Przewidywanie opóźnień lotów

Spis treści

[Ćwiczenia 1 1](#_Toc182948608)

[Ćwiczenia 2 2](#_Toc182948609)

[Zadanie 1 2](#_Toc182948610)

[Zadanie 2 4](#_Toc182948611)

[Zadanie 3 5](#_Toc182948612)

[Ćwiczenia 3 6](#_Toc182948613)

# Ćwiczenia 1

Cel projektu: Celem jest stworzenie modelu, który będzie przewidywać prawdopodobieństwo opóźnienia lotu na podstawie różnych cech, takich jak: linia lotnicza, lotnisko początkowe, lotnisko docelowe, godzina odlotu, dzień odlotu.

Problem projektów: Możliwość lepszego planowania lotów.

Źródło danych: - [[Link-dataset]](https://www.kaggle.com/datasets/jimschacko/airlines-dataset-to-predict-a-delay) – około 539383 linii danych.

Kolumny:

- Airline

- Flight

- Airport From

- Airport To

- DayOfWeek

- Time

Cele projektu:   
- Zbudowanie modelu będącego w dokładny sposób przewidywać możliwe opóźnienia lotów.

- Analiza i eksploracja danych.

- Trenowanie oraz walidowanie modelu.

- Dokształcanie modelu.

- Publikacja oraz prezentacja.

# Ćwiczenia 2

## Zadanie 1

EDA zostalo przeprowadzone przy pomocy Sweetviz, pandas, seaborn i matplotlib.

Obraz zawierający tekst, zrzut ekranu, kwadrat, Prostokąt

Opis wygenerowany automatycznie

Obraz zawierający diagram, tekst, zrzut ekranu, Wykres

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, Wykres

Opis wygenerowany automatycznie

Analiza raportu sweetviz:  
Dane zawierają informacje o lotach, takie jak linia lotnicza, numer lotu, lotnisko wylotu i przylotu, dzień tygodnia, czas wylotu, długość lotu oraz status opóźnienia. Zmienna docelowa Delay wskazuje, czy dany lot był opóźniony (1) czy nie (0), co czyni ten zestaw danych odpowiednim do klasyfikacji binarnej. Kolumny kategoryczne, takie jak Airline, Flight, AirportFrom, AirportTo i DayOfWeek, wymagają zakodowania, aby mogły być użyte w modelu. Nie wykryto brakujących danych, co upraszcza przetwarzanie. Analiza wskazuje również, że czas wylotu oraz dzień tygodnia mogą wpływać na ryzyko opóźnień, co może być istotnym czynnikiem w modelu predykcyjnym.

## Zadanie 2

**Najlepsze Modele**

1. **XGBoost Classifier (Extreme Gradient Boosting)**:
   * **Accuracy**: 0.6638
   * **AUC**: 0.7153
   * **Recall**: 0.5006
   * **Precision**: 0.6621
   * **F1 Score**: 0.5701
   * **Opis**: XGBoost to wydajny model gradient boosting, który często osiąga dobre wyniki na dużych zbiorach danych. Jego wysoka dokładność i AUC sugerują, że dobrze rozpoznaje wzorce w danych dotyczących opóźnień lotów.
2. **LightGBM Classifier (Light Gradient Boosting Machine)**:
   * **Accuracy**: 0.6576
   * **AUC**: 0.7071
   * **Recall**: 0.4549
   * **Precision**: 0.6705
   * **F1 Score**: 0.5421
   * **Opis**: LightGBM jest zoptymalizowanym modelem gradient boosting, który działa szybko przy dużych zbiorach danych. Model wykazuje dobrą dokładność oraz przyzwoitą precyzję, co czyni go konkurencyjnym modelem dla XGBoost.
3. **Gradient Boosting Classifier**:
   * **Accuracy**: 0.6475
   * **AUC**: 0.6922
   * **Recall**: 0.4066
   * **Precision**: 0.6726
   * **F1 Score**: 0.5068
   * **Opis**: Klasyczny Gradient Boosting również okazał się skuteczny, choć nieco słabszy niż XGBoost i LightGBM. Model nadal może być wartościowy, jeśli zależy nam na stabilnych wynikach w klasyfikacji opóźnień lotów.

**Wybór Modelu i Tuning**

* **Wybrany model**: Na podstawie najwyższej dokładności oraz AUC, **XGBoost** jest najlepszym modelem.
* **Tuning**: Próba dostrojenia nie poprawiła wyników, co oznacza, że wybrana konfiguracja pierwotna była optymalna.

## Zadanie 3

Podstawowe metryki jakości modelu na zbiorze walidacyjnym:

Dokładność (Accuracy): 0.6650

F1-Score: 0.5709

AUC: 0.7170

Średni błąd bezwzględny (MAE): 0.3350

Ćwiczenia 3  
  
Celem projektu było stworzenie dwóch zautomatyzowanych potoków danych (DAG-ów) w Apache Airflow:

DAG 1: Pobieranie danych, dzielenie ich na zbiory treningowe i testowe oraz zapisywanie ich do Google Sheets.

DAG 2: Pobieranie przetworzonych danych, ich czyszczenie i standaryzacja, a następnie zapisanie wyników z mapowaniami klucz-wartość do Google Sheets.

**DAG 1: Przetwarzanie i podział danych**

**Funkcjonalność**

1. **Pobieranie danych z pliku lokalnego**:
   * Dane w formacie .csv są wczytywane z lokalnej ścieżki.
2. **Podział danych**:
   * Dane są dzielone przy użyciu funkcji train\_test\_split z biblioteki scikit-learn na:
     + Zbiór modelowy (70%)
     + Zbiór douczeniowy (30%)
3. **Zapis do Google Sheets**:
   * Oba zbiory są zapisywane do osobnych arkuszy w Google Sheets (Zbiór\_Modelowy i Zbiór\_Douczeniowy).

**DAG 2: Czyszczenie i standaryzacja danych**

**Funkcjonalność**

1. **Pobieranie danych z Google Sheets**:
   * Wczytywane są dane z arkusza Zbiór\_Modelowy.
2. **Czyszczenie danych**:
   * Usuwanie duplikatów.
   * Obsługa brakujących wartości poprzez ich zastąpienie wartością Unknown.
3. **Standaryzacja danych**:
   * Zmienne kategoryczne są zamieniane na wartości liczbowe za pomocą LabelEncoder.
   * Tworzenie mapowań klucz-wartość dla zmiennych kategorycznych w formacie JSON (label\_mappings.json).
4. **Zapis danych**:
   * Przekształcone dane zapisywane są do Google Sheets w arkuszu Przetworzone\_Dane.

**Konfiguracja środowiska**

**Apache Airflow**

* Konteneryzowany za pomocą docker-compose.
* Kluczowe zależności:
  + gspread i oauth2client do obsługi Google Sheets.
  + scikit-learn do podziału danych i standaryzacji.
  + pandas do manipulacji danymi.
* Plik docker-compose.yaml został zaktualizowany o wymagane biblioteki.

**Google Sheets API**

* Utworzono konto serwisowe na platformie Google Cloud Console.
* Pobieranie danych za pomocą biblioteki gspread.
* Autoryzacja z użyciem pliku JSON z kluczem serwisowym.

**Pliki pomocnicze**

1. **flightdelay.json**:
   * Klucz serwisowy Google używany do autoryzacji.
2. **label\_mappings.json**:
   * Zawiera mapowania klucz-wartość dla zmiennych kategorycznych.

# Ćwiczenia 4

Stworzono 2 dagi model\_training\_dag.py i process\_data\_dag.py, dodano wymagane foldery w /airflow tj. models,processed\_data, reports i visualizations.

**process\_data\_dag.py:**

* Pobiera dane
* Usuwa duplikaty, uzupelnia braki
* Zapisuje przetworzone dane do processed\_data.csv w folderze procesed\_data
* Tworzy wizualizacje przy użyciu matplota

**Model\_training\_dag.py**

* Pobiera przetworozne dane
* Ustawia docelowa zmienna (Delay)
* Dzieli dane i trenuje model
* Zapisuje raport i model

**Wyniki:**

**Model**: XGBoostClassifier

**Dokładność: 58.26%**

**Macro Average**:

* Precision: 0.58
* Recall: 0.58
* F1-score: 0.57

**Weighted Average**:

* Precision: 0.58
* Recall: 0.58
* F1-score: 0.58

# Ćwiczenia 5

Stworzono daga do monitorowania, nie udało się ustawić wysyłania mail’a.  
  
  
Ćwiczenia 6

Stworzono api przyjmujące dane w JSON np.  
 **{**

**"Airline": 1,**

**"AirportFrom": 1,**

**"AirportTo": 1,**

**"DayOfWeek": 1,**

**"Time”:1,**

**"Lenght": 100**

**}**

I zwracajacę

**{**

**„prediction”:0 lub 1**

**}**

Dodano dockerfile z portem serwisu 5000.Dodano obraz do dockerhub’a.

Uruchomienie docker:

docker pull bartoszkunc/flight\_pred\_app:latest

docker run -p 5000:5000 bartoszkunc/flight\_pred\_app:latest  
w przeglądrace: http://localhost:5000/docs

Postman:  
New -> HTTP

Zmiana pakietu na POST  
adres: <http://127.0.0.1:5000/predict>  
raw JSON ustawiony w zakładce body np.:  
{

    "Airline": 2,

    "AirportFrom": 8,

    "AirportTo": 197,

    "DayOfWeek":2,

    "Time": 30,

    "Lenght": 202

}

Output:  
{

    "prediction": 0

}