# Cel projektu

Celem projektu jest stworzenie aplikacji do prognozowania sprzedaży produktów w ujęciu kwartalnym, na podstawie danych historycznych z ostatnich 8 lat. Uzyskane prognozy mają wspierać podejmowanie decyzji biznesowych dotyczących planowania produkcji.

# Aplikacja

Aplikacja umożliwia generowanie prognozy sprzedaży dla wybranego produktu, wykorzystując dane historyczne. W tym celu stosowany jest model XGBoost trenowany na pełnym zbiorze danych sprzedażowych. Do danych wejściowych dodawane są dodatkowe cechy, takie jak:

* średnia sprzedaż z 3 ostatnich miesięcy,
* średnia sprzedaż z 6 miesięcy,
* trend czasowy (np. różnica względem poprzedniego miesiąca),
* odchylenie standardowe z ostatnich 3 miesięcy.

Dane wejściowe z pliku „sprzedaż.csv” mają format jak poniżej:

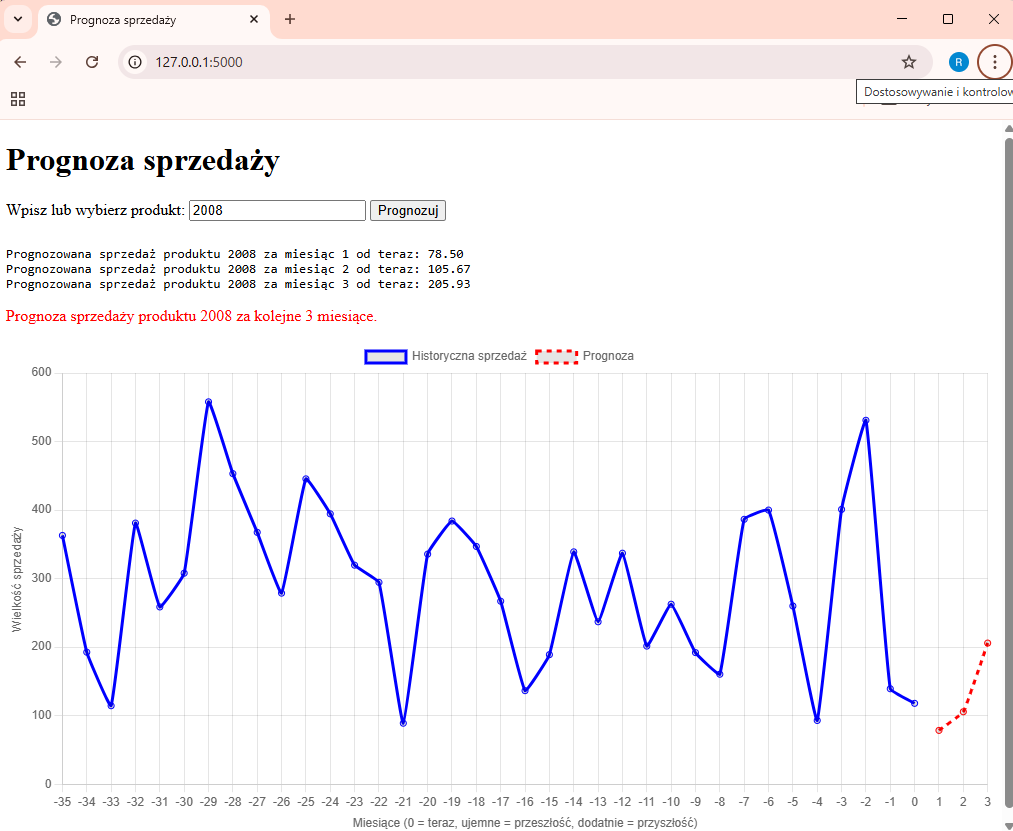
0001;100SZT;1196;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;25.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000

