COMARCH
Szkolenia

Python w pracy z MS Excel

Automatyzuj codzienną pracę z plikami Excel z wykorzystaniem Pythona!

Paweł Goleń

Trener

Paweł Goleń

AI enabled automation developer

COMARCH
Szkolenia

Agenda

1. Wprowadzenie do programowania w Pythonie, praca z notatnikami Jupyter
2. Wprowadzenie do Pythona w kontekście pracy z plikami Excela
3. Wprowadzenie do biblioteki pandas
4. Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Agenda

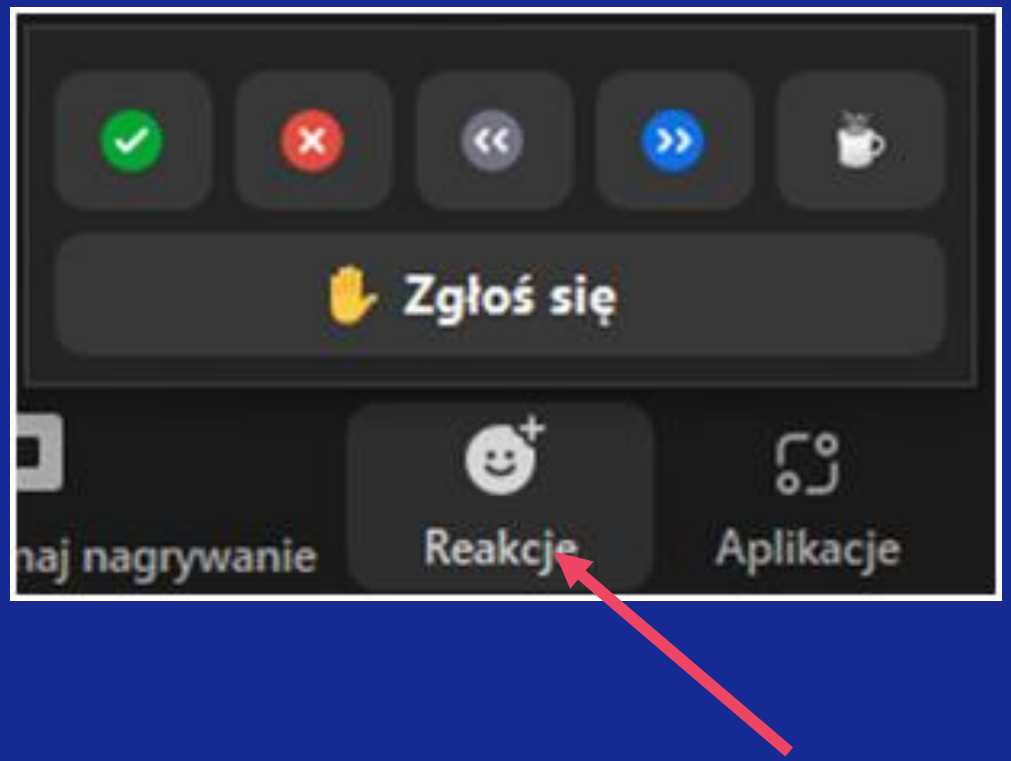
5 Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

6 Zaawansowane operacje na danych

7 Podsumowanie i zakończenie szkolenia

8 Test oraz ankiety

Szkolenia zdalne - Reakcje



COMARCH

Wprowadzenie

Omówienie celów i zakresu szkolenia:

- Celem szkolenia jest dostarczenie uczestnikom wiedzy i umiejętności niezbędnych do efektywnego wykorzystania pakietów Pythona w kontekście pracy z plikami Excela. Szkolenie ma na celu zapoznanie uczestników z różnymi bibliotekami i narzędziami dostępnymi w języku Python, które umożliwiają łatwe importowanie, eksportowanie, modyfikowanie i analizowanie danych z arkuszy Excela.

Wprowadzenie

Na czym polega analiza danych i dlaczego jest istotna:

- Analiza danych to kluczowy element podejmowania decyzji biznesowych. Chodzi o wydobywanie cennych informacji z danych, które pomagają zrozumieć trendy, prognozować wyniki i podejmować świadome decyzje. Bez analizy danych trudno jest efektywnie działać w dzisiejszym złożonym środowisku biznesowym.

Rola Pythona w analizie danych:

- Python stał się liderem w dziedzinie analizy danych, a to z kilku powodów. Jego czytelna składnia, ogromna społeczność, oraz bogactwo bibliotek, takich jak Pandas czy Numpy, sprawiają, że jest doskonałym narzędziem dla analityków danych. Podczas szkolenia zgłębimy, dlaczego Python jest tak popularny w tej dziedzinie.

Wprowadzenie

Omówienie środowiska pracy - efektywne korzystanie z Jupyter Notebook:

- Jupyter Notebook to potężne narzędzie, które ułatwia interaktywną pracę z danymi i kodem. Dzięki niemu możliwe jest eksperymentowanie, wizualizacja i dokumentacja pracy. Podczas szkolenia przekazana zostanie wiedza jak korzystać z różnych funkcji Jupyter Notebook, aby móc skutecznie przeprowadzać analizę danych.

Przygotowanie środowiska pracy - Jupyter Notebook

Instalacja Anacondy:

- Krok 1: Pobierz Anacondę
  - Przejdź na oficjalną stronę Anacondy: https://www.anaconda.com/products/distribution

- Krok 2: Wybierz wersję
  - Pobierz odpowiednią wersję Anacondy dla swojego systemu operacyjnego (Windows, macOS, Linux). Zazwyczaj poleca się pobranie najnowszej stabilnej wersji.

Przygotowanie środowiska pracy - Jupyter Notebook

Instalacja Anacondy:

- Krok 3: Uruchom instalator
  - Uruchom pobrany plik instalacyjny Anacondy i postępuj zgodnie z instrukcjami na ekranie.
- Krok 4: Wybierz opcje instalacji
  - Podczas instalacji możesz zostawić domyślne opcje, chyba że masz konkretne preferencje dotyczące instalacji.
- Krok 5: Zakończ instalację
  - Po zakończeniu instalacji Anacondy powinna być gotowa do użycia.

Przygotowanie środowiska pracy - Jupyter Notebook

Uruchomienie Jupyter Notebook:

- Krok 1: Uruchom Anacondę Navigator
  - Po zainstalowaniu Anacondy, otwórz Anaconda Navigator. Możesz znaleźć go w menu Start (Windows) lub w terminalu (Linux/macOS) wpisując anaconda-navigator i naciskając Enter.

- Krok 2: Uruchom Jupyter Notebook
  - W Anaconda Navigatorze, znajdź i uruchom Jupyter Notebook. Kliknij na ikonę "Launch" obok Jupyter Notebook.

Przygotowanie środowiska pracy - Jupyter Notebook

Uruchomienie Jupyter Notebook:

- Krok 3: Przeglądarka Jupyter Notebook
  - Po chwili powinna otworzyć się przeglądarka internetowa z interfejsem Jupyter Notebook. Możesz przeglądać swoje pliki, tworzyć nowe notatniki i uruchamiać kod.

- Krok 4: Utwórz nowy notatnik
  - Kliknij na przycisk "New" i wybierz "Python 3". Otworzy się nowy notatnik, gdzie możesz wprowadzać kod.

Przygotowanie środowiska pracy - Jupyter Notebook

Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:

- Komórki (Cells):
  - Jupyter Notebook jest podzielony na komórki, które można traktować jako jednostki kodu lub tekstu. Komórki kodu są przeznaczone do wprowadzania i wykonywania kodu, natomiast komórki tekstu mogą zawierać opisy, instrukcje czy też równania matematyczne.

- Typy Komórek:
  - Code (Kod): Komórki, w których wprowadza się i wykonuje kod Pythona.
  - Markdown: Komórki, które zawierają tekst sformatowany w języku Markdown. Wykorzystywane do dodawania komentarzy, opisów czy instrukcji.

Przygotowanie środowiska pracy - Jupyter Notebook

Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:

- Uruchamianie Komórek:
  - Aby uruchomić kod w komórce, możesz użyć przycisku "Run" lub skrótu klawiszowego Shift + Enter.
  - Wynik działania kodu będzie wyświetlany poniżej komórki.

- Interaktywność:
  - Jupyter Notebook pozwala na interaktywną pracę z danymi. Możesz eksperymentować z kodem, zmieniać wartości i natychmiast obserwować rezultaty.

Przygotowanie środowiska pracy - Jupyter Notebook

Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:

- Wizualizacje:
  - Możesz używać bibliotek takich jak Matplotlib czy Seaborn do tworzenia wykresów i wizualizacji danych. Wykresy są wyświetlane w notatniku, co ułatwia analizę.

Przygotowanie środowiska pracy - Jupyter Notebook

Omówienie środowiska, wyglądu i funkcji Jupyter Notebook:

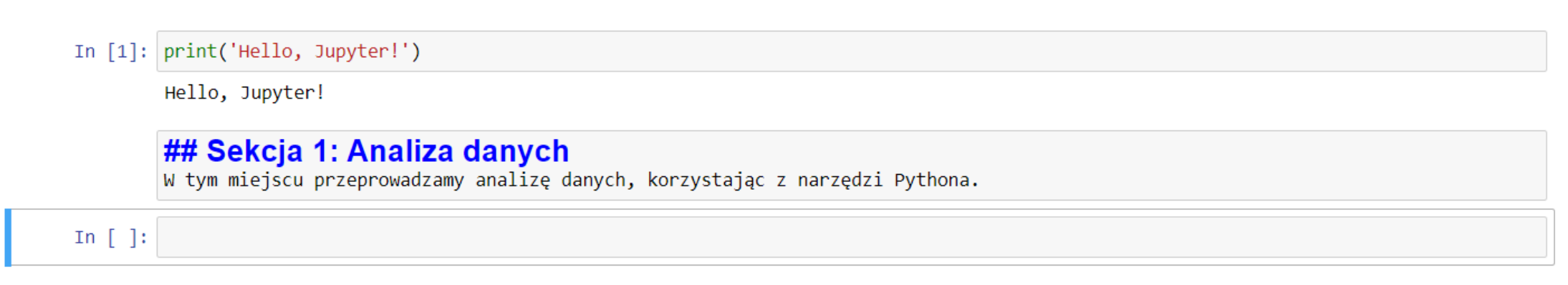
- Zapisywanie i Eksport:
  - Notatniki można zapisywać w formatach .ipynb (Jupyter Notebook), .html, .pdf, .py (skrypt Pythona) i innych.
  - Można eksportować notatniki do różnych formatów, co ułatwia ich udostępnianie.

- Obsługa Notatnika:
  - Notatnik zachowuje wyniki i zmienne nawet po ponownym uruchomieniu komórki. To ułatwia analizę i eksperymentowanie bez konieczności ponownego uruchamiania wszystkich komórek.

Omówienie środowiska pracy - Jupyter Notebook

Przykład:

Poniżej znajduje się prosty przykład notatnika Jupyter, składającego się z komórek kodu i tekstu:



Omówienie środowiska pracy - Jupyter Notebook

Kilka przydatnych skrótów klawiszowych w Jupyter Notebook, które mogą zwiększyć efektywność pracy:

- Shift + Enter: Uruchamia aktualną komórkę i przechodzi do następnej. Jeśli ostatnia komórka, to dodaje nową komórkę poniżej.
- Ctrl + Enter: Uruchamia aktualną komórkę, ale nie przechodzi do następnej, pozostając w bieżącej komórce.
- Esc + A: Dodaje nową komórkę powyżej aktualnej.
- Esc + B: Dodaje nową komórkę poniżej aktualnej.
- Esc + M: Zmienia typ komórki na Markdown (tekst).
- Esc + Y: Zmienia typ komórki na Code (kod).
- Esc + D, D: Kasuje aktualną komórkę.
- Esc + Z: Cofa ostatnią zmianę (przywraca skasowaną komórkę).

