8. Analiza dużych zbiorów danych

Leszek Klich

8.1 Wstęp

Znaczenie Big Data staje się coraz bardziej popularne i kojarzy się głównie z pracą analityków, którzy mając dostęp do olbrzymich zbiorów danych poszukują rozwiązań dla biznesu. Istnieje kilka definicji terminu Big Data, jednak część z nich nie wydaje się precyzyjnie oddawać rzeczywistość. Otóż terminem Big Data określa się duże zbiory danych. Słowo „duże” jest jednak nieprecyzyjne, zatem uznajmy, że są to zbiory o takim rozmiarze, że trudno jest poddać je analizie przy pomocy tradycyjnych metod. To także odniesienie do wszelkich czynności szukania, pobierania, gromadzenia i przetwarzania danych. Rezultatem analizy może być wiedza, która może być wygenerowanym profilem konsumenta lub firmy, a w konsekwencji przekłada się na zwiększenie konkurencyjności oraz większych dochodów. Analiza danych, wsparta algorytmami sztucznej inteligencji, może być także przydatna do wszelkiego typu predykcji niezbędnej w przemyśle.

8.2 Źródła danych i podstawowe problemy

Współczesne systemy informacyjne i produkcyjne gromadzą ogromne ilości danych, które stale się powiększają. Są one dostępne w postaci baz danych, plików tekstowych i binarnych, zaś ich źródłem mogą

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli” współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

być witryny internetowe, zawierające dane w postaci np. tabel. Jednakże część zebranych danych może okazać się mniej wartościowa, ponieważ mamy w tym przypadku do czynienia z terminem „szumu informacyjnego". Termin ten oznacza dane nieistotne, niezweryfikowane, które mogą zaburzyć wyniki analizy. Innym problemem jest uszkodzenie danych, np. na skutek błędów transmisji. Z tego powodu, jednym z pierwszych zadań dla analityka jest prawidłowe wyczyszczenie zbioru i posegregowanie danych, a następnie dokonanie ich analizy. Należy także zwrócić uwagę na filtrowanie danych, czyli redukcję, która przyspiesza proces analizy, a czasem w ogóle ją umożliwia. Jest to szczególnie istotne przy pracy z bardzo dużymi zbiorami danych. Czyszczenie, segregacja, naprawa błędów i filtrowanie danych może odbywać się przy pomocy półautomatycznych narzędzi, zaś czasochłonność tego etapu jest o wiele większa niż sam etap analizy. W tym rozdziale nie będziemy zajmować się tematyką procesu przygotowania danych. Zamiast tego, dane będą generowane lub dostarczone jako prawidłowe pliki zawierające zestawy danych.

8.3 Język Python i biblioteki do analizy danych

Standardowa instalacja języka Python nie zawiera wystarczająco wydajnych obliczeniowo złożonych typów danych. Mamy do dyspozycji listy czy słowniki, jednakże ze względu na ich uniwersalność charakteryzują się one powolnym działaniem w przypadku przechowywania dużej ilości wpisów. Z tego powodu nie są one przeznaczone do szybkich obliczeń numerycznych. Na szczęście istnieje szybka, wydajna i darmowa biblioteka, która niweluje tę przypadłość języka. Biblioteka NumPy, bo o niej będzie teraz mowa, została stworzona, aby umożliwić szybkie i sprawne operacje na macierzach. Czym różni się od standardowej listy w języku Python? Przypomina ona tablice znane z innych języków programowania, czyli przechowuje elementy tego samego typu (zazwyczaj typy liczbowe). Poza tym listy są jednowymiarowe, zaś tablice NujmPy - wielowymiarowe. Oprócz tego, listy umożliwiają przechowywanie różnych typów zmiennych, co czyni je znacznie wolniejszymi od tablic obecnych w NumPy.

Drugą z bibliotek, którą warto się zainteresować jest Pandas. Została ona stworzona na bazie poprzedniej. Pandas to jeden z najbardziej rozbudowanych pakietów przeznaczonych do analizy danych dla języka Python. Pozwala między innymi na odczyt danych z wielu źródeł, czyszczenie, filtrowanie i grupowanie danych oraz oczywiście na ich analizę. Biblioteka jest niezwykle popularna wśród analityków oraz

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

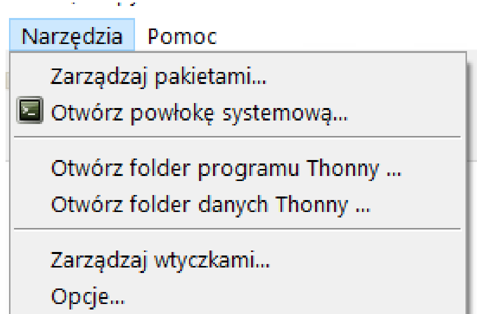
naukowców ze względu na jej dynamiczny rozwój oraz możliwości. Zasada działania obiektu Pandas, który nosi nazwę DataFrame, jest analogiczna do tabeli w programie Excel, jednakże biblioteka działa w trybie tekstowym i do jej obsługi wymagana jest podstawowa znajomość języka Python. Jest jednak bardzo uniwersalna, szybka i przy tym darmowa. Oto kluczowe funkcjonalności biblioteki Pandas:

- bardzo szybki obiekt DataFrame, który automatycznie indeksuje zbiór;
- zawiera narzędzia do ładowania danych z różnych formatów plików oraz baz danych;
- narzędzia do wypełniania brakujących danych;
- praca z zestawami dat w różnych formatach;
- przycinanie, indeksowanie i tworzenie podzbiorów dużych zbiorów danych;
- możliwość usuwania i wstawiania danych w strukturze;
- grupowanie i przekształcenia danych;
- obsługa szeregów czasowych.