0001;100SZT;2511;0.0000;20.0000;0.0000;5.0000;0.0000;10.0000;0.0000;10.0000;10.0000;0.0000;0.0000;10.0000;20.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;20.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;0.0000;20.0000;50.0000;0.0000;0.0000;0.0000;0.0000;25.0000;0.0000;10.0000;10.0000;10.0000;0.0000;0.0000;0.0000;0.0000;15.0000;20.0000;0.0000;5.0000;0.0000;35.1500;0.0000;0.0000;10.0000;0.0100;24.3300;5.6600;0.0000;20.0000;0.0000;25.0000;10.0000;0.0000;10.0000;10.0000;20.0000;30.0000;10.0000;0.0000;10.0000;16.6000;43.4000;0.0000;10.0000;10.0000;6.0000;6.0000;0.0000;0.0000;10.0000;20.0000;10.0000;0.0000;10.0000;13.0000;10.0000;0.0000;22.0000;0.0000;0.0000;22.0000;0.0000;8.0000;0.0000;0.0000

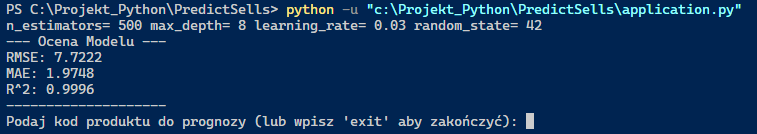
W aplikacji użytkownik może wybrać produkt, a aplikacja wygeneruje prognozę sprzedaży na następny kwartał.”

Projekt składa się z 4 plików:

* dataloader.py - klasa odpowiedzialna za wczytanie i przygotowanie danych do uczenia
* predictor.py – klasa odpowiedzialna za przygotowanie modelu oraz prognozowania
* app.py – aplikacja Flask z interfejsem do wybierania produktu a następnie wizualizacji wyników w przeglądarce



* application.py – aplikacja do oceny modelu oraz prognozowania w terminalu



# Kod źródłowy

dataloader.py

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

class DataLoader:

    def \_\_init\_\_(self):

        self.sellsData = pd.DataFrame()

        self.productsList = []

        self.le = LabelEncoder()

    def load\_sells\_data\_from\_file(self, path):

        self.sellsData = pd.read\_csv(path, sep=';', header=None, low\_memory=False)

        self.sellsData.columns = ['KodProduktu', 'JM', 'Klient'] + [str(i) for i in range(0, 98)]

    def prepare\_sells\_data(self):

        cols\_to\_drop = [self.sellsData.columns[1], self.sellsData.columns[3]]

        self.sellsData = self.sellsData.drop(cols\_to\_drop, axis=1)

        month\_cols = [str(i) for i in range(1, 97)]

        present\_months = [col for col in month\_cols if col in self.sellsData.columns]

        reversed\_months = list(reversed(present\_months))

        other\_cols = [col for col in self.sellsData.columns if col not in present\_months]

        self.sellsData = self.sellsData[other\_cols[:2] + reversed\_months]

        allowed\_prefixes = ('0', '1', '2', '3', '4', '5', '7', '9', 'UW', 'US')

        self.sellsData = self.sellsData[

            self.sellsData['KodProduktu'].astype(str).str.startswith(allowed\_prefixes)

        ]

        return self.sellsData

    def prepare\_products\_list(self):

        self.productsList = self.sellsData['KodProduktu'].unique().tolist()

        return self.productsList

    def prepare\_training\_data(self):

        month\_columns = [str(i) for i in range(96, 0, -1)]

        df\_agg = self.sellsData.groupby("KodProduktu")[month\_columns].sum().reset\_index()

        df\_long = df\_agg.melt(id\_vars="KodProduktu", var\_name="miesiac\_ago", value\_name="sprzedaz")

        df\_long["miesiac\_ago"] = df\_long["miesiac\_ago"].astype(int)

        df\_long["czas"] = 97 - df\_long["miesiac\_ago"]

        lags = list(range(1, 37))

        for lag in lags:

            df\_long[f"lag\_{lag}"] = df\_long.groupby("KodProduktu")["sprzedaz"].shift(lag)

        df\_long["rolling\_mean\_3"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(lambda x: x.rolling(3).mean())

        df\_long["rolling\_mean\_6"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(lambda x: x.rolling(6).mean())

        df\_long["trend\_24"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(lambda x: x.diff().rolling(24).mean())

        df\_long["trend\_6"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(lambda x: x.diff().rolling(6).mean())

        df\_long["std\_3"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(lambda x: x.rolling(3).std())

        df\_long["std\_6"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(lambda x: x.rolling(6).std())

        for i in range(1, 4):

            df\_long[f"sum\_year\_{i}"] = df\_long.groupby("KodProduktu")["sprzedaz"].transform(

                lambda x: x.shift(i \* 12).rolling(12).sum()