Omówienie środowiska pracy - Jupyter Notebook

Kilka przydatnych skrótów klawiszowych w Jupyter Notebook, które mogą zwiększyć efektywność pracy:

- Shift + Tab: Pokazuje podpowiedzi dotyczące funkcji lub metody (po umieszczeniu kursora wewnątrz nawiasów).
- Ctrl + S: Zapisuje notatnik.
- Esc + 1-6: Zmienia poziom nagłówka w komórce Markdown (1- najwyższy, 6- najniższy).
- Esc + H: Wyświetla listę wszystkich dostępnych skrótów klawiszowych.
- Ctrl + Shift + P: Otwiera polecenie paska wyszukiwania, pozwalając na szybkie wyszukiwanie i uruchamianie poleceń.
- Shift + M: Łączy zaznaczone komórki w jedną.
- Ctrl + Z: Cofa ostatnią zmianę.

Omówienie środowiska pracy - Jupyter Notebook

Te skróty klawiszowe mogą być bardzo przydatne podczas korzystania z Jupyter Notebook, przyspieszając pisanie kodu i nawigację w notatniku.

Python - Wybrane Elementy

W tej sekcji skupimy się na kilku kluczowych elementach języka Python, które są istotne w kontekście analizy danych i programowania w obszarze Data Science.

Python - Wybrane Elementy

Różnica między metodą a funkcją w Pythonie dotyczy przede wszystkim kontekstu, w którym są one używane:

Python - Wybrane Elementy

Metoda:

- Metoda jest funkcją, która jest związana z konkretnym obiektem lub typem danych.
- Jest wywoływana na rzecz konkretnego obiektu przy użyciu notacji kropkowej (.).
- Metoda może mieć dostęp do danych zawartych w obiekcie, na którym jest wywoływana, za pomocą argumentu self.

Przykładem metody jest .append() dla list, która dodaje element do listy, lub .capitalize() dla łańcuchów, która zmienia pierwszą literę na wielką.

Python - Wybrane Elementy

Funkcja:

- Funkcja jest blokiem kodu, który wykonuje określone zadanie i może być wywoływany z dowolnego miejsca w programie.
- Nie jest związana z konkretnym obiektem ani typem danych.
- Funkcje są definiowane przy użyciu słowa kluczowego def i mogą przyjmować argumenty.

Przykładem funkcji jest print(), która wypisuje wartość na standardowe wyjście, lub len(), która zwraca długość obiektu.

Python - Wybrane Elementy

- W skrócie, metoda jest specjalnym rodzajem funkcji, która jest związana z konkretnym obiektem, podczas gdy funkcja jest niezależnym blokiem kodu, który może być wywoływany w dowolnym kontekście.

- Metody mają dostęp do danych obiektu, na którym są wywoływane, podczas gdy funkcje nie mają takiego dostępu, chyba że dane są przekazywane jako argumenty.

Python - Wybrane Elementy

Przypomnienie podstawowych typów i struktur danych

- Liczby całkowite (int) i liczby zmiennoprzecinkowe (float):
  - Przykład:

```
# Int & Float

x = 5 # Int
y = 3.14 # Float
```

Python - Wybrane Elementy

Przypomnienie podstawowych typów i struktur danych

- Napisy (str):
  - Przykład:

```
# String

text = "Hello, Data Science!"
```

Python - Wybrane Elementy

Przypomnienie podstawowych typów i struktur danych

Listy:

- Przykład:

```
# Lista

numbers = [1, 2, 3, 4, 5]
```

Python - Wybrane Elementy

Przypomnienie podstawowych typów i struktur danych

Krotki (tuple):

- Przykład:

```
# Tuple

coordinates = (3, 4)
```

Python - Wybrane Elementy

Przypomnienie podstawowych typów i struktur danych

Słowniki (dict):

- Przykład:

```
# Dictionary

student = {'name': 'John',
           'age': 20,
           'grade': 'A'
          }
```

Python - Wybrane Elementy

Indeksowanie, Slicing i Iteracja

Indeksowanie:

 Indeksowanie w Pythonie zaczyna się od 0. 

- Przykład:

```
# Indeksowanie

numbers = [10, 20, 30, 40, 50]
first_element = numbers[0]  # Pobiera pierwszy element (10)
```

Python - Wybrane Elementy

Indeksowanie, Slicing i Iteracja

Slicing:

Pozwala na pobieranie fragmentów listy lub napisu.

- Przykład:

```
# Slicing

numbers = [10, 20, 30, 40, 50]
sliced_numbers = numbers[1:4]  # Pobiera elementy od indeksu 1 do 3
```

Python - Wybrane Elementy

Indeksowanie, Slicing i Iteracja

Iteracja:

- Przykład:

```
# Iteracja

numbers = [10, 20, 30, 40, 50]
for num in numbers:
    print(num)
```

```
10
20
30
40
50
```

Python - Wybrane Elementy

Funkcje, Funkcje Anonimowe (Lambda)

- Definiowanie Funkcji:
  - Przykład:

```
# Definiowanie funkcji

def add_numbers(x, y):
    return x + y
```

```
print(add_numbers(3, 2))
```

```
5
```

Python - Wybrane Elementy

Funkcja Anonimowa (Lambda):

- Python Lambda, jest jednolinijkową, anonimową funkcją. Nie jest skomplikowana. Jest to funkcja która nie ma nazwy. Poprzez użycie słowa kluczowego 'lambda' informujemy Python, że właśnie taką anonimową funkcję chcemy utworzyć. Następnie podajemy listę parametrów, które chcemy aby przyjmowała, używamy „:", oraz definiujemy jej zawartość.

- W przeciwieństwie do poprzednich funkcji funkcja lambda nie jest funkcją wyższego rzędu, aby definiować funkcje „na stałe". Służy do wykorzystania ad hoc i do ułatwienia sobie życia. Jeżeli natomiast przypiszemy funkcję do zmiennej, tak jak w przykładzie będziemy mogli się do niej później odwołać.

Python - Wybrane Elementy

Funkcja Anonimowa (Lambda):

- Przykład:

```
# Funkcja Lambda

multiply = lambda x, y: x * y
```

```
print(multiply(2, 2))
```

```
4
```

```
print(multiply(2, 4))
```

```
8
```

```
(lambda x, y: x * y)(3, 3)
```

```
9
```

```
print((lambda x, y: x * y)(3, 3))
```

```
9
```

Python - Wybrane Elementy

List Comprehension

- Umożliwia zwięzłe tworzenie list w jednej linii.
  - Przykład:

```
# List Comprehension

squares = [x**2 for x in range(1, 6)]

print(squares)
```

```
[1, 4, 9, 16, 25]
```

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje matematyczne:

- abs(), round(), max(), min(), sum()

Funkcje tekstowe:

- len(), str(), upper(), lower(), startswith(), endswith()

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje statystyczne, **należy zrobić import statistics**:

- mean(), median(), mode(), stdev()

Funkcje data i czas:

- datetime.now(), strftime(), strptime()

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje data i czas:

- Dlaczego importujemy najpierw z from datetime a później import datetime?

  - Moduł datetime zawiera głównie jedną klasę o nazwie datetime, która jest powszechnie używana.
  - Importując jedynie tę klasę za pomocą from datetime import datetime, można bezpośrednio odwoływać się do niej bez konieczności używania prefiksu nazwy modułu.
  - Użycie datetime.now() jest bardziej zwięzłe niż datetime.datetime.now(), co poprawia czytelność kodu.

Python - Wybrane Elementy

Wybrane funkcje wbudowane

W wielu przypadkach importowanie tylko potrzebnej klasy za pomocą from ... import ... jest bardziej praktyczne, ponieważ ogranicza ilość pisania kodu i poprawia czytelność. Jednakże, w niektórych sytuacjach, zwłaszcza gdy moduł datetime zawiera więcej niż jedną interesującą klasę lub funkcję, można zdecydować się na import całego modułu import datetime.

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje matematyczne:

- Przykład:

```
# Funkcje matematyczne

a = abs(-7.25) # Zwraca wartość bezwzględną.

b = round(5.76543, 2) # Zaokrągla liczbę do wybranej ilości miejsc po przecinku (nawias)

c = max(2, 3, 4, 5, 10, 15, 20) # Zwraca największą wartość z podanej tupli, listy.

d = min(2, 3, 4, 5, 10, 15, 20) # Zwraca najmniejszą wartość z podanej tupli, listy.

e = (2, 3, 4, 5, 10, 15, 20)

f = sum(e) # Sumuje wszystkie wartości z podanej zmiennej 'e'
```

```
print(f'{a},\n{b},\n{c},\n{d},\n{f}')
```

```
7.25,
5.77,
20,
2,
59
```

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje tekstowe:

- Przykład:

```
# Funkcje tekstowe

tekst = 'Przykładowy tekst do sprawdzenia.'
int = 4

g = len(tekst) # Zwraca długość znaków w podanym tekście.

h = str(int) # Zmienia wartość liczbową lub teks którego nie jesteśmy pewni na stringa

i = tekst.upper() # Zwraca tekst z powiększonymi znakami.

j = tekst.lower() # Zwraca tekst z pomniejszonymi znakami.

k = tekst.startswith('Przykład') # Zwraca wartość Prawda/Fałsz (Bolean)

l = tekst.startswith('Hello') # Zwraca wartość Prawda/Fałsz (Bolean)

m = tekst.endswith('.') # Zwraca wartość Prawda/Fałsz (Bolean)

n = tekst.endswith('World!') # Zwraca wartość Prawda/Fałsz (Bolean)
```

```
print(f'{g},\n{h},\n{i},\n{j},\n{k},\n{l},\n{m},\n{n}')
```

```
33,
4,
PRZYKŁADOWY TEKST DO SPRAWDZENIA.,
przykładowy tekst do sprawdzenia.,
True,
False,
True,
False
```

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje statystyczne:

- Przykład:

```
# Funkcje statystyczne

import statistics

numbersLong = [2, 4, 2, 5 ,6 ,7 , 8, 8 ,8 ,8 ,8 , 9, 10, 11, 11, 11, 11, 11, 11, 11, 12, 13, 2, 5, 6, 7]

print(f'''{statistics.mean(numbersLong)},
{statistics.median(numbersLong)},
{statistics.mode(numbersLong)},
{statistics.stdev(numbersLong)}''')

# .mean() - Średnia
# .median() - Mediana
# .mode() - Dominanta wartość najczęściej występująca w zbiorze.
# .stdev() - Odchylenie standardowe
```

```
7.961538461538462,
8.0,
11,
3.23086080456301
```

Python - Wybrane Elementy

Wybrane funkcje wbudowane

Funkcje data/czas:

- Przykład:

  (ze względu na ograniczone miejsce, przykłady na kolejnym slajdzie)

Python - Wybrane Elementy

Funkcje data/czas: Strftime()

```python
# Funkcje data i czas
# https://docs.python.org/3/library/datetime.html#format-codes

# Różnica pomiędzy strftime() a strptime - z strptime() stringa robimy obiekt, a z strftime() tworzymy stringa

from datetime import datetime

now = datetime.now() # current date and time

year = now.strftime("%Y")
print("year:", year)

month = now.strftime("%m")
print("month:", month)

day = now.strftime("%d")
print("day:", day)

time = now.strftime("%H:%M:%S")
print("time:", time)

date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date and time:", date_time)
print(now)
```

```
year: 2024
month: 01
day: 22
time: 21:06:22
date and time: 01/22/2024, 21:06:22
2024-01-22 21:06:22.394635
```