8.4 Instalacja bibliotek

Zakładam, że język Python został już zainstalowany zgodnie z instrukcją zawartą w załączniku zatytułowanym „Instrukcja instalacji oprogramowania Python".

Podczas nauki analizy danych będziemy korzystać z bibliotek: NumPy, Pandas oraz Matplotlib. Są to biblioteki przeznaczone do analizy i wizualizacji danych. Aby zainstalować biblioteki, należy uruchomić środowisko Thonny i z menu **Narzędzia** wybrać polecenie **Otwórz powłokę systemową**, co widać na rysunku 8.1.



Rysunek 8.1: Uruchamianie powłoki systemowej w środowisku Thonny

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

Następnie w oknie powłoki systemowej należy zainstalować biblioteki, wykonując poplecenia : **pip install numpy** <Enter>, następnie **pip install pandas** <Enter>, a na koniec **pip install matplotlib** <Enter>. Po każdym wywołaniu polecenia **pip** nastąpi proces pobierania i instalacji bibliotek, co może potrwać w zależności od prędkości łącza internetowego.

Jeśli instalator pakietu wykryje, że wymagana biblioteka lub zależność została już wcześniej zainstalowana, pominie jego instalację i wyświetli odpowiednią informację. Po zainstalowaniu pakietów NumPy, Pandas i Matplotlib, proces instalacji środowiska i bibliotek jest zakończony.

8.5 Podstawy biblioteki NumPy

Podstawowym typem danych NumPy jest typ ndarray (N-dimmentional array), czyli tablica wielowymiarowa. Jest to w rzeczywistości tablica zawierająca jeden typ danych, analogicznie jak w innych języków programowania. Oczywiście NumPy pozwala także na tworzenie tablic jednowymiarowych i służą do tego następujące komendy:

- numpy.array() - podajemy wszystkie wartości tablicy;
- numpy.arange() - podajemy zakres, jakim uzupełniamy tablicę oraz odstęp liczbowy pomiędzy nimi;
- numpy.linspace() - przydatny podczas tworzenia wykresów, gdzie podajemy zakres liczbowy oraz podział liczbowy.

W tym rozdziale zajmiemy się wyłącznie generowaniem tablicy jednowymiarowej, przypiszemy do niej wartości funkcji sinus i wygenerujemy prosty wykres. W pierwszym etapie wygenerujmy tablicę jednowymiarową przy pomocy polecenia arange, której parametrami są: liczba początkowa (domyślnie 0), liczba końcowa oraz krok (odstęp między wartościami). Domyślny rozmiar kroku wynosi 1, lecz ustawiamy go na wartość rzeczywistą 0.01.

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0.0, 2.0, 0.01) # zmienna t to czas
s = np.sin(2 * np.pi * t) # s = wartości funkcji sinus

print(t)
```

Listing 8.1: Generowanie tablicy jednowymiarowej

Norway grants

Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).

Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

Na koniec wyświetlamy wyniki zmiennej t, oznaczającej w tym przypadku czas. Otrzymane wyniki to zawartość tablicy z krokiem 0.01:

[0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.2 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.3 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.4 0.41 0.42 0.43 0.44 0.45 0.46 0.47 ... 9.74 9.75 9.76 9.77 9.78 9.79 9.8 9.81 9.82 9.83 9.84 9.85 9.86 9.87 9.88 9.89 9.9 9.91 9.92 9.93 9.94 9.95 9.96 9.97 9.98 9.99].

Ze względu na bardzo długi ciąg liczb tablicy (wynikający z podanego kroku 0.01), część wartości zostało pominięte.

Obiekt **ndarray x** jest tworzony z funkcji np.arange() i są to wartości dla osi x (czas). Wartości funkcji sinus na osi y są przechowywane w innym obiekcie (ndarray y). Obserwując wartości tablicy w postaci liczbowej, bardzo trudno jest je człowiekowi interpretować. Można je jednak umieścić na wykresie przy pomocą funkcji plot() - dostępnej w bibliotece Matplotlib. Aby tego dokonać, rozbudujmy powyższy przykład programu o kod wyświetlający prosty wykres:

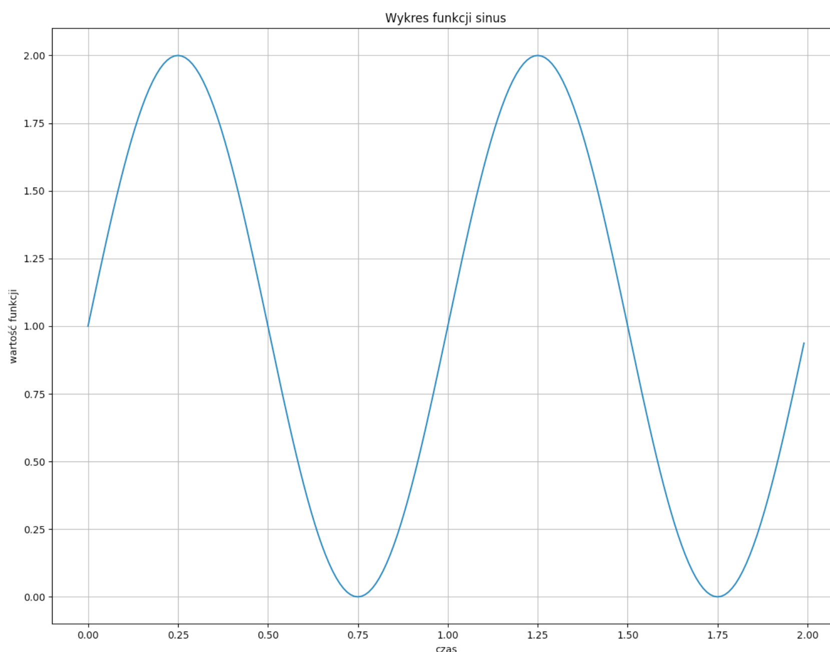
```
fig, ax = plt.subplots()
ax.plot(t, s)