            )

        df\_lags = df\_long.dropna().copy()

        def has\_recent\_sales(group):

            return group.sort\_values("czas", ascending=False).head(6)["sprzedaz"].sum() > 0

        df\_lags = df\_lags.groupby("KodProduktu").filter(has\_recent\_sales)

        df\_lags["KodProduktu\_enc"] = self.le.fit\_transform(df\_lags["KodProduktu"])

        additional\_features = [

            "rolling\_mean\_3", "rolling\_mean\_6",

            "trend\_24", "trend\_6",

            "std\_3", "std\_6",

            "sum\_year\_1", "sum\_year\_2", "sum\_year\_3"

        ]

        X = df\_lags[[f"lag\_{l}" for l in lags] + additional\_features + ["KodProduktu\_enc"]]

        y = df\_lags["sprzedaz"]

        return df\_lags, X, y

predictor.py

import pandas as pd

import xgboost as xgb

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

import numpy as np

class Predictor:

    def \_\_init\_\_(self, n\_estimators=100, max\_depth=5, learning\_rate=0.1, random\_state=42):

        self.model = xgb.XGBRegressor(

            n\_estimators=n\_estimators,

            max\_depth=max\_depth,

            learning\_rate=learning\_rate,

            random\_state=random\_state

        )

        print("n\_estimators=", n\_estimators, "max\_depth=", max\_depth, "learning\_rate=", learning\_rate, "random\_state=", random\_state)

    def model\_train(self, X, y):

        self.model.fit(X, y)

    def evaluate\_model(self, X\_test, y\_test):

        predictions = self.model.predict(X\_test)

        rmse = np.sqrt(mean\_squared\_error(y\_test, predictions))

        mae = mean\_absolute\_error(y\_test, predictions)

        r2 = r2\_score(y\_test, predictions)

        print("--- Ocena Modelu ---")

        print(f"RMSE: {rmse:.4f}")

        print(f"MAE: {mae:.4f}")

        print(f"R^2: {r2:.4f}")

        print("--------------------")

        return rmse, mae, r2

    def forecasting(self, product, df\_lags, le):

        if product not in le.classes\_:

            print(f"Produkt {product} nie był użyty podczas treningu. Brak prognozy.")

            df\_product = df\_lags[df\_lags["KodProduktu"] == product].sort\_values("czas", ascending=False)

            last\_values = df\_product.head(36)["sprzedaz"].tolist()[::-1]

            if len(last\_values) < 1:

                print("Brak danych historycznych do wyświetlenia.")

                return

            if len(last\_values) < 36:

                last\_values = [0] \* (36 - len(last\_values)) + last\_values

            czas\_axis\_hist = list(range(-35, 1))

            plt.figure(figsize=(12, 6))

            plt.plot(czas\_axis\_hist, last\_values, label="Historyczna sprzedaż", marker='o')

            plt.axvline(x=0, color='gray', linestyle='--', label='Teraz')

            plt.title(f"Tylko dane historyczne – produkt {product}")

            plt.xlabel("Miesiące (0 = teraz, ujemne = przeszłość)")

            plt.ylabel("Wielkość sprzedaży")

            plt.legend()

            plt.grid(True)

            plt.tight\_layout()

            plt.show()

            return

        product\_enc = le.transform([product])[0]

        df\_product = df\_lags[df\_lags["KodProduktu"] == product].sort\_values("czas", ascending=False)

        last\_values = df\_product.head(36)["sprzedaz"].tolist()[::-1]

        preds = []

        for i in range(3):

            lag\_features = last\_values[-36:]

            rolling\_mean\_3 = pd.Series(last\_values[-3:]).mean()

