AKADEMIA ŁOMŻYŃSKA
WYDZIAŁ NAUK INFORMATYCZNO-TECHNOLOGICZNYCH

DOKUMENTACJA PROJEKTOWA

Opracowanie przygotowane na zajęcia z przedmiotu

Wydziałowy projekt zespołowy

Zespół autorski

Gracjan Eryk Penk

Damian Piotrowski

Prowadzący zajęcia projektowe

dr inż. Janusz Rafałko

Informatyka

Studia stacjonarne I stopnia, rok IV, semestr VII

Rok akademicki: 2024/2025

SPIS TREŚCI

WSTĘP

Głównym celem projektu jest stworzenie modelu prognozowania zapotrzebowania na produkty spożywcze, wykorzystując dane historyczne oraz techniki uczenia maszynowego. Prognozy te mogą być stosowane do optymalizacji zarządzania zapasami, redukcji kosztów logistycznych i ograniczenia marnotrawstwa żywności. Projekt ma na celu opracowanie narzędzia, które pozwoli na przewidywanie ilości zamawianych produktów na podstawie danych sprzedażowych i innych czynników wpływających na popyt, takich jak sezonowość, promocje czy lokalne wydarzenia.

Prognozowanie zapotrzebowania na żywność jest istotnym problemem dla wielu firm z sektora spożywczego, ponieważ wpływa na zarządzanie łańcuchem dostaw, koszty operacyjne oraz minimalizację strat związanych z nadprodukcją lub niedoborami produktów. Celem projektu jest zbudowanie modelu predykcyjnego, który będzie analizował dane historyczne, wykrywał wzorce i prognozował przyszły popyt. Model ten zostanie opracowany z wykorzystaniem narzędzi takich jak regresja, sieci neuronowe, LSTM (Long Short-Term Memory) czy modele szeregów czasowych (np. ARIMA, Prophet).

1. WSTĘPNE ZAŁOŻENIA

1.1. Podział zadań

Osoba 1:

- Przygotowanie danych do pracy.
- Wstępna implementacja modeli predykcyjnych oraz algorytmów uczenia maszynowego
- Współpraca przy tworzeniu ostatecznego modelu.
- Przygotowanie dokumentacji.

Osoba 2:

- Stworzenie ostatecznego modelu.
- Przeprowadzenie walidacji i optymalizacji modeli.
- Analiza wyników i ich interpretacja.
- Prezentacja wyników pracy.

1.2. Wybór języka programowania

Projekt zostanie zrealizowany w języku Python, który jest szeroko stosowany w analizie danych oraz uczeniu maszynowym. Python oferuje bogaty ekosystem narzędzi, które ułatwią analizę, przetwarzanie danych oraz budowę modeli predykcyjnych.

1.3. Dobór narzędzi do przetwarzania danych

Wstępna propozycja doboru narzędzi do przetwarzania danych:

- Pandas – do analizy i manipulacji danymi.
- NumPy – do operacji numerycznych.
- Scikit-learn – do implementacji modeli klasycznych, jak regresja, drzewa decyzyjne.
- Matplotlib/Seaborn – do wizualizacji danych i wyników modelowania.
- xgboost – model XGBoostRegressor

1.4. Przegląd literatury

1.4.1. FOOD DEMAND PREDICTION USING MACHINE LEARNING

Artykuł „Food Demand Prediction Using Machine Learning" opublikowany w International Research Journal of Engineering and Technology (IRJET) opisuje metody prognozowania popytu na produkty spożywcze w restauracjach za pomocą algorytmów uczenia maszynowego i analizy statystycznej. Celem pracy jest przewidywanie liczby zamówień, co pomaga restauracjom zarządzać zapasami składników o krótkim terminie przydatności, zmniejszając tym samym marnotrawstwo żywności i poprawiając efektywność operacyjną.

Artykuł wskazuje, że prognozowanie popytu na żywność jest trudnym zadaniem z powodu licznych czynników wpływających na zmienność zamówień, takich jak sezonowość, preferencje klientów, promocje czy pogoda. Celem artykułu jest opracowanie modeli, które pozwolą przewidywać liczbę zamówień w restauracjach, co pomoże w efektywniejszym zarządzaniu zapasami.

W badaniu wykorzystano różne techniki uczenia maszynowego, w tym:

- Bayesian Linear Regression: Model oparty na sieciach Bayesowskich, który używa rozkładów prawdopodobieństwa do prognozowania zmiennych. Model ten umożliwia przewidywanie na podstawie zależności warunkowych między zmiennymi losowymi.