# nazwy etykiet x i y oraz tytuł wykresu
ax.set(xlabel='czas', ylabel='wartości funkcji', title='
    Funkcja sinus')

# dodanie siatki do wykresu
ax.grid()

# wyświetlenie wykresu
plt.show()
```

Listing 8.2: Generowanie wykresu funkcji sinus

Efekt działania kompletnego programu ilustruje obrazek 8.2. Obserwując wykres, można łatwo zauważyć, że że mamy do czynienia z sinusoidą.

W ramach ćwiczeń można zmienić argumenty funkcji arange poprzez zmianę kroku (np. na 0.1), zakresu czy wykorzystać inne funkcje dostępne w NumPy (np. wygenerować wykres funkcji cosinus). Dzięki temu będziemy mogli sprawdzić, jak zmienia się wykres oraz automatyczne dopasowanie obszaru wykresu do danych dzięki bibliotece Matplotlib.

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.



Rysunek 8.2: Wykres funkcji sinus

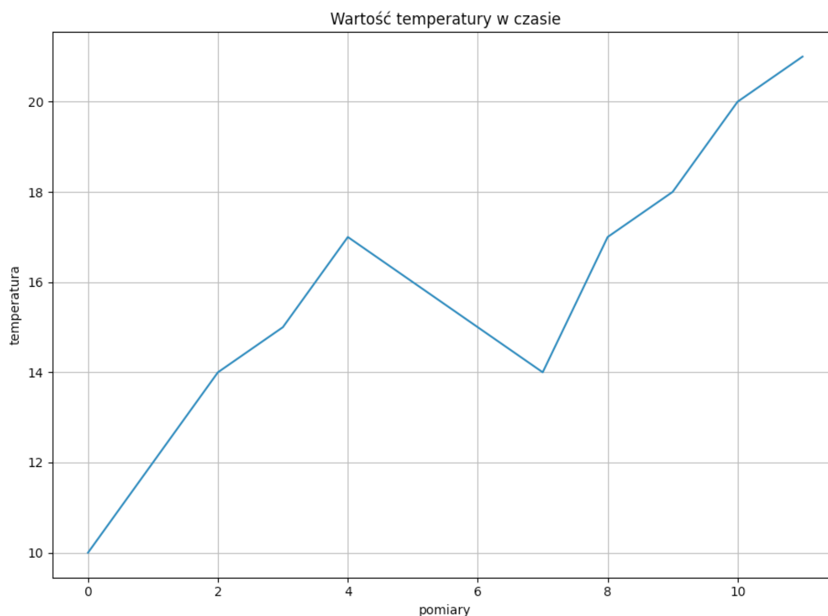
W kolejnym przykładzie zajmiemy się wizualizacją wartości temperatury w pewnym okresie czasu. W tym celu ponownie wygenerujemy zakres przy pomocy funkcji **arange()**, lecz tym razem przypiszemy do niego dane wartości temperatur przy pomocy tablicy **temperatura**.

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0, 12, 1)
temperatura = [10, 12, 14, 15, 17, 16 , 15, 14, 17, 18, 20,
    21]

fig, ax = plt.subplots()
ax.plot(t, temp)
ax.set(xlabel='pomiary', ylabel='temperatura', title='Wykres
    temperatury w czasie')
ax.grid()
plt.show()
```

Listing 8.3: Generowanie wykresu temperatur

Efektem działania programu jest wykres z rysunku 8.3. Obserwując wykres można zauważyć, że liczba pomiarów jest stosunkowo niewielka, zaś ręczne umieszczanie kolejnych pomiarów w tablicy **temperatura** jest niewygodne i wymaga zmiany kodu źródłowego programu.

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.



Rysunek 8.3: Wykres temperatury

O wiele lepszym sposobem na przechowywanie danych pomiarowych jest plikowa baza danych. Pliki zawierające dane zapisane w postaci kolumn i wierszy są bardzo popularnym sposobem przechowywania danych przez różnego typu urządzenia. Pliki te można nazwać logami, ponieważ zapisują one zdarzenia czy pomiary z różnego rodzaju czujników w postaci wierszy. Najbardziej popularnym typem pliku tekstowego jest typ CSV, który zawiera kolumny oraz pola oddzielone umownym znakiem (zwykle przecinkiem). Zestaw pól to wiersz, który może zawierać dowolną ilość pól. Pliki CSV są typu tekstowego, co oznacza, że można je otwierać i edytować przy pomocy dowolnego edytora tekstowego (np. Notatnika). Do odczytania plików tego typu wykorzystamy bibliotekę Pandas.