            rolling\_mean\_6 = pd.Series(last\_values[-6:]).mean()

            trend\_24 = pd.Series(last\_values).diff().rolling(24).mean().iloc[-1] if len(last\_values) >= 25 else 0

            trend\_6 = pd.Series(last\_values).diff().rolling(6).mean().iloc[-1] if len(last\_values) >= 7 else 0

            std\_3 = pd.Series(last\_values[-3:]).std()

            std\_6 = pd.Series(last\_values[-6:]).std()

            sum\_year\_1 = sum(last\_values[-12:])

            sum\_year\_2 = sum(last\_values[-24:-12])

            sum\_year\_3 = sum(last\_values[-36:-24])

            x\_input = lag\_features + [

                rolling\_mean\_3, rolling\_mean\_6,

                trend\_24, trend\_6,

                std\_3, std\_6,

                sum\_year\_1, sum\_year\_2, sum\_year\_3,

                product\_enc

            ]

            y\_pred = self.model.predict([x\_input])[0]

            y\_pred = max(0, y\_pred)

            preds.append(y\_pred)

            last\_values.append(y\_pred)

            print(f"Prognozowana sprzedaż produktu {product} za miesiąc {i+1} od teraz: {y\_pred:.2f}")

        history\_length = 36

        future\_steps = len(preds)

        historic\_sell = last\_values[-(history\_length + future\_steps):-future\_steps]

        forecast\_sell = preds

        time\_axis\_hist = list(range(-history\_length + 1, 1))

        time\_axis\_pred = list(range(1, future\_steps + 1))

        plt.figure(figsize=(12, 6))

        plt.plot(time\_axis\_hist, historic\_sell, label="Historyczna sprzedaż", marker='o')

        plt.plot(time\_axis\_pred, forecast\_sell, label="Prognoza", marker='x', linestyle='--', color='red')

        plt.axvline(x=0, color='gray', linestyle='--', label='Teraz')

        plt.title(f"Sprzedaż i prognoza produktu {product}")

        plt.xlabel("Miesiące (0 = teraz, ujemne = przeszłość, dodatnie = przyszłość)")

        plt.ylabel("Wielkość sprzedaży")

        plt.legend()

        plt.grid(True)

        plt.tight\_layout()

        plt.show()

    def forecasting\_www(self, product, df\_lags, le):

        if product not in le.classes\_:

            text = f"Brak danych lub prognozy dla produktu {product}."

            return None, None, text

        product\_enc = le.transform([product])[0]

        df\_product = df\_lags[df\_lags["KodProduktu"] == product].sort\_values("czas", ascending=False)

        last\_values = df\_product.head(36)["sprzedaz"].tolist()[::-1]

        preds = []

        if sum(last\_values[-6:]) == 0:

            text = f"Brak sprzedaży przez ostatnie 6 miesięcy dla produktu {product}. Prognoza = 0."

            # Przygotuj dane do wykresu: historia + 3 prognozy zero

            history\_length = 36

            future\_steps = 3

            historic\_sell = last\_values[-history\_length:]

            forecast\_sell = [0]\*future\_steps

            return historic\_sell, forecast\_sell, text

        for i in range(3):

            lag\_features = last\_values[-36:]

            rolling\_mean\_3 = pd.Series(last\_values[-3:]).mean()

            rolling\_mean\_6 = pd.Series(last\_values[-6:]).mean()

            trend\_24 = pd.Series(last\_values).diff().rolling(24).mean().iloc[-1] if len(last\_values) >= 25 else 0

            trend\_6 = pd.Series(last\_values).diff().rolling(6).mean().iloc[-1] if len(last\_values) >= 7 else 0

            std\_3 = pd.Series(last\_values[-3:]).std()

            std\_6 = pd.Series(last\_values[-6:]).std()

            sum\_year\_1 = sum(last\_values[-12:])

            sum\_year\_2 = sum(last\_values[-24:-12])

            sum\_year\_3 = sum(last\_values[-36:-24])

            x\_input = lag\_features + [

                rolling\_mean\_3, rolling\_mean\_6,

                trend\_24, trend\_6,

                std\_3, std\_6,

                sum\_year\_1, sum\_year\_2, sum\_year\_3,

                product\_enc

            ]

            y\_pred = self.model.predict([x\_input])[0]

            y\_pred = max(0, y\_pred)

            preds.append(y\_pred)

            last\_values.append(y\_pred)

        text = f"Prognoza sprzedaży produktu {product} za kolejne 3 miesiące."

