Zadanie Nr1

Preparacja danych w bibliotece Pandas

Dane pozyskiwane z różnorodnych obserwacji naszego niedoskonałego świata, zbierane, gromadzone i przetwarzane, by wydzierać mu jego tajemnice, rzadko kiedy mają postać nadającą się wprost do przetwarzania: zwykle mają format niezbyt wygodny do gromadzenia, wyszukiwania i obliczeń, bywają często niekompletne, a często i błędne ze względu na niewłaściwe lub niestaranne przeprowadzanie eksperymentów pomiarowych. Jak pokazuje doświadczenie, w procesie mającym na celu zrobienie rzeczywistego użytku z zebranych danych, nawet 75% czasu zajmuje przygotowanie tych danych do przetworzenia, przed właściwym ich przetworzeniem.

Takie przygotowanie danych do analizy nazywane jest z angielska **data munging** lub **data wrangling** — będziemy traktować te pojęcia jak synonimy.

Dwa najważniejsze kroki opisanej preparacji danych to ich **czyszczenie** (ang. *data cleaning*) i **przekształcanie** (ang. *data transforming*) do formatu optymalnego z perspektywy ich przechowywania w bazach danych, a także wykorzystywanie przez różne pakiety statystyczne.

Na czyszczenie danych składają się najczęściej:

- usuwanie niekompletnych danych pomiarowych,
- zastępowanie brakujących danych rozsądnymi substytutami,
- usuwanie błędnych danych pomiarowych,
- zastępowanie błędnych danych rozsądnymi substytutami,
- usuwanie danych nadmiernie odbiegających od średniej lub trendu (jeśli jest ono uzasadnione),
- eliminacja duplikatów (gdy duplikaty nie są dopuszczalne),
- specjalne postępowanie z danymi niespójnymi.

Czyszczenie danych jest procesem nie tylko żmudnym i skomplikowanym, lecz również wysoce odpowiedzialnym: błędne decyzje na tym etapie mogą nie tylko prowadzić do wyników obarczonych zbyt dużym błędem, ale w skrajnych wypadkach mogą uczynić wątpliwym cały sens eksperymentu i zniweczyć poniesione w związku z nim inwestycje. Zagrożenie to jest tym większe, iż dane Data Science są przeważnie danymi **empirycznymi**, różniącymi się co do natury od danych **teoretycznych**: zliczanie statystyki występowania poszczególnych cyfr w długich rozwinięciach dziesiętnych liczby π jest jakościowo czymś innym niż ocena skuteczności terapeutycznej nowego leku w zależności od sposobu jego dawkowania, na podstawie systematycznych pomiarów parametrów życiowych wykonywanych na grupie pacjentów-ochotników.

Czyszczenie danych nie jest bynajmniej procesem rutynowym, lecz specyficznym dla danego eksperymentu i próbki zebranych danych.

Drugi aspekt preparacji danych eksperymentalnych — ich transformacja — obejmuje najczęściej:

■ eliminację cech nieistotnych z punktu widzenia eksperymentu (cechami zajmować się będziemy w dalszej części wykładów),
■ łączenie cech zależnych od siebie,
■ próbkowanie danych (ang. *data sampling*) w celu otrzymania ich reprezentatywnego podzbioru (w dalszej części wyjaśnimy, dlaczego najbardziej efektywne jest *próbkowanie losowe* — ang. *random sampling*);
■ standaryzacja formatu danych,
■ grupowanie danych.

Ponieważ preparacja danych jest zabiegiem w dużych stopniu heurystycznym, a więc podatnym na błędy, jej rezultat może okazać się chybiony; rozsądnie jest zatem zachowywać oryginalne dane, by w razie potrzeby móc preparację tę powtarzać — być może wielokrotnie — aż do osiągnięcia zadowalających rezultatów. W kolejnych sekcjach opiszemy przykłady preparacji danych zapisanych w postaci szeregu i ramki danych biblioteki *Pandas*.

Czyszczenie danych

Niepoprawne i brakujące dane mogą mieć duży wpływ na proces analizy danych. Wielu specjalistów-analityków sprzeciwia się stanowczo zastępowaniu takich danych „rozsądnymi wartościami", proponując w zamian specjalne ich oznaczanie i powierzanie pakietom analitycznym rozwiązywania związanych z nimi problemów. Wielu innych analityków dopuszcza wspomnianą substytucję, jednocześnie jednak zalecają daleko posuniętą ostrożność.