Wykorzystanie biblioteki Pandas

W tym podrozdziale zajmiemy się bardziej poważną analityką. Oznacza to, że będziemy pracować z danymi, dostarczonymi w postaci plików .CSV. Jest to jeden z bardziej rozbudowanych pakietów, który umożliwia analizę danych w Python. Co więcej - przy pomocy Pandas można wczytywać pliki, czyścić, modyfikować i analizować zbiory danych.

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli"
współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

Zanim przejdziemy do odczytu danych z zewnętrznych plików, warto na początek zrozumieć złożone typy danych, które dostarcza pakiet Pandas. Pierwszy z typów jest seria, zaś drugim - tzw. DataFrame. Serię można porównać do kolumny w programie Excel. Aby utworzyć serię danych, należy użyć metody **pandas.Series()**, zaś jako argumenty podać wartości numeryczne. Utwórzmy zatem serię danych.

```
import pandas as pd

seria = pd.Series([-3, -1, 5, 15, 19, 21, 23])
```

Listing 8.4: Tworzenie serii danych przy pomocy Pandas

Utworzoną serię danych można wyświetlić na ekranie przy pomocy **print(seria)**, jednakże w tym przypadku pakiet daje nam do dyspozycji kilka dodatkowych metod.

- **head()** - metoda wyświetlająca początek obiektu DataFrame (domyślnie jest to 5 wierszy);
- **tail()** - wyświetla domyślnie 5 ostatnich wierszy, lecz jako argument można podać inną liczbę;
- **sample(5)** - wyświetli 5 losowych wierszy.

Wyświetlenie zawartości danych umożliwia wstępne zapoznanie się ze zbiorem. Obie metody mogą przyjmować opcjonalny argument w postaci liczby całkowitej, która oznacza liczbę wyświetlanych wierszy. Aby wyświetlić pierwsze 10 wierszy, wystarczy wywołać: print(seria.head(10)). Wynikiem działania będzie następujący rezultat:

```
0 -3
1 -1
2 5
3 15
4 19
5 21
6 23
dtype: int64
```

Jak widać, Pandas dodał automatycznie dodatkową kolumnę index, która pozwala na identyfikację dowolnego wiersza. Dodatkowo metoda head() wyświetliła na końcu typ danych (int64). Jest to typ całkowity używany przez bibliotekę do przechowywania bardzo dużych wartości. Oczywiście gdybyśmy zadeklarowali serię, w której użylibyśmy wartości rzeczywistej, Pandas dopasowałby typ jako float64.

Aby dowiedzieć się więcej o serii, można posłużyć się metodą info(), np. **print(df.info())**. Wyświetlone zostaną informację na temat indeksów, serii, typów danych oraz zajętości pamięci.

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

```
RangeIndex: 7 entries, 0 to 6
Series name: None
Non-Null Count  Dtype
--------------  -----
7 non-null      int64
dtypes: int64(1)
memory usage: 184.0 bytes
```

Teraz dla przykładu wykonamy na obiekcie metodę describe() i wygenerujemy podstawowe statystyki, takie jak ilość danych w serii, wartość średnią, minimalną, maksymalną, odchylenie standardowe oraz kwantyle: 25%, 50% (mediana), 75%. Statystykę można wygenerować przy pomocy jednego wiersza: **print(seria.describe())**.

```
count 7.000000
mean 11.285714
std 10.796825
min -3.000000
25% 2.000000
50% 15.000000
75% 20.000000
max 23.000000
dtype: float64
```

Nic nie stoi na przeszkodzie, aby do tworzenia obiektu DataFrame wykorzystać standardową listę dostępną w języku Python. Należy jej zawartość przypisać do nowo tworzonego obiektu. Dzięki temu standardowe typy złożone, dostępne w języku Python, można poddawać analizie statystycznej.

```
lista = [['Patrycja',16],['Monika',12],['Andrzej',28],['
    Leszek',9]]

df = pd.DataFrame(lista)
df.columns = 'Imię', 'Wiek'

print(df)
```

Listing 8.5: Tworzneie DataFrame z listy Python

```
       Imię  Wiek
0  Patrycja    16
1    Monika    12
2   Andrzej    28
3    Leszek     9
```

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli"
współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

W wyniku uruchomienia programu otrzymamy obiekt typu DataFrame. Można także wygenerować obiekt typu DataFrame ze standardowo dostępnego słownika w języku Python.

```
slownik = {'Imię':['Leszek', 'Monika', 'Patrycja', 'Andrzej'
    ], 'Miasto': ['Warszawa', 'Kraków', 'Gdańsk', 'Poznań']}

df = pd.DataFrame(slownik)

print(df)
```

Listing 8.6: Tworzenie DataFrame ze słownika Python

W wyniku uruchomienia programu również otrzymamy obiekt typu DataFrame.

```
       Imię    Miasto
0     Leszek  Warszawa
1     Monika    Kraków
2   Patrycja    Gdańsk
3    Andrzej    Poznań
```

Powyższe przykłady ilustrują, w jaki sposób istniejący program napisany w języku Python może współpracować z pakietem statystycznym Pandas.