Python - Wybrane Elementy

Funkcje data/czas: Strptime()

```
# Funkcje data i czas

date_string = "21 June, 2018"

print("date_string =", date_string)
print("type of date_string =", type(date_string))

date_object = datetime.strptime(date_string, "%d %B, %Y")

print("date_object =", date_object)
print("type of date_object =", type(date_object))
```

```
date_string = 21 June, 2018
type of date_string = <class 'str'>
date_object = 2018-06-21 00:00:00
type of date_object = <class 'datetime.datetime'>
```

Python - Wybrane Elementy

Funkcje data/czas: Strptime()

```
# Funkcje data i czas

dt_string = "12/11/2018 09:15:32"

# Considering date is in dd/mm/yyyy format
dt_object1 = datetime.strptime(dt_string, "%d/%m/%Y %H:%M:%S")
print("dt_object1 =", dt_object1)

# Considering date is in mm/dd/yyyy format
dt_object2 = datetime.strptime(dt_string, "%m/%d/%Y %H:%M:%S")
print("dt_object2 =", dt_object2)
```

```
dt_object1 = 2018-11-12 09:15:32
dt_object2 = 2018-12-11 09:15:32
```

Python - Wybrane Elementy

Kontrola Przepływu

- Pętla While:
  - Przykład:

```
# Pętla While

i = 0
while i < 5:
    print(i)
    i += 1
```

```
0
1
2
3
4
```

Python - Wybrane Elementy

Kontrola Przepływu

- Pętla For:
  - Przykład:

```
# Pętla For

numbers = [1, 2, 3, 4, 5]
for i in numbers:
    print(i)
```

```
1
2
3
4
5
```

Python - Wybrane Elementy

Kontrola Przepływu

- Instrukcje Warunkowe (if, elif, else):
  - Przykład

```
# Instrukcje warunkowe (if, elif, else)

x = 10
if x > 0:
    print("Dodatnie")
elif x < 0:
    print("Ujemne")
else:
    print("Zero")
```

```
Dodatnie
```

Python - Wybrane Elementy

Warsztat

Python i jego pakiety

Prezentacja pakietów używanych do pracy z plikami Excela - Wstęp:

- pandas: wszechstronna biblioteka do manipulacji danymi.
- openpyxl: pakiet do pracy z plikami Excel w formacie xlsx.
- xlsxwriter: biblioteka do tworzenia nowych plików Excel z rozbudowanymi opcjami formatowania.
- xlrd: pakiet do odczytu plików Excel w formacie xls i xlsx.
- xlwt: biblioteka do tworzenia nowych plików Excel w formacie xls.
- xlutils: narzędzia wspomagające dla xlrd i xlwt.
- pyexcel: pakiet wspomagający pracę z różnymi formatami arkuszy kalkulacyjnych.

Python i jego pakiety

Omówienie zalet i zastosowań każdego z pakietów:

- pandas: łatwe wczytywanie, analizowanie i manipulowanie danymi; idealne do pracy z dużymi zbiorami danych.
- openpyxl: obsługa plików xlsx, możliwość odczytu i zapisu, modyfikacja istniejących plików.
- xlsxwriter: zaawansowane opcje formatowania, tworzenie wykresów, obsługa wielu arkuszy.
- xlrd: szybki odczyt plików xls, obsługa starych formatów Excel.
- xlwt: tworzenie plików xls, kompatybilność ze starszymi wersjami Excela.
- xlutils: narzędzia do kopiowania, modyfikacji i porównywania arkuszy.
- pyexcel: prostota i wszechstronność, obsługa różnych formatów arkuszy (csv, ods, xls, xlsx).

Python i jego pakiety

Instalacja pakietów:

Co należy zrobić aby zainstalować odpowiednie pakiety do przyszłej pracy?

Python i jego pakiety

Aby zainstalować odpowiednie pakiety należy wykorzystać pip, czyli – oficjalny oraz domyślny system zarządzania pakietami dla środowiska języka Python, korzystający z dedykowanego repozytorium pakietów o nazwie Python Package Index lub z innych zdalnych oraz lokalnych repozytoriów.

Kod:

```
!pip install pandas openpyxl xlsxwriter xlrd xlwt xlutils pyexcel
```

Python i jego pakiety

Krótkie przykłady użycia każdego pakietu

Python i jego pakiety

Przykład 1: Wczytanie pliku Excel do DataFrame

```
# Wczytanie pliku Excel do DataFrame:

import pandas as pd
df = pd.read_excel('sample.xlsx')
print(df.head())
```

Python i jego pakiety

Przykład 2: Eksportowanie DataFrame do pliku Excel

```
# Eksportowanie DataFrame do pliku Excel:

df.to_excel('output.xlsx', index=False)
```

Python i jego pakiety

Przykład 3: Openpyxl: Tworzenie nowego pliku Excel

```
# Openpyxl: Tworzenie nowego pliku Excel

from openpyxl import Workbook
wb = Workbook()
ws = wb.active
ws['A1'] = "Hello, Openpyxl!"
wb.save('openpyxl_example.xlsx')
```

Python i jego pakiety

Przykład 4: xlsxwriter: Tworzenie nowego pliku Excel z formatowaniem

```python
# xlsxwriter: Tworzenie nowego pliku Excel z formatowaniem

import xlsxwriter
workbook = xlsxwriter.Workbook('xlsxwriter_example_reminder.xlsx')
worksheet = workbook.add_worksheet()
worksheet.write('A1', 'Hello, XlsxWriter!')
format = workbook.add_format({'bold': True, 'font_color': 'red'})
worksheet.write('A2', 'Formatted text', format)
workbook.close()
```

Python i jego pakiety

Przykład 5: xlrd: Odczytywanie danych z pliku Excel

```
# xlrd: Odczytywanie danych z pliku Excel

import xlrd
workbook = xlrd.open_workbook('sample.xls')
sheet = workbook.sheet_by_index(0)
print(sheet.cell_value(0, 0))
```

Python i jego pakiety

Przykład 6: xlwt: Tworzenie nowego pliku Excel

```
# xlwt: Tworzenie nowego pliku Excel

import xlwt
workbook = xlwt.Workbook()
sheet = workbook.add_sheet('Sheet 1')
sheet.write(0, 0, 'Hello, Xlwt!')
workbook.save('xlwt_example.xls')
```

Python i jego pakiety

Przykład 7: xlutils: Kopiowanie arkusza

```
# xlutils: Kopiowanie arkusza

from xlrd import open_workbook
from xlutils.copy import copy
rb = open_workbook(r'C:\Users\PabloPapito\Downloads\jobsInData\jobsInDataXLS.xls') # Przykładowa ścieżka
wb = copy(rb)
ws = wb.get_sheet(0)
ws.write(0, 1, 'Modified by xlutils')
wb.save(r'C:\Users\PabloPapito\Downloads\jobsInData\xlutils_example.xls') # Przykładowa ścieżka
```

Python i jego pakiety

Przykład 8: pyexcel: Wczytywanie pliku Excel

```
# pyexcel: Wczytywanie pliku Excel

import pyexcel_xlsx as p
sheet = p.get_data(r'C:\Users\PabloPapito\Downloads\jobsInData\jobsInData.xlsx') # Przykładowa ścieżka
# print(sheet) - zwróci bardzo duży output
```

Python i jego pakiety

Przykład 9: pyexcel: Zapisanie pliku Excel ze stworzonego zakresu danych

```
import pyexcel as py
data = [[1, 2, 3], [4, 5, 6]]
p.save_as(array=data, dest_file_name='pyexcel_example.xlsx')
```

Python i jego pakiety

Przykład 10: pyexcel: Wczytanie arkusza z wcześniej zapisanego pliku Excel

```
py.get_sheet(file_name='pyexcel_example.xlsx')
```

```
pyexcel_sheet1:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
```

Wprowadzenie do biblioteki pandas

Series i DataFrame w Pandas

- Omówienie Pandas
  - Pandas to potężna biblioteka w języku Python przeznaczona do manipulacji i analizy danych. Dwa główne obiekty w Pandas to Series i DataFrame. Series reprezentuje jednowymiarową strukturę danych, podczas gdy DataFrame to dwuwymiarowa tabela danych.

Wprowadzenie do biblioteki pandas

Series i DataFrame w Pandas

- Struktura Series:
  - Jednowymiarowy obiekt zawierający dane, indeks i etykiety.

- Struktura DataFrame:
  - Dwuwymiarowa tabela danych z etykietowanymi kolumnami i indeksem.

Wprowadzenie do biblioteki pandas

Tworzenie DataFrame w Pandas na podstawie słownika:

```
# Tworzenie DataFrame z Dictionary
df_slownik = pd.DataFrame({'A': 1.0,
                           'B': pd.Timestamp('20220101'),
                           'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                           'D': np.array([3] * 4, dtype='int32'),
                           'E': pd.Categorical(["test", "train", "test", "train"]),
                           'F': 'foo'})
```

Wprowadzenie do biblioteki pandas

Wczytywanie i Zapis Danych w Różnych Formatach

Wczytywanie Danych w Pandas:

- Pandas oferuje wiele funkcji do wczytywania danych z różnych źródeł, takich jak pliki CSV, Excel, SQL, a nawet strony internetowe. Poniżej znajdują się przykłady wczytywania danych z pliku CSV i Excel:

Wprowadzenie do biblioteki pandas

Przykład:

Dane w csv (Dane w csv (Pliki są dostępne na GitHub)

```python
import pandas as pd

# Wczytywanie danych z pliku CSV
df_csv = pd.read_csv('nazwa_pliku.csv')

# Wczytywanie danych z pliku Excel
df_excel = pd.read_excel('nazwa_pliku.xlsx', sheet_name='Arkusz1')
```

Wprowadzenie do biblioteki pandas

Wczytywanie i Zapis Danych w Różnych Formatach

Zapis Danych w Pandas:

- Podobnie jak z wczytywaniem, Pandas umożliwia zapisywanie danych do różnych formatów. Poniżej znajdują się przykłady zapisywania danych do pliku CSV i Excel:

Wprowadzenie do biblioteki pandas

Przykład:

```
df_slownik.to_csv('NowyPlikSlownik.csv', index=False)
df_slownik.to_excel('NowyPlikExcel.xlsx', sheet_name='Slownik', index=False)
```

```
# Sprawdzenie Current Working Directory (CWD) w JupyterNotebook
# metoda 1
import os

notebook_path = os.getcwd()
print(notebook_path)

# metoda 2
from ipykernel import get_connection_file
# import os - to już zaimportowaliśmy więc nie ma potrzeby powielać, natomiast jest niezbędnę do wykorzystania metody 2

connection_file = get_connection_file()
notebook_path = os.path.dirname(connection_file)
print(notebook_path)
```

```
C:\Users\PabloPapito\PythonWAnalizieDanych\DayTwo
C:\Users\PabloPapito\AppData\Roaming\jupyter\runtime
```

Wprowadzenie do biblioteki pandas

Podstawowe Atrybuty DataFrame w Pandas

- Struktura DataFrame w Pandas:
  - DataFrame w Pandas to dwuwymiarowa struktura danych, która składa się z wierszy i kolumn. Przedstawmy kilka podstawowych atrybutów, które pozwalają zrozumieć strukturę danych w ramach DataFrame.

Wprowadzenie do biblioteki pandas

Atrybut shape:

- Atrybut shape zwraca krotkę reprezentującą liczbę wierszy i kolumn w DF.

```python
import pandas as pd

# Przykładowe utworzenie DataFrame
data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data)

# Sprawdzenie kształtu DataFrame
ksztalt = df.shape
print("Kształt DataFrame:", ksztalt)
```

```
Kształt DataFrame: (3, 2)
```

Wprowadzenie do biblioteki pandas

Atrybut index:

- Atrybut index zwraca indeksy wierszy (etykiety).

```
# Sprawdzenie indeksów wierszy DataFrame
indeksy = df.index
print("Indeksy wierszy DataFrame:", indeksy)
```

```
Indeksy wierszy DataFrame: RangeIndex(start=0, stop=3, step=1)
```

Wprowadzenie do biblioteki pandas

Atrybut columns:

- Atrybut columns zwraca etykiety kolumn.

```
# Sprawdzenie etykiet kolumn DataFrame
etykiety_kolumn = df.columns
print("Etykiety kolumn DataFrame:", etykiety_kolumn)
```

```
Etykiety kolumn DataFrame: Index(['A', 'B'], dtype='object')
```

Wprowadzenie do biblioteki pandas

Atrybut dtypes:

- Atrybut dtypes zwraca informacje o typach danych w poszczególnych kolumnach.

```
# Sprawdzenie typów danych kolumn DataFrame
typy_danych = df.dtypes
print("Typy danych kolumn DataFrame:\n", typy_danych)
```

```
Typy danych kolumn DataFrame:
 A    int64
B    int64
dtype: object
```

Wprowadzenie do biblioteki pandas

Przydatne Funkcje w Pandas

- W Pandas istnieje wiele przydatnych funkcji do szybkiego podglądu i analizy danych. Poniżej omówione są niektóre z tych funkcji:

Wprowadzenie do biblioteki pandas

Funkcja describe()

- Funkcja describe() dostarcza podsumowania statystyczne dla kolumn w DF, takie jak średnia, odchylenie standardowe, minimum, maksimum i kwartyle.

```
# Przykład użycia describe
opis_statystyczny = df.describe()
print("Opis statystyczny DataFrame:\n", opis_statystyczny)
```

```
Opis statystyczny DataFrame:
              A    B
count  3.0  3.0
mean   2.0  5.0
std    1.0  1.0
min    1.0  4.0
25%    1.5  4.5
50%    2.0  5.0
75%    2.5  5.5
max    3.0  6.0
```

Wprowadzenie do biblioteki pandas

Funkcja info()

- Funkcja info() wyświetla podstawowe informacje o DF, takie jak ilość niepustych wartości i typy danych dla każdej kolumny.

```
# Przykład użycia info
informacje_o_danych = df.info()
print("Informacje o danych w DataFrame:\n", informacje_o_danych)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       3 non-null      int64
 1   B       3 non-null      int64
dtypes: int64(2)
memory usage: 180.0 bytes
Informacje o danych w DataFrame:
 None
```

Wprowadzenie do biblioteki pandas

Funkcja head(n)

- Funkcja head(n) zwraca pierwsze n wierszy DF, co jest przydatne do szybkiego podglądu danych.

```
# Przykład użycia head
pierwsze_wiersze = df.head(3)  # Zwróci trzy pierwsze wiersze
print("Pierwsze trzy wiersze DataFrame:\n", pierwsze_wiersze)
```

```
Pierwsze trzy wiersze DataFrame:
    A  B
0  1  4
1  2  5
2  3  6
```

Wprowadzenie do biblioteki pandas

Funkcja tail(n)

- Funkcja tail(n) zwraca ostatnie n wierszy DF, co pomaga przy weryfikacji danych na końcu zbioru.

```
# Przykład użycia tail
ostatnie_wiersze = df.tail(3)  # Zwróci trzy ostatnie wiersze
print("Ostatnie trzy wiersze DataFrame:\n", ostatnie_wiersze)
```

```
Ostatnie trzy wiersze DataFrame:
    A  B
0  1  4
1  2  5
2  3  6
```

Wprowadzenie do biblioteki pandas

Funkcja sample(n)

- Funkcja sample(n) zwraca losowe n wierszy z DF.

```
# Przykład użycia sample
losowe_wiersze = df.sample(3)  # Zwróci trzy losowe wiersze
print("Losowe trzy wiersze DataFrame:\n", losowe_wiersze)
```

```
Losowe trzy wiersze DataFrame:
    A  B
1  2  5
2  3  6
0  1  4
```

Wprowadzenie do biblioteki pandas

Czyszczenie wartości zduplikowanych w Pandas

Sprawdzanie duplikatów:

- W analizie danych często konieczne jest sprawdzenie i usunięcie zduplikowanych wierszy. Pandas dostarcza funkcji do tego celu.

Wprowadzenie do biblioteki pandas

Czyszczenie wartości zduplikowanych w Pandas

```python
import pandas as pd

# Przykładowe utworzenie DataFrame z zduplikowanymi danymi
data = {'A': [1, 2, 2, 3, 4],
        'B': ['a', 'b', 'b', 'c', 'd']}
df2 = pd.DataFrame(data)

# Sprawdzenie duplikatów w całych wierszach
duplikaty_wiersze = df2.duplicated()

# Sprawdzenie duplikatów w kolumnie 'A'
duplikaty_kolumna_A = df2['A'].duplicated()

print("Duplikaty wierszy:\n", duplikaty_wiersze)
print("Duplikaty w kolumnie 'A':\n", duplikaty_kolumna_A)
```

Wprowadzenie do biblioteki pandas

Czyszczenie wartości zduplikowanych w Pandas

```
# Usunięcie zduplikowanych wierszy (zostawiając pierwszy wystąpienie)
df_bez_duplikatow = df2.drop_duplicates()

# Usunięcie zduplikowanych wierszy bazując na konkretnej kolumnie
df_bez_duplikatow_kolumna_A = df2.drop_duplicates(subset='A')

print("DataFrame bez duplikatów:\n", df_bez_duplikatow)
print("DataFrame bez duplikatów w kolumnie 'A':\n", df_bez_duplikatow_kolumna_A)
```

```
DataFrame bez duplikatów:
    A  B
0  1  a
1  2  b
3  3  c
4  4  d
DataFrame bez duplikatów w kolumnie 'A':
    A  B
0  1  a
1  2  b
3  3  c
4  4  d
```

Wprowadzenie do biblioteki pandas

Wartości brakujące (None/Null) - Różne podejścia do radzenia sobie z nimi

- W analizie danych często spotykamy się z wartościami brakującymi (NaN lub None). Pandas oferuje różne metody radzenia sobie z tymi wartościami.

Wprowadzenie do biblioteki pandas

Wartości brakujące (None/Null) - Różne podejścia do radzenia sobie z nimi

- Sprawdzenie wartości brakujących:

```python
import pandas as pd

# Przykładowe utworzenie DataFrame z wartościami brakującymi
data = {'A': [1, 2, None, 4],
        'B': ['a', 'b', 'c', None]}
df = pd.DataFrame(data)

# Sprawdzenie, czy istnieją wartości brakujące w DataFrame
brakujace_wartosci = df.isnull()

print("Wartości brakujące w DataFrame:\n", brakujace_wartosci)
```

```
Wartości brakujące w DataFrame:
        A      B
0  False  False
1  False  False
2   True  False
3  False   True
```

Wprowadzenie do biblioteki pandas

Wartości Brakujące (None/Null) - Różne podejścia do radzenia sobie z nimi

- Usuwanie wartości brakujących:

```
# Usunięcie wierszy zawierających przynajmniej jedną wartość brakującą
df_bez_brakujacych_wierszy = df.dropna()

# Usunięcie kolumn zawierających przynajmniej jedną wartość brakującą
df_bez_brakujacych_kolumn = df.dropna(axis=1)

print("DataFrame bez wierszy zawierających wartości brakujące:\n", df_bez_brakujacych_wierszy)
print("DataFrame bez kolumn zawierających wartości brakujące:\n", df_bez_brakujacych_kolumn)
```

```
DataFrame bez wierszy zawierających wartości brakujące:
      A  B
0  1.0  a
1  2.0  b
DataFrame bez kolumn zawierających wartości brakujące:
 Empty DataFrame
Columns: []
Index: [0, 1, 2, 3]
```

Wprowadzenie do biblioteki pandas

Wartości brakujące (None/Null) - Różne podejścia do radzenia sobie z nimi

- Uzupełnianie wartości brakujących:

```
In [75]: # Uzupełnienie wartości brakujących określoną wartością (np. średnią)
         srednia_wartosc_A = df['A'].mean()
         df_uzupelnione = df.fillna({'A': srednia_wartosc_A, 'B': 'brak_danych'})

         print("DataFrame po uzupełnieniu wartości brakujących:\n", df_uzupelnione)
```

```
DataFrame po uzupełnieniu wartości brakujących:
          A            B
0  1.000000            a
1  2.000000            b
2  2.333333            c
3  4.000000  brak_danych
```

Wprowadzenie do biblioteki pandas

Wykrywanie wartości odstających w Pandas

Wartości odstające (ang. outliers) są danymi, które znacząco różnią się od reszty zbioru danych i mogą wpływać na wyniki analizy. Pandas pozwala na wykrywanie tych wartości i podejmowanie odpowiednich działań.

Wprowadzenie do biblioteki pandas

Wykrywanie wartości odstających w Pandas

```python
import pandas as pd

# Przykładowe utworzenie DataFrame z wartościami odstającymi
data = {'A': [1, 2, 3, 100],
        'B': [4, 5, 6, 200]}
df = pd.DataFrame(data)

# Wykrywanie wartości odstających na podstawie kwantyli
kwantyl_25 = df.quantile(0.25)
kwantyl_75 = df.quantile(0.75)
rozstep_miedzykwartylowy = kwantyl_75 - kwantyl_25

# Definiowanie zakresu wartości "normalnych"
dolny_limit = kwantyl_25 - 1.5 * rozstep_miedzykwartylowy
gorny_limit = kwantyl_75 + 1.5 * rozstep_miedzykwartylowy

# Wykrywanie wartości odstających
odstajace_wartosci = (df < dolny_limit) | (df > gorny_limit)

print("Wartości odstające w DataFrame:\n", odstajace_wartosci)
```

```
Wartości odstające w DataFrame:
        A      B
0  False  False
1  False  False
2  False  False
3   True   True
```

Wprowadzenie do biblioteki pandas

Zastępowanie wartości odstających:

```
# Zastępowanie wartości odstających wartością graniczną
df_bez_odstajacych = df.mask(odstajace_wartosci, gorny_limit, axis=1)

print("DataFrame po zastąpieniu wartości odstających:\n", df_bez_odstajacych)
```

```
DataFrame po zastąpieniu wartości odstających:
      A        B
0   1.0    4.000
1   2.0    5.000
2   3.0    6.000
3  65.5  129.125
```

Wprowadzenie do biblioteki pandas

Sortowanie danych w Pandas

Sortowanie danych jest ważnym krokiem w analizie danych, umożliwiając uporządkowanie danych według określonych kryteriów. Pandas oferuje funkcje umożliwiające sortowanie zarówno wierszy, jak i kolumn.

Wprowadzenie do biblioteki pandas

Sortowanie wierszy:

```python
import pandas as pd

# Przykładowe utworzenie DataFrame do sortowania wierszy
data = {'A': [3, 1, 4, 2],
        'B': ['c', 'a', 'd', 'b']}
df = pd.DataFrame(data)

# Sortowanie wierszy według kolumny 'A'
df_posortowany = df.sort_values(by='A')

print("DataFrame po posortowaniu wierszy według kolumny 'A':\n", df_posortowany)
```

```
DataFrame po posortowaniu wierszy według kolumny 'A':
    A  B
1  1  a
3  2  b
0  3  c
2  4  d
```

Wprowadzenie do biblioteki pandas

Sortowanie kolumn:

```
# Sortowanie kolumn według etykiet kolumn
df_kolumny_posortowane = df.sort_index(axis=1)

print("DataFrame po posortowaniu kolumn według etykiet:\n", df_kolumny_posortowane)
```

```
DataFrame po posortowaniu kolumn według etykiet:
    A  B
0  3  c
1  1  a
2  4  d
3  2  b
```

Wprowadzenie do biblioteki pandas

Sortowanie malejące:

```
# Sortowanie wierszy malejąco według kolumny 'A'
df_posortowany_malejaco = df.sort_values(by='A', ascending=False)

print("DataFrame po posortowaniu malejąco według kolumny 'A':\n", df_posortowany_malejaco)
```

```
DataFrame po posortowaniu malejąco według kolumny 'A':
    A  B
2  4  d
0  3  c
3  2  b
1  1  a
```

Wprowadzenie do biblioteki pandas

Filtrowanie danych w Pandas

Filtrowanie danych to proces wybierania jedynie tych danych, które spełniają określone kryteria. W Pandas istnieje kilka metod do filtrowania danych, a każda z nich ma swoje zastosowanie.

Wprowadzenie do biblioteki pandas

Filtrowanie za pomocą loc:

```
import pandas as pd

# Przykładowe utworzenie DataFrame do filtrowania
data = {'A': [1, 2, 3, 4],
        'B': ['a', 'b', 'c', 'd']}

dfBezIndeksow = pd.DataFrame(data)

dfBezIndeksow.index += 1 # by default indeksy są od 0 - tutaj możemy ustawić od 1

df = pd.DataFrame(data, index=['x', 'y', 'z', 'w'])

# Filtrowanie wierszy o indeksach 'x' i 'y' oraz kolumny 'A'
df_filtr_loc = df.loc[['x', 'y'], 'A']


print('Nasz zbiór danych:\n', data, '\n\n----------------------\n')

print('Nasz zbiór danych bez indeksów literowych tylko cyfrowe od 0:\n', dfBezIndeksow, '\n\n----------------------\n')



print('Dodanie indeksów wierszy do danych:\n', df, '\n\n----------------------\n')

print("Wynik filtrowania za pomocą loc:\n", df_filtr_loc)
```

Wprowadzenie do biblioteki pandas

Filtrowanie za pomocą iloc:

```
# Filtrowanie pierwszych dwóch wierszy oraz pierwszej kolumny
df_filtr_iloc = df.iloc[:2, 0] # pierwszy argument to wiersz, a drugi to kolumna

print("Wynik filtrowania za pomocą iloc:\n", df_filtr_iloc)
```

```
Wynik filtrowania za pomocą iloc:
 x    1
y    2
Name: A, dtype: int64
```

Wprowadzenie do biblioteki pandas

Filtrowanie za pomocą query:

```
# Filtrowanie wierszy, gdzie wartość w kolumnie 'A' jest większa niż 2
df_filtr_query = df.query('A > 2')

print("Wynik filtrowania za pomocą query:\n", df_filtr_query)
```

```
Wynik filtrowania za pomocą query:
    A  B
z   3  c
w   4  d
```

Wprowadzenie do biblioteki pandas

Filtrowanie za pomocą where:

```
# Filtrowanie, zwracając nowy DataFrame z wartościami, które spełniają warunek, a reszta to NaN
df_filtr_where = df.where(df['A'] > 2)

print("Wynik filtrowania za pomocą where:\n", df_filtr_where)
```

```
Wynik filtrowania za pomocą where:
     A    B
x  NaN  NaN
y  NaN  NaN
z  3.0    c
w  4.0    d
```

Wprowadzenie do biblioteki pandas

Filtrowanie za pomocą isin:

```
# Filtrowanie wierszy, gdzie wartość w kolumnie 'B' jest jedną z określonych wartości
df_filtr_isin = df[df['B'].isin(['a', 'c'])]

print("Wynik filtrowania za pomocą isin:\n", df_filtr_isin)
```

```
Wynik filtrowania za pomocą isin:
    A  B
x   1  a
z   3  c
```

Wprowadzenie do biblioteki pandas

Tabele Przestawne w Pandas

- Tabele przestawne to narzędzie umożliwiające podsumowanie i analizę danych w formie tabeli. Pandas oferuje funkcję do tworzenia tabel przestawnych, co ułatwia analizę różnych aspektów danych.

Wprowadzenie do biblioteki pandas

Tworzenie tabeli przestawnej:

```python
import pandas as pd

# Przykładowe utworzenie DataFrame do tabeli przestawnej
data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Value': [10, 20, 30, 40, 50, 60]}
df = pd.DataFrame(data)

# Tworzenie tabeli przestawnej
table_przestawna = pd.pivot_table(df, values='Value', columns='Category', aggfunc='sum')

print("Tabela przestawna:\n", table_przestawna)
```

```
Tabela przestawna:
 Category   A    B
Value      90  120
```

Wprowadzenie do biblioteki pandas

Określanie indeksów i kolumn tabeli przestawnej:

```
# Określanie indeksów i kolumn dla tabeli przestawnej
table_przestawna_indeks_kolumny = pd.pivot_table(df, values='Value', index='Category', columns='Category', aggfunc='sum')

print("Tabela przestawna z określonymi indeksami i kolumnami:\n", table_przestawna_indeks_kolumny)
```

```
Tabela przestawna z określonymi indeksami i kolumnami:
 Category     A      B
Category
A         90.0    NaN
B          NaN  120.0
```

Wprowadzenie do biblioteki pandas

Określanie funkcji agregującej:

```
# Określanie funkcji agregującej dla tabeli przestawnej (np. średnia)
table_przestawna_srednia = pd.pivot_table(df, values='Value', index='Category', aggfunc='mean')

print("Tabela przestawna z określoną funkcją agregującą (średnia):\n", table_przestawna_srednia)
```

```
Tabela przestawna z określoną funkcją agregującą (średnia):
          Value
Category
A            30
B            40
```

Wprowadzenie do biblioteki pandas

Uzupełnianie wartości brakujących:

```
# Uzupełnianie wartości brakujących w tabeli przestawnej (np. wartością 0)
table_przestawna_uzupelniona = pd.pivot_table(df, values='Value', index='Category', fill_value=0, aggfunc='sum')

print("Tabela przestawna z uzupełnionymi wartościami brakującymi:\n", table_przestawna_uzupelniona)
```

```
Tabela przestawna z uzupełnionymi wartościami brakującymi:
          Value
Category
A            90
B           120
```

Wprowadzenie do biblioteki pandas

Grupowanie danych w Pandas

- Grupowanie danych to proces dzielenia danych na grupy w oparciu o określone kryteria, a następnie wykonywania operacji na każdej z tych grup. W Pandas, do grupowania danych używamy funkcji groupby.

Wprowadzenie do biblioteki pandas

Grupowanie na podstawie jednej kolumny:

```python
import pandas as pd

# Przykładowe utworzenie DataFrame do grupowania
data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
        'Value': [10, 20, 30, 40, 50, 60]}
df = pd.DataFrame(data)

# Grupowanie na podstawie kolumny 'Category' i obliczanie sumy dla każdej grupy
grupowanie_jedna_kolumna = df.groupby('Category').sum()

print("Wynik grupowania na podstawie jednej kolumny:\n", grupowanie_jedna_kolumna)
```

```
Wynik grupowania na podstawie jednej kolumny:
          Value
Category
A            90
B           120
```

Wprowadzenie do biblioteki pandas

Grupowanie na podstawie wielu kolumn:

```
# Grupowanie na podstawie wielu kolumn i obliczanie sumy dla każdej grupy
grupowanie_wiele_kolumn = df.groupby(['Category', 'Value']).size().reset_index(name='Count')

print("Wynik grupowania na podstawie wielu kolumn:\n", grupowanie_wiele_kolumn)
```

```
Wynik grupowania na podstawie wielu kolumn:
   Category  Value  Count
0         A     10      1
1         A     30      1
2         A     50      1
3         B     20      1
4         B     40      1
5         B     60      1
```

Wprowadzenie do biblioteki pandas

Wykonywanie różnych operacji dla każdej grupy:

```
# Grupowanie i obliczanie różnych statystyk dla każdej grupy
grupowanie_statystyki = df.groupby('Category').agg({'Value': ['sum', 'mean', 'count']})

print("Wynik grupowania z różnymi operacjami dla każdej grupy:\n", grupowanie_statystyki)
```

```
Wynik grupowania z różnymi operacjami dla każdej grupy:
          Value
            sum  mean count
Category
A            90  30.0     3
B           120  40.0     3
```

Wprowadzenie do biblioteki pandas

Łączenie danych w Pandas

- Łączenie danych to proces łączenia dwóch lub więcej DataFrame'ów w celu utworzenia jednego DataFrame'u na podstawie wspólnych kolumn lub indeksów. W Pandas, do łączenia danych używamy funkcji merge lub concat.

Wprowadzenie do biblioteki pandas

Łączenie na podstawie wspólnej kolumny:

```
import pandas as pd

# Przykładowe utworzenie dwóch DataFrame'ów do łączenia
df1 = pd.DataFrame({'ID': [1, 2, 3],
                    'Value1': ['A', 'B', 'C']})

df2 = pd.DataFrame({'ID': [2, 3, 4],
                    'Value2': ['X', 'Y', 'Z']})

# łączenie na podstawie wspólnej kolumny 'ID'
laczenie_kolumna = pd.merge(df1, df2, on='ID')

print("Wynik łączenia na podstawie wspólnej kolumny:\n", laczenie_kolumna)
```

```
Wynik łączenia na podstawie wspólnej kolumny:
    ID Value1 Value2
0   2      B      X
1   3      C      Y
```

Wprowadzenie do biblioteki pandas

Łączenie na podstawie wspólnego indeksu:

```
# Przykładowe utworzenie dwóch DataFrame'ów do łączenia na podstawie indeksu
df3 = pd.DataFrame({'Value3': ['M', 'N', 'O']}, index=[1, 2, 3])

# łączenie na podstawie wspólnego indeksu
laczenie_indeks = pd.concat([df1, df3], axis=1)

print("Wynik łączenia na podstawie wspólnego indeksu:\n", laczenie_indeks)
```

```
Wynik łączenia na podstawie wspólnego indeksu:
     ID Value1 Value3
0   1.0      A    NaN
1   2.0      B      M
2   3.0      C      N
3   NaN    NaN      O
```

Wprowadzenie do biblioteki pandas

Łączenie na podstawie wspólnego indeksu w kierunku pionowym:

```
# łączenie na podstawie wspólnego indeksu w kierunku pionowym
laczenie_pionowe = pd.concat([df1, df3], axis=0)

print("Wynik łączenia na podstawie wspólnego indeksu w kierunku pionowym:\n", laczenie_pionowe)
```

```
Wynik łączenia na podstawie wspólnego indeksu w kierunku pionowym:
     ID Value1 Value3
0   1.0      A    NaN
1   2.0      B    NaN
2   3.0      C    NaN
1   NaN    NaN      M
2   NaN    NaN      N
3   NaN    NaN      O
```

Wprowadzenie do biblioteki pandas

Tworzenie nowych atrybutów w Pandas

- Tworzenie nowych atrybutów (kolumn) to kluczowy krok w analizie danych, który pozwala na dostosowywanie danych do konkretnych potrzeb i celów analizy. W Pandas, nowe atrybuty można dodawać poprzez różne operacje na istniejących danych.

Wprowadzenie do biblioteki pandas

Dodawanie nowego atrybutu na podstawie istniejących kolumn:

```python
import pandas as pd

# Przykładowe utworzenie DataFrame
df = pd.DataFrame({'Value1': [10, 20, 30],
                   'Value2': [5, 15, 25]})

# Dodawanie nowego atrybutu 'Sum' na podstawie istniejących kolumn
df['Sum'] = df['Value1'] + df['Value2']

print("DataFrame z dodanym atrybutem 'Sum':\n", df)
```

```
DataFrame z dodanym atrybutem 'Sum':
    Value1  Value2  Sum
0       10       5   15
1       20      15   35
2       30      25   55
```

Wprowadzenie do biblioteki pandas

Dodawanie nowego atrybutu na podstawie warunków logicznych:

```
# Dodawanie nowego atrybutu 'Category' na podstawie warunków logicznych
df['Category'] = ['High' if x > 15 else 'Low' for x in df['Sum']]

print("DataFrame z dodanym atrybutem 'Category':\n", df)
```

```
DataFrame z dodanym atrybutem 'Category':
    Value1  Value2  Sum Category
0       10       5   15      Low
1       20      15   35     High
2       30      25   55     High
```

Wprowadzenie do biblioteki pandas

Utworzenie nowego atrybutu przy użyciu funkcji:

```
# Utworzenie nowego atrybutu 'Product' przy użyciu funkcji
def categorize_product(sum_value):
    if sum_value > 30:
        return 'Premium'
    elif sum_value > 20:
        return 'Standard'
    else:
        return 'Basic'

df['Product'] = df['Sum'].apply(categorize_product)

print("DataFrame z dodanym atrybutem 'Product':\n", df)
```

```
DataFrame z dodanym atrybutem 'Product':
    Value1  Value2  Sum Category  Product
0      10       5   15      Low    Basic
1      20      15   35     High  Premium
2      30      25   55     High  Premium
```

Python - Wybrane Elementy

Warsztat

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych za pomocą openpyxl:

openpyxl to bardzo popularna biblioteka, która pozwala na odczytywanie, modyfikację i tworzenie plików Excel w formacie .xlsx. Daje również możliwość pracy z zaawansowanymi funkcjami, takimi jak formatowanie komórek, tworzenie wykresów i obsługa formuł.

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych za pomocą openpyxl:

Przykład:

```python
from openpyxl import Workbook

# Tworzymy nowy obiekt Workbook - jest to "skoroszyt" w Excelu.
wb = Workbook()

# Aktywujemy domyślny arkusz (worksheet), który jest tworzony automatycznie w nowym skoroszycie.
ws = wb.active

# Dodajemy dane do komórek, podobnie jak w Excelu, ws['A1'] oznacza komórkę A1.
ws['A1'] = 'Imię'          # Nagłówek kolumny A
ws['B1'] = 'Wynagrodzenie'  # Nagłówek kolumny B

# Dodajemy kolejne wiersze za pomocą metody `append`, która automatycznie przechodzi do kolejnych wierszy.
ws.append(['Anna', 5000])   # Wiersz z danymi
ws.append(['Piotr', 6000])  # Kolejny wiersz z danymi

# Pętla zaczynająca od wiersza 3, aby nie nadpisać wierszy z "Anna" i "Piotr"
for row in range(3, 10):  # Zaczynamy od wiersza 3
    ws[f'A{row}'] = f'Imię {row}'          # Wypełniamy kolumnę A imionami
    ws[f'B{row}'] = 5000 + row * 100       # W kolumnie B dodajemy wynagrodzenia

# Zapisujemy plik jako 'przyklad.xlsx' - zapisuje cały skoroszyt na dysk.
wb.save("przyklad.xlsx")
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych za pomocą openpyxl:

W tym przykładzie:

- Utworzono nowy skoroszyt w Excelu.
- Dodano nagłówki oraz dwa wiersze danych za pomocą metody .append().
- W pętli dodano kolejne wiersze z danymi.
- Zapisano plik jako przyklad.xlsx.

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych z pomocą xlsxwriter:

xlsxwriter to kolejna popularna biblioteka służąca do tworzenia plików Excel w formacie .xlsx, oferująca zaawansowane funkcje formatowania, takie jak style, wykresy, tabele, ale jest ograniczona do zapisu (nie wspiera odczytu plików).

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych z pomocą xlsxwriter:

```
import xlsxwriter

# Tworzymy nowy obiekt Workbook, co odpowiada skoroszytowi w Excelu.
workbook = xlsxwriter.Workbook('przyklad.xlsx')

# Dodajemy nowy arkusz do skoroszytu.
worksheet = workbook.add_worksheet()

# Wstawiamy nagłówki do komórek. Metoda write() pozwala dodawać dane do konkretnych komórek.
worksheet.write('A1', 'Imię')             # Nagłówek w kolumnie A
worksheet.write('B1', 'Wynagrodzenie')    # Nagłówek w kolumnie B

# Dodajemy dane do poszczególnych wierszy.
worksheet.write('A2', 'Anna')    # Imię w wierszu 2
worksheet.write('B2', 5000)      # Wynagrodzenie w wierszu 2

# Podobnie jak w openpyxl, możemy dodawać dane w pętli dla większej ilości danych.
for row in range(3, 10):
    worksheet.write(row, 0, f'Imię {row}')  # Wstawiamy dane do kolumny A
    worksheet.write(row, 1, 5000 + row * 100)  # Wstawiamy dane do kolumny B

# Zapisujemy skoroszyt jako 'przyklad.xlsx'.
workbook.close()
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych z pomocą xlsxwriter:

W tym przykładzie:

- Utworzono nowy skoroszyt za pomocą biblioteki xlsxwriter.
- Dodano nowy arkusz do tego skoroszytu.
- Wstawiono nagłówki „Imię” i „Wynagrodzenie” do komórek A1 i B1.
- Wstawiono dane do wiersza 2: „Anna” i 5000.
- W pętli wstawiono kolejne wiersze danych od 1 do 9 w kolumnach A i B, gdzie w kolumnie A dodano imiona (np. „Imię 1”), a w kolumnie B dodano wynagrodzenia (5000 + row * 100).
- Zapisano skoroszyt do pliku o nazwie przyklad.xlsx.

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych z pomocą xlrd i xlwt:

xlrd służy do odczytywania danych z plików Excel w formacie .xls,

natomiast **xlwt** jest używany do tworzenia i zapisywania tychże plików.

Obie biblioteki są już przestarzałe i zaleca się korzystanie z nowocześniejszych rozwiązań jak openpyxl dla plików .xlsx.

Jednak jeśli musisz pracować z plikami .xls, oto krótki przykład:

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Dodawanie danych z pomocą xlrd:

```python
import xlwt

# Tworzymy nowy skoroszyt.
wb = xlwt.Workbook()

# Dodajemy nowy arkusz do skoroszytu.
ws = wb.add_sheet('Arkusz1')

# Wstawiamy nagłówki do pierwszego wiersza.
ws.write(0, 0, 'Imię')           # Kolumna A
ws.write(0, 1, 'Wynagrodzenie')   # Kolumna B

# Dodajemy dane do kolejnych wierszy.
ws.write(1, 0, 'Anna')    # Imię w kolumnie A
ws.write(1, 1, 5000)      # Wynagrodzenie w kolumnie B

# Pętla do wstawiania danych w wielu wierszach.
for row in range(1, 10):
    ws.write(row, 0, f'Imię {row}')   # Kolumna A
    ws.write(row, 1, 5000 + row * 100)  # Kolumna B

# Zapisujemy skoroszyt w formacie .xls.
wb.save('przyklad.xls')
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

xlrd - odczyt danych (tylko dla plików .xls):

```python
import xlrd

# Otwieramy istniejący plik .xls do odczytu.
wb = xlrd.open_workbook('przyklad.xls')

# Wybieramy pierwszy arkusz z pliku.
sheet = wb.sheet_by_index(0)

# Pobieramy dane z konkretnej komórki (wiersz 0, kolumna 0).
cell_value = sheet.cell_value(0, 0)
print(f'Zawartość komórki A1: {cell_value}')

# Możemy iterować przez wszystkie wiersze i kolumny.
for row in range(sheet.nrows):
    for col in range(sheet.ncols):
        print(sheet.cell_value(row, col))
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Modyfikowanie danych z pomocą openpyxl oraz xlwt/xlrd:

Modyfikacja danych w plikach Excel zależy od używanej biblioteki oraz formatu pliku. Należy zwrócić uwagę że, można to zrobić w openpyxl (dla plików .xlsx) i w xlwt/xlrd (dla plików .xls).

Biblioteka xlsxwriter nie obsługuje bezpośredniej modyfikacji istniejących plików, dlatego nie będziemy jej tutaj używać.

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Modyfikowanie danych z pomocą openpyxl:

openpyxl obsługuje zarówno odczyt, jak i modyfikację plików Excel w formacie .xlsx. Oto jak to zrobić:

Modyfikowanie istniejących danych:

- Otwieramy plik .xlsx.
- Wybieramy arkusz, który chcemy zmodyfikować.
- Odczytujemy, modyfikujemy, a następnie zapisujemy zmienione dane.

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Modyfikowanie danych z pomocą openpyxl:

```python
# Modyfikacja danych za pomocą openpyxl (pliki .xlsx)

from openpyxl import load_workbook

# Otwieramy istniejący plik Excel (.xlsx)
wb = load_workbook('przyklad.xlsx')

# Wybieramy arkusz, który chcemy zmodyfikować
ws = wb.active

# Modyfikujemy dane w komórkach. np. zmiana wartości w komórce B2
ws['B2'] = 7000  # Zmieniamy wynagrodzenie dla "Anna"

# Dodajemy więcej danych
ws['A10'] = 'Karol'  # Nowe imię w komórce A10
ws['B10'] = 8000  # Nowe wynagrodzenie w komórce B10

# Zapisujemy zmiany w tym samym pliku
wb.save('przyklad.xlsx')
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Modyfikowanie danych z pomocą openpyxl:

Modyfikowanie danych w pętli:

- Możemy przejść przez wiersze i zmienić wartości w komórkach w pętli.

```
for row in range(2, ws.max_row + 1):  # Przechodzimy przez wszystkie wiersze od 2 do ostatniego
    current_salary = ws[f'B{row}'].value  # Odczytujemy bieżącą wartość wynagrodzenia
    ws[f'B{row}'] = current_salary + 500  # Dodajemy 500 do każdego wynagrodzenia
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Modyfikacja danych w plikach .xls przy użyciu xlwt i xlrd

Pliki .xls są starszym formatem, a xlrd służy tylko do odczytu. Dlatego, jeśli chcemy modyfikować pliki .xls, musimy:

- Odczytać dane za pomocą xlrd.
- Zrobić kopię za pomocą xlwt (ponieważ xlwt służy tylko do zapisu).

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Odczyt i modyfikacja danych w plikach .xls (xlrd + xlwt):

```python
# Odczyt i modyfikacja danych w plikach .xls (xlrd + xlwt):

import xlrd
import xlwt
from xlutils.copy import copy  # Potrzebujemy tej funkcji, aby skopiować skoroszyt

# Otwieramy istniejący plik .xls do odczytu
workbook = xlrd.open_workbook('przyklad.xls')

# Otwieramy pierwszy arkusz
sheet = workbook.sheet_by_index(0)

# Kopiujemy skoroszyt, aby go modyfikować
wb_copy = copy(workbook)

# Wybieramy arkusz, który chcemy modyfikować
ws_copy = wb_copy.get_sheet(0)

# Modyfikujemy komórkę B2 (zmieniamy wynagrodzenie)
ws_copy.write(1, 1, 7000)  # Zmieniamy wynagrodzenie w wierszu 2, kolumna 2 (B2)

# Dodajemy nowe dane
ws_copy.write(9, 0, 'Karol')  # Nowe imię w wierszu 10, kolumna 1 (A10)
ws_copy.write(9, 1, 8000)  # Nowe wynagrodzenie w wierszu 10, kolumna 2 (B10)

# Zapisujemy plik jako nową kopię
wb_copy.save('przyklad_modyfikowany.xls')
```

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Podsumowanie:

W przypadku plików .xls, każda modyfikacja wymaga skopiowania pliku przy pomocy xlwt i zapisania go jako nowy plik, ponieważ xlrd nie obsługuje bezpośredniej modyfikacji plików.

Podsumowanie:

- **openpyxl** (pliki .xlsx): Możemy bezpośrednio modyfikować dane w istniejącym pliku Excel.
- **xlrd** + **xlwt** (pliki .xls): Modyfikacja wymaga skopiowania pliku i zapisania zmian w nowym pliku.

openpyxl jest bardziej wszechstronną i nowoczesną biblioteką, zwłaszcza jeśli pracujemy z plikami w formacie .xlsx.

Praca z pakietami openpyxl, xlsxwriter, xlrd i xlwt

Podsumowanie:

- **openpyxl**: Najbardziej wszechstronna biblioteka do odczytu i zapisu plików .xlsx. Idealna do pracy z większymi i nowszymi plikami Excel.
- **xlsxwriter**: Świetna do generowania i formatowania nowych plików .xlsx, szczególnie dla złożonych formatów i wykresów.
- **xlrd**: Używana głównie do odczytu starych plików .xls, ale już nie obsługuje formatu .xlsx.
- **xlwt**: Używana do zapisu starych plików .xls, ale ograniczona funkcjonalnością w porównaniu do nowszych bibliotek.

Każda z tych bibliotek ma swoje specyficzne zastosowanie, a wybór zależy od formatu pliku oraz poziomu złożoności danych, z którymi chcesz pracować.

Python - Wybrane Elementy

Warsztat

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Ćwiczenie 5.

- Tworzenie pliku
- Wczytanie danych z istniejącego pliku Excel za pomocą pandas
- Tworzenie różnych typów wykresów
  - Wykres kolumnowy
  - Wykres liniowy
  - Wykres kołowy
  - Wykres obszarowy
  - Wykres scatter (punktowy)
  - Wykres radarowy
  - Wykres słupkowy poziomy

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do tworzenia pliku:

```
# Tworzenie pliku

import xlsxwriter

# Tworzenie nowego pliku Excel
workbook = xlsxwriter.Workbook('xlsxwriter_example_with_multiple_charts.xlsx')
worksheet = workbook.add_worksheet('DataSheet')

# Dodawanie nagłówków do arkusza
headers = ['ID', 'Name', 'Age', 'Department', 'Salary']
header_format = workbook.add_format({'bold': True, 'font_color': 'white', 'bg_color': 'blue'})
for col, header in enumerate(headers):
    worksheet.write(0, col, header, header_format)
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do tworzenia pliku:

```
# Dodawanie danych do arkusza
data = [
    [1, 'Alice', 30, 'HR', 50000],
    [2, 'Bob', 24, 'Engineering', 55000],
    [3, 'Charlie', 29, 'Marketing', 60000],
    [4, 'David', 35, 'Finance', 70000],
    [5, 'Eva', 28, 'IT', 65000],
    [6, 'Frank', 33, 'HR', 52000],
    [7, 'Grace', 26, 'Engineering', 58000],
    [8, 'Hank', 31, 'Marketing', 61000],
    [9, 'Ivy', 38, 'Finance', 72000],
    [10, 'Jack', 27, 'IT', 66000]
]

cell_format = workbook.add_format({'text_wrap': True, 'valign': 'top', 'border': 1})
number_format = workbook.add_format({'num_format': '0.00', 'border': 1})

for row, entry in enumerate(data, start=1):
    for col, value in enumerate(entry):
        if col == 4:  # Apply number format to Salary
            worksheet.write(row, col, value, number_format)
        else:
            worksheet.write(row, col, value, cell_format)

workbook.close()
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do wczytania danych:

```
# Wczytanie danych z istniejącego pliku Excel za pomocą pandas
df = pd.read_excel('xlsxwriter_example_with_multiple_charts.xlsx', sheet_name='DataSheet')
df
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu kolumnowego:

```python
# Wykres kolumnowy

from openpyxl import load_workbook
from openpyxl.chart import BarChart, Reference

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu kolumnowego
bar_chart = BarChart()
data = Reference(ws, min_col=5, min_row=1, max_row=11)
categories = Reference(ws, min_col=2, min_row=2, max_row=11)
bar_chart.add_data(data, titles_from_data=True)
bar_chart.set_categories(categories)
bar_chart.title = "Employee Salaries"
bar_chart.x_axis.title = "Employee"
bar_chart.y_axis.title = "Salary"

# Dodanie wykresu do arkusza
ws.add_chart(bar_chart, "G2")

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu liniowego:

```
# Wykres liniowy

from openpyxl.chart import LineChart

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu liniowego
line_chart = LineChart()
data = Reference(ws, min_col=5, min_row=1, max_row=11)
categories = Reference(ws, min_col=2, min_row=2, max_row=11)
line_chart.add_data(data, titles_from_data=True)
line_chart.set_categories(categories)
line_chart.title = "Employee Salaries Over Time"
line_chart.x_axis.title = "Employee"
line_chart.y_axis.title = "Salary"

# Dodanie wykresu do arkusza
ws.add_chart(line_chart, "Q2")

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu kołowego:

```
# Wykres kołowy

from openpyxl.chart import PieChart

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu kołowego
pie_chart = PieChart()
data = Reference(ws, min_col=5, min_row=1, max_row=11)
categories = Reference(ws, min_col=4, min_row=2, max_row=11)
pie_chart.add_data(data, titles_from_data=True)
pie_chart.set_categories(categories)
pie_chart.title = "Department Salary Distribution"

# Dodanie wykresu do arkusza
ws.add_chart(pie_chart, "G14")

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu obszarowego:

```
# Wykres obszarowy

from openpyxl.chart import AreaChart

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu obszarowego
area_chart = AreaChart()
data = Reference(ws, min_col=5, min_row=1, max_row=11)
categories = Reference(ws, min_col=2, min_row=2, max_row=11)
area_chart.add_data(data, titles_from_data=True)
area_chart.set_categories(categories)
area_chart.title = "Employee Salaries (Area Chart)"
area_chart.x_axis.title = "Employee"
area_chart.y_axis.title = "Salary"

# Dodanie wykresu do arkusza
ws.add_chart(area_chart, "Q14")

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu scatter:

```python
# Wykres scatter

from openpyxl.chart import ScatterChart, Series

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu scatter
scatter_chart = ScatterChart()
xvalues = Reference(ws, min_col=2, min_row=2, max_row=11)
yvalues = Reference(ws, min_col=5, min_row=2, max_row=11)
series = Series(yvalues, xvalues, title="Salary")
scatter_chart.series.append(series)
scatter_chart.title = "Employee Salaries (Scatter Chart)"
scatter_chart.x_axis.title = "Employee"
scatter_chart.y_axis.title = "Salary"

# Dodanie wykresu do arkusza
ws.add_chart(scatter_chart, "G30")

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu radarowego:

```
# Wykres radarowy

from openpyxl.chart import RadarChart

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu radarowego
radar_chart = RadarChart()
data = Reference(ws, min_col=5, min_row=1, max_row=11)
categories = Reference(ws, min_col=2, min_row=2, max_row=11)
radar_chart.add_data(data, titles_from_data=True)
radar_chart.set_categories(categories)
radar_chart.title = "Employee Salaries (Radar Chart)"

# Dodanie wykresu do arkusza
ws.add_chart(radar_chart, "Q30")

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zapisywanie danych w arkuszu, formatowanie i modyfikowanie wykresów

Rozwiązanie do stworzenia wykresu słupkowego poziomego:

```
# Wykres słupkowy poziomy

# Wczytanie istniejącego pliku Excel za pomocą openpyxl
wb = load_workbook('xlsxwriter_example_with_multiple_charts.xlsx')
ws = wb['DataSheet']

# Tworzenie wykresu słupkowego
bar_chart = BarChart()
bar_chart.type = "bar"  # Zmiana typu wykresu na 'bar'
data = Reference(ws, min_col=5, min_row=1, max_row=11)
categories = Reference(ws, min_col=2, min_row=2, max_row=11)
bar_chart.add_data(data, titles_from_data=True)
bar_chart.set_categories(categories)
bar_chart.title = "Employee Salaries (Bar Chart)"
bar_chart.x_axis.title = "Salary"
bar_chart.y_axis.title = "Employee"

# Dodanie wykresu do arkusza
ws.add_chart(bar_chart, "G46")

# Zmiana szerokości kolumn i wysokości wierszy w nowym arkuszu
bar_chart.width = 32  # szerokość w centymetrach
bar_chart.height = 10  # wysokość w centymetrach

# Zapisanie pliku
wb.save('xlsxwriter_example_with_multiple_charts.xlsx')
```

Zaawansowane operacje na danych

Ćwiczenie 6.

- Tworzenie danych wejściowych
  - Dwa pliki xlsx i jeden csv
- Wczytywanie danych z plików Excel
- Wczytywanie danych z pliku CSV
- Łączenie danych z różnych arkuszy i plików w jeden DataFrame
- Przekształcanie danych
- Grupowanie danych według działu i obliczanie miar statystycznych
- Filtrowanie danych według złożonych kryteriów
- Uzupełnianie brakujących danych
- Łączenie przekształconych danych w jeden DataFrame

Zaawansowane operacje na danych

Ćwiczenie 6 cd.

- Usuwanie duplikatów
- Sortowanie danych według wynagrodzenia
- Zapisanie przetworzonych danych do nowego pliku Excel
- Tworzenie wykresów
  - Dodanie wykresu słupkowego dla średnich wynagrodzeń
  - Dodanie wykresu liniowego dla wynagrodzeń
  - Dodanie wykresu kołowego dla rozkładu wynagrodzeń
  - Dodanie wykresu punktowego dla wynagrodzeń

Zaawansowane operacje na danych

Rozwiązanie do tworzenia danych wejściowych:

```python
import pandas as pd

# Dane do pliku Excel 1
data1_sheet1 = {
    'ID': [1, 2, 3, 4, 5],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Department': ['HR', 'Engineering', 'Marketing', 'Finance', 'IT'],
    'Salary': [50000, 55000, 60000, 70000, 65000]
}

data1_sheet2 = {
    'ID': [1, 2, 3, 4, 5],
    'Date': ['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-04', '2024-01-05'],
    'Hours Worked': [8, 7.5, 8, 6, 8]
}

df1_sheet1 = pd.DataFrame(data1_sheet1)
df1_sheet2 = pd.DataFrame(data1_sheet2)

with pd.ExcelWriter('data1.xlsx') as writer:
    df1_sheet1.to_excel(writer, sheet_name='Sheet1', index=False)
    df1_sheet2.to_excel(writer, sheet_name='Sheet2', index=False)
```

Zaawansowane operacje na danych

Rozwiązanie do tworzenia danych wejściowych cd:

```python
# Dane do pliku Excel 2
data2_sheet1 = {
    'ID': [6, 7, 8, 9, 10],
    'Name': ['Frank', 'Grace', 'Hank', 'Ivy', 'Jack'],
    'Department': ['HR', 'Engineering', 'Marketing', 'Finance', 'IT'],
    'Salary': [52000, 58000, 61000, 72000, 66000]
}

data2_sheet2 = {
    'ID': [6, 7, 8, 9, 10],
    'Date': ['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-04', '2024-01-05'],
    'Hours Worked': [8, 7, 8, 6.5, 8]
}

df2_sheet1 = pd.DataFrame(data2_sheet1)
df2_sheet2 = pd.DataFrame(data2_sheet2)

with pd.ExcelWriter('data2.xlsx') as writer:
    df2_sheet1.to_excel(writer, sheet_name='Sheet1', index=False)
    df2_sheet2.to_excel(writer, sheet_name='Sheet2', index=False)
```

Zaawansowane operacje na danych

Rozwiązanie do tworzenia danych wejściowych cd:

```
data_csv = {
    'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Project': ['Project A', 'Project B', 'Project C', 'Project D', 'Project E', 'Project F', 'Project G', 'Project H', 'Project I', 'Project J'],
    'Hours': [10, 12, 8, 15, 10, 14, 11, 9, 16, 13]
}

df_csv = pd.DataFrame(data_csv)
df_csv.to_csv('data.csv', index=False)
```

Zaawansowane operacje na danych

Rozwiązanie do wczytania danych:

```
import numpy as np
from openpyxl import load_workbook
from openpyxl.chart import BarChart, LineChart, PieChart, ScatterChart, Series, Reference
from openpyxl.chart.label import DataLabelList

# Wczytywanie danych z plików Excel
file1 = 'data1.xlsx'
file2 = 'data2.xlsx'

df1_sheet1 = pd.read_excel(file1, sheet_name='Sheet1')
df1_sheet2 = pd.read_excel(file1, sheet_name='Sheet2')
df2_sheet1 = pd.read_excel(file2, sheet_name='Sheet1')
df2_sheet2 = pd.read_excel(file2, sheet_name='Sheet2')
```

```
# Wczytywanie danych z pliku CSV
df_csv = pd.read_csv('data.csv')
```

Zaawansowane operacje na danych

Rozwiązanie do łączenia danych z różnych arkuszy i plików:

```
# łączenie danych z różnych arkuszy i plików w jeden DataFrame
df_employees = pd.concat([df1_sheet1, df2_sheet1], ignore_index=True)
df_hours = pd.concat([df1_sheet2, df2_sheet2], ignore_index=True)
```

Zaawansowane operacje na danych

Rozwiązanie do przekształcenia danych:

```
# Przekształcanie danych
pivot_hours = df_hours.pivot_table(index='ID', columns='Date', values='Hours Worked', aggfunc='sum', fill_value=0)
pivot_hours
```

Zaawansowane operacje na danych

Rozwiązanie do grupowania danych:

```
# Grupowanie danych według działu i obliczanie średniego wynagrodzenia, mediany i odchylenia standardowego
grouped_salary = df_employees.groupby('Department')['Salary'].agg(['mean', 'median', 'std']).reset_index()
grouped_salary
```

Zaawansowane operacje na danych

Rozwiązanie do filtrowania danych:

```
# Filtrowanie danych według złożonych kryteriów
filtered_employees = df_employees[(df_employees['Salary'] > 50000) & (df_employees['Department'] == 'Engineering')]
filtered_employees
```

Zaawansowane operacje na danych

Rozwiązanie do uzupełnienia brakujących danych:

```
# Uzupełnianie brakujących danych
df_employees.fillna({'Salary': df_employees['Salary'].mean()}, inplace=True)
```

Zaawansowane operacje na danych

Rozwiązanie do łączenia przekształconych danych w jeden DataFrame:

```
# łączenie przekształconych danych w jeden DataFrame
merged_df = pd.merge(df_employees, pivot_hours, on='ID', how='left')
merged_df
```

Zaawansowane operacje na danych

Rozwiązanie do usuwania duplikatów:

```
# Usuwanie duplikatów
merged_df = merged_df.drop_duplicates()
merged_df
```

Zaawansowane operacje na danych

Rozwiązanie do sortowania danych:

```
# Sortowanie danych według wynagrodzenia
sorted_df = merged_df.sort_values(by='Salary', ascending=False)
sorted_df
```

Zaawansowane operacje na danych

Rozwiązanie do zapisania danych do nowego pliku Excel:

```
# Zapisanie przetworzonych danych do nowego pliku Excel
with pd.ExcelWriter('processed_data.xlsx') as writer:
    sorted_df.to_excel(writer, sheet_name='Merged Data', index=False)
    grouped_salary.to_excel(writer, sheet_name='Grouped Salary', index=False)
    pivot_hours.to_excel(writer, sheet_name='Pivot Hours', index=True)
    filtered_employees.to_excel(writer, sheet_name='Filtered Employees', index=False)

processedDataFile = pd.read_excel('processed_data.xlsx')
processedDataFile
```

Zaawansowane operacje na danych

Rozwiązanie do Tworzenia wykresów, ładujemy workbook:

```
# Tworzenie wykresów
wb = load_workbook('processed_data.xlsx')
```

Zaawansowane operacje na danych

Rozwiązanie do Tworzenia wykresów, wykres słupkowy dla średnich wynagrodzeń:

```
# Dodanie wykresu słupkowego dla średnich wynagrodzeń
ws = wb['Grouped Salary']
bar_chart = BarChart()
data = Reference(ws, min_col=2, min_row=1, max_row=ws.max_row, max_col=2)
categories = Reference(ws, min_col=1, min_row=2, max_row=ws.max_row)
bar_chart.add_data(data, titles_from_data=True)
bar_chart.set_categories(categories)
bar_chart.title = "Average Salary by Department"
bar_chart.x_axis.title = "Department"
bar_chart.y_axis.title = "Average Salary"
ws.add_chart(bar_chart, "E5")
```

Zaawansowane operacje na danych

Rozwiązanie do Tworzenia wykresów, wykresu liniowego dla wynagrodzeń:

```
# Dodanie wykresu liniowego dla wynagrodzeń
ws = wb['Merged Data']
line_chart = LineChart()
data = Reference(ws, min_col=5, min_row=1, max_row=ws.max_row)
categories = Reference(ws, min_col=1, min_row=2, max_row=ws.max_row)
line_chart.add_data(data, titles_from_data=True)
line_chart.set_categories(categories)
line_chart.title = "Salaries Over Time"
line_chart.x_axis.title = "ID"
line_chart.y_axis.title = "Salary"
ws.add_chart(line_chart, "L5")
```

Zaawansowane operacje na danych

Rozwiązanie do Tworzenia wykresów, wykresu kołowego dla rozkładu wynagrodzeń:

```
# Dodanie wykresu kołowego dla rozkładu wynagrodzeń
ws = wb['Grouped Salary']
pie_chart = PieChart()
data = Reference(ws, min_col=2, min_row=2, max_row=ws.max_row)
categories = Reference(ws, min_col=1, min_row=2, max_row=ws.max_row)
pie_chart.add_data(data, titles_from_data=True)
pie_chart.set_categories(categories)
pie_chart.title = "Salary Distribution by Department"
ws.add_chart(pie_chart, "E20")
```

Zaawansowane operacje na danych

Rozwiązanie do Tworzenia wykresów, wykresu punktowego dla wynagrodzeń:

```
# Dodanie wykresu punktowego dla wynagrodzeń
scatter_chart = ScatterChart()
xvalues = Reference(ws, min_col=1, min_row=2, max_row=ws.max_row)
yvalues = Reference(ws, min_col=2, min_row=2, max_row=ws.max_row)
series = Series(yvalues, xvalues, title="Salaries")
scatter_chart.series.append(series)
scatter_chart.title = "Salaries Distribution"
scatter_chart.x_axis.title = "Department"
scatter_chart.y_axis.title = "Salary"
ws.add_chart(scatter_chart, "E35")
```

Ankieta

Adres strony:

https://www.erp.comarch.pl/Szkolenia/Ankiety/survey/MDKZNM

COMARCH

Ścieżka kształcenia

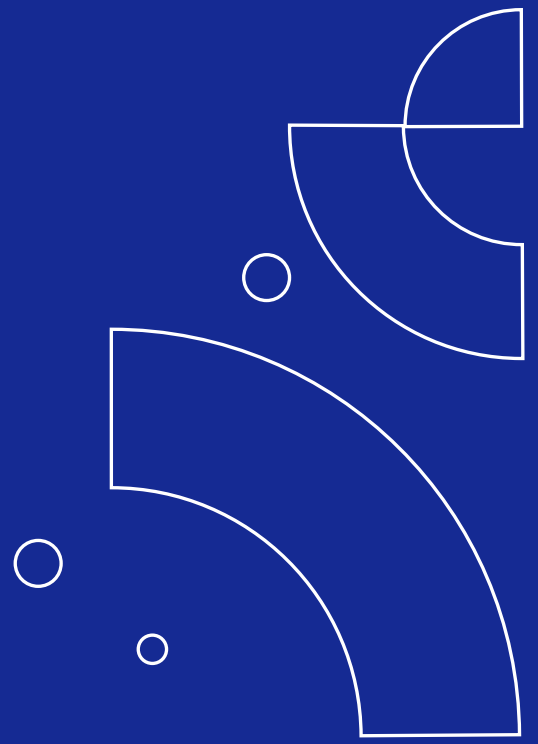
1. Wstęp do Machine Learning i Deep Learning w języku Python
   - https://www.comarch.pl/szkolenia/programowanie/python/wstep-do-machine-learning-i-deep-learning/
2. Wstęp do Machine Learning i Deep Learning w języku Python
   - https://www.comarch.pl/szkolenia/programowanie/python/wstep-do-machine-learning-i-deep-learning/
3. Machine Learning z użyciem języka Python. Zagadnienia zaawansowane
   - https://www.comarch.pl/szkolenia/programowanie/python/machine-learning-z-uzyciem-jezyka-python/

COMARCH
Szkolenia

Literatura przedmiotu

1. Źródła z wykorzystaniem których została stworzona niniejsza prezentacja:
   - https://www.kaggle.com/ - zbiory danych
   - AUTOMATYZACJA NUDNYCH ZADAŃ Z PYTHONEM: Nauka programowania, Sweigart Al.
   - PROGRAMOWANIE W PYTHONIE DLA ŚREDNIO ZAAWANSOWANY: Najlepsze praktyki tworzenia czystego kodu, Sweigart Al.
   - https://www.python.org/dev/
   - https://www.geeksforgeeks.org/python-programming-language/
   - https://www.w3schools.com/python/python_intro.asp

2. Dalsza ścieżka kształcenia
   - W głównej mierze można skupić się na darmowych rozwiązaniach aby pogłębiać wiedzę.
   - Sololearn – interesująca platforma do zdobywania wiedzy
   - Udemy
   - Coursera
   - Comarch



COMARCH

COMARCH
Szkolenia

Dziękujemy za udział w szkoleniu

Python w pracy z MS Excel.

Paweł Goleń

Trener

Centrum Szkoleniowe Comarch

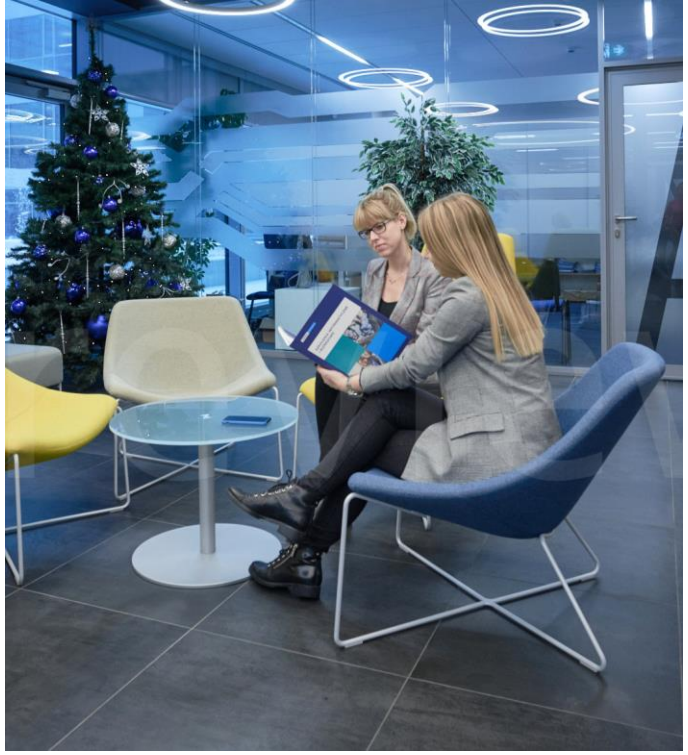
ul. Prof. M.Życzkowskiego 33

31-864 Kraków

Tel. +48 (12) 687 78 11

E-Mail: szkolenia@comarch.pl

www.szkolenia.comarch.pl



COMARCH
Szkolenia

COMARCH
Szkolenia

www.szkolenia.comarch.pl