- Random Forest: Technika wykorzystująca wiele drzew decyzyjnych do prognozowania. Model ten dobrze radzi sobie zarówno z danymi liniowymi, jak i nieliniowymi.

- Support Vector Machine (SVM): Popularny algorytm do klasyfikacji, który tworzy hiperpłaszczyznę w celu rozdzielenia danych na różne klasy.

- LASSO Regression: Technika regresji stosująca „kurczenie" zmiennych, co pozwala na uproszczenie modelu i wybór najistotniejszych zmiennych.

- XGBoost: Zaawansowany algorytm oparty na gradient boosting, który jest wyjątkowo wydajny pod względem obliczeń i często daje lepsze wyniki w porównaniu do innych technik.

W badaniu przeprowadzono prognozowanie liczby zamówień na podstawie danych wewnętrznych (takich jak liczba zamówień z poprzednich tygodni) oraz danych zewnętrznych (np. promocje). Stwierdzono, że algorytm XGBoost osiągnął najlepsze wyniki, zapewniając najwyższą dokładność prognozowania. Inne algorytmy, takie jak regresja Bayesowska i LASSO, miały niższą dokładność.

Dodatkowo, artykuł podkreśla znaczenie inżynierii cech, czyli procesu przekształcania danych w formy, które są bardziej użyteczne dla algorytmów uczenia maszynowego. Na przykład, użyto kodowania etykiet dla danych kategorycznych.

Autorzy zauważają, że dokładność modeli można jeszcze poprawić, uwzględniając dodatkowe czynniki, takie jak zwyczaje kulturowe czy święta religijne. W przyszłości planowane jest rozszerzenie metod na inne branże, takie jak prognozowanie zapotrzebowania na pracowników.

Artykuł pokazuje, że zastosowanie algorytmów uczenia maszynowego w prognozowaniu popytu na żywność może znacznie poprawić dokładność przewidywań. Zastosowanie algorytmu XGBoost okazało się najskuteczniejsze, ale istnieje potencjał na dalsze ulepszenia poprzez uwzględnienie dodatkowych danych kontekstowych.

1.4.2. A CASE ANALYSIS OF A SUSTAINABLE FOOD SUPPLY CHAIN DISTRIBUTION SYSTEM – A MULTI-OBJECTIVE APPROACH

Badanie „A case analysis of a sustainable food supply chain distribution system—A multi-objective approach" przeprowadzone przez 'Management Group' ze Szkoły Biznesu Uniwersytetu Miasta Dublin dotyczy analizy zrównoważonego systemu dystrybucji w łańcuchu dostaw żywności, ze szczególnym naciskiem na przemysł mleczarski w Irlandii. Przedstawiono model dystrybucji dwuwarstwowej, którego celem jest minimalizacja emisji CO2 oraz kosztów związanych z transportem mleka.

Zielony system dystrybucji uwzględnia zarówno wpływ na środowisko, jak i efektywność ekonomiczną. Model równoważy redukcję emisji węglowych z kontrolą kosztów.

Model uwzględnia również alternatywne scenariusze, które zakładają otwarcie zamkniętych dotąd tras dystrybucji. Ma to na celu zwiększenie odporności systemu dystrybucji na różne nieprzewidziane zmiany, takie jak zamknięcie tras lub problemy operacyjne.

Wyniki modelu są także przedstawiane w formie geograficznej, co pozwala na wizualizację optymalnych tras transportowych. Geograficzne rozmieszczenie tras pomaga lepiej zrozumieć, które trasy są najbardziej zrównoważone z punktu widzenia zarówno kosztów, jak i emisji węglowych.

Technologie wykorzystane w badaniu:

W badaniu zastosowano trzy różne **algorytmy genetyczne** – **NSGA-II**, **MOGA-II** oraz **metodę hybrydową**. Algorytmy genetyczne to techniki inspirowane procesami ewolucji biologicznej, które są używane do znajdowania optymalnych rozwiązań w problemach z dużą liczbą zmiennych i ograniczeń, takich jak logistyka transportu. W tym przypadku, algorytmy te pomogły w identyfikacji najlepszych tras dystrybucji, które znajdują się na tzw. froncie Pareto, czyli zestawie rozwiązań, gdzie żadna z nich nie jest gorsza od innych pod względem wszystkich celów optymalizacyjnych. Wyniki analizy wykazały, że algorytm NSGA-II (ang. Non-Dominated Sorting Genetic Algorithm II) okazał się najbardziej efektywny w porównaniu z pozostałymi metodami.