Załóżmy, że w pewnym szpitalu cztery razy dziennie mierzy się pacjentom temperaturę, a dzienny zapis tego pomiaru zawiera imię i nazwisko pacjenta oraz cztery wartości typu *float*, na przykład

```
['Laura Filon', 37.0, 36.9, 37.1, 0.0]
```

Wartość 0.0 nie oznacza oczywiście, że pacjentowi przytrafiło się nieszczęście z powodu awarii ogrzewania, lecz wynika z braku wieczornego pomiaru (bo na przykład zepsuł się jedyny na oddziale termometr). Średnia wartość z trzech udokumentowanych pomiarów wynosi 37.0 i nie stanowi powodu do niepokoju, jednakże bezmyślne uśrednienie wszystkich czterech wartości wykazuje ewidentną hipotermię (27.75) i równie bezmyślnie zinterpretowana może stanowić impuls do podjęcia działań, które (nic niepodejrzewającemu) pacjentowi mogą poważnie zaszkodzić.

Czy wobec tego mamy prawo zastąpić brakujący pomiar średnią z trzech poprzednich (37.0) jako „rozsądną" wartością? A jeśli w czasie wykonywania nieudanego pomiaru pacjent miał wysoką gorączkę, na przykład w wyniku skonsumowania mało apetycznej kolacji? Lekarz dyżurny, który przybył w środku nocy wskutek skarg pacjenta na złe samopoczucie, uzna, że nie ma powodu do obaw, bo przecież wieczorem pacjent miał temperaturę 37.0. Takie sztuczne „polepszenie" stanu zdrowia pacjenta przez beztroską pielęgniarkę może się dla niego źle skończyć.

Weryfikacja poprawności danych

Rozpocznijmy od skonstruowania słownika, którego kluczami są nazwy miast, w których pewna firma posiada oddziały, a wartościami kody pocztowe lokalizacji firmy w tych miastach. Na podstawie tegoż słownika skonstruujemy następnie szereg (*Series*) biblioteki *Pandas*.

```
In [1]: import pandas as pd

In [2]: kody pocztowe = pd.Series({'Bytom': '41-900', 'Gliwice': '4410'})

In [3]: kody pocztowe
Out[3]:
Bytom      41-900
Gliwice      4410
dtype: object
```

Chociaż obiekt *kody_pocztowe* przypomina wyglądem tablicę dwuwymiarową, w rzeczywistości jest indeksowaną tablica jednowymiarową: indeksami są nazwy miast (klucze oryginalnego słownika), a wartościami — wartości ze słownika odpowiadające tym kluczom.

Zastosujmy wyrażenie regularne do zweryfikowania poprawności (walidacji) opisanych danych.

Klasa Series udostępnia atrybut *str* umożliwiający wykonywanie rozmaitych operacji łańcuchowych na elementach szeregu, na przykład wywołanie metody *match* sprawdzającej zgodność wartości elementu ze wzorcem danym w postaci wyrażenia regularnego.

```
In [4]: kody_pocztowe.str.match(r'\d{2}-\d{3}')
Out[4]:
Bytom       True
Gliwice    False
dtype: bool
```

Metoda *match* wykonuje sprawdzenie dopasowania kolejno dla każdej wartości szeregu. Programista uwolniony jest od programowania iteracji po elementach szeregu, bo zadanie to bierze na siebie atrybut *str* organizujący we własnym zakresie iterowanie wewnętrzne. Rezultatem tego iterowania jest nowy szereg, którego indeksy są tożsame z indeksami szeregu oryginalnego, a wartością elementu jest **True**, gdy jego oryginalna wartość jest zgodna z podanym wyrażeniem regularnym, i **False** w przeciwnym razie. W naszym przykładzie kod pocztowy Gliwic ma ewidentnie błędną strukturę.

Z błędami w danych można radzić sobie na różne sposoby. Najbardziej oczywistym jest ponowne pozyskanie przedmiotowych danych u źródła — oczywistym, ale przeważnie niemożliwym.

Jeżeli dane pochodzą z szybkiego czujnika działającego w ramach internetu rzeczy, to nie da się cofnąć czasu, by ponownie zmierzyć parametry zjawiska,

które przeminęło. Nie da się wówczas zrekonstruować brakujących wartości i należy raczej zająć się czyszczeniem danych.
W przypadku kodów pocztowych sprawa jest prostsza; wystarczy sięgnąć do ich wykazu, a jeżeli nie mamy dostępu do takowego, można przeszukać inne zasoby zebranych danych w nadziei natrafienia na kody rozpoczynające się od 44-1 czy nawet 44-.
Niekiedy kryteria walidacji danych są mniej rygorystyczne, na przykład zamiast badać dopasowanie wartości do wzorca, można ją badać na okoliczność zawierania pewnego podłańcucha. Załóżmy, że każdej miejscowości przyporządkowano (oprócz kodu pocztowego) trzyliterowy skrót, na przykład tak:

```
[
 'Bytom, BTM 41-900',
 'Gliwice, centrala, GLC 44-100',
 'Bielsko Biała, BLB 43-300 (w budowie)'
]
```

Wartość uznamy za poprawną, jeśli zawiera w sobie podłańcuch dający się dopasować do wzorca postaci „trzy wielkie litery, ciąg białych znaków, dwie cyfry, myślnik, trzy cyfry", czyli

```
'[A-Z]{3}[\s]+\d{2}-\d{3}'
```

Jako że badamy wartości szeregu nie pod kątem dopasowania, ale *zawierania dopasowania*, zamiast metody *match* musimy użyć metody *contains*.

```
In [5]: miasta = pd.Series(
   ...:   [
   ...:    'Bytom, BTM 41-900',
   ...:    'Gliwice, centrala, GLC 44-100',
   ...:    'Bielsko Biała, BLB 43-300 (w budowie)'
   ...:   ]
   ...: )
   ...:

In [6]: miasta
Out[6]:
0                        Bytom, BTM 41-900
1            Gliwice, centrala, GLC 44-100
2    Bielsko Biała, BLB 43-300 (w budowie)
dtype: object

In [7]: wzorzec = '[A-Z]{3}[\s]+\d{2}-\d{3}'
```

```
In [8]: miasta.str.contains(wzorzec)
Out[8]:
0    True
1    True
2    True
dtype: bool

In [9]: miasta.str.match(wzorzec)
Out[9]:
0    False
1    False
2    False
dtype: bool
```

Ponieważ tworzyliśmy szereg na bazie listy, nie na bazie słownika, nie określiliśmy indeksów elementów, więc są one indeksowane kolejnymi liczbami całkowitymi począwszy od zera. I ponieważ struktura wartości może być dowolna poza fragmentem opisanym przez wzorzec, nie jest możliwe weryfikowanie zawartości szeregu za pomocą metody *match*.

Reformatowanie danych

Często zdarza się tak, że nienagannej treści towarzyszy niewłaściwa forma — dane pochodzące z aparatury pomiarowej lub z lektury dokumentów źródłowych mają format różny od tego, w jakim mają być magazynowane lub przekazywane aplikacjom do przetwarzania. W procesie preparacji danych nie musimy ich czyścić, za to staje się konieczne ich **przeformatowanie**.

Jest ono znacznie ułatwione, gdy wykonujemy je za pomocą połączenia odpowiednich struktur danych, funkcyjnego stylu programowania oraz wyrażeń regularnych.

W charakterze prostego przykładu rozpatrzmy zbiór numerów telefonicznych w USA, które zwyczajowo zapisywane są w formacie ###-###-#### (# oznacza cyfrę dziesiętną), lecz we wspomnianym zbiorze dla oszczędności miejsca są one zapisane jako ciągi 10 cyfr bez separatorów. Rozpocznijmy od zapisania zawartości wspomnianego zbioru w formie ramki danych biblioteki *Pandas*.

```
In [10]: kontakty = [
    ...:             ['Mike Green', 'demo1@deitel.com', '5555555555'],
    ...:             ['Sue Brown', 'demo2@deitel.com', '5555551234']
    ...:            ]
    ...:

In [11]: kontakty df = pd.DataFrame(kontakty,
    ...:                            columns=['Nazwisko', 'E-mail', 'Telefon'])
    ...:

In [12]: kontakty df
Out[12]:
     Nazwisko            E-mail     Telefon
0  Mike Green  demo1@deitel.com  5555555555
1   Sue Brown  demo2@deitel.com  5555551234
```

Ponieważ nie zdefiniowaliśmy indeksów wierszy ramki, są one numerowane począwszy od zera. Zgodnie z przyjętym standardem ramki danych, zawartość jej kolumn wyrównywana jest *prawostronnie*, *niezależnie od typu*; jest to konwencja odmienna od standardów Pythona, zgodnie z którymi wartości *nienumeryczne* wyrównywane są *lewostronnie*.

Konwersję formatu danych z kolumny Telefon przeprowadzimy za pomocą mapowania (które jest jednym z paradygmatów funkcyjnego stylu programowania). Argumentem tego mapowania jest funkcja sformatuj_nr_telefonu, dokonująca formatowania pojedynczej wartości,

```
In [13]: import re

In [14]: def sformatuj nr telefonu(numer):
    ...:     result = re.fullmatch(r'(\d{3})(\d{3})(\d{4})', numer)
    ...:     return '-'.join(result.groups()) if result else numer
    ...:
```

Funkcja ta rozpoczyna swe działanie od sprawdzenia, czy wartość źródłowa istotnie jest ciągiem 10 cyfr dziesiętnych: jeżeli tak, to funkcja *fullmatch* zwraca obiekt *Match*, którego metoda *groups* zwraca wynik podziału owego ciągu na trzy grupy cyfr; grupy te, po połączeniu za pomocą myślnika (metoda *join*), dadzą żądany format wynikowy.

Jeśli funkcja *fullmatch* zwróci wartość *None*, będzie to oznaczać, że wartość źródłowa nie jest ciągiem 10-cyfrowym; jej obróbka przekracza kompetencje funkcji sformatuj_nr_telefonu, i ta zwraca ją w niezmienionej postaci. Opisywane formatowanie jest więc procedurą powtarzalną („idempotentną”) — próba formatowania numeru już sformatowanego nie powoduje żadnych skutków.

Każda kolumna ramki danych jest szeregiem, reprezentowanym przez klasę *Series*; metoda *map* tej klasy zwraca nowy szereg, którego elementy stanowią

rezultat mapowania. W naszym przykładzie szeregiem źródłowym jest kolumna Telefon ramki danych.

```
In [15]: nr_telefonu_sformatowany = kontakty_df['Telefon'].map(sformatuj_nr_telefonu)

In [16]: nr_telefonu_sformatowany
Out[16]:
0    555-555-5555
1    555-555-1234
Name: Telefon, dtype: object
```

Nowy szereg wynikowy, którego referencję zawiera zmienna nr_telefonu_sformatowany, wystarczy już tylko podstawić w miejsce oryginalnej kolumny Telefon.

```
In [17]: kontakty_df['Telefon'] = nr_telefonu_sformatowany

In [18]: kontakty df
Out[18]:
     Nazwisko            E-mail       Telefon
0  Mike Green  demo1@deitel.com  555-555-5555
1   Sue Brown  demo2@deitel.com  555-555-1234
```

ZADANIE

Zestawy danych

https://data.gov/ to portal otwartych danych rządu Stanów Zjednoczonych.

https://www.ncdc.noaa.gov/cag/ - portal klimatyczny NOAA

https://www.esrl.noaa.gov/psd/data/timeseries/ - Portal badań Ziemi administracja laboratoriów (ESRL, Earth System Research Laboratory) NOAA dostarcza miesięczne i sezonowe z danymi klimatycznymi.

https://www.quandl.com/search - Quandl udostępnia setki bezpłatnych szeregów czasowych z danymi finansowymi, a także zestawy danych z płatnym dostępem.

https://datamarket.com/data/list/?q=provider:tsdl - biblioteka danych TSDL (biblioteka danych szeregów czasowych) zawiera linki do setek tymczasowych zbiorów danych rzędy w wielu obszarach przemysłowych.

http://archive.ics.uci.edu/ml/datasets.html - repozytorium uczenia maszynowego Uniwersytetu Kalifornijskiego w Irvine (UCI) zawiera dziesiątki zestawów danych szeregów czasowych z różnych obszary.

https://dane.gov.pl/pl - polskie dane statystyczne

1. Pobrać dane z dowolnego z podanych źródeł.

Zgodnie ze wzorcem zrób:

2. Czyszczenie danych

3. Weryfikację poprawności danych

4. Reformatowanie danych

5. W MS Teams (zakładka Class Notebook, zakładka odpowiadająca Twojemu Imieniu, zakładka Homework) utwórz folder 27.10.2023_Zadanie1. Wklej plik danych, plik kodu, raport (.docx lub .doc) w którym pokaż działanie kodu programu i wyjaśnij swoje działania.