do odbierania żądań typu POST i zwracania predykcji, czyli przewidywanej daty dotarcia zamówienia do klienta. Domyślnym portem działania aplikacji Flask jest port 5000.

Nasz mikroserwis umożliwia także realizację eksperymentów A/B. W tym celu dokonuje logowania – zapisu zarówno żądań, jak i zwracanych predykcji do plików .csv. Pliki te można znaleźć w folderze *logs*:

*Requests.csv* - zawiera treści wysłanych requestów

*RandomForestPredictions.csv* - zawiera predykcje algorytmu Random Forest

*RidgeCVPredictions.csv* - zawiera predykcje algorytmu Ridge CV

Aby móc zidentyfikować, które predykcje odpowiadają którym żądaniom, wiersze każdego z plików opatrzone są identyfikatorem.

## 6. Mikroserwis - testy

W tej sekcji znajdują się materiały dowodzące, że implementacja działa.

Zaimplementowany mikroserwis przetestowaliśmy przy pomocy kilku przykładowych plików typu .json z danymi.

Pliki te dostępne są w folderze */example\_posts*.

Testy obsługi żądań zostały wykonane przy pomocy programu Postman.

Poniżej znajdują się zrzuty ekranów zawierające treści requestów oraz odpowiedzi serwera. Oprócz tego zamieściliśmy obrazki potwierdzające, że aplikacja uruchamia się i odbiera żądania.

### Uruchomienie aplikacji

```

PS C:\Users\user\Desktop\IUM_repo\ium-projekt> flask run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)

```

### Test 1 – plik example\_post.json

```

1  [{
2    ...."purchase_id": 20000,
3    ...."purchase_timestamp": "2021-11-07T06:15:13",
4    ...."delivery_company": 620,
5
6    ...."product_id": 1156,
7    ...."product_name": "Słuchawki nauszne Corsair HS35",
8    ...."category_path": "Mikrofony i słuchawki;Słuchawki przewodowe",
9    ...."price": 159.0,
10   ...."brand": "Corsair",
11   ...."weight_kg": 0.52,
12   ...."optional_attributes": {
13     ...."color": "czarny"
14   },
15
16   ...."session_id": 125,
17   ...."event_type": "BUY_PRODUCT",
18   ...."offered_discount": 15,
19   ....
20   ...."user_id": 102,
21   ...."name": "Monika Forysiak",
22   ...."city": "Poznań",
23   ...."street": "plac Dębowa 11/53"
24   ....
25  }]

```

Odpowiedź mikroservisu:

Body Cookies Headers (5) Test Results 🌐 Status: 200 OK Time: 1237 ms Size: 285 B

Pretty Raw Preview Visualize JSON 🔍

```

1  [
2    "RandomForestRegressor prediction": "Tue, 09 Nov 2021 18:48:17 GMT",
3    "Ridge prediction": "Tue, 09 Nov 2021 21:41:22 GMT"
4  ]

```

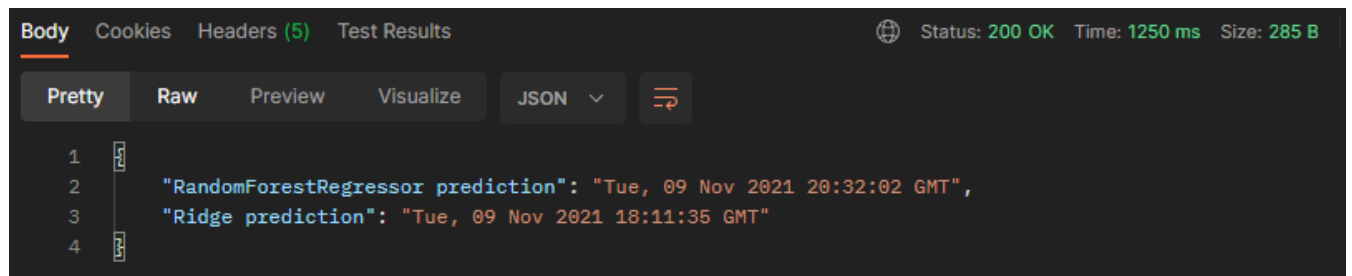
### Test 2 – plik example\_post2.json

```

1  [{
2    ...."purchase_id": 20000,
3    ...."purchase_timestamp": "2021-11-07T06:15:13",
4    ...."delivery_company": 516,
5
6    ...."product_id": 1156,
7    ...."product_name": "Słuchawki nauszne Corsair HS35",
8    ...."category_path": "Mikrofony i słuchawki;Słuchawki przewodowe",
9    ...."price": 159.0,
10   ...."brand": "Corsair",
11   ...."weight_kg": 0.52,
12   ...."optional_attributes": {
13     ...."color": "czarny"
14   },
15
16   ...."session_id": 125,
17   ...."event_type": "BUY_PRODUCT",
18   ...."offered_discount": 10,
19   ....
20   ...."user_id": 223,
21   ...."name": "Ewa Dynak",
22   ...."city": "Wrocław",
23   ...."street": "plac Fiołkowa 95/00"
24   ....
25  }]

```

Odpowiedź mikroserwisu:



The screenshot shows the 'Body' tab of a web browser's developer tools. The response is a JSON object with two keys: 'RandomForestRegressor prediction' and 'Ridge prediction', both containing timestamps. The status is 200 OK, the time is 1250 ms, and the size is 285 B.

```

1  {
2    "RandomForestRegressor prediction": "Tue, 09 Nov 2021 20:32:02 GMT",
3    "Ridge prediction": "Tue, 09 Nov 2021 18:11:35 GMT"
4  }

```

Test 3 – plik example\_post3.json

```

1  [{
2    ...."purchase_id": 20000,
3    ...."purchase_timestamp": "2021-11-14T12:22:13",
4    ...."delivery_company": 620,
5
6    ...."product_id": 1368,
7    ...."product_name": "Skaner Brother ADS-2400N",
8    ...."category_path": "Skanery;Profesjonalne",
9    ...."price": 680.0,
10   ...."brand": "Brother",
11   ...."weight_kg": 6.2,
12   ...."optional_attributes": {},
13
14   ...."session_id": 124,
15   ...."event_type": "BUY_PRODUCT",
16   ...."offered_discount": 5,
17   ....
18   ...."user_id": 102,
19   ...."name": "Monika Forysiak",
20   ...."city": "Wrocław",
21   ...."street": "plac Dębowa 11/53"
22   ....
23  }]

```

Odpowiedź mikroserwisu:

Body
Cookies
Headers (5)
Test Results

Status: 200 OK
Time: 1310 ms
Size: 285 B

Pretty
Raw
Preview
Visualize
JSON

```

1  {
2    "RandomForestRegressor prediction": "Wed, 17 Nov 2021 20:38:35 GMT",
3    "Ridge prediction": "Wed, 17 Nov 2021 20:24:19 GMT"
4  }

```

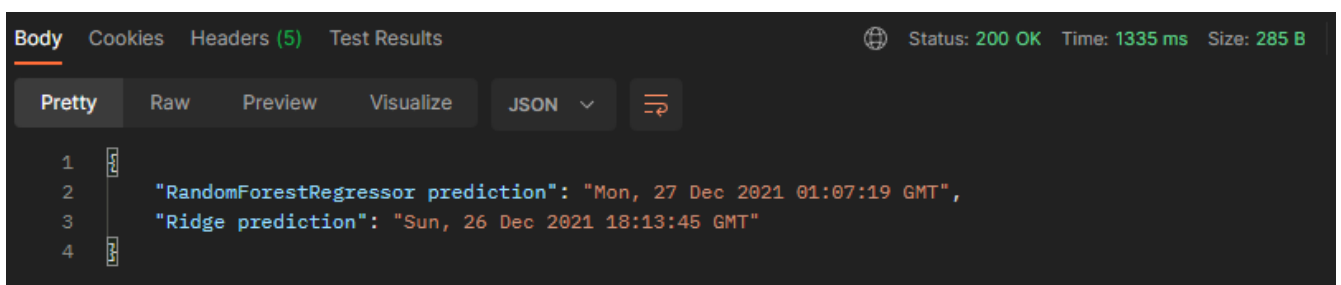
Test 4 – plik example\_post4.json

```

1  [{
2    ...."purchase_id": 20000,
3    ...."purchase_timestamp": "2021-12-24T02:15:13",
4    ...."delivery_company": 620,
5
6    ...."product_id": 1156,
7    ...."product_name": "Słuchawki nauszne Corsair HS35",
8    ...."category_path": "Mikrofony i słuchawki;Słuchawki przewodowe",
9    ...."price": 159.0,
10   ...."brand": "Corsair",
11   ...."weight_kg": 0.52,
12   ...."optional_attributes": {
13     ...."color": "czarny"
14   },
15
16   ...."session_id": 125,
17   ...."event_type": "BUY_PRODUCT",
18   ...."offered_discount": 0,
19   ....
20   ...."user_id": 102,
21   ...."name": "Monika Forysiak",
22   ...."city": "Poznań",
23   ...."street": "plac Dębowa 11/53"
24   ....
25  }]

```

Odpowiedź mikroserwisu:



```

Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON
1  {
2    "RandomForestRegressor prediction": "Mon, 27 Dec 2021 01:07:19 GMT",
3    "Ridge prediction": "Sun, 26 Dec 2021 18:13:45 GMT"
4  }

```

Zrzut ekranu logów aplikacji po obsłużeniu jednego z żądań:

```

Predictions    y_test  RidgeCV prediction  RandomForestRegressor prediction
0      0.0      215782.628909      217176.64
Predicted delivery (Random Forest):  2021-11-09 18:34:49.640000
Predicted delivery (Ridge CV):  2021-11-09 18:11:35.628909
127.0.0.1 - - [03/Jun/2022 22:30:29] "POST / HTTP/1.1" 200 -

```

## 7. Repozytorium kodu



Implementacje algorytmów oraz wykorzystywane dane będziemy umieszczać w repozytorium kodu Gitlab, do którego odnośnik znajduje się poniżej:

<https://gitlab-stud.elka.pw.edu.pl/mlopatto/ium-projekt>