NSGA-II (Non-Dominated Sorting Genetic Algorithm II): Jest to jeden z najczęściej używanych algorytmów do optymalizacji wielokryterialnej. NSGA-II sortuje populację rozwiązań na podstawie ich dominacji względem innych rozwiązań – szuka tych, które nie są gorsze w żadnym z badanych aspektów (w tym przypadku, emisje i koszty). Algorytm generuje zbiór rozwiązań, który jest rozłożony na froncie Pareto, dając wiele opcji wyboru.

MOGA-II (Multi-Objective Genetic Algorithm II): Jest to inna wersja algorytmu genetycznego do rozwiązywania problemów wielokryterialnych. Podobnie jak NSGA-II, MOGA-II szuka optymalnych rozwiązań na froncie Pareto, ale może różnić się w sposobie zarządzania populacją i selekcji rozwiązań.

Metoda hybrydowa (GA + Sequential Quadratic Programming): Jest to kombinacja algorytmów genetycznych z bardziej tradycyjnymi metodami optymalizacji numerycznej, takimi jak programowanie kwadratowe sekwencyjne (SQP). Podejście hybrydowe często stosuje się, aby połączyć zalety obu metod – eksploracyjnych właściwości GA z precyzją optymalizacji numerycznej.

TOPSIS (Technique for Order Preference by Similarity to Ideal Solution): Jest to narzędzie wielokryterialne służące do porównywania różnych opcji decyzyjnych. Umożliwia ocenę, które rozwiązanie jest najbliższe do "idealnego" (najlepszego możliwego), bazując na odległości od hipotetycznego idealnego punktu w wielowymiarowej przestrzeni decyzji. W badaniu TOPSIS pomógł ocenić trasy pod kątem emisji CO_2 i kosztów, tworząc hierarchię najbardziej efektywnych tras.

To badanie podkreśla rosnącą potrzebę wprowadzania bardziej ekologicznych i wydajnych systemów dystrybucji w łańcuchach dostaw żywności. Zastosowane algorytmy genetyczne i technologie optymalizacyjne stanowią skuteczne narzędzia, które umożliwiają jednoczesne zarządzanie kosztami i emisją węglową. Rozwiązania proponowane w modelu mogą być szerzej stosowane w innych sektorach, które muszą balansować między zrównoważonym rozwojem a efektywnością operacyjną.

2. ANALIZA I PRZYGOTOWANIE DANYCH

2.1. Źródło i charakterystyka danych

Dane wykorzystywane w projekcie pochodzą z systemu zamówień klienta i obejmują historyczne zapisy dotyczące zapotrzebowania na posiłki w połączeniu z konkretnymi centrami dystrybucji. Zestaw danych historycznych obejmuje okres od tygodnia 1 do 145, co pozwala na identyfikację wzorców sezonowych oraz trendów.

Dane obejmują:

- Zapotrzebowanie na dany posiłek w centrach dystrybucji – historyczne zapotrzebowanie (liczba zamówień) dla każdej kombinacji posiłku i centrum w tygodniach 1-145.
- Cechy posiłków – informacje takie jak kategoria, podkategoria, obecna cena oraz ewentualny rabat na posiłek. Te cechy mogą być pomocne w analizie wpływu sezonowych rabatów i kategorii posiłków na poziom zamówień.
- Informacje o centrach dystrybucji – szczegółowe dane o centrach dystrybucji, w tym miasto, obszar centrum oraz kod regionu. Lokalizacja i specyfika centrum mogą wpływać na popyt, np. centra w obszarach o dużym natężeniu ruchu mogą wykazywać wyższy poziom zapotrzebowania.

2.2. Wstępne przetwarzanie danych

Przetwarzanie danych jest kluczowe dla poprawy jakości wejść do modelu, eliminacji nieścisłości oraz zminimalizowania wpływu błędów w danych na końcowe wyniki. Kroki przetwarzania danych obejmują:

- Uzupełnienie lub usunięcie braków danych: Dane zostały sprawdzone pod kątem brakujących wartości, które mogłyby negatywnie wpłynąć na wyniki modelu. W zależności od ich liczby i istotności, braki zostały usunięte lub uzupełnione.
- Transformacja zmiennych numerycznych: W celu ujednolicenia wartości, takie zmienne jak ceny czy liczba zamówień zostały znormalizowane. To pozwala na lepszą interpretację różnic w wartościach i zapobiega dominacji jednej cechy nad innymi.