        history\_length = 36

        future\_steps = len(preds)

        historic\_sell = last\_values[-(history\_length + future\_steps):-future\_steps]

        forecast\_sell = preds

        return historic\_sell, forecast\_sell, text

app.py

from flask import Flask, render\_template, request

from dataloader import DataLoader

from predictor import Predictor

app = Flask(\_\_name\_\_)

loader = DataLoader()

loader.load\_sells\_data\_from\_file("sprzedaz.csv")

loader.prepare\_sells\_data()

loader.prepare\_products\_list()

df\_lags, X, y = loader.prepare\_training\_data()

predictor = Predictor(n\_estimators=300, max\_depth=8, learning\_rate=0.3, random\_state=42)

predictor.model\_train(X, y)

@app.route("/", methods=["GET", "POST"])

def index():

    products = loader.productsList

    product = None

    message = ""

    history = []

    forecast = []

    forecast\_text = ""

    if request.method == "POST":

        product = request.form.get("product\_code", "").strip()

        if product not in products:

            message = "❌ Nieprawidłowy kod produktu lub brak danych."

        else:

            history, forecast, message = predictor.forecasting\_www(product, df\_lags, loader.le)

            history = [float(x) for x in history] if history else []

            forecast = [float(x) for x in forecast] if forecast else []

            forecast\_lines = [

                f"Prognozowana sprzedaż produktu {product} za miesiąc {i+1} od teraz: {pred:.2f}"

                for i, pred in enumerate(forecast)

            ]

            forecast\_text = "\n".join(forecast\_lines)

    return render\_template(

        "index.html",

        products=products,

        product=product,

        message=message,

        history=history,

        forecast=forecast,

        forecast\_text=forecast\_text

    )

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

application.py

from dataloader import DataLoader

from predictor import Predictor

from sklearn.model\_selection import train\_test\_split

loader = DataLoader()

loader.load\_sells\_data\_from\_file("sprzedaz.csv")

loader.prepare\_sells\_data()

loader.prepare\_products\_list()

df\_lags, X, y = loader.prepare\_training\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, shuffle=False)

predictor = Predictor(n\_estimators=500, max\_depth=8, learning\_rate=0.03, random\_state=42)

predictor.model\_train(X, y)

predictor.evaluate\_model(X\_test, y\_test)

while True:

    product\_code = input(

        "Podaj kod produktu do prognozy (lub wpisz 'exit' aby zakończyć): "

    ).strip()

    if product\_code.lower() == "exit":

        print("Zakończono prognozowanie.")

        break

    if product\_code not in loader.productsList:

        print("❌ Nieprawidłowy kod produktu lub brak danych. Spróbuj ponownie.")

        continue

    try:

        predictor.forecasting(product\_code, df\_lags, loader.le)

    except Exception as e:

        print(f"Błąd podczas prognozy: {e}")

index.html

<!DOCTYPE html>

<html lang="pl">

<head>

  <meta charset="UTF-8" />

  <title>Prognoza sprzedaży</title>

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body>

  <h1>Prognoza sprzedaży</h1>

  <form method="POST">

    <label for="product\_code">Wpisz lub wybierz produkt:</label>

    <input list="produkty" name="product\_code" value="{{ product or '' }}">

    <datalist id="produkty">

      {% for code in products %}

      <option value="{{ code }}">

      {% endfor %}

    </datalist>

    <button type="submit">Prognozuj</button>

  </form>

    <div id="forecast-text" style="margin-top:10px; white-space: pre-line; font-family: monospace;">

