Wydział Elektroniki
i Technik Informacyjnych

POLITECHNIKA WARSZAWSKA

Raport z wykonania laboratorium z Inżynierii Uczenia Maszynowego

EShopping - Przewidywanie czasu dostawy

Autor:
Zuzanna Santorowska
Magdalena Majkowska

Opiekun:
dr inż. Paweł Zawistowski

6 sierpnia 2021

1 Zadanie Biznesowe

1.1 Obecna sytuacja

W firmie eSzoppping nie ma informacji dotyczących czasu realizacji dostawy - występują opóźnienia w realizacji zamówienia, co negatywnie wpływa na zadowolenie klientów.

1.2 Co należy zmienić?

Implementacja systemu, który na podstawie danych historycznych i zakupionych produktów umożliwia przewidzenie czasu dostawy. Rozwiązanie będzie wdrożone na serwerze sklepu internetowego i przewidywania będą dostępne dla pracowników i wyświetlane przy finalizacji zamówienia.

1.3 Jakie właściwości powinno mieć docelowe rozwiązanie?

- założenia i oczekiwania
  - System będzie utworzony w języku programowania Python
  - System będzie przenośny pomiędzy platformami
  - System będzie dostarczał przewidywania czasu dostaw z nie większym błędem niż 24h
  - System będzie łatwy w utrzymaniu i będzie umożliwiał poprawę i rozbudowę w przyszłości
  - Projekt musi zostać zakończony do 06.06.2021
- zasoby:
  - W firmie dostępna jest infrastruktura potrzebna do uruchomienia systemu i pozwalająca na cyklicznie pobieranie nowych danych uczących

1.4 Biznesowe kryteria sukcesu

- Dzięki wprowadzeniu systemu o 90% ilość negatywnych opinii dotyczących dostawy.
- System pozwoli na poprawienie stosunków z klientami oraz zyskanie nowych - dzięki dobrym przewidywaniom ci klienci, którzy byli niezadowoleni z dostawy mogą ponownie zacząć kupować

2 Zdefiniowanie Zadania Modelowania

2.1 Mapowanie Zadania Domenowego na Analityczne

System będzie realizowany jako zadanie regresji. Decyzja projektowa dotycząca wyboru modelu analitycznego jest pokierowana tym, że nie zależy nam na zbyt dużej dokładności modelu - z punktu widzenia klienta sklepu internetowego interesuje nas przybliżony a nie dokładny czas dostawy. Ponadto czas dostawy jest wynikiem bardzo wielu czynników, na które nie mamy bezpośrednio wpływu, dlatego niemożliwe jest, żeby wyliczone wartości zgadzały się z rzeczywistością.

2.1.1 Analityczne kryteria sukcesu

- Prawidłowa predykcja czasu dostaw na poziomie wyższym niż 50% - jest to wartość osiągana przez prosty model regresji liniowej.

3 Wymagania Funkcjonalne

1. System podczas finalizacji transakcji dostarcza klientowi sklepu informacji dotyczącej szacowanego czasu dostawy.

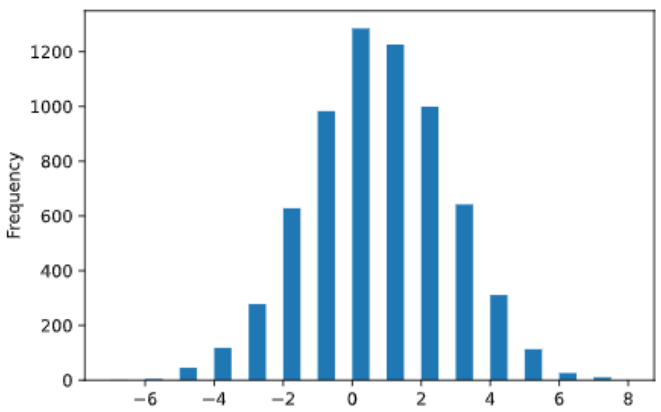
4 Wymagania Niefunkcjonalne

1. System umożliwia przewidywanie długości dostawy z dokładnością do 24h.
2. System będzie w łatwy sposób można rozbudować o inne moduły i nie będzie sprawiał problemów w późniejszym utrzymaniu.
3. System będzie uczył się nie dłużej niż 15 min.
4. System będzie dostarczał informacji na temat długości dostawy w 99% przypadków w czasie mniejszym niż 1s.

5 Analiza otrzymanych danych

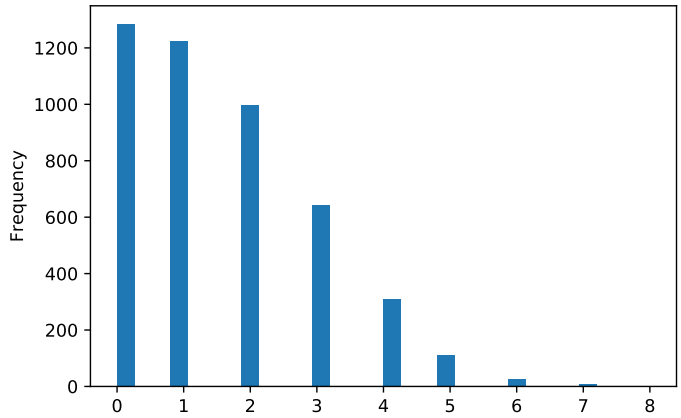
5.1 Napotkane problemy i ich rozwiązania

- Kodowanie danych - polskie znaki diakrytyczne w pierwotnej wersji danych były zapisane w surowej formie tzn.'not escaped'. Dlatego przy czytaniu ich z pliku zastosowałyśmy funkcję escape z pakietu *html*.
- Ujemne ceny produktów - dla niektórych produktów w zbiorze danych ceny były ujemne. Podczas analizy udało nam się jednak zauważyć pewną prawidłowość: gdyby wziąć wartość absolutną cen, nie odstawałyby one od reszty cen produktów z tej samej kategorii. Podzieliłyśmy się naszymi spostrzeżeniami z klientem, który potwierdził ich słuszność i od tej pory w programie używałyśmy wartości absolutnej cen.
- Ujemne czasy dostaw - w przypadku aż 31% wierszy początkowego zestawu danych zauważyłyśmy, że czas dostawy, liczony jako odjęcie od siebie czasu dostarczenia i zakupu jest ujemny. Ze względu na wielkość problemu stanowiłoby to duży problem w trenowaniu modelu, ponieważ podawałby ujemne predykcje dla czasu dostawy. Podejrzewałyśmy, że znacznik czasowy dostawy i zakupu zostały zamienione ze sobą podczas procesu zbierania danych. Postanowiłyśmy przyjrzeć się bliżej rozkładowi tej zmiennej:

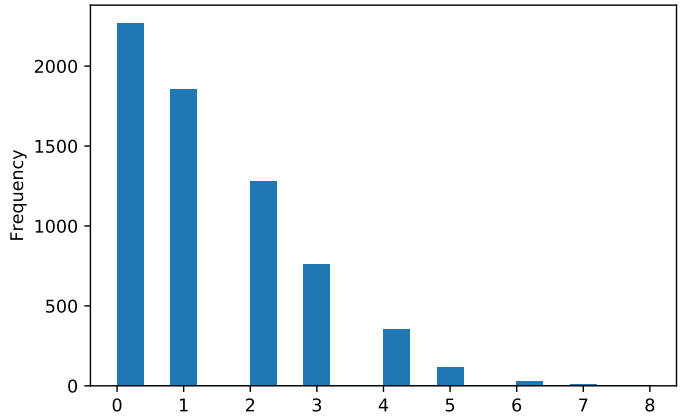


Rysunek 1: Rozkład zmiennej czasu dostawy

Przypomina on rozkład normalny z wartością oczekiwaną około 0. Sprawdziliśmy czy gdy użyjemy wartości absolutnej to rozkład będzie w jakiś sposób charakterystyczny np: czy co któryś element jest taki itp.



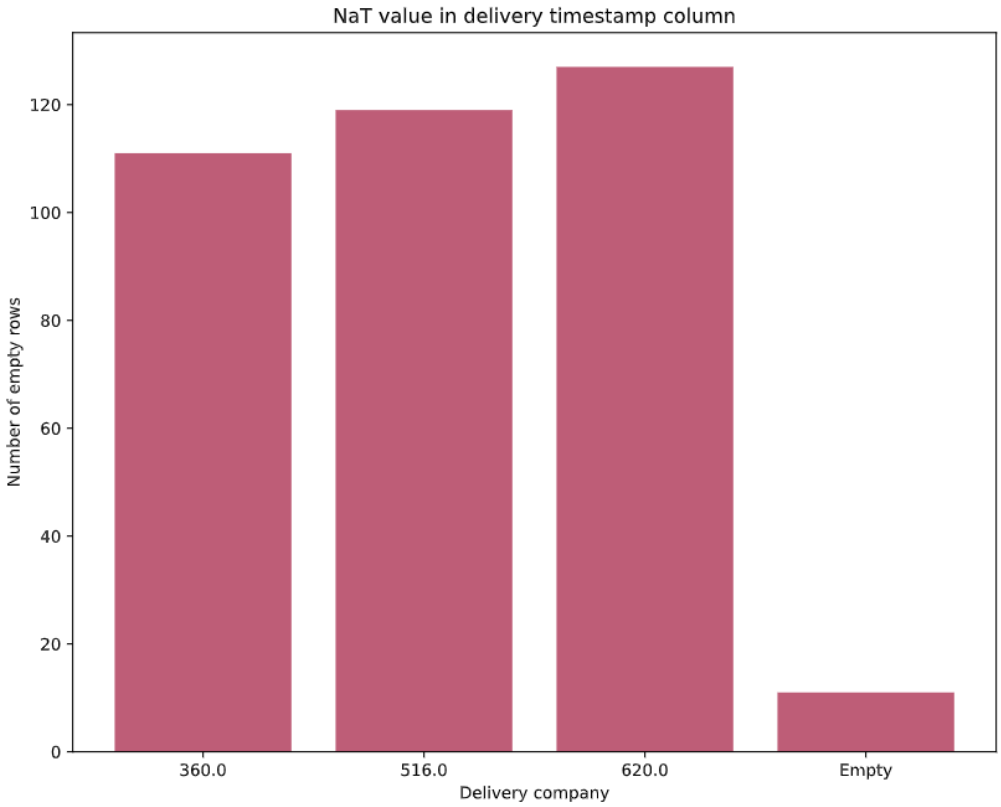
Rysunek 2: Rozkład zmiennej wartości większej od 0 czasu dostawy



Rysunek 3: Rozkład zmiennej czasu dostawy po wzięciu wartości absolutnej

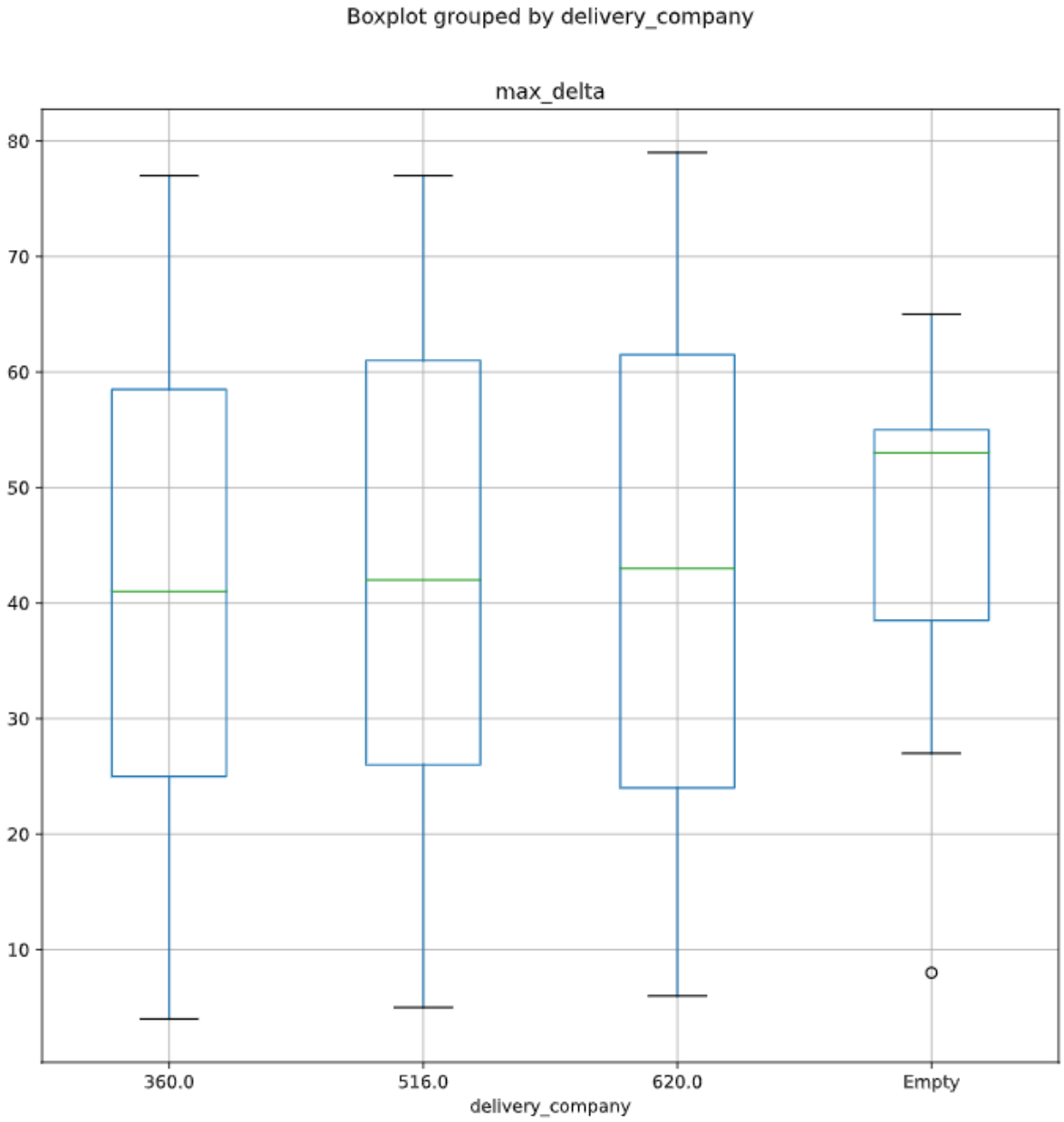
Sądziliśmy że problemy mogą wynikać z wpisania daty dostarczenia i zamówienia na odwrót. Po kontakcie z klientem nasze przypuszczenia okazały się słuszne i otrzymałyśmy nowy zestaw danych, którego analizie poświęcony jest oddzielny rozdział dokumentacji.

- Kategorie produktów w postaci wieloczłonowej zmiennej tekstowej - wydzieliłyśmy każdy człon kategorii na oddzielną kolumnę w ramce danych za pomocą funkcji *split()*. Dzięki temu mamy dużo łatwiejszy dostęp do poszczególnych kategorii i podkategorii, co znacznie ułatwiło późniejsze tworzenie wykresów eksploracyjnych.
- Zły typ podczas wczytywania dat z pliku danych - przekonwertowanie na typ *datetime64*
- W danych było 368 rzędów które nie miały danych w delivery_timestamp i 353 gdzie 11 pokrywa się z tymi co nie miały w delivery_timestamp rzędy bez informacji o firmie dostawczej. Braki w wartościach delivery_timestamp:



Rysunek 4: Ilość wierszy bez wartości w kolumnie delivery_timestamp

Widać wyraźnie że nie jest to problem dotyczący większości zbioru danych - odsetek rzędów bez informacji o dostawcy lub o dacie dostawy stanowi 5% całości. Postanowiłyśmy się temu jednak przyjrzeć bliżej. Początkowo pomyślałyśmy, że to mogą być po prostu przesyłki jeszcze niedostarczone. Dlatego znalazłyśmy dla każdego dostawy najpóźniejszą datę dostarczenia przesyłki i odjęłyśmy czas zakupu dla tych wierszy, które nie mają informacji w timestamp_delivery. Niestety nasze przewidywania okazały się nie być prawdziwe - dla każdego z dostawców nawet najniższe wartości przekraczają 8 dni (jest to wartość najwyższa w rozkładzie ogólnym dostaw). Zatem te przesyłki albo nie zostały nigdy dostarczone, albo wydarzyło się coś złego po stronie dostawców - tzn. błąd w rejestrowaniu momentu dostarczenia. Przy okazji sprawdziłyśmy, że po stronie sklepu internetowego problem z rejestrowaniem nie występuje - tzn. nie ma w zbiorze danych wierszy, którym brakowałoby znacznika zakupu.

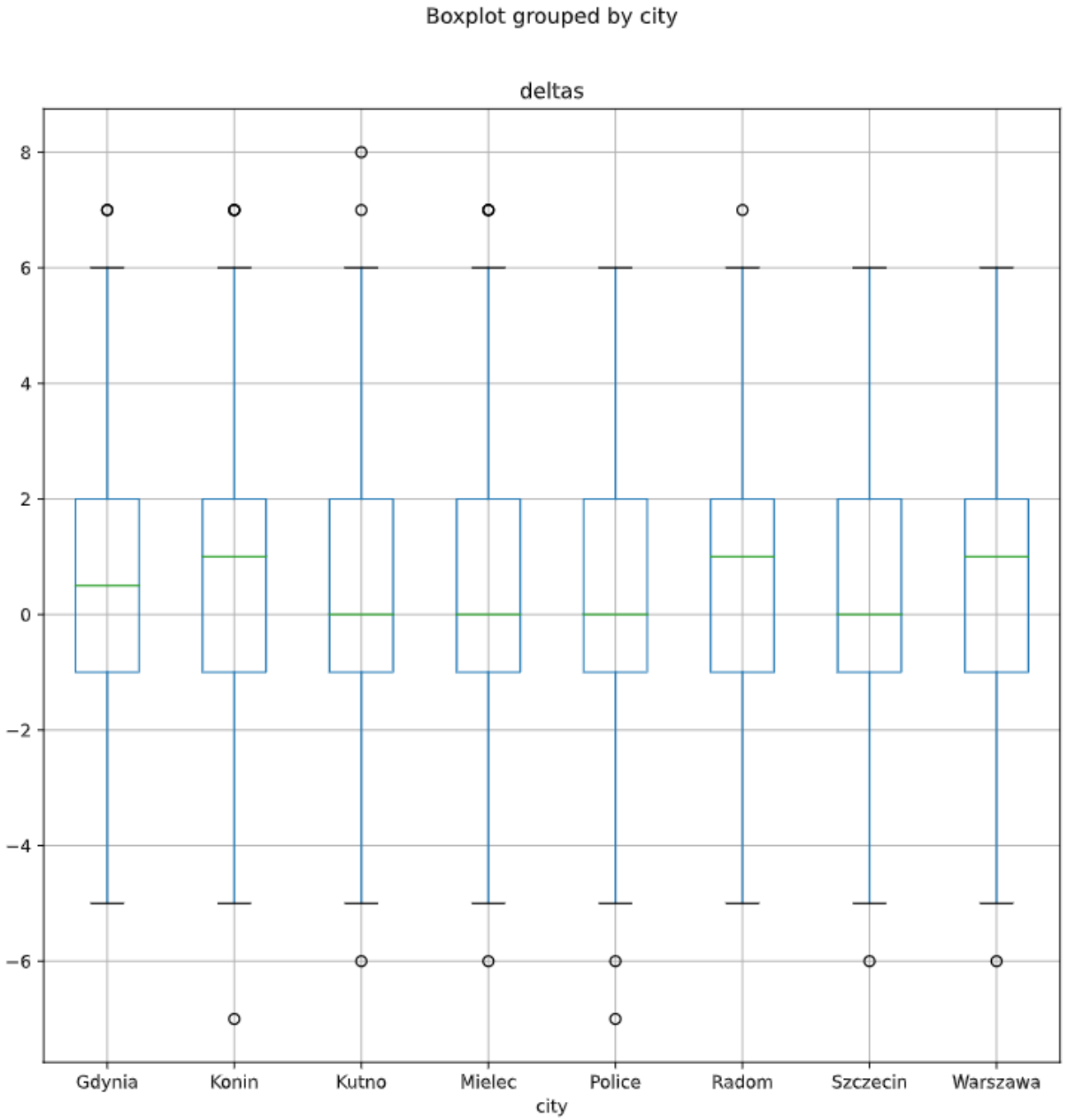


Rysunek 5: Boxplot przedstawiający rozkład długości dostaw

Ze względu na mały rozmiar problemu, brak widocznych powiązań pomiędzy pustymi wartościami, a resztą atrybutów oraz podobieństwo rozkładu tych atrybutów tych wierszy do tych kompletnych, ostatecznie postanowiłyśmy je usunąć ze zbioru danych.

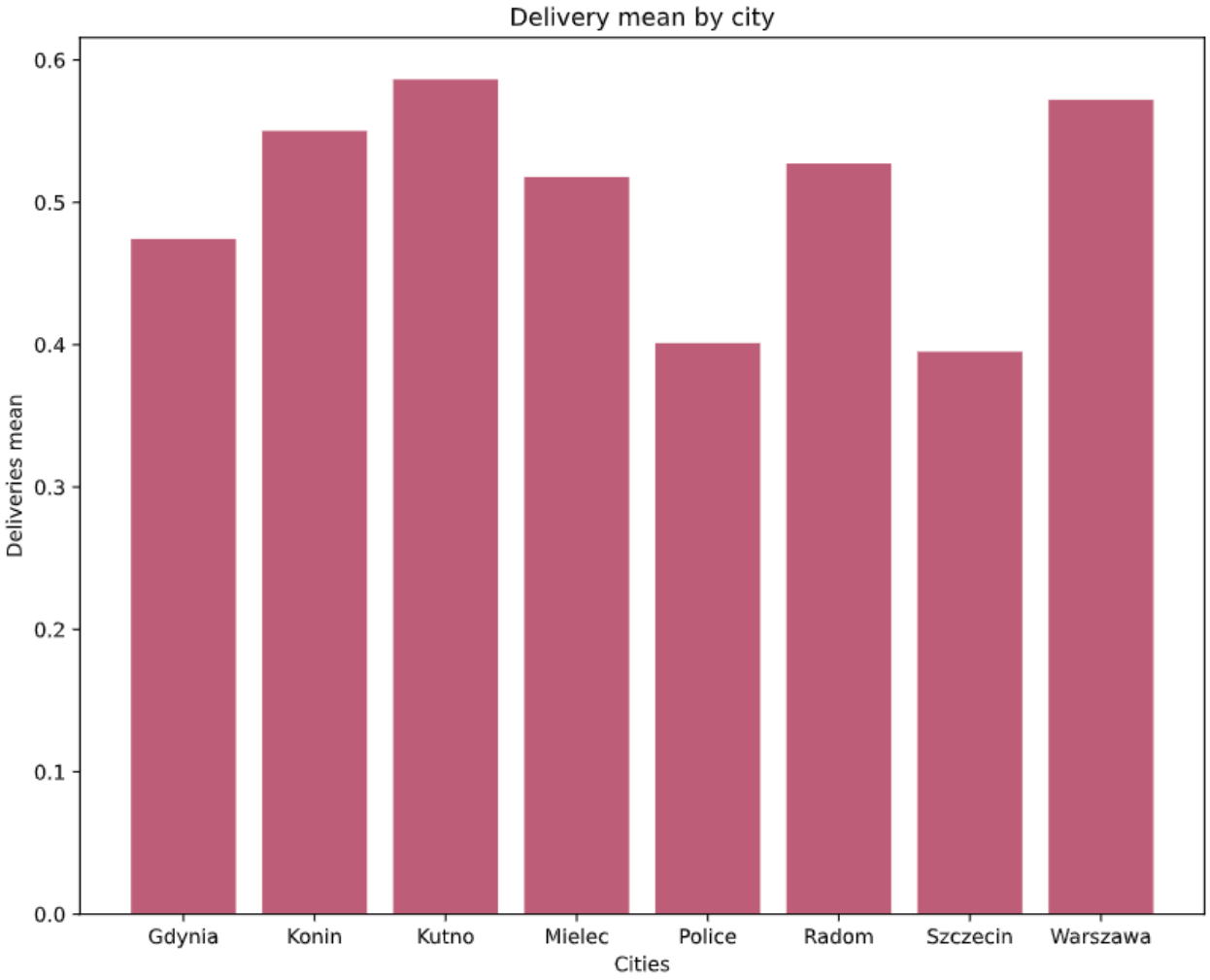
5.2 Analiza czasu dostaw

5.2.1 W zależności od miasta dostawy



Rysunek 6: Wykres obrazujący rozkład czasu dostaw w zależności od miasta dostawy

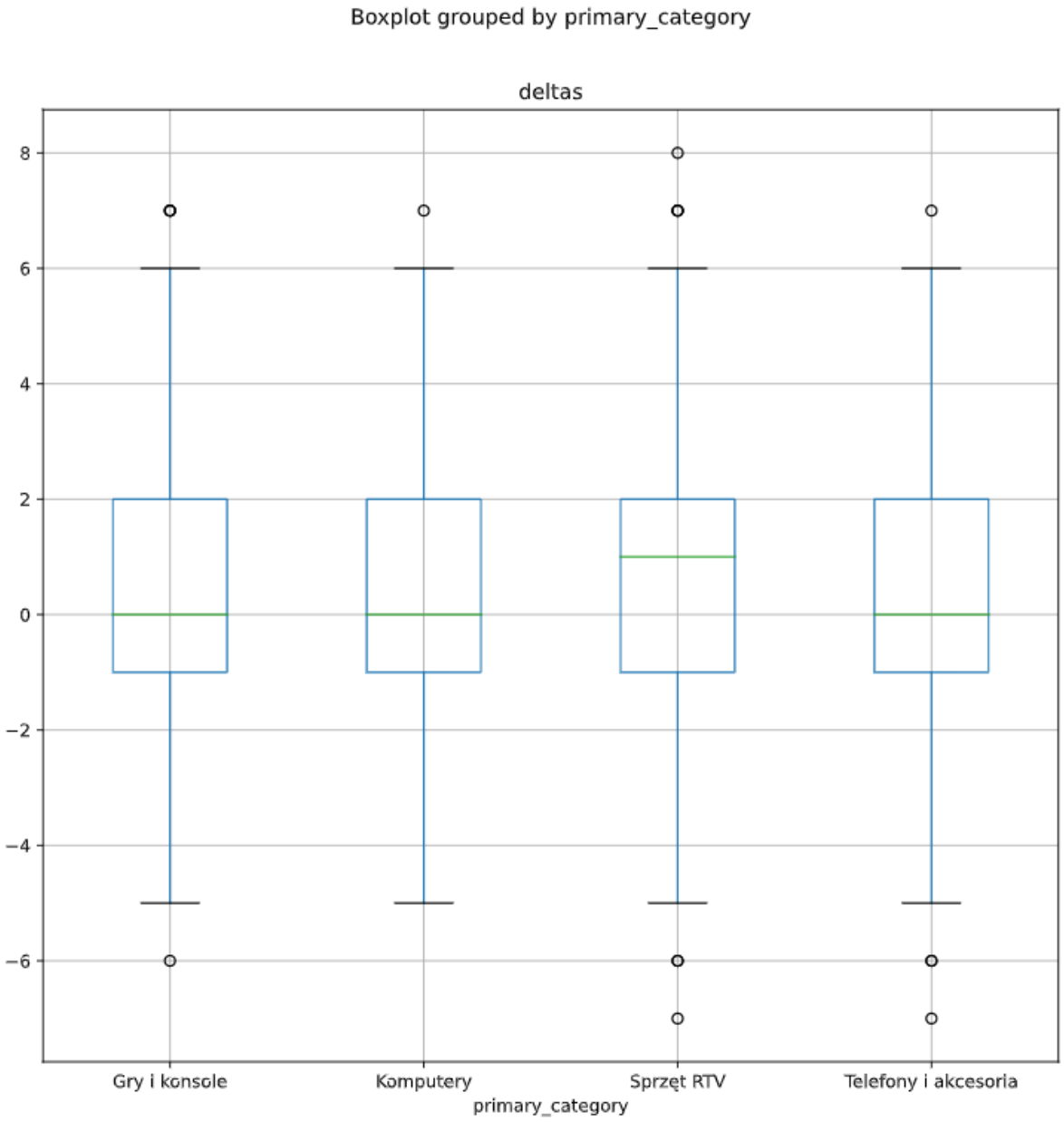
Z wykresu pudełkowego niewiele wynika - rozkłady są do siebie bardzo podobne, niemal identycznie i różnią się położeniem mediany oraz paroma wartościami odstającymi. (Możemy jednak wysnuć pewne domysły o tym jak generowane były dane do projektu :))



Rysunek 7: Wykres obrazujący średnią czasu dostaw w zależności od miasta dostawy

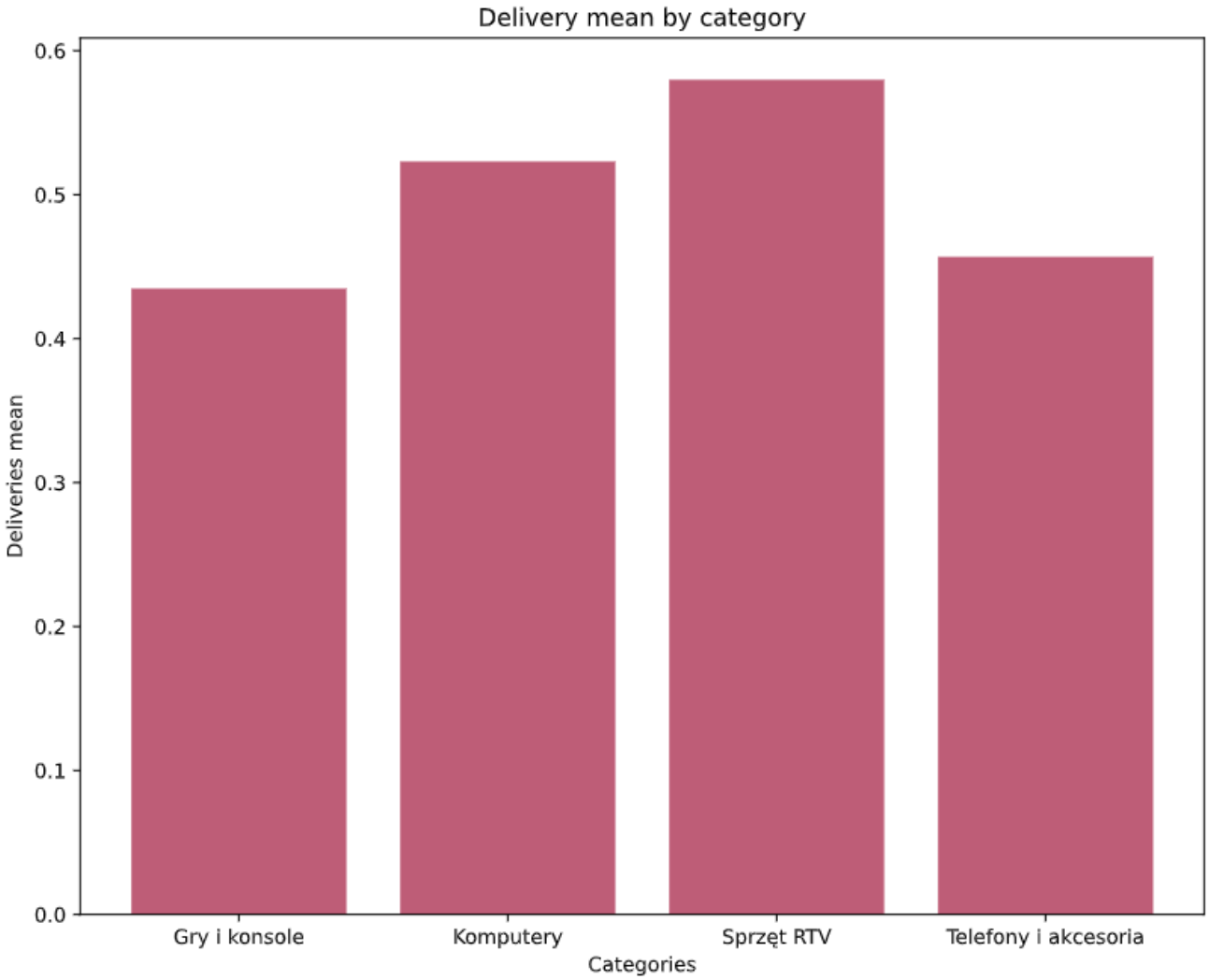
Wykres przedstawiający średnie czasów dostawy daje nam nieco lepszy ogląd na sytuację. Jednakże przedstawione w nim dane różną się od spodziewanych - zarówno dla dużych jak i dla małych miast czas dostawy potrafi być wysoki. Zaś najniższą średnią wartość dostawy obserwujemy dla Szczecina oraz Polic, które należą do najmniej ludnych spośród badanych.

5.2.2 W zależności od kategorii produktu



Rysunek 8: Wykres obrazujący rozkład czasu dostaw w zależności od kategorii produktu

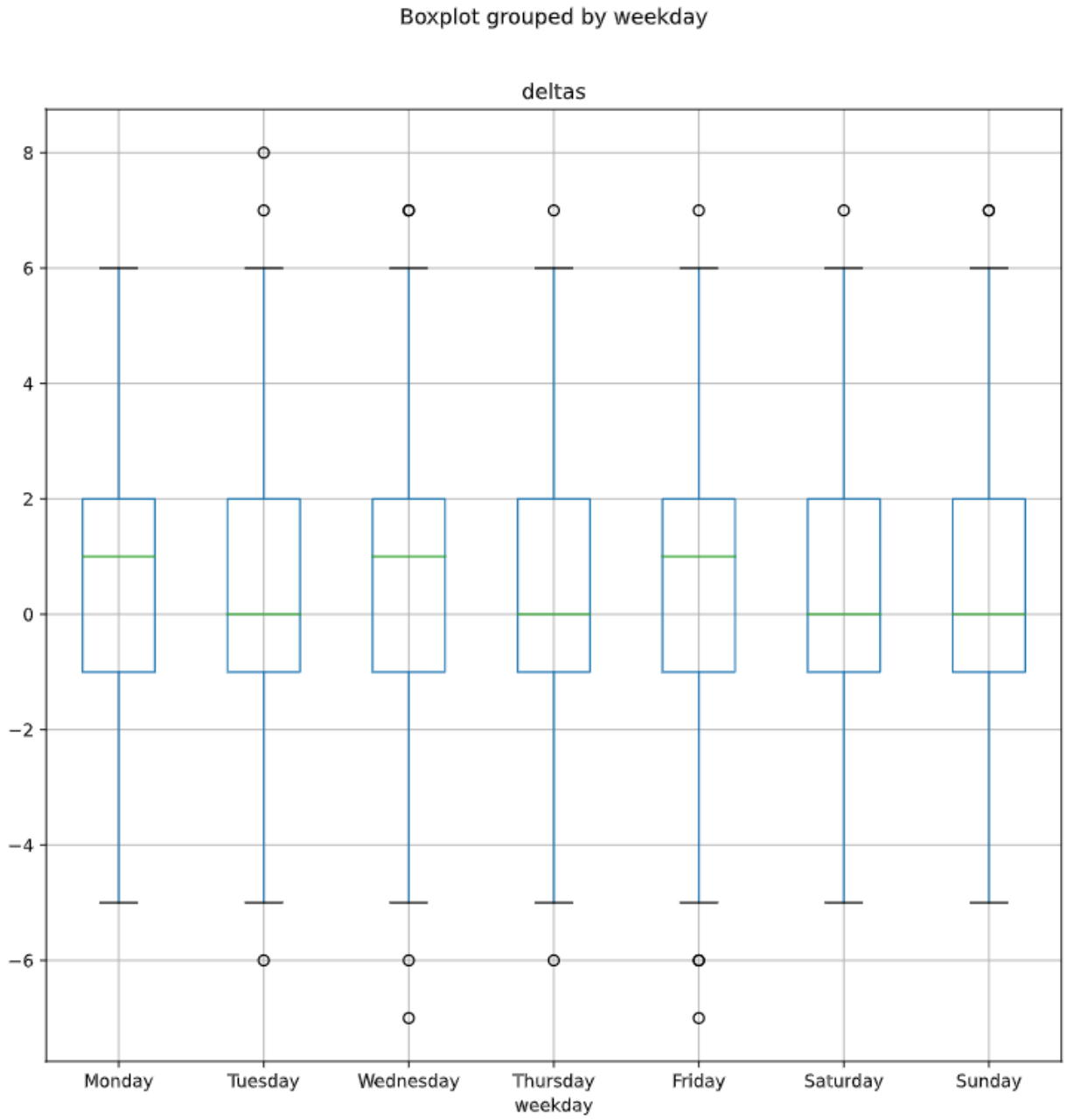
Na wykresie pudełkowym podobnie jak wcześniej nie obserwujemy zależności pomiędzy czasem dostawy a kategorią.



Rysunek 9: Wykres obrazujący średnią czasu dostaw w zależności od kategorii produktu

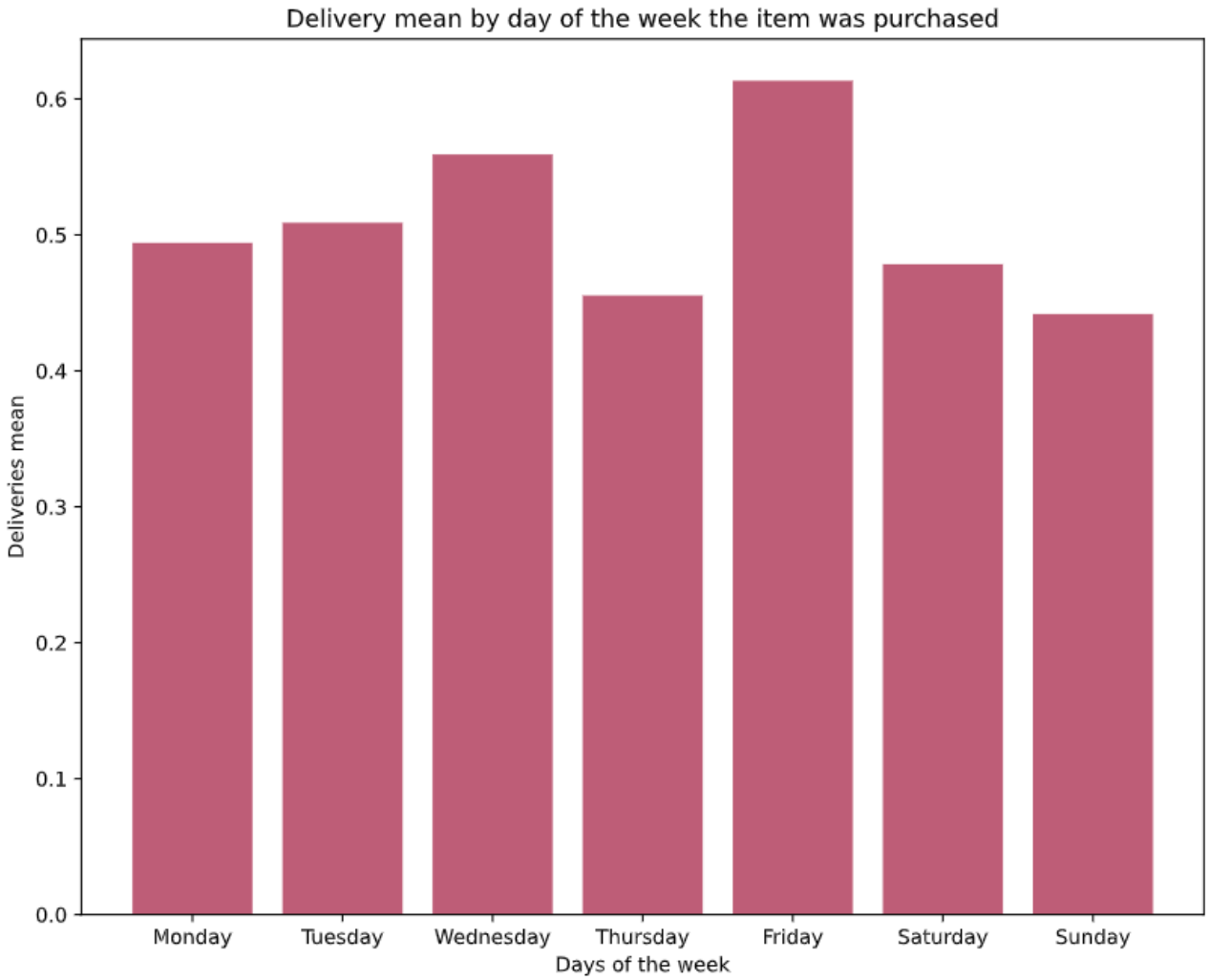
Na wykresie średnich możemy zauważyć, że im większy gabaryt produktu, tym dłuższy średni czas dostawy, co pokrywa się z przewidywaniami.

5.2.3 W zależności od dnia zakupu produktu



Rysunek 10: Wykres obrazujący rozkład czasu dostawy w zależności od dnia zakupu produktu

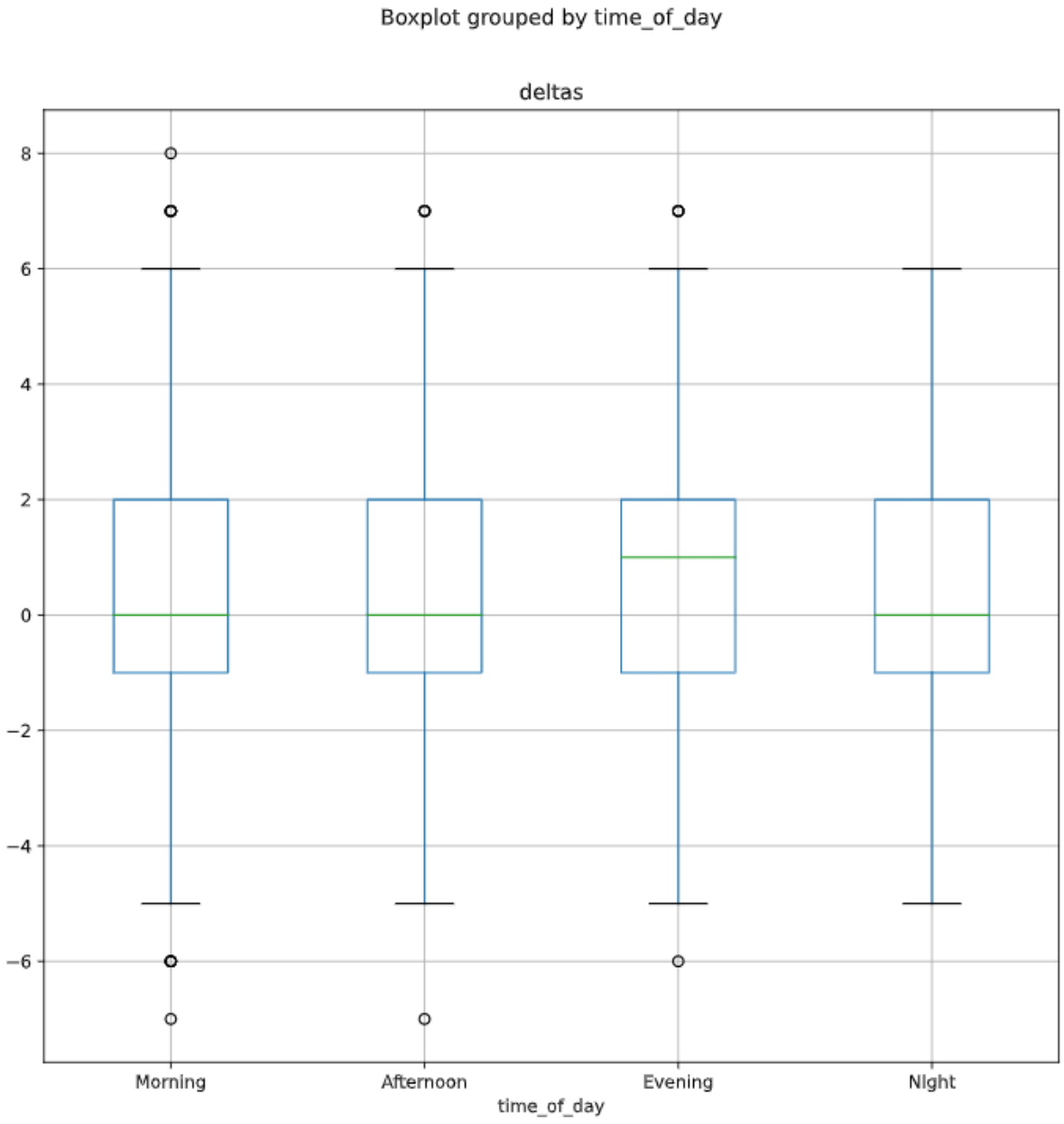
Z tego wykresu możemy wysnuć takie same wnioski co we wcześniejszych rozważaniach.



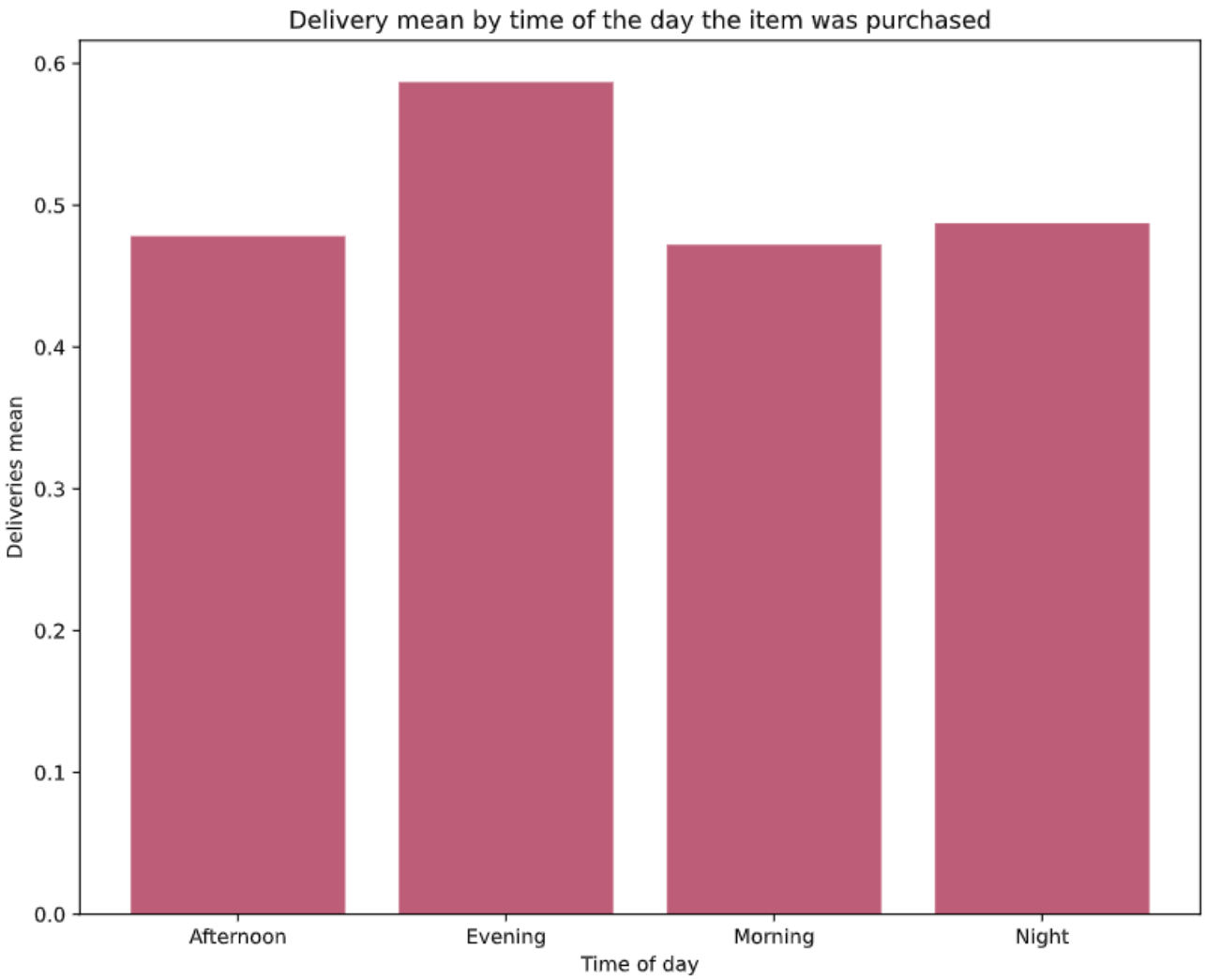
Rysunek 11: Wykres obrazujący średnią czasu dostaw w zależności od dnia zakupu produktu

Ponownie wykres przedstawiający średnie jest bardziej pomocny - najkrótsze dostawy mają produkty zakupione w weekend oraz na początku tygodnia (poniedziałek, wtorek), zaś im późniejszy dzień zakupu, tym dłuższa dostawa. Obserwujemy największą wartość średniej dla piątku, wydaje się to być logiczne - produkty zakupione w piątek mają szansę być dostarczone albo tego samego dnia, co jest nieprawdopodobne(bo oprócz samego przejazdu wysyłki na miejsce, należy ją spakować i nadać), albo na początku następnego tygodnia. Zatem do sumarycznego czasu wysyłki naliczone dodatkowe dni weekendu, gdy poczta nie dowozi przesyłek. Ciekawy jest nagły spadek średniego czasu dostawy w czwartek - jest on niewiele wyższy od niedzieli i odrobinę mniejszy niż dla soboty. Najprawdopodobniej jest to spowodowane pośpiechem w wysyłce, aby zdążyć przed weekendem, w przeciwnym wypadku czas dostawy znacznie się opóźnia, co wpłynie negatywnie na zadowolenie klienta.

5.2.4 W zależności od pory dnia zakupu produktu



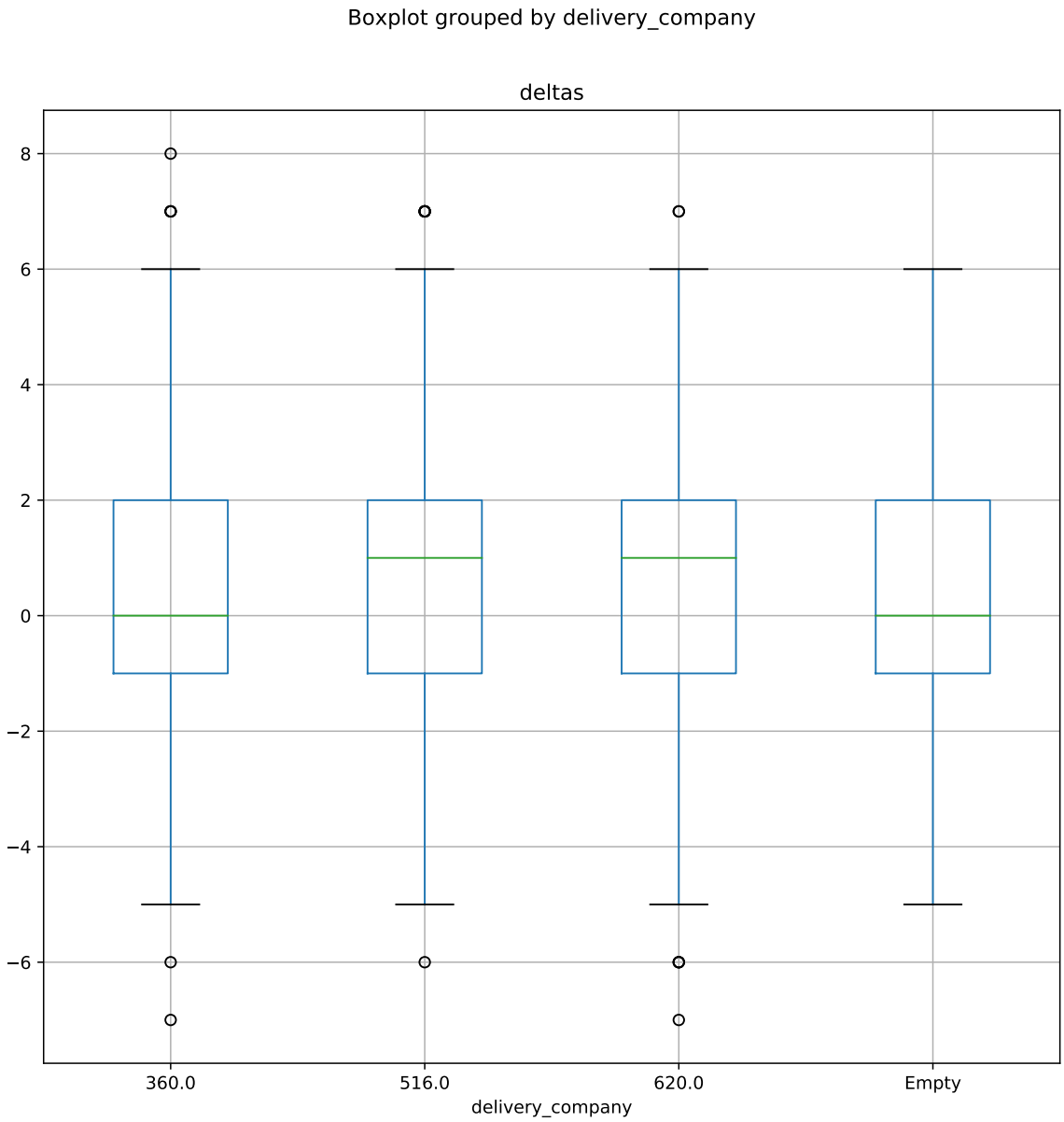
Rysunek 12: Wykres obrazujący rozkład czasu dostawy w zależności od pory dnia zakupu produktu



Rysunek 13: Wykres obrazujący średnią czasu dostaw w zależności od pory dnia zakupu produktu

Tutaj obserwujemy, że średni czas dostawy nie różni się zbytnio pomiędzy porami dnia, oprócz dla produktów kupionych wieczorem, co wynika z tego, że takie produkty mogą zostać nadane dopiero następnego dnia, więc ich wysyłka się wydłuża.

5.3 W zależności od firmy dostawczej

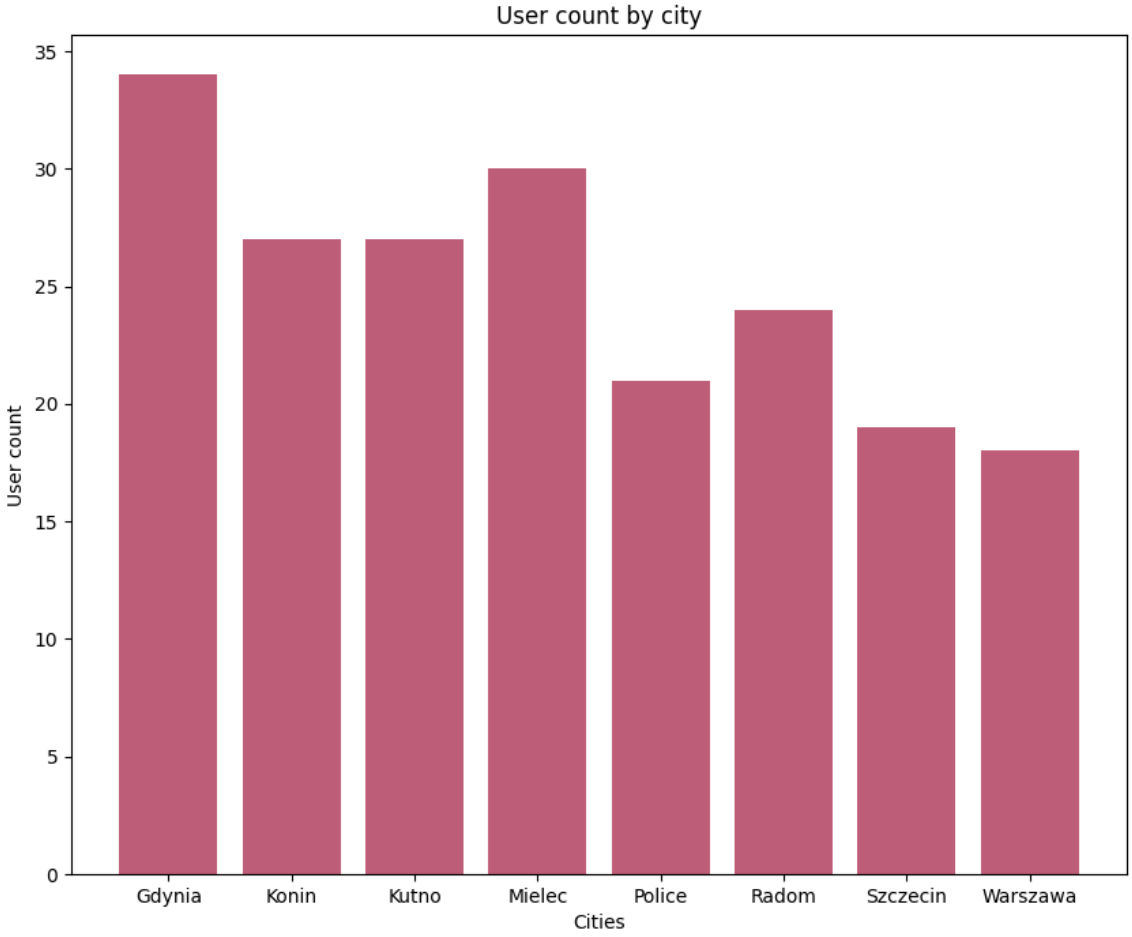


Rysunek 14: Wykres obrazujący rozkład czasu dostaw w zależności od firmy jaka dostarczała produkt

Widać, że rozkłady czasu dostaw są bardzo podobne niezależnie od firmy, różnią się jedynie średnią oraz wartościami odstającymi. Można stąd wyciągnąć wniosek że czasy dostaw nie zależą od rodzaju firmy jaka te dostawy realizuje. Na obrazku znajduje się również coś co nazywa się Empty, są to wiersze w których nie było wartości(NaN). Widać że rozkład pustych wierszy jest podobny do pozostałych rozkładów przez co nasuwa się wniosek że można bezpiecznie usunąć te dane.

5.4 Użytkownicy sklepu ESzopping

Na podstawie wstępnej analizy danych, rozkład zamieszkania użytkowników wygląda następująco:



Rysunek 15: Wykres obrazujący rozkład użytkowników wedle miasta zamieszkania

Tak zaś wygląda rozkład tychże miast na podstawie liczby mieszkańców:

- Miasta powyżej 500 tyś mieszkańców: Warszawa
- Miasta powyżej 250 tyś mieszkańców: Szczecin
- Miasta powyżej 100 tyś mieszkańców: Gdynia, Radom
- Miasta powyżej 50 tyś mieszkańców: Konin, Mielec
- Miasta poniżej 50 tyś mieszkańców: Kutno, Police

Jest to o tyle istotne w analizie czasu dostaw, że dostawy do większych miast (powyżej 250 tyś mieszkańców) będą znacznie szybsze - w nich zazwyczaj znajdują się oddziały przeładunkowe/sortujące paczki, w nich pracuje największa ilość kurierów, do nich prowadzą drogi szybkiego ruchu. W danych które otrzymałyśmy niestety przestrzeń miast w Polsce jest niereprezentatywna. Znaczną większość użytkowników stanowią ci mieszkających w małych miasteczkach, więc możemy się spodziewać predykcji przekłamanych dla dużych miast takich jak np. Warszawa.

6 Analiza późniejszych danych dostarczonych przez klienta

6.1 Czy problemy występujące w pierwotnych danych nadal występują?

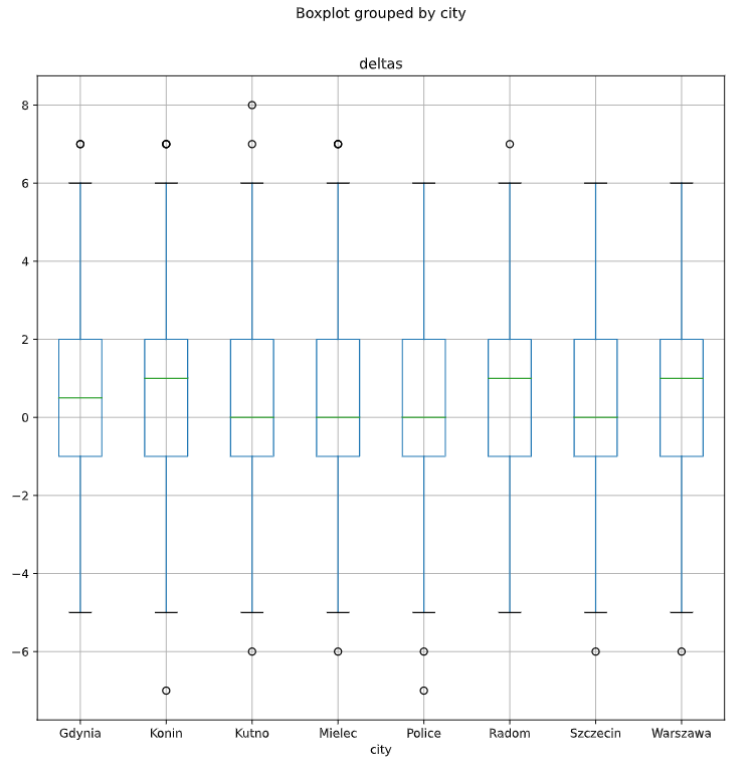
- Kodowanie danych - w danych o produktach nadal znajdowały się polskie znaki w formie surowej, ale jest to zapewne po prostu specyfika bazy danych. Wprowadziłyśmy zatem postępowanie takie jak we wcześniejszym zestawie.
- Ujemne ceny produktów - brak
- Ujemne czasy dostaw - brak
- Kategorie produktów w postaci wieloczłonowej zmiennej tekstowej - podobnie jak w przypadku kodowania jest to najwyraźniej specyfika zapisywania danych do bazy, ponieważ również i dla tego zestawu podzieliłyśmy dane na podkategorie.
- Braki w kolumnie delivery_timestamp - w tym zestawie nie pojawiły się już braki wartości w kolumnie znacznika czasu dostawy.
- Braki w kolumnie delivery_company - brak
- Zły typ dat podczas wczytywania z pliku - tu napotkałyśmy ponownie problemy, które rozwiązałyśmy tak jak przy poprzednich danych.

6.1.1 Podsumowanie

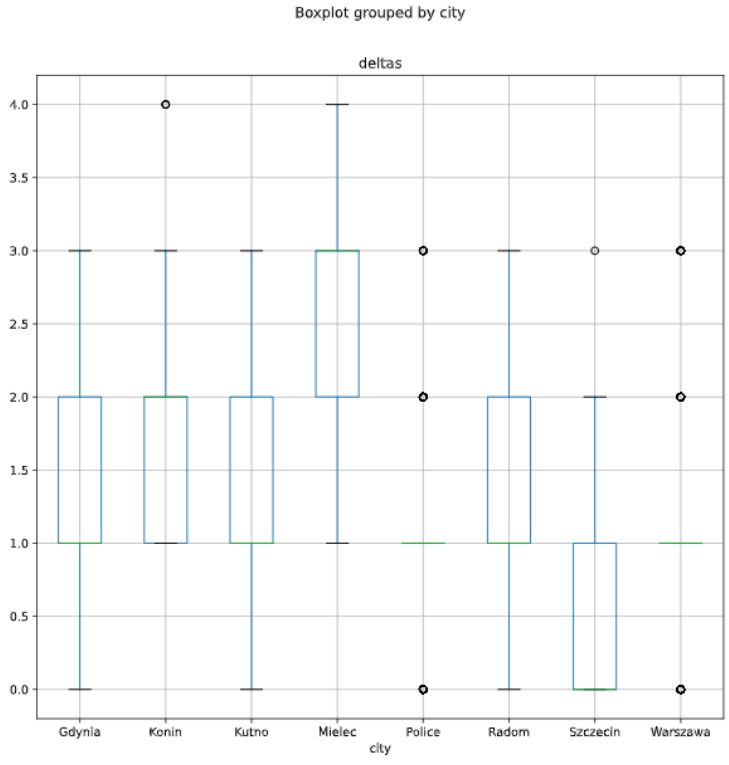
Po wstępnej analizie pod kątem ujemnych, bądź brakujących wartości można z łatwością stwierdzić, że ten zestaw jest czystszy i dużo lepszej jakości od poprzedniego, dlatego że większość problemów z poprzedniego nie występuje dla tego zbioru. W następnym paragrafie znajduje się szczegółowa analiza dotycząca wpływu poszczególnych parametrów na czas dostaw.

6.2 Analiza porównawcza czasu dostaw w nowych i starych danych

6.2.1 W zależności od miasta dostawy

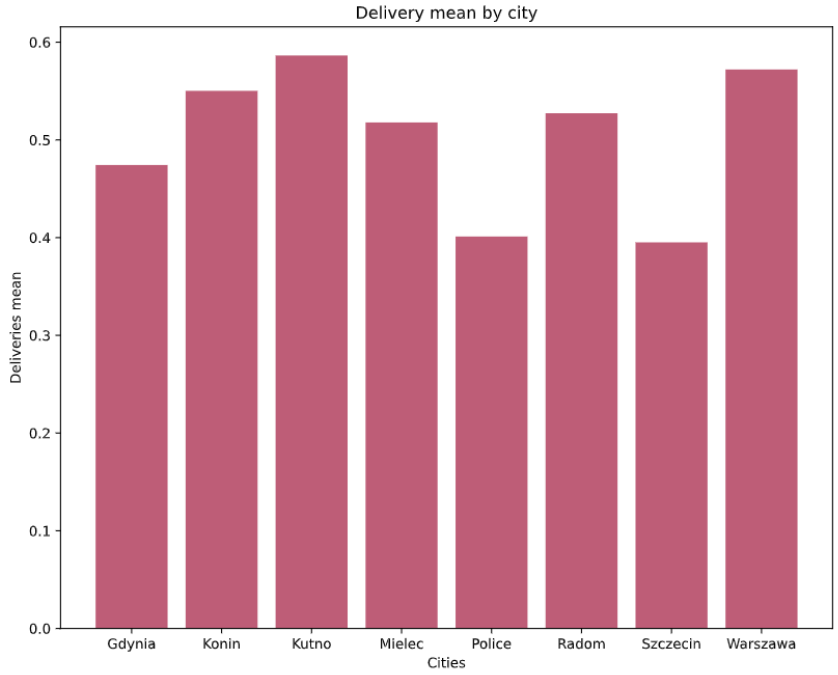


(a) Stare dane

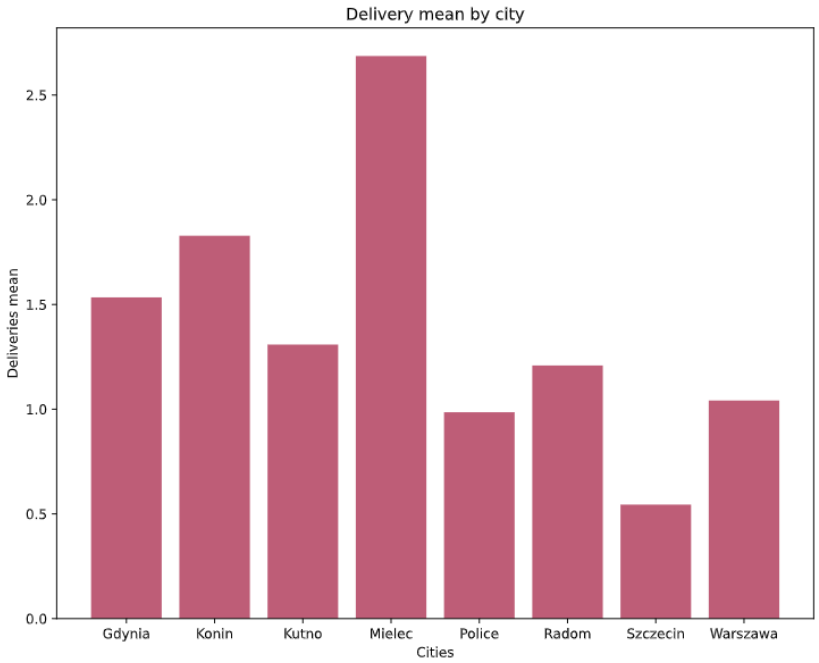


(b) Nowe dane

Rysunek 16: Porównanie wykresów pudełkowych czasów dostaw w zależności od miasta



(a) Stare dane

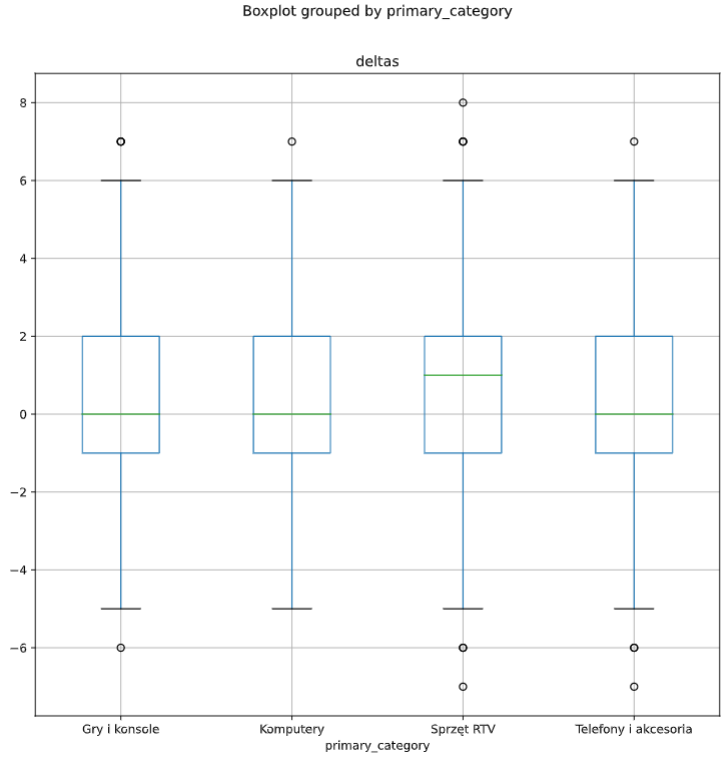


(b) Nowe dane

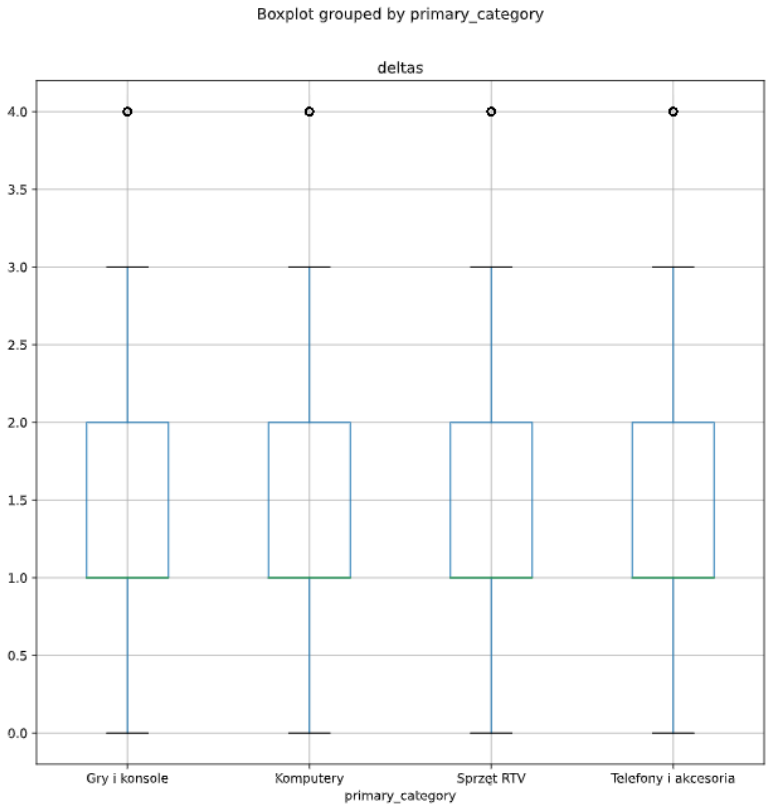
Rysunek 17: Porównanie średnich czasów dostaw w zależności od miasta

Obserwujemy, że czasy dostaw dla nowych danych są dużo bardziej zróżnicowane i intuicyjne - dla Warszawy i Polic mediany wynoszą około 1 dnia, zaś najdłuższa dostawa jest do Mielec. Wyniki analizy średnich czasów są bardziej intuicyjne - dla większych miast takich jak Warszawa czy Szczecin czas dostawy jest krótszy, niż dla tych mniejszych.(Mielec) Krótkie czasy dostaw dla Kutna i Polic możemy w łatwy sposób wytłumaczyć bliskością położenia od dużych miast - Police znajdują się niedaleko Szczecina, dla którego odnotowujemy najkrótszą dostawę, zaś Kutno jest położone niedaleko Warszawy.