- Standaryzacja jednostek czasowych: Dla zachowania spójności dane o zapotrzebowaniu są analizowane w rozbiciu na tygodnie, co odpowiada cyklowi zaopatrzeniowemu i pozwala przewidywać popyt w przyszłości.

2.3. Inżynieria cech

Inżynieria cech była kluczowym krokiem w celu wzbogacenia modelu o nowe zmienne:

- Kwota rabatu – różnica między ceną podstawową a ceną końcową, odzwierciedlająca wysokość zniżki.
- Procent rabatu – wartość procentowa rabatu, informująca, jak dużą zniżkę uzyskał klient.
- Zmienne binarne – wskaźnik, czy posiłek był objęty zniżką.
- Kodowanie zmiennych kategorycznych: Zmieniono cechy tekstowe (center_type, category, cuisine) na wartości numeryczne za pomocą pd.get_dummies().

3. DOBÓR I IMPLEMENTACJA MODELI

3.1. Wybór modeli uczenia maszynowego

Do prognozowania popytu na jedzenie wybrano trzy modele:

- Random Forest Regressor – model lasu losowego, który agreguje wyniki wielu drzew decyzyjnych. Charakteryzuje się odpornością na przeuczenie i jest efektywny przy danych o dużej liczbie cech.
- Gradient Boosting Regressor – model wzmocnienia gradientowego, który buduje kolejne drzewa decyzyjne, minimalizując błędy wcześniejszych drzew. Dzięki tej metodzie osiąga wyższą dokładność niż las losowy, ale jest bardziej zasobochłonny.
- XGBoost Regressor – zoptymalizowany model gradientowego wzmocnienia, znany z wysokiej dokładności i wydajności, szczególnie przy dużych zbiorach danych.

3.2. Trening modeli i parametryzacja

Dane historyczne z tygodni 1–145 zostały wykorzystane jako zbiór treningowy, a dane z tygodni 146–155 jako zbiór testowy. Modele były trenowane z różną liczbą estymatorów (n_estimators=100 i n_estimators=500) oraz ustawioną liczbą losową (random_state=42) w celu porównania ich wyników.

3.3. Pseudokod implementacji

```
# Importowanie bibliotek
Wczytaj numpy, pandas, matplotlib.pyplot, RandomForestRegressor,
GradientBoostingRegressor, mean_squared_error z odpowiednich pakietów
Wczytaj XGBRegressor z xgboost

# Ładowanie danych
Wczytaj pliki: 'train.csv', 'fulfilment_center_info.csv', 'meal_info.csv',
'test_QoiMO9B.csv'

# Łączenie danych
Połącz data i test wzdłuż osi wierszy
Połącz data i center na podstawie 'center_id'
Połącz data i meal na podstawie 'meal_id'

# Inżynieria cech
Dla każdej obserwacji oblicz:
    'discount_amount' jako różnica między 'base_price' a 'checkout_price'
    'discount_percent' jako procent rabatu
    'discount_y/n' jako 1, jeśli base_price > checkout_price, inaczej 0

# Kodowanie zmiennych kategorycznych
Zastosuj kodowanie `pd.get_dummies()` do kolumn: 'center_type', 'category',
'cuisine'

# Podział na zbiór treningowy i testowy
Utwórz train jako dane dla tygodni 1-145
Utwórz test jako dane dla tygodni 146-155

# Selekcja cech i etykieta
Utwórz X_train jako wszystkie kolumny z train poza 'num_orders', 'id',
'week'
Utwórz y_train jako kolumna 'num_orders' z train
Utwórz X_test jako wszystkie kolumny z test poza 'num_orders', 'id', 'week'

# Trenowanie modeli
Dla modelu RandomForestRegressor z n_estimators=100 i random_state=42:
    Dopasuj model do X_train i y_train
    Wykonaj prognozy na X_test

Dla modelu GradientBoostingRegressor z n_estimators=100 i random_state=42:
    Dopasuj model do X_train i y_train
    Wykonaj prognozy na X_test

Dla modelu XGBoostRegressor z n_estimators=100 i random_state=42:
    Dopasuj model do X_train i y_train
    Wykonaj prognozy na X_test

# Ocena modeli na zbiorze treningowym
Dla każdego modelu oblicz RMSE na podstawie prognoz dla X_train i
rzeczywistych wartości y_train

# Wizualizacja wyników
Wykres słupkowy porównujący RMSE dla każdego modelu
Wykres liniowy porównujący wartości rzeczywiste i przewidywane dla
wybranych próbek testowych
```

Listing 3.1 Pseudokod projektu

3.4. Kod projektu

```
# Importowanie bibliotek
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from xgboost import XGBRegressor

# Wczytywanie danych
train_data = pd.read_csv('train.csv')
center_info = pd.read_csv('fulfilment_center_info.csv')
meal_info = pd.read_csv('meal_info.csv')
test_data = pd.read_csv('test_QoiMO9B.csv')

# Łączenie danych
train_data = pd.concat([train_data, test_data], axis=0)
train_data = train_data.merge(center_info, on='center_id', how='left')
train_data = train_data.merge(meal_info, on='meal_id', how='left')

# Inżynieria cech
train_data['discount_amount'] = train_data['base_price']-
train_data['checkout_price']
train_data['discount_percentage'] = (train_data['discount_amount'] /
train_data['base_price']) * 100
train_data['discount_y/n'] = (train_data['base_price'] >
train_data['checkout_price']).astype(int)

# Kodowanie zmiennych kategorycznych
train_data = pd.get_dummies(train_data, columns=['center_type', 'category',
'cuisine'], drop_first=True)

# Podział na zbiór treningowy i testowy
train = train_data[train_data['week'].isin(range(1, 146))]
test = train_data[train_data['week'].isin(range(146, 156))]

# Selekcja cech i etykiety
x_train = train.drop(['num_orders', 'id', 'week'], axis=1)
y_train = train['num_orders']
x_test = train.drop(['num_orders', 'id', 'week'], axis=1)

# Trenowanie modelu RandomForestRegressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(x_train, y_train)
rf_predictions = rf_model.predict(x_test)

# Trenowanie modelu GradientBoostingRegressor
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_model.fit(x_train, y_train)
gb_predictions = gb_model.predict(x_test)

# Trenowanie modelu XGBoostRegressor
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model.fit(x_train, y_train)
xgb_predictions = xgb_model.predict(x_test)

# Ocena modeli na zbiorze treningowym
rf_train_predictions = rf_model.predict(x_train)
```

```
gb_train_predictions = gb_model.predict(x_train)
xgb_train_predictions = xgb_model.predict(x_train)

rf_rmse = mean_squared_error(y_train, rf_train_predictions, squared=False)
gb_rmse = mean_squared_error(y_train, gb_train_predictions, squared=False)
xgb_rmse = mean_squared_error(y_train, xgb_train_predictions,
squared=False)

# Wizualizacja wynikóww RMSE dal każdego modelu
plt.figure(figsize=(10, 5))
plt.bar(['Random Forest', 'Gradient Boosting', 'XGBoostRegressor'],
[rf_rmse, gb_rmse, xgb_rmse], color=['blue', 'green', 'purple'])
plt.ylabel('RMSE')
plt.title('Porównanie błędu RMSE dla Random Forest, Gradient Boosting i
XGBoostRegressor')
plt.show()

# Wizualizacja porównania wartości rzeczywistych i przewidywanych
# Używamy tylko próbek z testu dla czytelności wykresu
plt.figure(figsize=(12, 6))
plt.plot(y_train.values[:50], label='Rzeczywiste', color='black')
plt.plot(rf_predictions[:50], label='Random Forest Przewidywania',
color='blue')
plt.plot(gb_predictions[:50], label='Gradient Boosting Przewidywania',
color='green')
plt.plot(xgb_predictions[:50], label='XGBoostRegressor Pzewidywanie',
color='purple')
plt.legend()
plt.xlabel('Indeks próby')
plt.ylabel('Liczba zamówień')
plt.title('Porównanie rzeczywistych i przewidywanych wartości liczby
zamówień')
plt.show()
```

Listing 3.2 Kod projektu

Importowanie bibliotek:

- numpy i pandas do pracy z danymi.
- RandomForestRegressor i GradientBoostingRegressor jako modele regresji.
- train_test_split z sklearn.model_selection do podziału danych.
- mean_squared_error do oceny błędu modelu.

Ładowanie danych: Wczytywane są cztery zbiory danych:

- data – zbiór treningowy zawierający informacje o zapotrzebowaniu na jedzenie.
- center – informacje o centrum realizacji.
- meal – informacje o posiłkach.
- test – zbiór testowy, który służy do prognozowania zapotrzebowania.

Łączenie danych:

- Funkcja pd.concat łączy zbiór treningowy i testowy, co pozwala na jednolite przetwarzanie cech.
- Następnie merge łączy dane data z danymi center i meal na podstawie kluczy center_id oraz meal_id.

Inżynieria cech:

- discount_amount – różnica między ceną podstawową a ceną końcową (czyli kwota zniżki).
- discount_percent – procentowa wartość zniżki.
- discount_y/n – wskaźnik 1, jeśli zniżka jest dostępna, i 0, jeśli nie ma zniżki.

Podział danych na zbiór treningowy i testowy:

- Dane są podzielone według numerów tygodnia. Tygodnie 1-145 stanowią zbiór treningowy, a tygodnie 146-155 są traktowane jako testowy.

Selekcja cech:

- X_train zawiera wszystkie kolumny poza num_orders (liczba zamówień, czyli zmienna docelowa), id oraz week.
- y_train zawiera wartości docelowe num_orders dla treningu.
- X_test zawiera cechy bez num_orders, id i week, na podstawie których przewidujemy zapotrzebowanie.

RandomForestRegressor:

- RandomForestRegressor to model lasu losowego z liczbą drzew (n_estimators) ustawioną na 100 i stałą losową (random_state) dla powtarzalności wyników.
- .fit(X_train, y_train) trenuje model na danych treningowych.
- .predict(X_test) przewiduje liczbę zamówień na danych testowych.

GradientBoostingRegressor:

- GradientBoostingRegressor to model bazujący na gradientowym wzmocnieniu, również z liczbą estymatorów ustawioną na 100 i stałą losową dla powtarzalności.
- .fit(X_train, y_train) – trenuje model na tych samych danych.

- .predict(X_test) – generuje przewidywania dla zbioru testowego.

XGBoostRegressor:

- XGBoostRegressor to zoptymalizowany model bazujący na gradientowym wzmocnieniu, również z liczbą estymatorów ustawioną na 100 i stałą losową dla powtarzalności.
- .fit(X_train, y_train) – trenuje model na tych samych danych.
- .predict(X_test) – generuje przewidywania dla zbioru testowego.

Ocena błędu przy użyciu RMSE (Root Mean Squared Error):

- -mean_squared_error(y_train, rf_train_predictions, squared=False) oblicza pierwiastek średniego błędu kwadratowego dla modelu RandomForest.
- -mean_squared_error(y_train, gb_train_predictions, squared=False) wykonuje to samo dla modelu GradientBoosting.
- -Wyniki RMSE wskazują, który model lepiej dopasowuje się do danych.

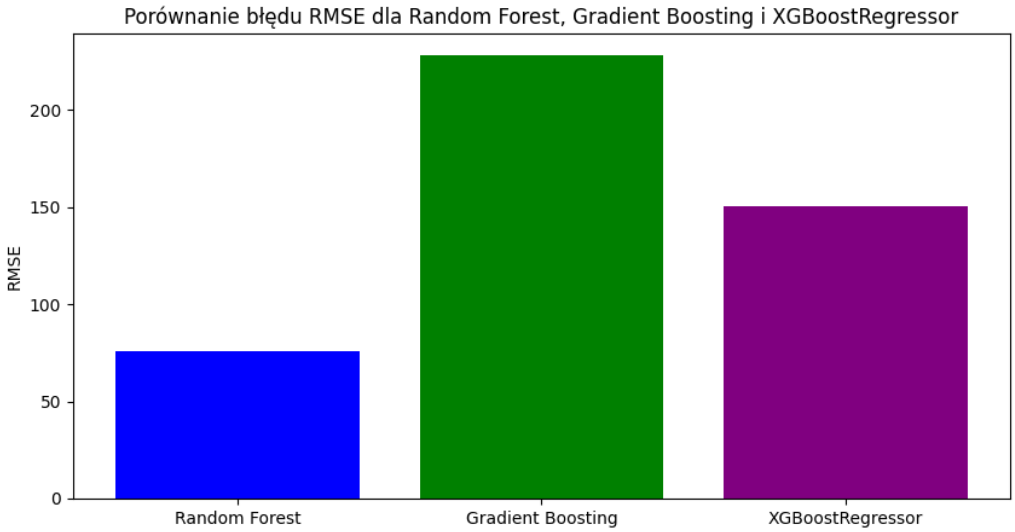
Wizualizacja wyników:

- -Porównanie RMSE – wykres słupkowy z błędami RMSE dla obu modeli.
- -Porównanie wartości rzeczywistych i przewidywanych – wykres liniowy dla próbek z testu, pokazujący, jak przewidywania obu modeli wypadają względem rzeczywistych wartości. Dla czytelność wykresu przedstawiono tylko wyniki 50 pierwszych próbek.

3.5. Wizualizacja wyników

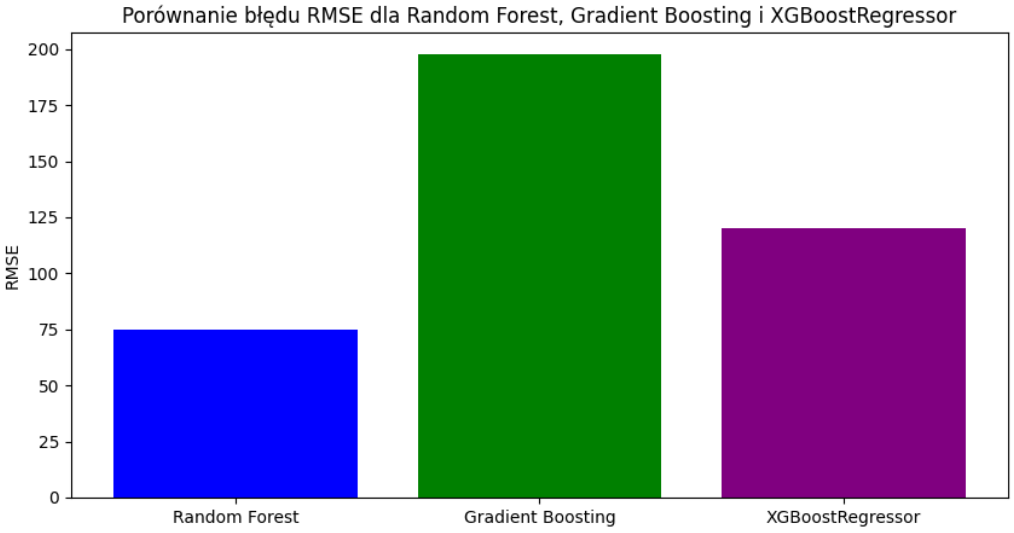
Dla lepszego zrozumienia wyników przewidywań zostały stworzone następujące wykresy:

- Porównanie RMSE dla obu modeli: Wykres słupkowy prezentujący błędy RMSE dla Random Forest, Gradient Boosting i XGBoost Regressor. Dzięki temu widzimy, który model osiągnął mniejszy błąd na zbiorze treningowym.
- Porównanie wartości rzeczywistych i przewidywanych: Wykresy liniowe przedstawiające przewidywania obu modeli dla próbek z testu w odniesieniu do rzeczywistych wartości liczby zamówień. Ułatwia to ocenę, który model lepiej odwzorowuje zmienność w danych.



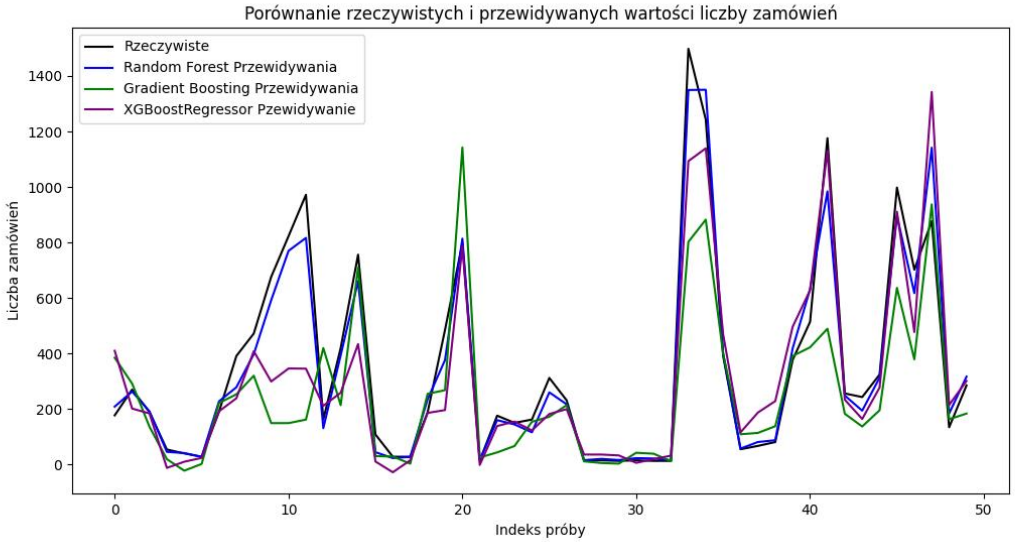
Rysunek 3.1 RMSE dla 100 estymatorów

Random Forest osiąga najniższy błąd RMSE, co sugeruje jego stabilność i odporność na szum w danych.



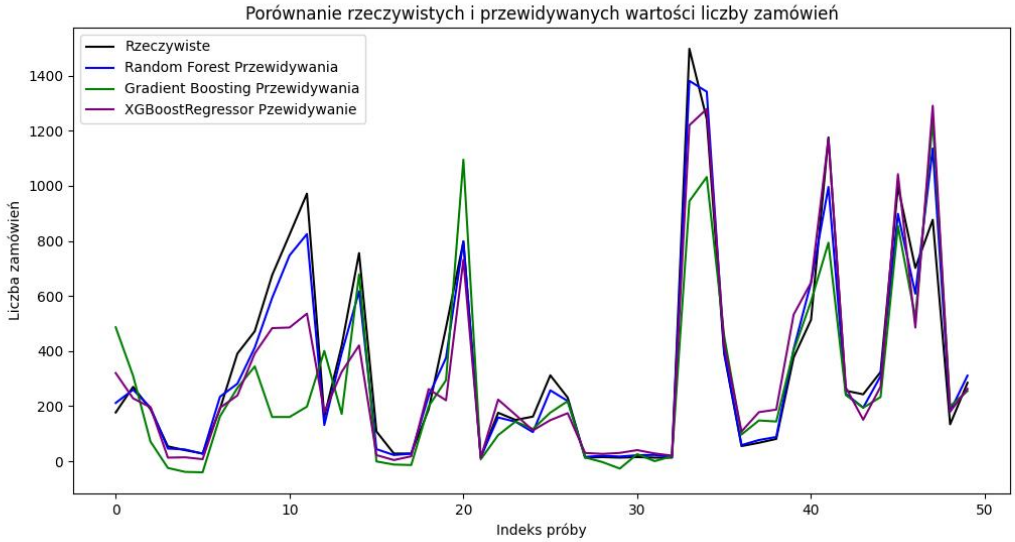
Rysunek 3.2 RMSE dla 500 estymatorów

XGBoost poprawia swoje wyniki wraz ze wzrostem liczby estymatorów, ale Random Forest nadal osiąga najlepszy wynik.



Rysunek 3.3 Przewidywania dla 100 estymatorów

Random Forest lepiej odwzorowuje rzeczywiste wartości w porównaniu z Gradient Boosting i XGBoost.



Rysunek 3.4 Przewidywania dla 500 estymatorów

Po zwiększeniu liczby estymatorów modele XGBoost i Gradient Boosting zbliżają się do dokładności Random Forest, ale nadal pozostają w tyle.

4. WNIOSKI I PRZYSZŁE PRACE

4.1. Podsumowanie wyników

Random Forest okazał się najbardziej efektywnym modelem dzięki swojej odporności na szum i zdolności do generalizacji. XGBoost, choć bardziej złożony, wymaga precyzyjnego strojenia parametrów. Gradient Boosting, choć wydajny, nie osiągnął takich wyników jak Random Forest..

4.2. Możliwości rozwoju

Projekt można rozwijać poprzez:

- Testowanie dodatkowych modeli: Włączenie algorytmów, takich jak LightGBM czy CatBoost, które również dobrze radzą sobie z danymi o dużej liczbie cech.
- Rozbudowę zbioru danych o nowe cechy: Uwzględnienie sezonowości, wydarzeń i trendów pogodowych, co może dodatkowo zwiększyć dokładność.
- Automatyzację aktualizacji modelu: Implementacja mechanizmu monitorującego dane i automatycznie aktualizującego model na podstawie najnowszych informacji.

Rozbudowa projektu może znacznie zwiększyć dokładność i zdolność modelu do adaptacji do zmieniających się warunków rynkowych, co będzie przydatne dla klienta w codziennym zarządzaniu łańcuchem dostaw.

SPIS RYSUNKÓW

SPIS LISTINGÓW

BIBLIOGRAFIA

1. International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 p-ISSN: 2395-0072 Volume: 07 Issue: 06, June 2020, Pages 3672-3675
2. International Journal of Production Economics Volume 152,June 2014,Pages 71-87
3. https://www.sciencedirect.com/science/article/abs/pii/S0925527314000437