        {{ forecast\_text or '' }}

    </div>

  <p style="color: red;">{{ message }}</p>

  {% if history and forecast %}

  <canvas id="salesChart" width="800" height="400"></canvas>

  <script>

    const ctx = document.getElementById('salesChart').getContext('2d');

    const historyData = {{ history | tojson }};

    const forecastData = {{ forecast | tojson }};

    const labels = [];

    const histLen = historyData.length;

    const futLen = forecastData.length;

    for(let i = -histLen + 1; i <= 0; i++) {

      labels.push(i.toString());

    }

    for(let i = 1; i <= futLen; i++) {

      labels.push(i.toString());

    }

    new Chart(ctx, {

      type: 'line',

      data: {

        labels: labels,

        datasets: [

          {

            label: 'Historyczna sprzedaż',

            data: historyData,

            borderColor: 'blue',

            fill: false,

            tension: 0.1,

            pointRadius: 3,

          },

          {

            label: 'Prognoza',

            data: Array(histLen).fill(null).concat(forecastData),

            borderColor: 'red',

            borderDash: [5, 5],

            fill: false,

            tension: 0.1,

            pointRadius: 3,

          }

        ]

      },

      options: {

        scales: {

          x: {

            title: {

              display: true,

              text: "Miesiące (0 = teraz, ujemne = przeszłość, dodatnie = przyszłość)"

            }

          },

          y: {

            beginAtZero: true,

            title: {

              display: true,

              text: "Wielkość sprzedaży"

            }

          }

        }

      }

    });

  </script>

  {% endif %}

</body>

</html>

# Opis działania aplikacji

Z zewnętrznego systemu ERP otrzymujemy plik sprzedaż.csv w którym są dane na historyczne sprzedaży poszczególnych detali do klientów z ostatnich 97 miesięcy. Plik nie zawiera nagłówków. Oraz kolumny z danymi sprzedażowymi są ustawione od obecnego miesiąca do 96.

## Aplikacja konsolowa „application.py”.

Po uruchomieniu następuje wczytanie danych z pliku. Następnie za pomocą funkcji prepare\_sells\_data() następuje obórka danych usunięcie zbędnych kolumn, pogrupowanie sprzedaży wg kodu produktu, utworzenie lag 36 z poprzedzających 36 miesięcy, usunięcie produktów które nie miały sprzedaży przez ostatnie 6 miesięcy oraz dodanie dodatkowych cech średnie 3, 6 miesięczne, trendy 24 i 6 miesięczne, odchylenia standardowe 3 i 6 miesięczne oraz roczne sumy sprzedaży z 3 ostatnich lat. Zostaje utworzona lista produktów w celu późniejszego sprawdzania czy podany kod produktu był prognozowany.

loader = DataLoader()

loader.load\_sells\_data\_from\_file("sprzedaz.csv")

loader.prepare\_sells\_data()

loader.prepare\_products\_list()

Po tym etapie następuje przygotowanie danych treningowych i testowych do oceny efektywności modelu:

df\_lags, X, y = loader.prepare\_training\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, shuffle=False)

Trenujemy model.

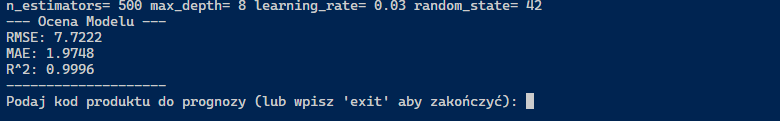
predictor = Predictor(n\_estimators=500, max\_depth=8, learning\_rate=0.03, random\_state=42)

predictor.model\_train(X, y)

Na koniec oceniamy:

predictor.evaluate\_model(X\_test, y\_test)

Wynikiem jest poniższa informacja:



Model uzyskał współczynnik determinacji R2 = 0,9996, co świadczy o jego bardzo dobrym dopasowaniu do danych testowych. Niskie wartości RMSE i MAE potwierdzają wysoką trafność prognoz.