Odczyt danych z pliku

Na tym etapie zajmiemy się operacją odczytu plików typu .CSV z dysku. Warto zaznaczyć, że Pandas może odczytywać różne typy plików, z czego najpopularniejsze to wspomniany .CSV oraz .JSON. Nic jednak nie stoi na przeszkodzie, aby odczytywać także pliki Excela. W tym podrozdziale zostanie omówiony wyłącznie odczyt typu tekstowego .CSV. Do odczytu plików tekstowych typu .CSV służy funkcja read_csv(). Funkcja ta może przyjmować szereg argumentów, lecz my skupimy się tylko na podstawowych. Przyjmijmy, że posiadamy plik z danymi temperatury w danym okresie, o następującej strukturze:

```
Data,Temp,Hum
2020/01/01, -12.0, 70
2020/01/02, -7.2, 80
...
2020/02/22, 5.1, 91
2020/02/24, 5.0, 91
2020/02/26, 8.1, 91
2020/02/28, 8.8, 91
```

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

Analizując początek pliku można zauważyć, że plik posiada nagłówek, w którym zostały zapisane nazwy kolumn, zaś same kolumny oddzielone są znakiem przecinka. Dodatkowo występują wartości w postaci liczb zmiennoprzecinkowych, w których znak kropki oddziela część całkowitą od części ułamkowej liczby. Oto przykład programu odczytującego plik o nazwie temp.csv wraz z niezbędnymi opcjami umożliwiającymi prawidłowy odczyt pliku:

```
df = pd.read_csv("temp.csv", decimal=".", delimiter=",")
print(df.info())
```

Listing 8.7: Tworzenie obiektu DataFrame i wypełnienie go danymi z pliku

Program tworzy obiekt DataFrame o nazwie df i w sposób automatyczny wczytuje do niego zwartość pliku **temp.csv**. Argument **decimal** oznacza znak rozdziału części całkowitej od ułamkowej liczby w kolumnie **Temp**, zaś opcja **delimiter** - znak rozdziału kolumn - w tym wypadku jest to przecinek. Podanie tej opcji jest konieczne, ponieważ domyślnym znakiem rozdziału jest średnik, zatem argument ten zmienia konfigurację funkcji **read_csv()** w taki sposób, aby dostosować się do konkretnego pliku, który chcemy analizować. Jeśli zaś chodzi o argument **decimal**, to należy również dostosować go do konwencji zawartej w pliku tekstowym. Na koniec wykorzystano funkcję info(), która wyświetli informacje na temat struktury pliku, jak poniżej:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Data    29 non-null     object
 1   Temp    29 non-null     float64
 2   Hum     29 non-null     int64
dtypes: float64(1), int64(1), object(1)
memory usage: 824.0+ bytes
```

Jak widać na podsumowaniu, plik zawiera 3 kolumny: datę, która została zinterpretowana przez bibliotekę jako typ obiektowy, temperaturę, która została przypisana jako typ float64 oraz wilgotność - typ int64. I faktycznie wszystko się zgadza oprócz pola data, która powinna być przypisana do typu datetime. Zmodyfikuj zatem wiersz odpowiedzialny za odczyt danych następująco:

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

```
df = pd.read_csv("temp.csv", decimal=".", \
     delimiter=",", parse_dates=['Data'])
```

Następnie ponownie uruchom program.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Data    29 non-null     datetime64[ns]
 1   Temp    29 non-null     float64
 2   Hum     29 non-null     int64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 824.0 bytes
```

Tym razem pole daty zostało poprawnie przypisane jako typ datetime64. Ma to istotne znaczenie podczas filtrowania rekordów po dacie czy generowaniu wykresów, które jako oś x wykorzystują typ daty.

Wróćmy jeszcze do nagłówka pliku (Date,Temp,Hum), który informuje użytkownika o nazwach poszczególnych kolumn. Zdarzają się pliki tekstowe, które nagłówka nie posiadają lub posiadają nagłówek w ilości większej niż jeden wiersz. Na przykład:

```
Plik tekstowy wygenerowany ze stacji pogodowej.
Data,Temp,Hum
2020/01/01, -12.0, 70
2020/01/02, -7.2, 80
```

W tym wypadku należy wskazać funkcji **read_csv**, ile wierszy należy opuścić podczas odczytu. Służy do tego argument **skiprows**, który domyślnie jest ustawiony na **None**. Po analizie nagłówka pliku .CSV wiemy, że należy opuścić jedną linię. Możemy dodać ten parametr jako **skiprows=1** w funkcji **read_csv()**, czyli nasza linia odczytu pliku będzie wyglądać tak:

```
df = pd.read_csv("temp.csv", decimal=".", \
     delimiter=",", parse_dates=['Data'], \
     skiprows=1)
```

Ostatnim argumentem, który chciałbym omówić jest **usecols**, przy pomocy którego można zdefiniować kolumny, które zostaną odczytane z pliku do obiektu DataFrame. Jest to przydatna opcja, gdy posiadamy

Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Norway grants
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

pliki tekstowe z danymi, charakteryzujące się dużą ilością kolumn, z których można wybrać wyłącznie kolumny, które będą podlegać analizie. Ma to duże znaczenie w celu oszczędzania pamięci podczas analizy bardzo dużych zbiorów danych. W przypadku naszego pliku, jeśli istotnymi kolumnami są data oraz temperatura, warto wyłączyć kolumnę wilgotności w sposób następujący:

```
# Odczyt wyłącznie kolumn Data oraz Temp
kolumny = ["Data", "Temp"]
df = pd.read_csv("temp.csv", decimal=".", delimiter=",",
    parse_dates = ['Data'], usecols = kolumny)

print(df.info())
```

Listing 8.8: Odczyt zdefiniowanych kolumn z pliku .CSV

Modyfikacja programu przy użyciu **usecols** nie tylko zredukuje ilość kolumn i nasz obiekt DataFrame stanie się bardziej czytelny, lecz dodatkowo zmniejszy ilość wykorzystywanej pamięci RAM (z 824 bajtów do 592 bajty).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29 entries, 0 to 28
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Data    29 non-null     datetime64[ns]
 1   Temp    29 non-null     float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 592.0 bytes
```

Pozostało jeszcze dopracowanie nazw kolumn, które nierzadko mogą przyjmować nieintuicyjne nazwy. W przypadku naszego pliku są to Data oraz Temp, zaś chcielibyśmy nazwać przynajmniej kolumnę Temp bardziej przystępnie. W tym celu wykorzystamy metodę **rename(columns = 'nazwa_w_pliku' : 'nowa_nazwa', inplace = True)**. Użycie słowa kluczowego inplace=True oznacza, że operacja zmodyfikuje dane wewnątrz istniejącego obiektu. Pracując z danymi umieszczonymi na dysku w postaci plików CSV, kod realizujący odczyt pliku będzie zwykle identyczny lub bardzo podobny jak poniżej:

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

```
kolumny = ["Data", "Temp"]
df = pd.read$\_$csv("temp.csv", decimal=".", delimiter=",",
    parse_dates = ['Data'], usecols = kolumny)

df.rename(columns={'Temp': 'Temperatura', 'Data': 'Data
    pomiaru'}, inplace=True)

print(df.info())
```

Listing 8.9: Ostateczna wersja programu do odczytu pliku

Wykonajmy teraz kilka podstawowych operacji statystycznych na naszym obiekcie. Warto jedynie dodać, że w przypadku, gdy analiza dotyczy wyłącznie jednej kolumny, należy się do niej odwołać poprzez df['nazwa_kolumny']. Na początek kilka dostępnych funkcji:

- df['nazwa_kolumny'].min() - wartość minimalna,
- df['nazwa_kolumny'].max() - wartość maksymalna,
- df['nazwa_kolumny'].mean() - wartość średnia,
- df['nazwa_kolumny'].var() - wariancja kolumny,
- df['nazwa_kolumny'].sum() - suma wszystkich wierszy,
- df['nazwa_kolumny'].abs() - bezwzględne wartości numeryczne każdego elementu.

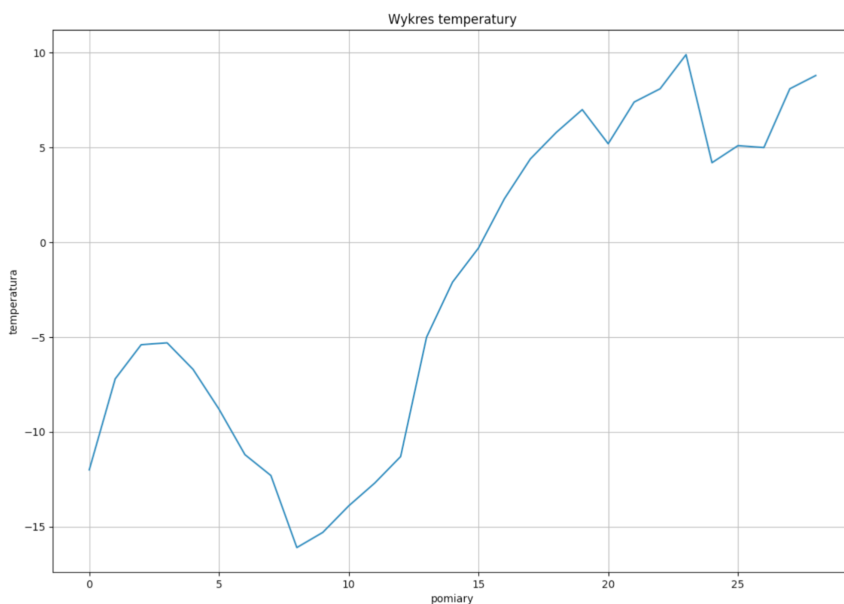
Umieśćmy zatem na wykresie pobrane z pliku wartości pomiarów temperatury, dopisując na końcu poprzedniego programu następujący kod:

```
fig, ax = plt.subplots()
ax.plot(df.index, df["Temperatura"])
ax.set(xlabel='pomiary', ylabel='temperatura', title='Wykres
    temperatury')
ax.grid()

plt.show()
```

Listing 8.10: Wykres temperatury z pliku .CSV

W efekcie otrzymamy wykres liniowy, gdzie oś x oznacza próbki pomiarów (w rzeczywistości index obiektu DataFrame), zaś na osi y zostanie umieszczony wykres wartości. Jak widać na rysunku 8.4, wykres wygenerował się poprawnie. Jednakże można zauważyć istotny problemem na osi x, na której nie widnieje data pomiaru.

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.



Rysunek 8.4: Wynik działania programu

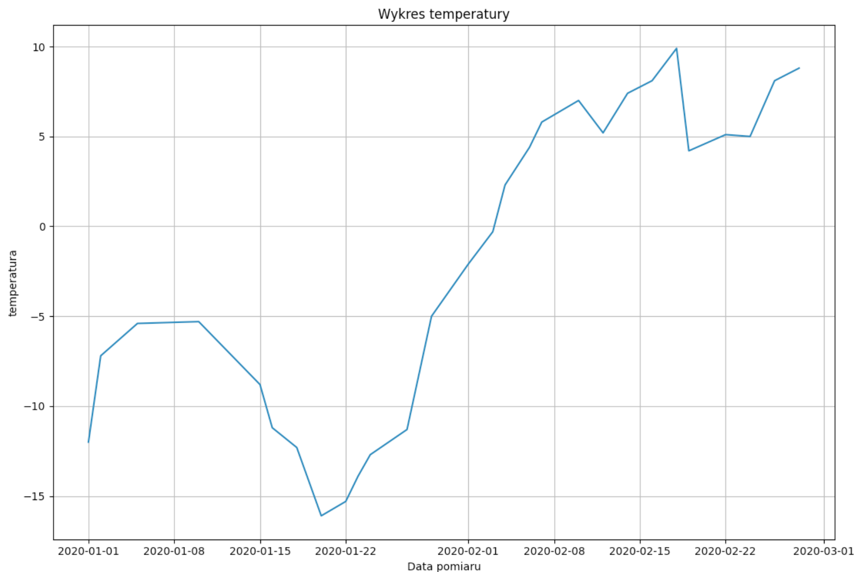
Warto zatem wykorzystać dodatkową kolumnę 'Data' w pliku, która wskazuje na konkretną datę, w której dokonano pomiaru. Wystarczy lekko zmodyfikować program, aby data była widoczna na wykresie. Dodatkowo, aby ulepszyć wykres, skorzystamy z funkcji **scatter()**, która doda na wykresie punkty pomiarowe na współrzędnych x, y.

```
fig, ax = plt.subplots()

ax.plot(df["Data pomiaru"], df["Temperatura"])
ax.set(xlabel='Data pomiaru', ylabel='temperatura', title='
    Wykres temperatury')
ax.grid()
plt.scatter(df["Data pomiaru"], df["Temperatura"])

plt.show()
```

Listing 8.11: Wykres temperatury z pliku .CSV z datą pomiaru

Od tego momentu wykres będzie zawierał oś x z naniesionymi datami poszczególnych pomiarów. Jak widać pakiet pakiet Pandas i Matplotlib pozwala na proste tworzenie wykresów.

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.



Rysunek 8.5: Wynik działania programu

Przykład - analiza porównawcza ocen uczniów z różnych szkół

Teraz zajmiemy się bardziej zaawansowaną analizą, której celem będzie odczyt danych z pliku tekstowego .CSV i wyświetlenie na wykresie porównania ocen, zdobytych przez uczniów z różnych szkół. Plik zawiera oceny uczniów z różnych przedmiotów. Analiza będzie polegać na obliczeniu wartości średniej ocen i wyświetlenie ich na wykresie jako porównanie wyników dla różnych szkół. Dzięki temu, będziemy w stanie porównać wyniki uczniów ze wszystkich szkół. Poniżej znajduje się przykładowy format pliku, w którym przechowywane są wyniki ocen.

```
szkola,nazwa,matematyka,polski,fizyka
PSP1,Jan Kowalski,5,4,3
PSP1,Krystian Nowak,4,2,4
...
PSP4,Piotr Mazur,4,4,4
PSP4,Paweł Dąbrowski,5,5,5
PSP4,Krzysztof Szymański,4,4,3
```

Jak widać plik zawiera nagłówek, który jest informacją o kolumnach, z których wynika, że oceny końcowe dotyczą przedmiotów z matematyki,

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli"
współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

języka polskiego oraz fizyki, zaś pierwsza kolumna jest oznaczeniem szkoły, do której uczęszczał uczeń.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read$\_$csv('school.csv', delimiter=',')

matma = df.groupby('szkola').agg(Średnia = ('matematyka', '
    mean'), Najwyższa = ('matematyka', 'max'), Minimum = ('
    matematyka','min'))
matma.plot(kind = 'bar', color=['r', 'g', 'b'], figsize =
    (10, 6))
plt.title('Podsumowanie z przedmiotu: matematyka', fontsize
    =18)
plt.ylabel('Ocena', fontsize=16)
plt.xlabel('Szkoły', fontsize=16)
plt.xticks(size = 14)
plt.yticks(size = 14)
plt.legend(fontsize=14, loc='upper center')
plt.show()

# Jeśli chcesz zapisać wykres do pliku, użyj metody savefig()
plt.savefig('matematyka.png')
```

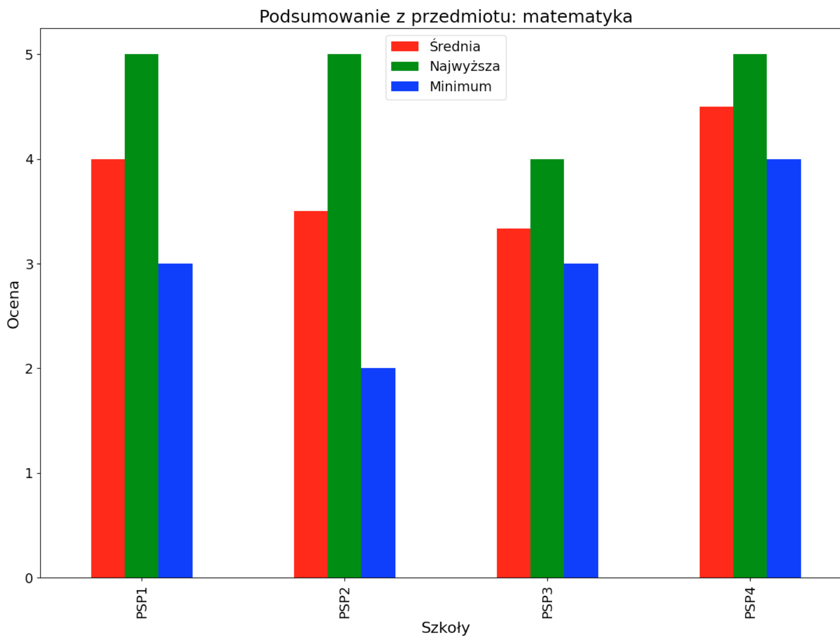
Listing 8.12: Porównanie ocen ze szkół

W programie generującym porównanie wyników ocen jest kilka nowych zapisów, które wymagają wyjaśnienia. Do tej pory używaliśmy domyślnych wartości odpowiedzialnych za rozmiar czcionki czy formatowanie wykresów. Teraz zatem je objaśnimy:

- **groupby()** - funkcja grupująca, która pozwala poznać oceny uczniów z podziałem na szkoły. Jako parametr przyjmuje nazwę kolumny, po której należy dokonać grupowania;
- **agg()** - wykonuje obliczenia dla każdej z grup. W tym przypadku wylicza dla każdej grupy średnią ocen oraz minimalną i maksymalną ocenę. Dodatkowo tego typu zapis umożliwia nadanie nazw dla każdej z kolumn, które są wykorzystane w wykresie;
- **plot()** - funkcja umieszcza na wykresie kolumny wraz z etykietami. Opcjonalnie podano także kolory poszczególnych belek wykresu przy pomocy tablicy ['r', 'g', 'b'], co oznacza: red, green, blue;
- **title()** - umieszcza tytuł wykresu. Choć jest to opcjonalne, ponieważ tytuł można podać bezpośrednio w funkcji plot(), to umieszczenie go w osobnej linii pozwala na czytelniejsze formatowanie czcionki;
- **xlabel()** - pozwala na zdefiniowanie etykiety osi x. Podobnie jak w przypadku tytułu, przeniesiono wywołanie funkcji do osobnej linii w celu zwiększenia czytelności programu i modyfikacji

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

czcionki;
- **ylabel()** - pozwala na zdefiniowanie etykiety osi y;
- **xtick()** - formatowanie wartości na osi x. W tym przypadku oznacza jedynie zmianę wielkości czcionki, jednak opcjonalnych argumentów jest o wiele więcej;
- **ytick()** - jak wyżej, lecz dotyczy formatowania wartości osi y;
- **legend()** - dotyczy legendy wyświetlanej nad wykresem. Matplotlib umieszcza legendę automatycznie, co czasami potrafi zaburzyć widok wykresu. W tej linii legenda zostałą przeniesiona do górnej części wykresu i wycentrowana;
- **savefig()** - dotychczas w przykładowych programach wyświetlaliśmy wykres przy pomocy funkcji **show()**. Wykres można także zapisać na dysk podając jako argument jego nazwę oraz rozszerzenie pliku. Funkcja zapisze plik w formacie podanym w rozszerzeniu pliku.



Rysunek 8.6: Wynik działania programu

8.6 Podsumowanie

W rozdziale zostały opisane podstawy wykorzystania bibliotek dla języka Python, które umożliwiają analizę danych. Tematyka jest nie-

Norway grants
Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).
Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.

zwykle szeroka, stąd opisano jedynie podstawowe zastosowania narzędzi. Mam nadzieję, że zaprezentowane podstawy będą zachętą do zgłębienia tematu, ponieważ analiza danych jest współcześnie podstawą działalności wielu przedsiębiorstw. Dzięki zaawansowanej analizie danych biznesowych można zwiększyć przychody firmy, zoptymalizować marketing i pozyskać więcej klientów poprzez szybszą reakcję na trendy rynkowe, a tym samym uzyskać przewagę konkurencyjną. Coraz bardziej popularne jest również łączenie analizy danych z algorytmami sztucznej inteligencji i wykorzystywanie tego typu systemów w zakładach produkcyjnych.

Norway grants

Projekt pt.: „MODELOWE ROZWIĄZANIA NA TRUDNE WYZWANIA – Plan Rozwoju Lokalnego i Instytucjonalnego Stalowej Woli" współfinansowany jest ze środków Norweskiego Mechanizmu Finansowego 2014-2021 (85%) oraz Budżetu Państwa (15%).

Wspólnie działamy na rzecz Europy zielonej, konkurencyjnej i sprzyjającej integracji społecznej.