6.2.2 W zależności od kategorii produktu

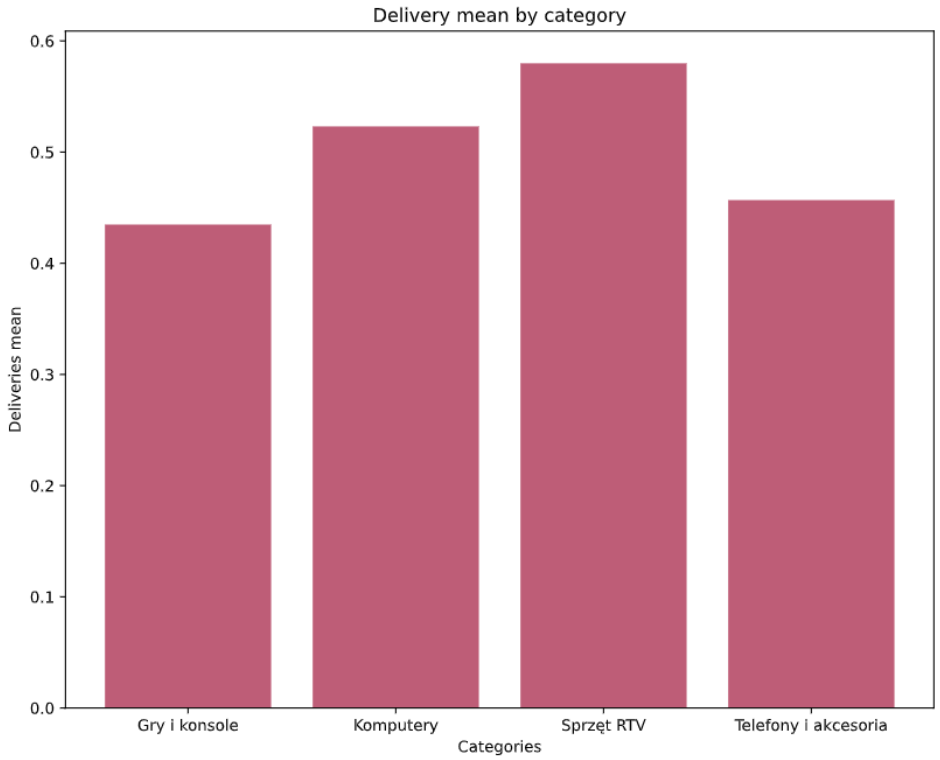


(a) Stare dane

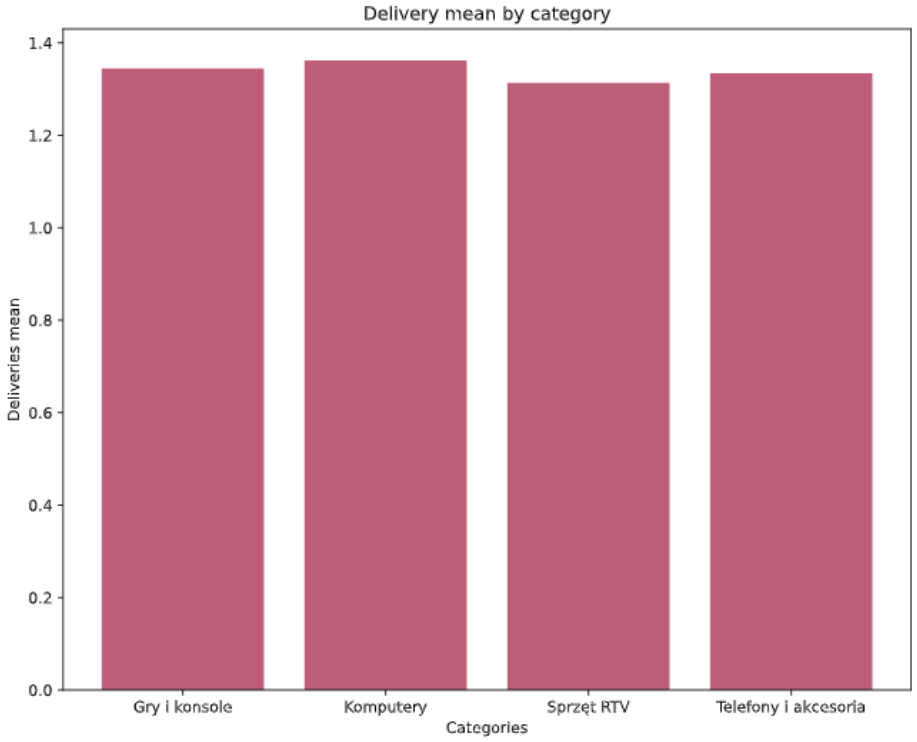


(b) Nowe dane

Rysunek 18: Porównanie wykresów pudełkowych czasów dostaw w zależności od kategorii kupowanego produktu



(a) Stare dane

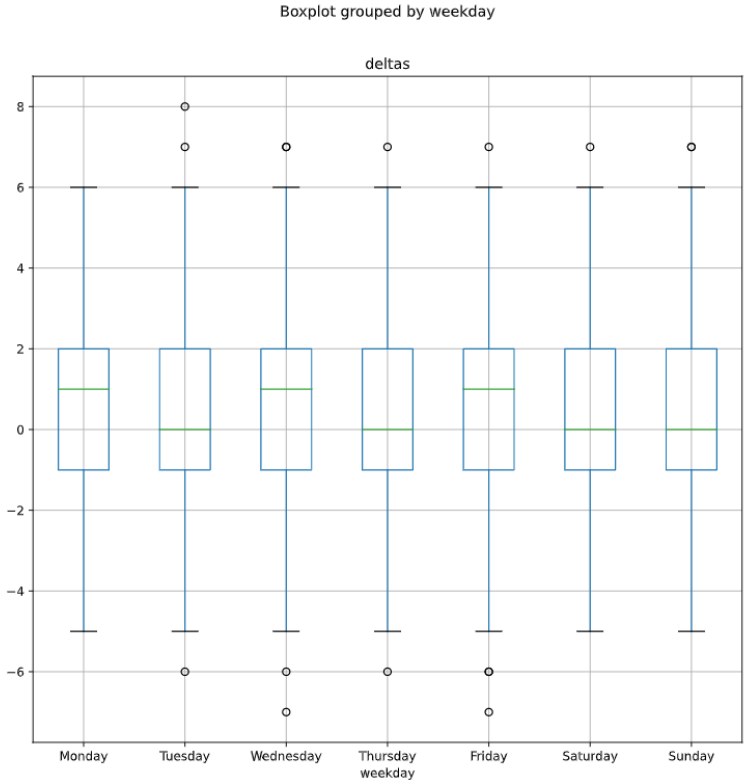


(b) Nowe dane

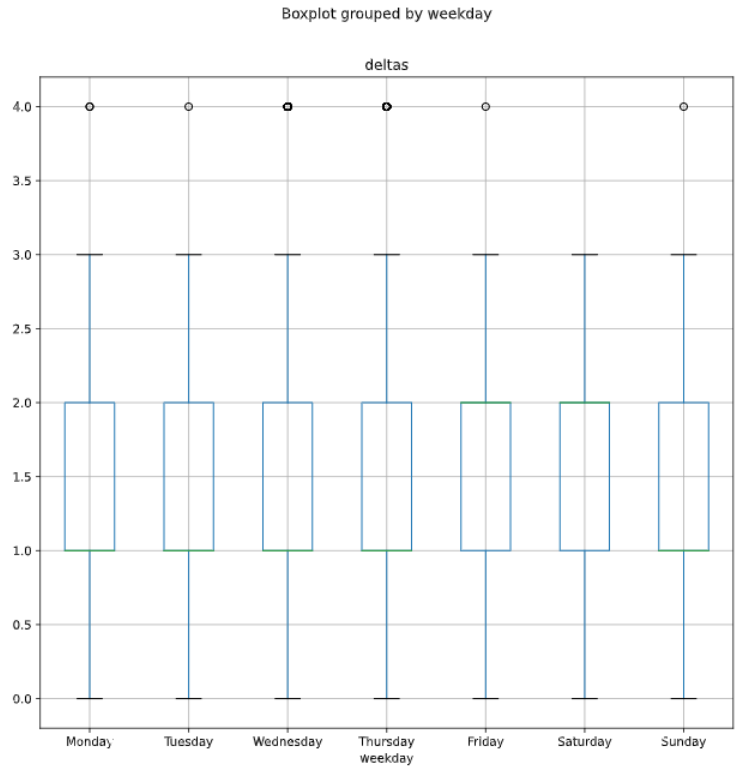
Rysunek 19: Porównanie średnich czasów dostaw w zależności od kategorii kupowanego produktu

W nowych danych widzimy, że wcześniej zauważona korelacja - im większy gabaryt produktu, tym dłuższa dostawa, tutaj nie występuje. Niezależnie od kategorii produktów średnie utrzymują się na poziomie około 1.3 dnia , co jest cenną wskazówką przy budowaniu modelu, że najprawdopodobniej kategoria produktu nie wpływa znacząco na czas dostawy.

6.2.3 w zależności od dnia zakupu produktu

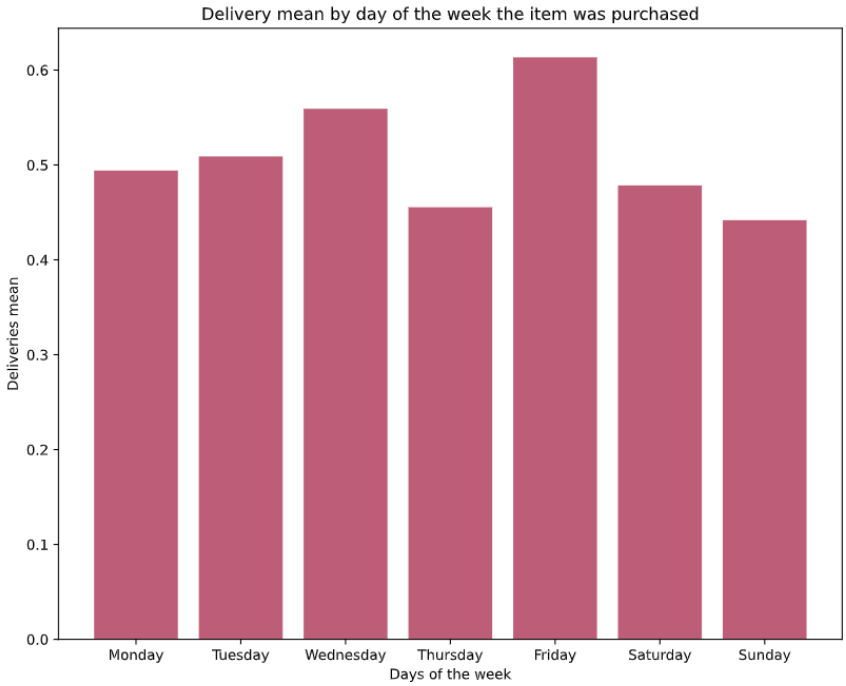


(a) Stare dane

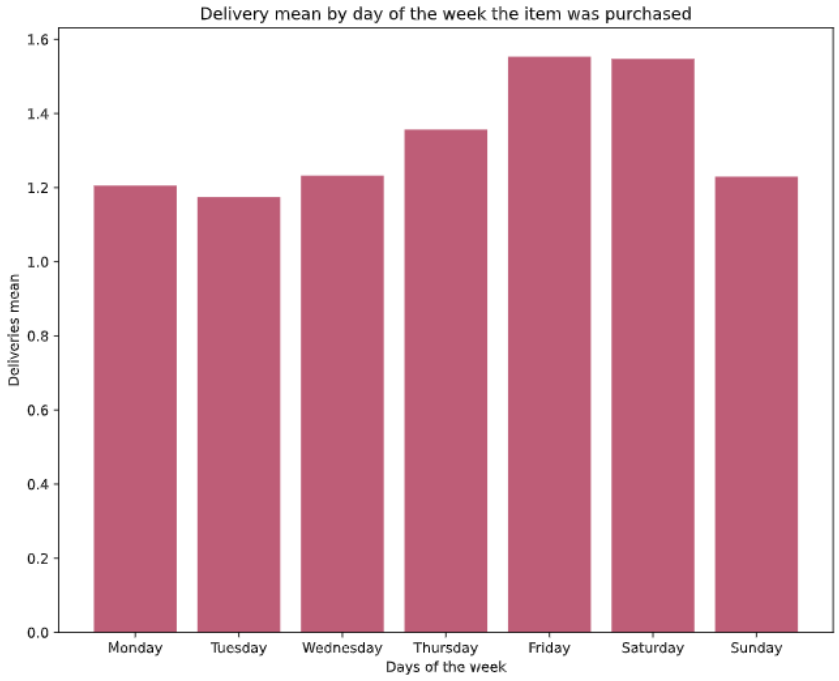


(b) Nowe dane

Rysunek 20: Porównanie wykresów pudełkowych czasów dostaw w zależności od dnia zakupu produktu



(a) Stare dane

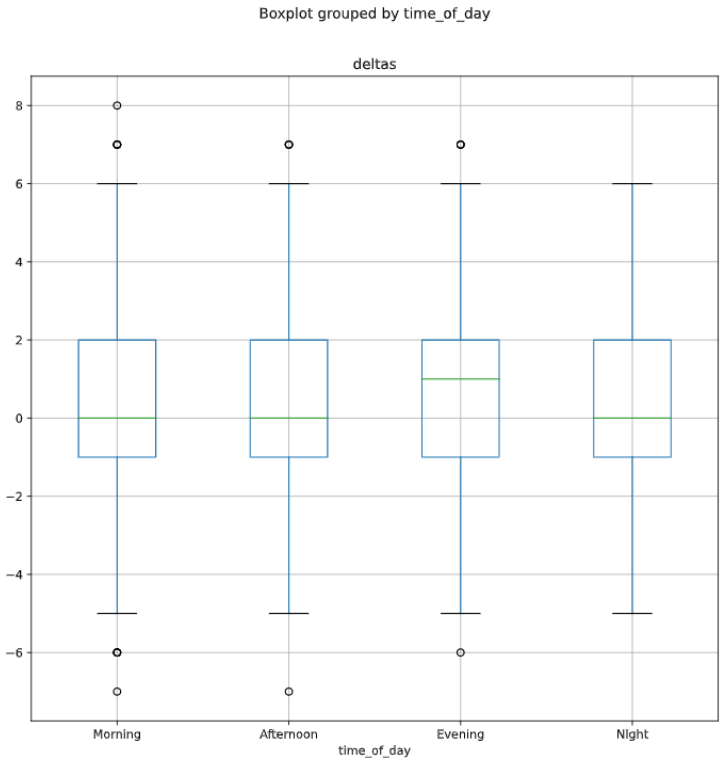


(b) Nowe dane

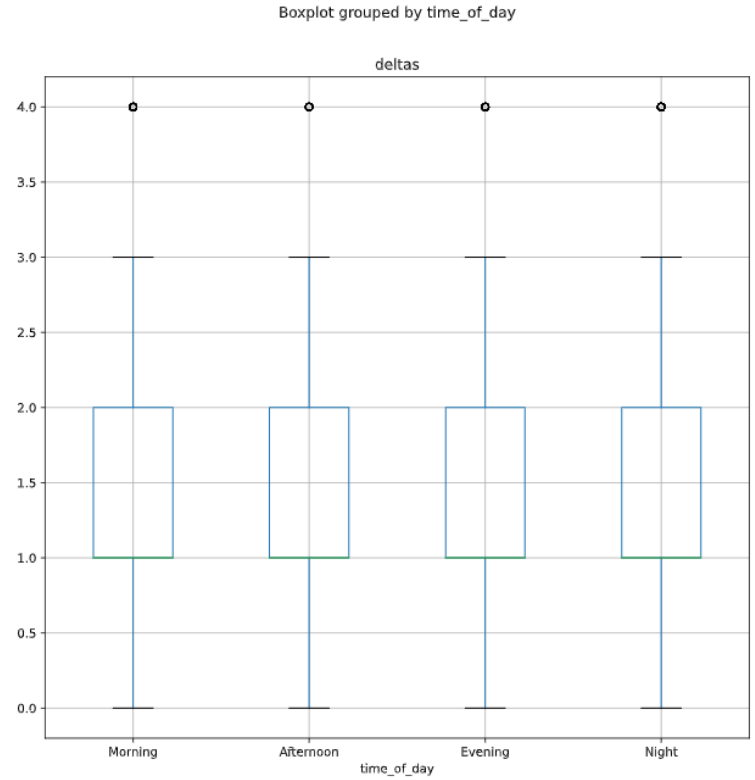
Rysunek 21: Porównanie średnich czasów dostaw w zależności od dnia zakupu produktu

Tym razem dużo więcej możemy odczytać z wykresu średnich niż pudełkowego. Zauważamy prawidłowość, że im bliżej końca tygodnia, tym dłuższa dostawa - co jest zgodne z przewidywaniami. Piątkowe zakupy nie dotrą do kupujących przed weekendem, zatem do czasu dostawy dodatkowo naliczone są dwa dni, sobotnie podobnie z różnicą, że dodatkowo jest naliczony jeden dzień a nie dwa, a produkty zakupione w niedzielę zostaną zapewne wysłane w poniedziałek, stąd niższy wynik niż dla soboty i piątku, ale nieco wyższy od poniedziałku.

6.2.4 W zależności od pory dnia zakupu produktu

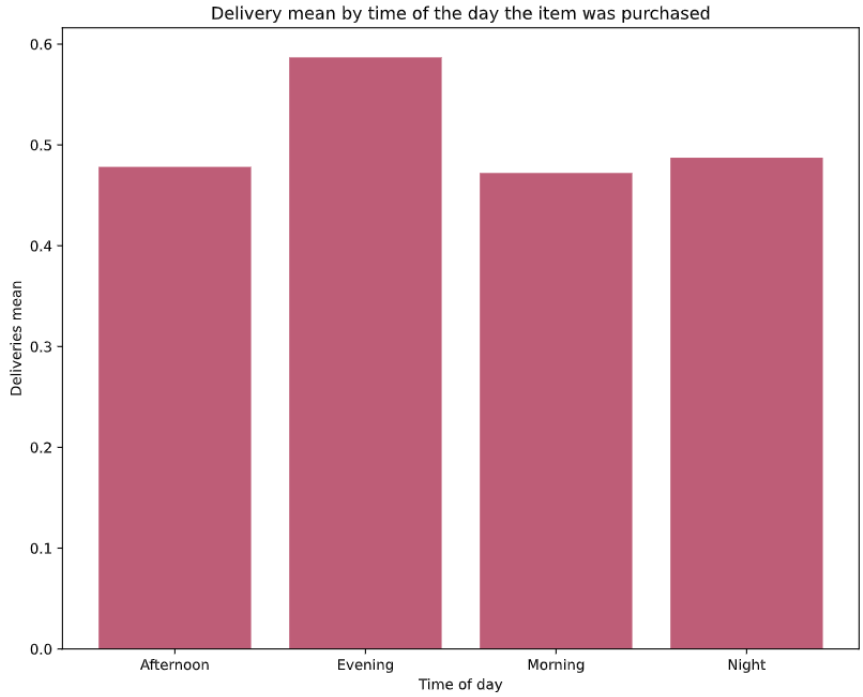


(a) Stare dane

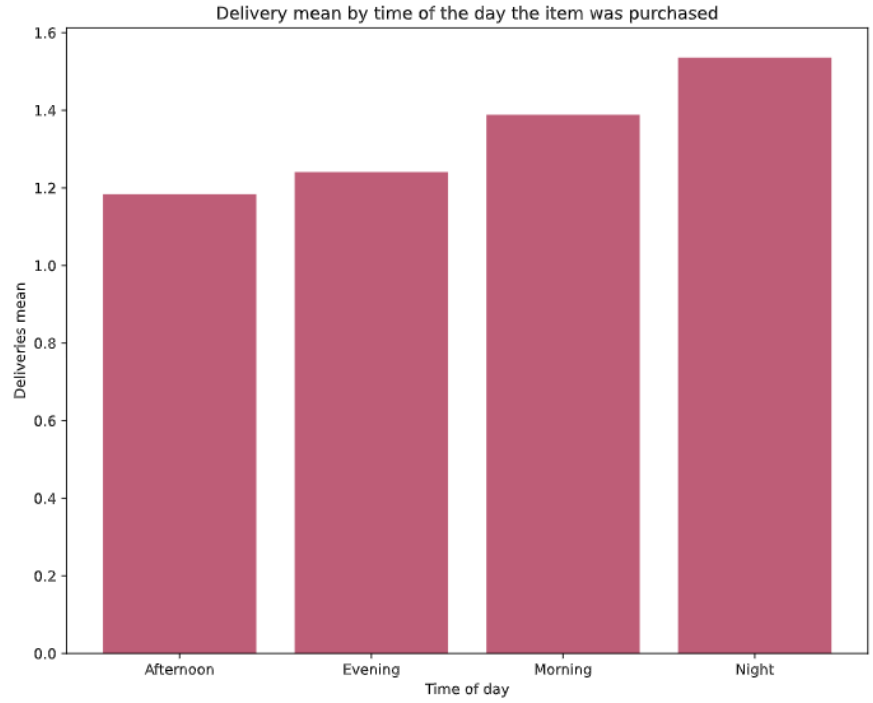


(b) Nowe dane

Rysunek 22: Porównanie wykresów pudełkowych czasów dostaw w zależności od pory dnia zakupu produktu



(a) Stare dane

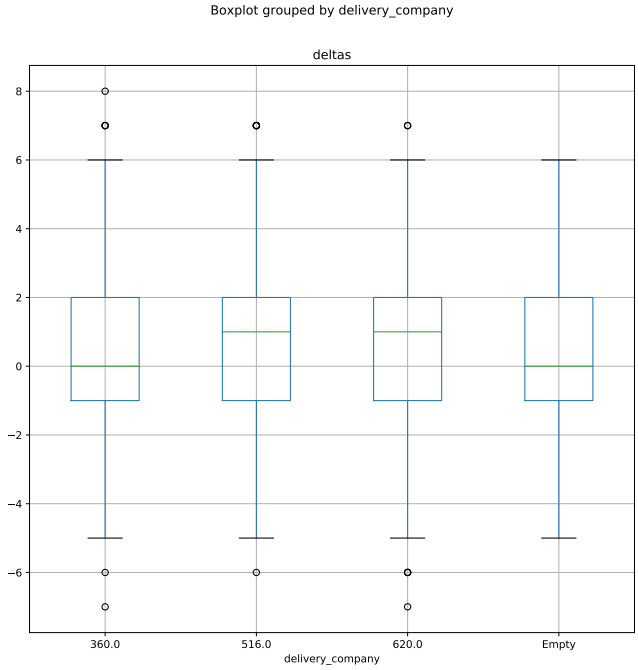


(b) Nowe dane

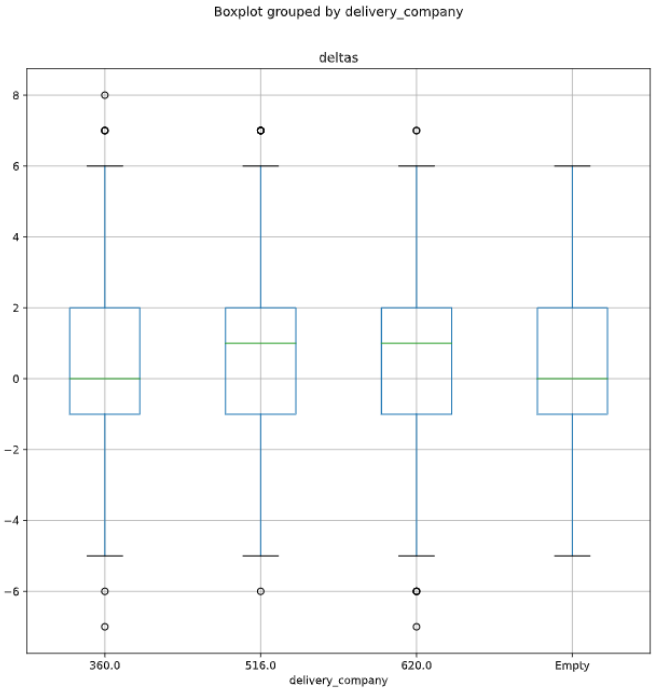
Rysunek 23: Porównanie średnich czasów dostaw w zależności od od pory dnia zakupu produktu

Ponownie wyniki analizy są nieco bardziej racjonalne - zakupy poczynione w nocy mają najdłuższy czas dostawy, bo wysyłka jest następnego dnia. Dziwi jedynie tak długi czas dostawy dla zakupów porannych, ale nie mamy wystarczających danych, aby wyciągnąć wnioski.

6.2.5 W zależności firmy dostawczej



(a) Stare dane

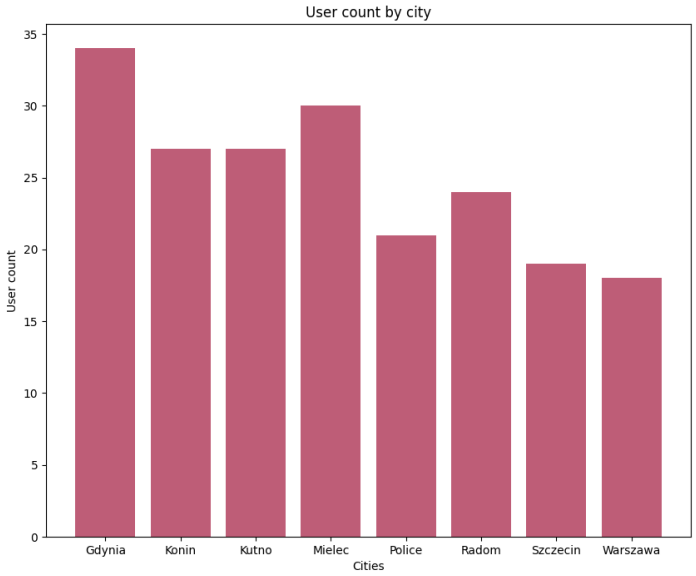


(b) Nowe dane

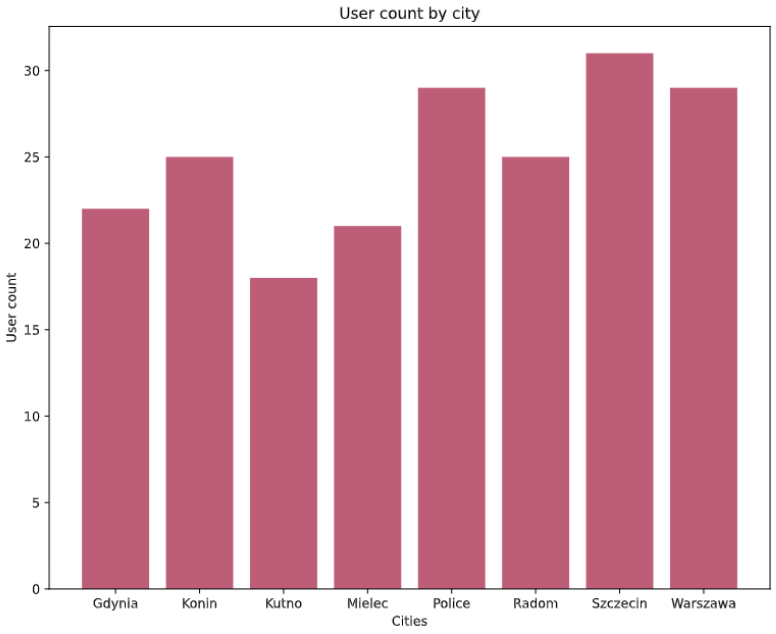
Rysunek 24: Porównanie wykresów pudełkowych czasów dostaw w zależności od firmy dostawczej

Tutaj nie zauważamy żadnych zmian w stosunku do poprzednich danych. Wykresy pudełkowe są dokładnie takie same.

6.3 Użytkownicy sklepu EScopping



(a) Stare dane



(b) Nowe dane

Rysunek 25: Porównanie rozkładu zamieszkania użytkowników korzystających ze sklepu EScopping

Przypomnienie podziału miast:

- Miasta powyżej 500 tyś mieszkańców: Warszawa
- Miasta powyżej 250 tyś mieszkańców: Szczecin
- Miasta powyżej 100 tyś mieszkańców: Gdynia, Radom
- Miasta powyżej 50 tyś mieszkańców: Konin, Mielec

- Miasta poniżej 50 tyś mieszkańców: Kutno, Police

Teraz widzimy, że w większości klienci pochodzą z większych miast - Szczecin, Warszawa, Radom. Zatem nie powinniśmy obserwować już dużego zawyżania przewidywań dla bardziej ludnych i lepiej skomunikowanych miejscowości. Nadal pozostaje jednak pytanie, czy teraz nie zauważymy symetrycznego problemu - zaniżania czasu dostaw dla małych miejscowości. Odpowiedź na to pytanie znajduje się w jednym z następnych paragrafów.

7 Model prosty - regresja liniowa

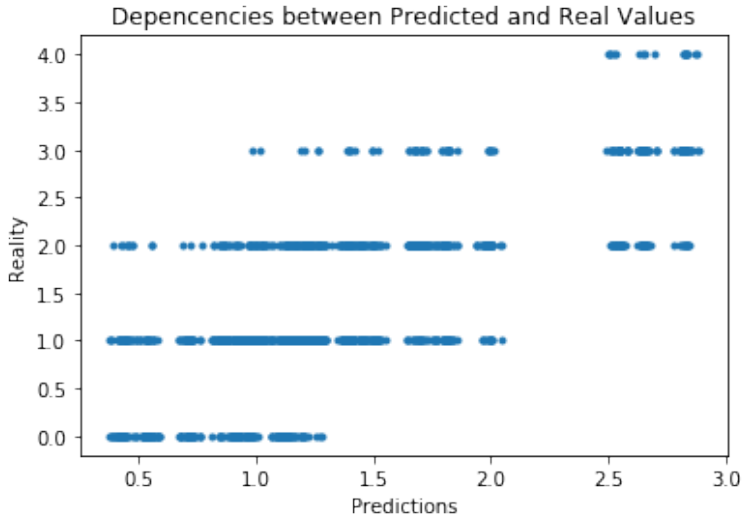
Aby ocenić na jakim poziomie można osiągnąć wyniki gdy zastosuje się rozwiązanie naiwne sięgnęłyśmy do modelu regresji liniowej. Chcemy ocenić jakie wyniki możemy osiągnąć stosując prosty model aby mieć punkt odniesienia do jakości naszego zaawansowanego modelu.

7.1 Wstępna obróbka danych

Zaczęłyśmy od złączenia danych które otrzymałyśmy w kilku plikach za pomocą instrukcji merge. Zbiory dało się połączyć ponieważ występowały w nich powtarzające się wartości takie jak "purchase_id"w deliveries i sessions, "user_id"w users i sessions oraz "product_id"w products i sessions.
Następnie przeszłyśmy do usunięcia se zbioru danych które są nieprzydatne bądź istniejące tylko w danych historycznych. Za nieprzydatne uznałyśmy kolumny: event_type - cała kolumna zawierała tylko transakcje zakupu ponieważ łączyłyśmy sessions w deliveries, name - nie powinno być związku między imionami użytkowników a tym jak szybko dostają oni swoje paczki, street - posiadamy informacje dotyczące miasta które jak sądzimy wystarczą do przewidywań bo gdy paczka dotrze do jakiegoś miasta to zakładamy że jest dostarczana w podobnym czasem co inna paczka z tego miasta, usunęłyśmy również wszystkie id oraz tmestamp sesji użytkownika. Delivery_timestamp oraz purchase_timestamp usunęłyśmy ponieważ z nich powstała zmienna wynikowa - czas dostawy.
Kolejnym krokiem było zastosowanie one hot encoding na zmiennych dyskretnych czyli miasto kategorie produktu oraz firma dostawcza a deltas czyli nasze label zrzutowałyśmy na datetime.days ponieważ regresja liniowa sklearn której użyłyśmy nie radzi sobie z datami. Po powyższych zabiegach otrzymaliśmy zbiór składający się z 43 featerów oraz jednego label.
Podzieliłyśmy zbiór dnych na zbiór testowy oraz treningowy (w proporcji 2:8) a następnie wytrenowałyśmy model regresji liniowej na tak przygotowanych danych. <po zbadaniu jak model radzi sobie na danych testowych otrzymałyśmy błąd średniokwadratowy wynoszący 0.38 oraz współczynnik determinacji na poziomie 0.51.
Starałam się uzyskać lepsze wyniki, stwierdziłąm że może przyczyną jest zaokrąglanie wyników do dni więc spróbowałam zamienić czas dostawy na sekundy i znormalizować go jednak wyniki okazały się znacznie gorsze.

7.2 Wnioski

Przygotowałyśmy wykres zależności wartości przewidzianych przez nasz model od tych które występują w rzeczywistości.

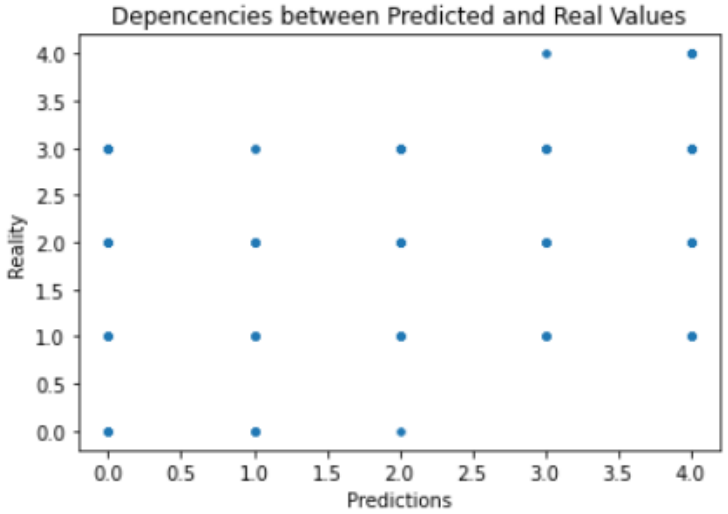


Rysunek 26: Zależność pomiędzy wartościami przewidzianymi oraz prawdziwymi.

Ustaliłyśmy że problemem może być rodzaj modelu jaki zastosowałyśmy - zmienna którą starałyśmy się przewidzieć była bardziej dyskretna dlatego należałoby zastosować raczej model klasyfikacji a nie regresji. Przystąpiłyśmy więc do tworzenia Naiwnego modelu klasyfikacji Bayesa.

8 Model prosty 2 - naiwny klasyfikator Bayesa

Zamieniłyśmy model prosty z regresji liniowej na naiwny klasyfikator Bayesa. Zastosowałyśmy dokładnie takie przekształcenia jak w modelu regresji liniowej jednak w tym wypadku wyniki modelu okazały się jeszcze słabsze. Prawdziwe wyniki model otrzymywał w 39% przypadków.



Rysunek 27: Zależność pomiędzy wartościami przewidzianymi oraz prawdziwymi.

Sprawdziłyśmy również dopasowanie modelu do danych treningowych w porównaniu do danych testowych. Zdaje się że model nie przeucza się ponieważ wyniki na zbiorze treningowym i testowym są zbliżone. Wyniki okazały się niezadowalające, prawdopodobnie przyczyną może być zbyt mała złożoność modelu który nie był w stanie uchwycić zależności bądź danych słabej jakości - nieinformatywnych. W modelu Bayesowskim użyliśmy dwóch cech więcej niż w sieci neuronowej ponieważ znacząco poprawiały one działanie modelu (z 27% do 39% bez wskazań na przeuczenie w obu przypadkach) jednak w sieci usunęliśmy te cechy ponieważ tam powodowały one przeuczenie.

	precision	recall	f1-score	support
0	0.31	0.94	0.47	823
1	0.63	0.33	0.44	2426
2	0.64	0.21	0.31	1487
3	0.43	0.51	0.46	462
4	0.10	1.00	0.18	47
accuracy			0.41	5245
macro avg	0.42	0.60	0.37	5245
weighted avg	0.56	0.41	0.41	5245

(a) Zbiór treningowy

	precision	recall	f1-score	support
0	0.29	0.94	0.44	327
1	0.61	0.32	0.42	1054
2	0.63	0.20	0.31	647
3	0.39	0.43	0.41	197
4	0.10	0.87	0.18	23
accuracy			0.39	2248
macro avg	0.41	0.55	0.35	2248
weighted avg	0.55	0.39	0.39	2248

(b) Zbiór testowy

Rysunek 28: Porównanie jakości predykcji dla zbioru treningowego i testowego

9 Model rozszerzony - sieć neuronowa

9.1 Wybór modelu

Bazując na doświadczeniach z budowania uproszczonego modelu - słabą skuteczność przewidywań czasu dostawy jako zadanie regresji, postanowiłyśmy stworzyć model sieci neuronowej w wykorzystaniu do zadania klasyfikacji. Do tego celu użyłyśmy MLPClassifier() z pakietu sckit-learn, z następujących powodów:

- Skorzystanie z gotowego otwartoźródłowego rozwiązania, pozwala na uniknięcie problemów implementacyjnych związanych z budowaniem modelu sieci neuronowej, które to nie są istotną częścią tego projektu
- Ten konkretny model jest polecany do zadań klasyfikacji w literaturze, którą udało nam się znaleźć w sieci
- Jest to rozwiązanie powszechnie używane nawet w dużych projektach, dzięki czemu możemy liczyć na łatwą integrację w przyszłości oraz stosunkowo refactoring kodu i brak większych problemów z utrzymaniem go (w przeciwieństwie do rozwiązania zaprojektowanego i implementowanego od zera).
- Dzięki otwartym źródłom i zaangażowanej społeczności pakiet sckit-learn jest ciągle rozwijany i może korzystać z doświadczenia wielu programistów, dzięki czemu możemy się spodziewać, że nawet jeśli coś zostało błędnie zaimplementowane, będzie to naprawione w następnej odsłonie.

W następnych podrozdziałach znajdują się szczegółowe opisy kolejnych iteracji ulepszania modelu.

9.2 Pierwsza iteracja budowania modelu

9.2.1 Dane podawane do modelu

Na podstawie wcześniejszej analizy danych, odgórnie wybrałyśmy te atrybuty, które wykazywały największy wpływ na długość dostawy. (analiza zależności na podstawie wykresów pudełkowych, wykresów średnich) Następnie zakodowałyśmy je za pomocą metody one-hot-encoding. Dane zostały dodatkowo znormalizowane, ze względu na wrażliwość sieci neuronowych na niezrównoważone wartości atrybutów i mogło by to doprowadzić do niestabilnego uczenia i gorszych wyników. Stosunek ilości danych testowych do danych treningowych wyniósł 3:7.

```
#Usuwanie atrybutów mało informatywnych
united = united.loc[:, united.columns != 'event_type']
united = united.loc[:, united.columns != 'name']
united = united.loc[:, united.columns != 'street']
united = united.loc[:, united.columns != 'product_name']
united = united.loc[:, united.columns != 'delivery_timestamp']
united = united.loc[:, united.columns != 'timestamp']
united = united.loc[:, united.columns != 'purchase_id']
united = united.loc[:, united.columns != 'product_id']
united = united.loc[:, united.columns != 'user_id']
united = united.loc[:, united.columns != 'session_id']
united = united.loc[:, united.columns != 'offered_discount']
united = united.loc[:, united.columns != 'price']
united = united.loc[:, united.columns != 'primary_category']
united = united.loc[:, united.columns != 'secondary_category']
united = united.loc[:, united.columns != 'tertiary_category']
united = united.loc[:, united.columns != 'quaternary_category']

united.loc[:,'time_of_day'] = united.apply(lambda row: labelTimeOfDay(row), axis=1)
united['weekday'] = united['purchase_timestamp'].dt.day_name()
united = united.loc[:, united.columns != 'purchase_timestamp']

#One hot encoding

y = pd.get_dummies(united.city, prefix='city')
united = united.join(other=y)
united = united.loc[:, united.columns != 'city']

y = pd.get_dummies(united.delivery_company, prefix='delivery_company')
united = united.join(other=y)
united = united.loc[:, united.columns != 'delivery_company']

y = pd.get_dummies(united.time_of_day, prefix='time_of_day')
united = united.join(other=y)
united = united.loc[:, united.columns != 'time_of_day']

y = pd.get_dummies(united.weekday, prefix='weekday')
united = united.join(other=y)
united = united.loc[:, united.columns != 'weekday']
```

```
united['deltas'] = pd.to_numeric(united['deltas'].dt.days, downcast='integer')

#Podział na zbiór testowy i uczący
X = united[predictors].values
y = united[target_column].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state
    ↪ =40)

#Skalowanie atrybutów
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

9.2.2 Model

Na początek zastosowałyśmy funkcję aktywacji RELU i solver Adam, podobnie jak w jednym z przykładów z literatury. Sieć miała trzy warstwy ukryte, każda po tyle neuronów ile było atrybutów ze zbioru uczącego.

```
#kod budujący model
mlpClassifier = MLPClassifier(hidden_layer_sizes=(sizePred,sizePred,sizePred),
    ↪ activation='relu', solver='adam', max_iter=500)
mlpClassifier.fit(X_train,y_train.ravel())
```

9.2.3 Wyniki oraz ich omówienie

	precision	recall	f1-score	support
0	0.71	0.61	0.66	823
1	0.77	0.77	0.77	2426
2	0.68	0.77	0.72	1487
3	0.74	0.64	0.69	462
4	0.79	0.47	0.59	47
accuracy			0.73	5245
macro avg	0.74	0.65	0.69	5245
weighted avg	0.73	0.73	0.73	5245

(a) Zbiór treningowy

	precision	recall	f1-score	support
0	0.62	0.58	0.60	327
1	0.74	0.75	0.74	1054
2	0.66	0.71	0.68	647
3	0.68	0.52	0.59	197
4	0.74	0.61	0.67	23
accuracy			0.69	2248
macro avg	0.69	0.63	0.66	2248
weighted avg	0.69	0.69	0.69	2248

(b) Zbiór testowy

Rysunek 29: Porównanie jakości predykcji dla zbioru treningowego i testowego

Wnioski:

- Ogólna jakość modelu oceniłyśmy na średnią około 69% skuteczności to nie jest oszałamiający wynik
- Błąd na zbiorze testowym jest stosunkowo niski, zatem model jest się w stanie ńauczyćńa danych, ale jako że obciążenie jest wysokie, a wariancja niska, znaczy to że architektura modelu jest niedopracowana, lub dane są słabej jakości.
- Dane są dosyć mocno niezbalansowane - jak widać 4-dniową dostawę odnotowujemy jedynie dla 47 wierszy. Zatem prawdopodobnie to jest przyczyną średniej jakości przewidywania. Dla kolejnej iteracji należy wyrównać to niezbalansowanie poprzez oversampling dla grupy z dostawą 4dniową i 3dniową i undersampling dla grupy 1-2dniowej.

9.3 Kolejne iteracje budowania modelu

W podrozdziałach będą opisane zmiany jakich dokonywałyśmy oraz ich wpływ na końcową jakość predykcji.

9.3.1 Próba ze zbalansowanymi danymi - bez łączenia najmniejszych podgrup

Aby zmniejszyć niezbalansowanie danych uczących postanowiłyśmy na początku dolosowywać w zbiorze treningowym wiersze, aby zwiększyć liczność najmniejszych podgrup. tzn. wierszy z dostawami 0,3,4 dniowymi. Uznałyśmy, że dobrą liczbą wierszy będzie około 1000 na każdą z podgrup, aby drastycznie nie zmniejszać ogólnej liczebności zbioru uczącego, ale też nie powielać zbyt wiele razy wierszy z mniejszych grup.

```
# LICZNOŚĆ PRZED ZMIANĄ
train_united_0 = train_united.loc[train_united['deltas'] == 0] #823
train_united_1 = train_united.loc[train_united['deltas'] == 1] #2426
train_united_2 = train_united.loc[train_united['deltas'] == 2] #1487
train_united_3 = train_united.loc[train_united['deltas'] == 3] #462
train_united_4 = train_united.loc[train_united['deltas'] == 4] #47


# LICZNOŚĆ PO ZMIANACH
train_united_0 = train_united.loc[train_united['deltas'] == 0] #1197
train_united_1 = train_united.loc[train_united['deltas'] == 1] #1140
train_united_2 = train_united.loc[train_united['deltas'] == 2] #1008
train_united_3 = train_united.loc[train_united['deltas'] == 3] #1122
train_united_4 = train_united.loc[train_united['deltas'] == 4] #1066
```

```
[[872 147  32   3   0]
 [163 772 140  31   0]
 [ 46 157 708 117  14]
 [  5  14  96 936  18]
 [  0   0   9  33  73]]
              precision    recall  f1-score   support

           0       0.80      0.83      0.81      1054
           1       0.71      0.70      0.70      1106
           2       0.72      0.68      0.70      1042
           3       0.84      0.88      0.86      1069
           4       0.70      0.63      0.66       115

    accuracy                           0.77      4386
   macro avg       0.75      0.74      0.75      4386
weighted avg       0.76      0.77      0.76      4386
```

(a) Zbiór treningowy

```
[[250  54  20   3   0]
 [198 638 182  34   2]
 [ 30 129 403  74  11]
 [  1  11  31 151   3]
 [  0   0   1   5  17]]
              precision    recall  f1-score   support

           0       0.52      0.76      0.62       327
           1       0.77      0.61      0.68      1054
           2       0.63      0.62      0.63       647
           3       0.57      0.77      0.65       197
           4       0.52      0.74      0.61        23

    accuracy                           0.65      2248
   macro avg       0.60      0.70      0.64      2248
weighted avg       0.67      0.65      0.65      2248
```

(b) Zbiór testowy

Rysunek 30: Porównanie jakości predykcji dla zbioru treningowego i testowego

Wnioski:

- Jakość modelu na danych treningowych spadła z 69% do 65%.

- Błąd na zbiorze testowym dosyć mocno wzrósł, zatem przesuwamy się w kierunku najmniej pożądanej sytuacji - duży błąd na zbiorze testowym i treningowym. Model utracił trochę zdolności do generalizacji (co widać po zwiększonym błędzie na zbiorze testowym) zatem zaczął się dopasowywać do danych treningowych.
- Jest to nieudana próba poprawy jakości modelu - najwyraźniej jest zbyt wiele duplikatów wierszy, więc w dalszym ciągu realizacji rezygnujemy z tego pomysłu.

9.3.2 Próba ze zbalansowanymi danymi, ale bez zmniejszania ich ogólnej liczebności

W tej próbie dolosowujemy odpowiednio dużo nowych wierszy w zbiorze treningowym dla grup dostaw 3 i 4 dniowych bez zmniejszania liczebności innych grup. W ten sposób w znaczny sposób powiększamy w ogólności zbiór danych, ale również udaje nam się zmniejszyć niezbalansowanie zbioru.

```
[[1575  201   75    7    0]
 [ 347 1282  347   76    0]
 [  87  265 1692  283   66]
 [   8   27  200 1633  155]
 [   0    0    0   38 2245]]
              precision    recall  f1-score   support

           0       0.78      0.85      0.81      1858
           1       0.72      0.62      0.67      2052
           2       0.73      0.71      0.72      2393
           3       0.80      0.81      0.80      2023
           4       0.91      0.98      0.95      2283

    accuracy                           0.79     10609
   macro avg       0.79      0.79      0.79     10609
weighted avg       0.79      0.79      0.79     10609
```

(a) Zbiór treningowy

```
[[252  52  21   2   0]
 [209 605 199  39   2]
 [ 24 108 426  73  16]
 [  1   7  31 145  13]
 [  0   0   0   4  19]]
              precision    recall  f1-score   support

           0       0.52      0.77      0.62       327
           1       0.78      0.57      0.66      1054
           2       0.63      0.66      0.64       647
           3       0.55      0.74      0.63       197
           4       0.38      0.83      0.52        23

    accuracy                           0.64      2248
   macro avg       0.57      0.71      0.62      2248
weighted avg       0.68      0.64      0.65      2248
```

(b) Zbiór testowy

Rysunek 31: Porównanie jakości predykcji dla zbioru treningowego i testowego

Wnioski:

- Błąd na zbiorze testowym ponownie wzrósł, zatem wnioskujemy podobnie co przy poprzedniej próbie.

9.3.3 Próba połączenia najmniejszych podgrup w większą na całych danych

Po przeanalizowaniu liczebności wierszy dla każdego przypadku, okazało się że grupa dostaw 3,4dniowych jest bardzo niewielka w porównaniu do reszty. Dlatego postanowiłyśmy połączyć ją we wspólną grupę 3+

dni dostawy. Dla wszystkich danych nie tylko dla treningowych!

```
#balancing data
united_0 = united.loc[united['deltas'] == 0] #1150
united_1 = united.loc[united['deltas'] == 1] #3480
united_2 = united.loc[united['deltas'] == 2] #2134
united_3 = united.loc[united['deltas'] == 3] #659
united_4 = united.loc[united['deltas'] == 4] #70
united_3_4 = united.loc[united['deltas'].isin([3,4])] #729
```

```
[[ 563  220   40    0]
 [ 227 1931  256   12]
 [  35  326 1067   59]
 [   1   52  148  308]]
              precision    recall  f1-score   support

           0       0.68      0.68      0.68       823
           1       0.76      0.80      0.78      2426
           2       0.71      0.72      0.71      1487
           3       0.81      0.61      0.69       509

    accuracy                           0.74      5245
   macro avg       0.74      0.70      0.72      5245
weighted avg       0.74      0.74      0.74      5245
```

(a) Zbiór treningowy

```
[[204  98  25   0]
 [127 793 126   8]
 [ 19 179 410  39]
 [  0  30  68 122]]
              precision    recall  f1-score   support

           0       0.58      0.62      0.60       327
           1       0.72      0.75      0.74      1054
           2       0.65      0.63      0.64       647
           3       0.72      0.55      0.63       220

    accuracy                           0.68      2248
   macro avg       0.67      0.64      0.65      2248
weighted avg       0.68      0.68      0.68      2248
```

(b) Zbiór testowy

Rysunek 32: Porównanie jakości predykcji dla zbioru treningowego i testowego

Wnioski:

- Jakość modelu jest nieco niższa niż w pierwotnej próbie, błąd na zbiorze testowym nieco wzrósł.
- Zmiana nie jest drastyczna i ciężko wyciągnąć wnioski - należy wykonać więcej eksperymentów na zmienionych danych np. próbę ich zbalansowania.

9.3.4 Próba połączenia najmniejszych podgrup w większą na całych danych + próba zbalansowania

```
# LICZNOŚĆ PRZED ZMIANĄ
train_united_0 = train_united.loc[train_united['deltas'] == 0] #823
```

```
train_united_1 = train_united.loc[train_united['deltas'] == 1] #2426
train_united_2 = train_united.loc[train_united['deltas'] == 2] #1487
train_united_3 = train_united.loc[train_united['deltas'] == 3] #509


# LICZNOŚĆ PO ZMIANACH
train_united_0 = train_united.loc[train_united['deltas'] == 0] #1039
train_united_1 = train_united.loc[train_united['deltas'] == 1] #1023
train_united_2 = train_united.loc[train_united['deltas'] == 2] #1152
train_united_3 = train_united.loc[train_united['deltas'] == 3] #1092
```

```
[[920  65  53   1]
 [201 599 180  43]
 [ 41 111 835 165]
 [  7   9  78 998]]
              precision    recall  f1-score   support

           0       0.79      0.89      0.83      1039
           1       0.76      0.59      0.66      1023
           2       0.73      0.72      0.73      1152
           3       0.83      0.91      0.87      1092

    accuracy                           0.78      4306
   macro avg       0.78      0.78      0.77      4306
weighted avg       0.78      0.78      0.77      4306
```

(a) Zbiór treningowy

```
[[262  38  26   1]
 [261 532 224  37]
 [ 37  93 430  87]
 [  3   6  26 185]]
              precision    recall  f1-score   support

           0       0.47      0.80      0.59       327
           1       0.80      0.50      0.62      1054
           2       0.61      0.66      0.64       647
           3       0.60      0.84      0.70       220

    accuracy                           0.63      2248
   macro avg       0.62      0.70      0.64      2248
weighted avg       0.67      0.63      0.63      2248
```

(b) Zbiór testowy

Rysunek 33: Porównanie jakości predykcji dla zbioru treningowego i testowego

Wnioski:

- Błąd na zbiorze testowym jest dosyć wysoki, podobnie jak przy poprzednich podobnych próbach z balansowaniem.
- Końcowa jakość modelu spadła, zatem rezygnujemy z tej ścieżki.

9.3.5 Wnioski końcowe części związanej z próbą poprawienia jakości modelu poprzez zmiany w danych

9.3.6 Próba zmiany architektury modelu - zmniejszenie ilości warstw ukrytych

```
[[ 577  210   36    0    0]
 [ 243 1914  235   34    0]
 [  41  338 1038   68    2]
 [   1   38  128  290    5]
 [   0    0   12   10   25]]
              precision    recall  f1-score   support

           0       0.67      0.70      0.68       823
           1       0.77      0.79      0.78      2426
           2       0.72      0.70      0.71      1487
           3       0.72      0.63      0.67       462
           4       0.78      0.53      0.63        47

    accuracy                           0.73      5245
   macro avg       0.73      0.67      0.69      5245
weighted avg       0.73      0.73      0.73      5245
```

(a) Zmiana architektury

	precision	recall	f1-score	support
0	0.71	0.61	0.66	823
1	0.77	0.77	0.77	2426
2	0.68	0.77	0.72	1487
3	0.74	0.64	0.69	462
4	0.79	0.47	0.59	47
accuracy			0.73	5245
macro avg	0.74	0.65	0.69	5245
weighted avg	0.73	0.73	0.73	5245

(b) Początkowy

```
[[215  94  17   1   0]
 [128 794 118  14   0]
 [ 22 180 409  32   4]
 [  0  21  65 108   3]
 [  0   0   5   4  14]]
              precision    recall  f1-score   support

           0       0.59      0.66      0.62       327
           1       0.73      0.75      0.74      1054
           2       0.67      0.63      0.65       647
           3       0.68      0.55      0.61       197
           4       0.67      0.61      0.64        23

    accuracy                           0.69      2248
   macro avg       0.67      0.64      0.65      2248
weighted avg       0.69      0.69      0.68      2248
```

(c) Zmiana architektury

	precision	recall	f1-score	support
0	0.62	0.58	0.60	327
1	0.74	0.75	0.74	1054
2	0.66	0.71	0.68	647
3	0.68	0.52	0.59	197
4	0.74	0.61	0.67	23
accuracy			0.69	2248
macro avg	0.69	0.63	0.66	2248
weighted avg	0.69	0.69	0.69	2248

(d) Początkowy

Rysunek 34: Porównanie jakości predykcji dla zbioru treningowego i testowego dwóch modeli o różnej architekturze

Liczba warstw ukrytych : 2
Wnioski:

- Patrząc na same wartości ogólne parametrów zauważamy, że model uproszczony praktycznie nie różni się od startowego. Obydwa mają 69% skuteczności na zbiorze testowym, na zbiorze treningowym osiągają 73%. Jednakże przyglądając się bardziej dokładnie widzimy, że dla uproszczonego modelu precyzja klasyfikacji dla każdej z podgrup zmalała o kilka punktów procentowych. Jest to niepożądany wynik, ale być może mniejsza złożoność obliczeniowa w zamian za jedynie kilka punktów procentowych straty jest dobrą zamianą. Łatwiejszy model jest szybciej wytrenować, mniej skomplikowane modele są też łatwiejsze w utrzymywaniu.

Postanowiłyśmy zmniejszyć liczbę warstw ukrytych do jednej, aby zauważyć, czy nastąpi zauważalne pogorszenie modelu.

```
[[ 514  267   42    0    0]
 [ 187 1967  244   28    0]
 [  34  353 1027   71    2]
 [   1   44  122  290    5]
 [   0    0   12   10   25]]
              precision    recall  f1-score   support

           0       0.70      0.62      0.66       823
           1       0.75      0.81      0.78      2426
           2       0.71      0.69      0.70      1487
           3       0.73      0.63      0.67       462
           4       0.78      0.53      0.63        47

    accuracy                           0.73      5245
   macro avg       0.73      0.66      0.69      5245
weighted avg       0.73      0.73      0.73      5245
```

(a) Zmiana architektury

	precision	recall	f1-score	support
0	0.71	0.61	0.66	823
1	0.77	0.77	0.77	2426
2	0.68	0.77	0.72	1487
3	0.74	0.64	0.69	462
4	0.79	0.47	0.59	47
accuracy			0.73	5245
macro avg	0.74	0.65	0.69	5245
weighted avg	0.73	0.73	0.73	5245

(b) Początkowy

```
[[194 114  18   1   0]
 [104 834 107   9   0]
 [ 17 179 407  40   4]
 [  0  29  60 105   3]
 [  0   0   4   5  14]]
              precision    recall  f1-score   support

           0       0.62      0.59      0.60       327
           1       0.72      0.79      0.75      1054
           2       0.68      0.63      0.65       647
           3       0.66      0.53      0.59       197
           4       0.67      0.61      0.64        23

    accuracy                           0.69      2248
   macro avg       0.67      0.63      0.65      2248
weighted avg       0.69      0.69      0.69      2248
```

(c) Zmiana architektury

	precision	recall	f1-score	support
0	0.62	0.58	0.60	327
1	0.74	0.75	0.74	1054
2	0.66	0.71	0.68	647
3	0.68	0.52	0.59	197
4	0.74	0.61	0.67	23
accuracy			0.69	2248
macro avg	0.69	0.63	0.66	2248
weighted avg	0.69	0.69	0.69	2248

(d) Początkowy

Rysunek 35: Porównanie jakości predykcji dla zbioru treningowego i testowego dwóch modeli o różnej architekturze

```
[[ 514  267   42    0    0]
 [ 187 1967  244   28    0]
 [  34  353 1027   71    2]
 [   1   44  122  290    5]
 [   0    0   12   10   25]]
              precision    recall  f1-score   support

           0       0.70      0.62      0.66       823
           1       0.75      0.81      0.78      2426
           2       0.71      0.69      0.70      1487
           3       0.73      0.63      0.67       462
           4       0.78      0.53      0.63        47

    accuracy                           0.73      5245
   macro avg       0.73      0.66      0.69      5245
weighted avg       0.73      0.73      0.73      5245
```

(a) 1 warstwa ukryta

```
[[ 577  210   36    0    0]
 [ 243 1914  235   34    0]
 [  41  338 1038   68    2]
 [   1   38  128  290    5]
 [   0    0   12   10   25]]
              precision    recall  f1-score   support

           0       0.67      0.70      0.68       823
           1       0.77      0.79      0.78      2426
           2       0.72      0.70      0.71      1487
           3       0.72      0.63      0.67       462
           4       0.78      0.53      0.63        47

    accuracy                           0.73      5245
   macro avg       0.73      0.67      0.69      5245
weighted avg       0.73      0.73      0.73      5245
```

(b) 2 warstwy ukryte

```
[[194 114  18   1   0]
 [104 834 107   9   0]
 [ 17 179 407  40   4]
 [  0  29  60 105   3]
 [  0   0   4   5  14]]
              precision    recall  f1-score   support

           0       0.62      0.59      0.60       327
           1       0.72      0.79      0.75      1054
           2       0.68      0.63      0.65       647
           3       0.66      0.53      0.59       197
           4       0.67      0.61      0.64        23

    accuracy                           0.69      2248
   macro avg       0.67      0.63      0.65      2248
weighted avg       0.69      0.69      0.69      2248
```

(c) 1 warstwa ukryta

```
[[215  94  17   1   0]
 [128 794 118  14   0]
 [ 22 180 409  32   4]
 [  0  21  65 108   3]
 [  0   0   5   4  14]]
              precision    recall  f1-score   support

           0       0.59      0.66      0.62       327
           1       0.73      0.75      0.74      1054
           2       0.67      0.63      0.65       647
           3       0.68      0.55      0.61       197
           4       0.67      0.61      0.64        23

    accuracy                           0.69      2248
   macro avg       0.67      0.64      0.65      2248
weighted avg       0.69      0.69      0.68      2248
```

(d) 2 warstwy ukryte

Rysunek 36: Porównanie jakości predykcji dla zbioru treningowego i testowego dwóch modeli o różnej architekturze

Liczba warstw ukrytych : 1

Wnioski:

- Dla niektórych klas model prostszy jest nieco lepszy, zwłaszcza na danych testowych. Dlatego też o ile dalsze próby nie wykażą, że bardziej skomplikowany model jest dużo lepszy to pozostaniemy przy takiej konfiguracji warstw ukrytych.

I dla braku warstw ukrytych:

```
[[ 542  241   40    0    0]
 [ 209 1958  235   24    0]
 [  37  352 1031   65    2]
 [   1   44  128  285    4]
 [   0    0   17   10   20]]
              precision    recall  f1-score   support

           0       0.69      0.66      0.67       823
           1       0.75      0.81      0.78      2426
           2       0.71      0.69      0.70      1487
           3       0.74      0.62      0.67       462
           4       0.77      0.43      0.55        47

    accuracy                           0.73      5245
   macro avg       0.73      0.64      0.68      5245
weighted avg       0.73      0.73      0.73      5245
```

(a) 0 warstw ukrytych

```
[[ 514  267   42    0    0]
 [ 187 1967  244   28    0]
 [  34  353 1027   71    2]
 [   1   44  122  290    5]
 [   0    0   12   10   25]]
              precision    recall  f1-score   support

           0       0.70      0.62      0.66       823
           1       0.75      0.81      0.78      2426
           2       0.71      0.69      0.70      1487
           3       0.73      0.63      0.67       462
           4       0.78      0.53      0.63        47

    accuracy                           0.73      5245
   macro avg       0.73      0.66      0.69      5245
weighted avg       0.73      0.73      0.73      5245
```

(b) 1 warstwa ukryta

```
[[202 107  17   1   0]
 [118 818 107  11   0]
 [ 18 179 415  33   2]
 [  1  29  62 102   3]
 [  0   0  10   4   9]]
              precision    recall  f1-score   support

           0       0.60      0.62      0.61       327
           1       0.72      0.78      0.75      1054
           2       0.68      0.64      0.66       647
           3       0.68      0.52      0.59       197
           4       0.64      0.39      0.49        23

    accuracy                           0.69      2248
   macro avg       0.66      0.59      0.62      2248
weighted avg       0.69      0.69      0.69      2248
```

(c) 0 warstw ukrytych

```
[[194 114  18   1   0]
 [104 834 107   9   0]
 [ 17 179 407  40   4]
 [  0  29  60 105   3]
 [  0   0   4   5  14]]
              precision    recall  f1-score   support

           0       0.62      0.59      0.60       327
           1       0.72      0.79      0.75      1054
           2       0.68      0.63      0.65       647
           3       0.66      0.53      0.59       197
           4       0.67      0.61      0.64        23

    accuracy                           0.69      2248
   macro avg       0.67      0.63      0.65      2248
weighted avg       0.69      0.69      0.69      2248
```

(d) 1 warstwa ukryta

Rysunek 37: Porównanie jakości predykcji dla zbioru treningowego i testowego dwóch modeli o różnej architekturze

Liczba warstw ukrytych : 0

Wnioski:

- Tutaj podobnie nie widać większych różnić, oprócz tego że ważona jakoś predykcji dla modelu bez warstw ukrytych jest niższa o 0.1 punktu procentowego.

9.3.7 Próba zmiany architektury modelu - zwiększenie ilości warstw ukrytych

```
[[ 512  251   60    0    0]
 [ 187 1882  330   27    0]
 [  21  251 1165   48    2]
 [   1   38  146  272    5]
 [   0    0   12   10   25]]
              precision    recall  f1-score   support

           0       0.71      0.62      0.66       823
           1       0.78      0.78      0.78      2426
           2       0.68      0.78      0.73      1487
           3       0.76      0.59      0.66       462
           4       0.78      0.53      0.63        47

    accuracy                           0.74      5245
   macro avg       0.74      0.66      0.69      5245
weighted avg       0.74      0.74      0.73      5245
```

(a) Zmiana architektury

	precision	recall	f1-score	support
0	0.71	0.61	0.66	823
1	0.77	0.77	0.77	2426
2	0.68	0.77	0.72	1487
3	0.74	0.64	0.69	462
4	0.79	0.47	0.59	47
accuracy			0.73	5245
macro avg	0.74	0.65	0.69	5245
weighted avg	0.73	0.73	0.73	5245

(b) Początkowy

```
[[187 107  33   0   0]
 [110 770 163  11   0]
 [ 11 141 462  29   4]
 [  0  23  70 101   3]
 [  0   0   5   4  14]]
              precision    recall  f1-score   support

           0       0.61      0.57      0.59       327
           1       0.74      0.73      0.74      1054
           2       0.63      0.71      0.67       647
           3       0.70      0.51      0.59       197
           4       0.67      0.61      0.64        23

    accuracy                           0.68      2248
   macro avg       0.67      0.63      0.64      2248
weighted avg       0.68      0.68      0.68      2248
```

(c) Zmiana architektury

	precision	recall	f1-score	support
0	0.62	0.58	0.60	327
1	0.74	0.75	0.74	1054
2	0.66	0.71	0.68	647
3	0.68	0.52	0.59	197
4	0.74	0.61	0.67	23
accuracy			0.69	2248
macro avg	0.69	0.63	0.66	2248
weighted avg	0.69	0.69	0.69	2248

(d) Początkowy

Rysunek 38: Porównanie jakości predykcji dla zbioru treningowego i testowego dwóch modeli o różnej architekturze

Liczba warstw ukrytych : 4
Wnioski:

- Tutaj podobnie jak w poprzedniej próbie modele są bardzo podobne do siebie. Widzimy, że dla 4 warstw ukrytych precyzja na zbiorze treningowym nieco wzrosła, podobnie jak błąd na zbiorze testowym. Prawdopodobnie jest to spowodowane zbyt skomplikowaną architekturą modelu do tego zadania.

9.3.8 Wnioski płynące ze zmian ilości warstw ukrytych

Nie zauważyłyśmy w żadnej z prób ani ogromnej poprawy, ani pogorszenia się modelu. Dlatego w dalszej części rozważań użyjemy najprostszego modelu tzn. bez warstw ukrytych.

9.3.9 Próby zmian parametrów uczenia

Testowałyśmy m.in wpływ zmiany funkcji aktywacji na jakość modelu:

- Identity - znaczne obniżenie jakości
- Tanh - brak widocznych zmian

- Logistic - brak widocznych zmian

oraz wpływ zmiany solvera na jakość modelu:

- lbfgs - bez zmian
- sgd - niewielka poprawa jakości modelu
- adam - pierwotny solver

9.4 Wnioski i ustalenia końcowe

Ponieważ próby balansowania zbioru uczącego nie przynosiły żadnych korzystnych skutków na jakość przewidywań oraz zmniejszenie ilości warstw ukrytych w modelu sieci neuronowej nie pogarszało jej w drastyczny sposób, uznałyśmy, że najbardziej wydajnym i najlepszym do tego konkretnego zadania będzie sieć neuronowa trenowana na oryginalnym zbiorze uczącym bez warstw ukrytych. Pozostawiłyśmy również pierwotną funkcję aktywacji, ale zmieniłyśmy solver na sgd, dla którego otrzymałyśmy poprawę modelu. Końcowa jakość modelu na danych testowych wynosi 70%.Niestety nie udało nam się poprawić jego jakości do tego stopnia w jakim chciałyśmy. Sądzimy jednak, że być może nie jest to do końca wina implementacji modelu, lecz danych, które otrzymałyśmy od klienta i wyboru zaawansowanego modelu.
Po wstępnych analizach łatwo było zauważyć, że niewiele z atrybutów było informatywnych - większość z nich minimalnie wpływała na długość dostawy, oczywiście oprócz miasta zamieszkania kupującego.
Być może również wybór sieci neuronowej do zadania klasyfikacji okazał się być nietrafiony. Gdybyśmy miały wystarczająco dużo czasu na wytrenowanie modelu np. drzewa klasyfikacji lub lasu losowego, mogłoby się okazać,że poradziłby on sobie lepiej na otrzymanych danych, ponieważ potrafiłby lepiej zauważyć niuanse pomiędzy atrybutami i lepiej oddzielić te bardziej informatywne od tych mniej. Niemniej jednak uważamy, że jakość modelu sieci neuronowej jest wystarczająca do zgrubnego przewidywania czasu dostaw w sklepie internetowym.

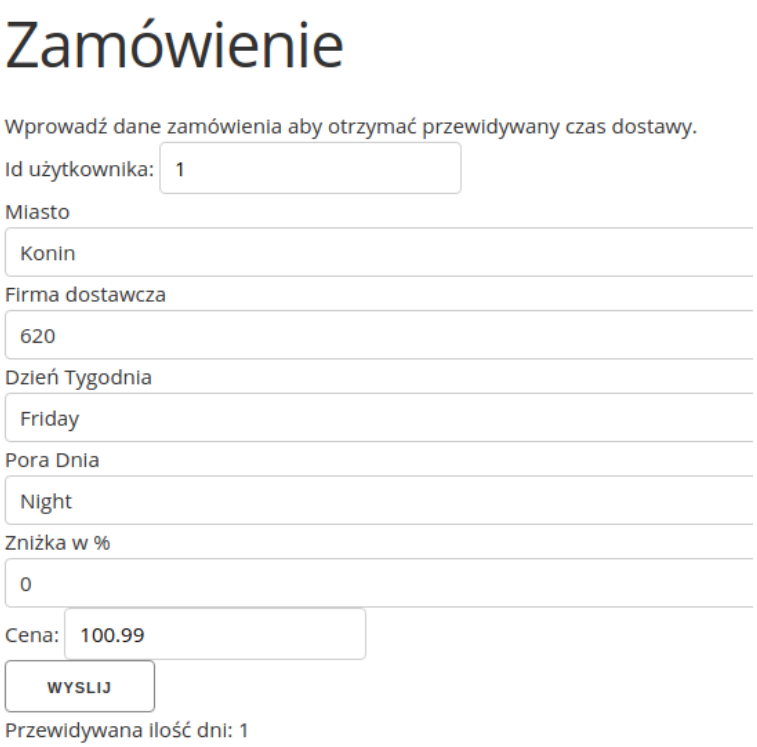
10 Mikroserwis

Aby umożliwić korzystanie z obu modeli napisałyśmy mikroserwis który pobiera od użytkownika dane na podstawie model przewiduje czas dostawy. Aby uniknąć błędów podczas wpisywania danych do formularza, użytkownik może wybierać spośród dostępnych dla niego opcji, które są odgórnie narzucone. Oczywiście możemy je w łatwy sposób modyfikować w razie potrzeby.

Trenowanie modelu od początku za każdym razem gdy klient zada pytanie byłoby niewydajne oraz wrażliwe zna bieżące zmiany w modelu dlatego użyłyśmy pickla który pozwala na zapisanie obiektów w pythonie, a następnie odtworzenie ich w dowolnym miejscu. Wytrenowany model zapisujemy w repozytorium a następnie gdy klient zadaje pytanie odtwarzamy obiekty aby móc za ich pomocą przewidywać czas dostawy. Dla odróżnienia kolejnych iteracji modeli zastosowałyśmy zasadę nazywania plików binarnych, tak aby za jej pomocą w łatwy sposób zauważyć zmiany: najpierw nazwa modelu - bayes lub neural_network a następnie liczby oddzielone od siebie kropkami:

- Pierwsza cyfra oznacza, która to iteracja API modelu, czyli np. dodawanie, usuwanie atrybutów.
- Druga cyfra - zmiana w atrybutach, np inne próbkowanie danych
- Trzecia cyfra - małe zmiany w modelu, drobne poprawki parametrów

Do stworzenia mikroserwisu użyłyśmy dasha aby rozwiązanie było proste w użyciu. Jest to narzędzie generujące kod w HTMLu oraz w przypadku bardziej skomplikowanych struktur również kod w javascrptcie (we frameworku reactowym).



Rysunek 39: Mikroserwis po uzupełnieniu.

10.1 Część mikroserwisu związana za zbieranie informacji do przeprowadzania eksperymentu A/B

Aby udogodnić przeprowadzanie eksperymentów A/B mikroserwis ma wbudowaną opcję zapisywania danych użytych w predykcji oraz samej predykcji razem ze znacznikiem czasowym do pliku. Klienci zostali podzieleni na dwie grupy, za pomocą prostej funkcji mieszającej, która na podstawie ich identyfikatorów (założyłyśmy że są unikalne) przydziela ich do grupy A lub B.

Program zapisuje predykcje dla modeli w oddzielnych plikach, w formie ramek danych (dataframów), w ten sposób w łatwy sposób możemy je później wczytać do pliku i obrobić np. za pomocą pakietu pandas. Rozdział na pliki ma też taką zaletę, że w łatwy sposób po załadowaniu możemy się przekonać ile predykcji zostało zebranych dla każdego z modeli, bez dodatkowego sortowania i rozdzielania. (których musielibyśmy dokonać, gdyby były w jednym). Znacznik czasowy jest po to, aby łatwo określić czy dana predykcja należy do wyznaczonego okresu eksperymentu, czy też nie.

W mikroserwisie nie jest zaimplementowane sprawdzanie wartości identyfikatora jaki podaje użytkownik, dlatego, że nie miałyśmy dostępu do pełnej bazy danych - należałoby to dopracować podczas wdrażania modelu.