Następnie przechodzimy do pętli while True. Gdzie możemy sprawdzać prognozy dla poszczególnych detali.

while True:

    product\_code = input(

        "Podaj kod produktu do prognozy (lub wpisz 'exit' aby zakończyć): "

    ).strip()

    if product\_code.lower() == "exit":

        print("Zakończono prognozowanie.")

        break

    if product\_code not in loader.productsList:

        print("❌ Nieprawidłowy kod produktu lub brak danych. Spróbuj ponownie.")

        continue

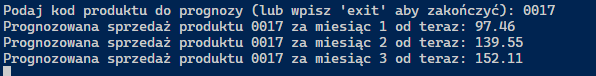
    try:

        predictor.forecasting(product\_code, df\_lags, loader.le)

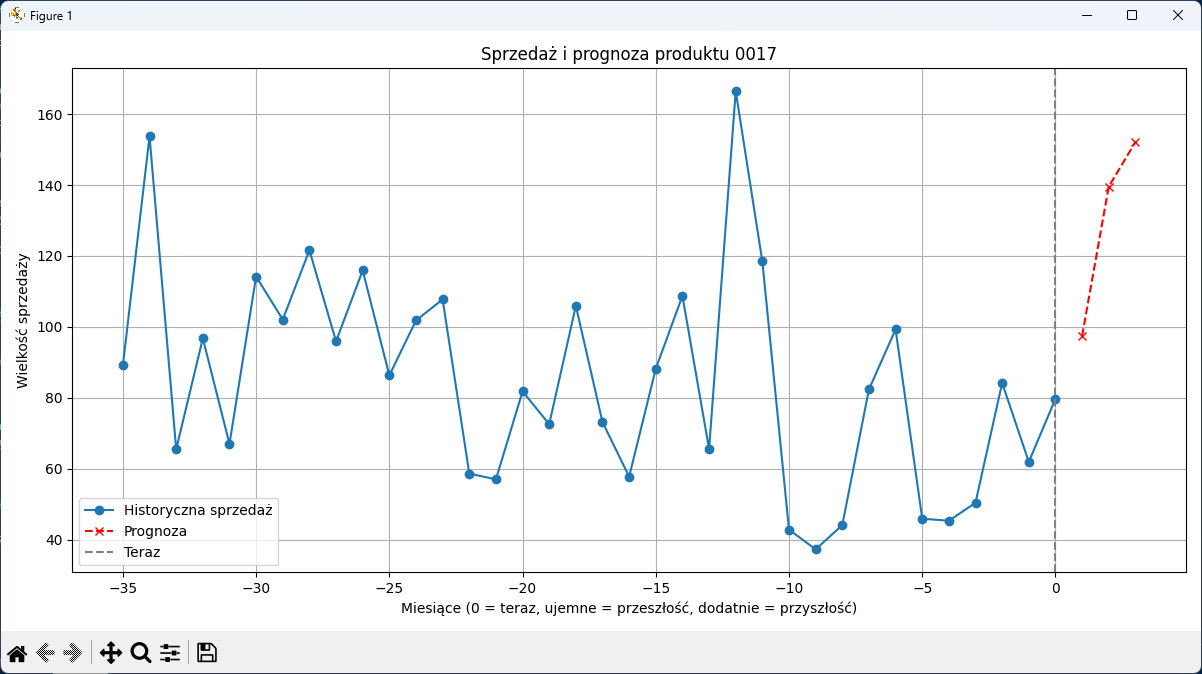
    except Exception as e:

        print(f"Błąd podczas prognozy: {e}")

Po podaniu kodu produktu np. 0017 otrzymujemy wynik prognozy:



Oraz wykres:



W przypadku podania nieprawidłowego kodu na przykład 0017.0154 otrzymujemy informację:



Po wpisaniu „exit” kończymy działanie programu:



## Aplikacja Flask app.py.

Podobnie jak w przypadku application.py następuje wczytanie i przygotowanie danych. Jednak w tej aplikacji pomijamy etap oceny modelu.

from flask import Flask, render\_template, request

from dataloader import DataLoader

from predictor import Predictor

app = Flask(\_\_name\_\_)

loader = DataLoader()

loader.load\_sells\_data\_from\_file("sprzedaz.csv")

loader.prepare\_sells\_data()

loader.prepare\_products\_list()

df\_lags, X, y = loader.prepare\_training\_data()

Następnie następuje trenowanie modelu:

predictor = Predictor(n\_estimators=300, max\_depth=8, learning\_rate=0.3, random\_state=42)

predictor.model\_train(X, y)

Poniżej endpoint do prognozowania detali:

@app.route("/", methods=["GET", "POST"])

def index():

    products = loader.productsList

    product = None

    message = ""

    history = []

    forecast = []

    forecast\_text = ""

    if request.method == "POST":

        product = request.form.get("product\_code", "").strip()

        if product not in products:

            message = "❌ Nieprawidłowy kod produktu lub brak danych."

        else:

            history, forecast, message = predictor.forecasting\_www(product, df\_lags, loader.le)

            history = [float(x) for x in history] if history else []

            forecast = [float(x) for x in forecast] if forecast else []

            forecast\_lines = [

                f"Prognozowana sprzedaż produktu {product} za miesiąc {i+1} od teraz: {pred:.2f}"

                for i, pred in enumerate(forecast)

            ]

            forecast\_text = "\n".join(forecast\_lines)

    return render\_template(

        "index.html",

        products=products,

        product=product,

        message=message,

        history=history,

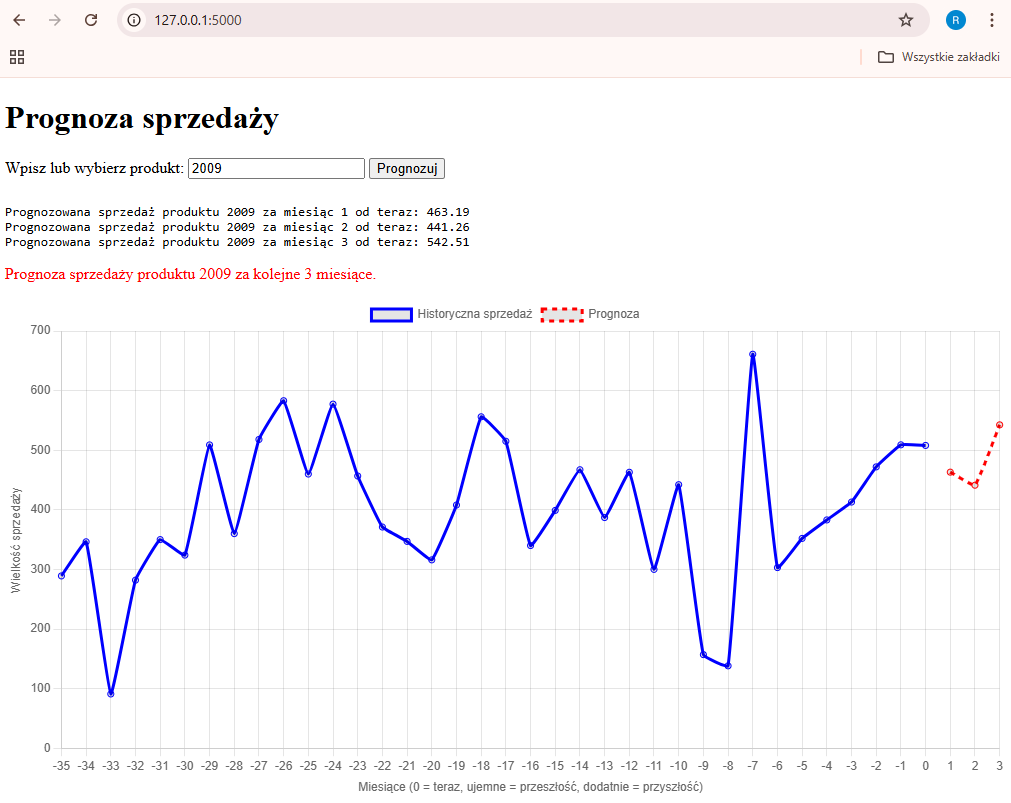
        forecast=forecast,

        forecast\_text=forecast\_text

    )

Widok z przeglądarki:

Po wpisaniu lub wybraniu z listy kodu produktu 2009 otrzymujemy prognozę.



W przypadku gdy wpiszemy błędny kod produktu, lub gdy nie był brany do trenowania otrzymujemy komunikat:

