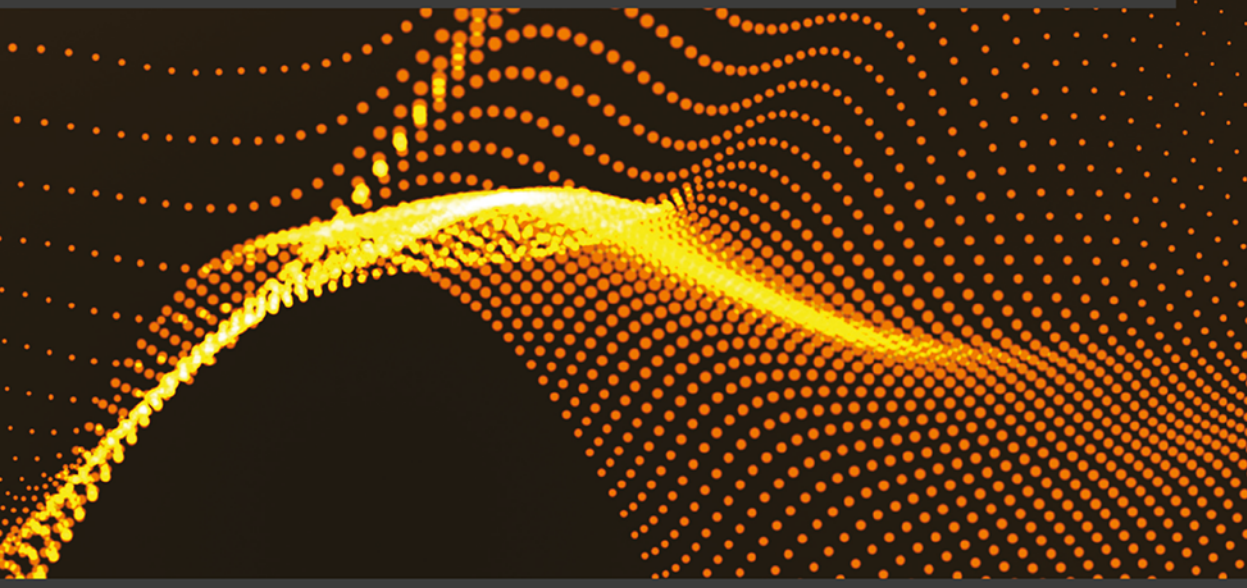
Dodaj mocy **Power BI**!

Jak za pomocą kodu w **Pythonie** i **R** pobierać, przekształcać i wizualizować dane



Luca Zavarella
Przedmowa Francesca Lazzeri

Helion

Tytuł oryginału: Extending Power BI with Python and R Ingest, transform, enrich, and visualize data using the power of analytical languages

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-9453-7

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/rozmoz
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/rozmoz.zip

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 5

Korzystanie z wyrażeń regularnych w usłudze Power BI

Wiele zadań czyszczenia danych obejmuje przeprowadzanie złożonych wyszukiwań i podstawień między ciągami znaków. Aby uzyskać pożądane wyniki, czasami nie wystarczą zwykłe narzędzia wyszukiwania i zastępowania. Załóżmy, że musisz dopasować ciągi znaków nie w sposób dokładny (na przykład z wykorzystaniem warunków równości), ale przy użyciu kryteriów podobieństwa między nimi. W projektach wymagających wysokiej jakości danych umiejętność korzystania z narzędzi takich jak wyrażenia regularne (znanych również jako regeksy) może mieć ogromne znaczenie. Dzięki językom R i Python możesz dodać te narzędzia do swojego arsenału.

W tym rozdziale omówię następujące tematy:

- Krótkie wprowadzenie do regeksów.
- Sprawdzanie poprawności danych przy użyciu regeksów w Power BI.
- Ładowanie złożonych plików logów przy użyciu regeksów w Power BI.
- Wyodrębnianie wartości z tekstu przy użyciu regeksów w Power BI.

Wymagania techniczne

Aby skutecznie studiować treść tego rozdziału, będziesz potrzebować połączenia z internetem i zainstalowanego na komputerze programu Power BI Desktop. Musisz poprawnie skonfigurować silniki języków R i Python oraz środowiska IDE, zgodnie z opisem w rozdziałach:

2., „Konfigurowanie języka R na potrzeby usługi Power BI”' i 3., „Konfigurowanie języka Python na potrzeby usługi Power BI”.

Krótkie wprowadzenie do regeksów

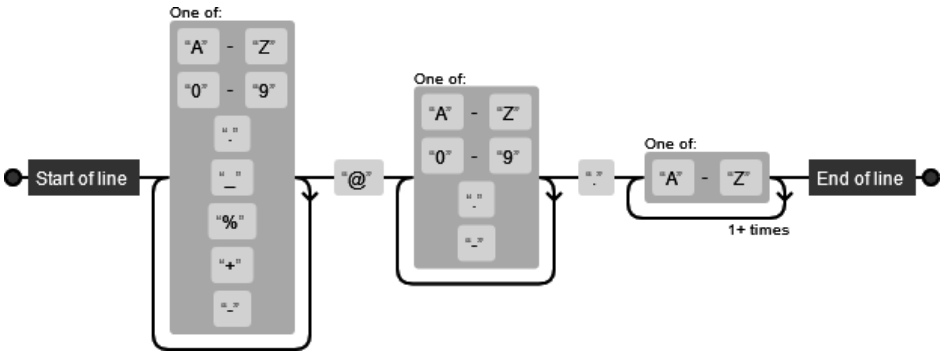
Wyrażenie regularne (zwykle określane skrótową nazwą **regex**) to ciąg znaków, które *identyfikują abstrakcyjny wzorzec wyszukiwania*. Ogólnie rzecz biorąc, jest to technika matematyczna, która została opracowana w 1951 roku przez ekspertów języka formalnego i informatyki teoretycznej. Służy do sprawdzania poprawności danych wejściowych lub wyszukiwania i wyodrębniania informacji z tekstów.

Jeśli nie znasz składni regeksów, na pierwszy rzut oka może ona sprawiać wrażenie bardzo złożonej (rysunek 5.1):

```
^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$
```

Rysunek 5.1. Przykład wzorca regeksa

Na szczęście istnieją internetowe narzędzia do wizualizacji regeksów, które ułatwiają zrozumienie tych wzorców (jeden z nich jest dostępny na stronie *https://regexper.com*). Na przykład regex wyróżniony na rysunku 5.1 można zwizualizować w następujący sposób (rysunek 5.2):



Rysunek 5.2. Wizualizacja regeksa

Na rysunku 5.2 widać, że regex z rysunku 5.1 pozwala zidentyfikować we fragmencie tekstu adresy e-mail.

Nauczenie się, jak korzystać z regeksów w sposób profesjonalny, z pewnością nie jest łatwe i nie jest celem tego podrozdziału. Wyjaśnię tutaj jedynie podstawowe zasady, które pozwolą Ci tworzyć proste, ale skuteczne wzorce wyszukiwania. Aby uzyskać więcej informacji, zapoznaj się z bibliografią na końcu tego rozdziału.

Podstawowe informacje o regeksach

Podstawowe zasady tworzenia regeksów postaramy się wyjaśnić za pomocą przykładów, co jest prawdopodobnie najbardziej bezpośrednim sposobem na rozpoczęcie ich używania. W każdym kolejnym podpunkcie wyjaśnię inną funkcję regeksów. Aby przetestować regex użyty w przykładzie, posłużymy się narzędziem dostępnym na stronie *https://www.regexpal.com/*. Zaczynamy!

Znaki literalne

Aby dołączyć jeden lub więcej znaków literalnych do regeksa, konieczne jest skorzystanie z funkcji „wyszukiwania". Spróbujmy wyszukać ciąg *ie* wewnątrz tekstu *Niech moc rozszerzenia Power BI za pomocą Pythona i R będzie z Tobą!*(rysunek 5.3):

Rysunek 5.3. Wyszukiwanie ciągu „ie" za pomocą regeksów

Zauważ, że narzędzie domyślnie używa flagi wyszukiwania globalnego. Konkretne flagi są oznaczane literami (w naszym przypadku `g`) zaraz za ogranicznikami regeksa, `/.../`. Oto lista dostępnych flag, z których możesz skorzystać:

- `g` (globalne): powoduje dopasowywanie wszystkich wystąpień z zachowaniem indeksu wystąpienia ostatniego.
- `m` (wielowierszowe — ang. *multiline*): gdy ta opcja jest włączona, kotwice ciągów (omówię je później) będą dopasowywane do początku i końca wiersza, a nie do całego ciągu.
- `i` (ignoruj wielkość liter): powoduje przeszukiwanie wzorca niezależnie od wielkości liter w ciągu (małych lub wielkich).

Należy zapamiętać, że nie we wszystkich językach programowania wykorzystywana jest wspomniana wcześniej składnia flag. Na przykład pakiet `re` dostępny dla Pythona (domyślny dla regeksów) pozwala na stosowanie parametrów w funkcjach `search`, `match` i `sub`:

```
re.search('test', 'TeSt', re.IGNORECASE)
re.match('test', 'TeSt', re.IGNORECASE)
re.sub('test', 'xxxx', 'TesTing', flags=re.IGNORECASE)
```

To samo dotyczy funkcji `regex()` z pakietu `stringr` języka R:

```
str_detect('tEsTuj to', regex('test', ignore_case=TRUE))
```

Można także korzystać z **modyfikatorów globalnych** bezpośrednio we wzorcu regeksa (w tzw. trybie *inline*). Są to modyfikatory: `(?i)` pozwalający określić ignorowanie wielkości liter i `(?m)` oznaczający ciągi wielowierszowe. Na przykład w języku R możesz uruchomić następujący skrypt:

```
str_detect('tEsTuj to', regex("(?i)test"))
```

Warto zwrócić uwagę, że Python nie pozwala na stosowanie modyfikatorów globalnych w trybie inline.

Znaki specjalne

W regeksach można korzystać z 12 znaków specjalnych (zwanych również **metaznakami**). Każdy z nich ma specjalne znaczenie. Są to znaki potoku (|); lewego ukośnika (\); dolara ($); znaku zapytania (?); karetki (^); gwiazdki (*); plusa (+); kropki (.); nawiasy zwykłe ((i)); otwierający nawias prostokątny ([) oraz otwierający nawias klamrowy ({).

Jeśli chcesz wyszukać jeden z wcześniej wymienionych znaków, musisz skorzystać z tzw. **znaku ucieczki** (ang. *escape character*), którym jest znak lewego ukośnika. Tak więc jeśli chcesz dokładnie dopasować ciąg `123$`, musisz użyć wzorca regex w postaci `123\$`.

Następnie zapoznamy się ze znaczeniem metaznaków.

Znaki zakotwiczenia ^ i $

Znaki zakotwiczenia mają specjalne znaczenie, ponieważ są używane do umieszczania dopasowania regeksa w określonej pozycji w ciągu. Znak karetki (^) służy do wskazania początku ciągu (lub wiersza), a znak dolara ($) służy do wskazania końca ciągu (lub wiersza). Przykładowa ilustracja jest warta tysiąca słów (rysunek 5.4):

Rysunek 5.4. Wyszukiwanie globalne bez uwzględniania wielkości liter

W przykładzie pokazanym na rysunku 5.4 została ustawiona flaga „ignoruj wielkość liter". Aby to zrobić, należy kliknąć ikonę *flags*, a następnie zaznaczyć opcję *ignore case*. Dzięki temu regex znajdzie oba wystąpienia dopasowania. Spróbujmy teraz dodać znak karetki (^) przed znakiem *n* (rysunek 5.5):

Rysunek 5.5. Wyszukiwanie globalne bez uwzględnienia wielkości liter oraz z zastosowaniem symbolu karetki (^)

W tym przypadku regex dopasuje tylko pierwsze wystąpienie (czyli to na początku ciągu).

Jeśli dodasz również znak dolara na końcu regeksa, to nie znajdzie on żadnego dopasowania. Byłoby to bowiem żądanie dopasowania ciągu *niech moc*, który jednocześnie rozpoczyna i kończy tekst.

Operatory OR

Czasami może zajść konieczność dopasowania jednego z wybranych zbiorów znaków lub ciągów. Na przykład, aby dopasować dowolny ze znaków *s* lub *t* za ciągiem *ko*, powinieneś użyć klasy znaków `[st]` wewnątrz regeksa `ko[st]`. Dzięki temu regex będzie pasował zarówno do ciągów *kos*, jak i *kot*. Klasy znaków mogą być również wykorzystywane do dopasowywania wystąpienia w zakresach znaków, które tworzymy za pomocą łącznika (`-`). Na przykład klasa `[0–9]` odpowiada pojedynczej cyfrze z zakresu od `0` do `9`, podczas gdy klasa `[A–Z]` dopasowuje pojedynczą wielką literę z zakresu od `A` do `Z`. Możesz także łączyć wiele zakresów w jedną klasę znaków. Na przykład klasa `[A-Z0-9]` pasuje do cyfry lub do wielkiej litery.

Aby za pomocą regeksa dopasować jeden z dwóch ciągów znaków, możesz użyć symbolu potoku (`|`), za pomocą którego oddzielisz ciągi umieszczone w nawiasach okrągłych, na przykład `(ciąg1|ciąg2)`. Oto przykład (rysunek 5.6):

Rysunek 5.6. Przykład zastosowania operatorów OR

Klasę znaków można również wykorzystać do dopasowania dowolnego znaku innego niż określony znak. Można to uzyskać dzięki zanegowaniu klasy znaków.

Zanegowane klasy znaków

Karetka występująca tuż za otwierającym nawiasem kwadratowym neguje zawartość klasy znaków. Na przykład regex `[^"]` pasuje do każdego znaku, który nie jest cudzysłowem.

Skróty klas znaków

Istnieją pewne klasy znaków, które są używane bardzo często. Z tego powodu zdefiniowano kilka skrótów pozwalających na ich szybkie włączenie do regeksów. Oto lista najczęściej używanych:

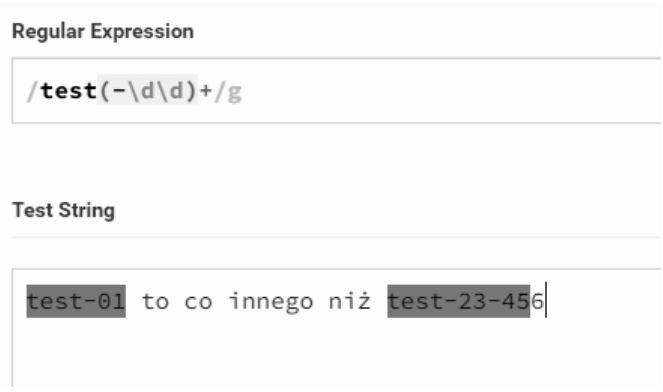
- `\w`: pasuje do wszystkich znaków alfanumerycznych oraz znaku podkreślenia.
- `\W`: przeciwieństwo klasy `\w`; pasuje do pojedynczego znaku niealfanumerycznego z wyłączeniem znaku podkreślenia. Na przykład pozwala dopasować spacje i znaki interpunkcyjne.
- `\d`: pasuje do pojedynczej cyfry.
- `\D`: przeciwieństwo klasy `\d`; pasuje do pojedynczego znaku niebędącego cyfrą.
- `\s`: pasuje do „znaków białych spacji", takich jak spacja, tabulator, znak podziału wiersza i powrót karetki.

W tym rozdziale będziemy dość często korzystać ze skrótów klas znaków.

Kwantyfikatory

Kwantyfikatory wskazują, ile razy musi zostać dopasowany *znak* lub *wyrażenie*. Oto lista najczęściej używanych kwantyfikatorów:

- `+`: pasuje do ciągu poprzedzającego kwantyfikator jeden lub więcej razy. Na przykład regex `test\d+` będzie pasować do ciągu *test*, za którym występuje jedna lub więcej cyfr. Regex `test(-\d\d)+` będzie z kolei pasował do ciągu *test*, po którym występuje jeden lub więcej razy myślnik, a za nim dwie cyfry (rysunek 5.7):



Rysunek 5.7. Powtarzanie grupy znaków za pomocą kwantyfikatora +

- `{n}`: pasuje do ciągu poprzedzającego ten symbol *n* razy. Na przykład regex `\d{4}` będzie pasować do dowolnej liczby całkowitej złożonej z 4 cyfr.
- `{n,m}`: pasuje do ciągu poprzedzającego ten kwantyfikator od *n* do *m* razy. Na przykład regex `prod-\d{2,6}` będzie pasować do ciągu *prod-*, za którym występuje liczba całkowita składająca się z od 2 do 6 cyfr.
- `{n,}`: pasuje do ciągu poprzedzającego ten kwantyfikator *n* lub więcej razy.
- `?`: pasuje do ciągu poprzedzającego ten kwantyfikator zero razy lub raz. Na przykład regex `Mar(zec)?` będzie pasować zarówno do ciągu *Marzec*, jak i *Mar*. Z kolei regex `pisa?nki` będzie pasował zarówno do ciągów *pisanki*, jak i *pisaki*.
- `*`: pasuje do ciągu poprzedzającego ten kwantyfikator zero lub więcej razy. Na przykład regex `kod\d*` będzie zgodny z ciągiem *kod*, *kod1*, a także *kod73846*.

Kropka

Kropka odpowiada pojedynczemu znakowi, niezależnie od tego, czym jest ten znak, z wyjątkiem znaków podziału wiersza. Jest to bardzo potężny metaznak stosowany w regeksach, który może nas rozleniwić. Właśnie dlatego trzeba uważać, aby go nie nadużywać, ponieważ czasami może to doprowadzić do uwzględnienia w dopasowaniach niezamierzonych wyników.

Dopasowania zachłanne i leniwe

Symbole `+`, `*` oraz powtórzenie symboli `{...}` to **kwantyfikatory zachłanne** (ang. *greedy quantifiers*). Zachłanność polega na tym, że kwantyfikatory te *skonsumują najdłuższy możliwy ciąg*. Załóżmy, że chcesz dopasować same tagi użyte w ciągu *Power BI wymiata*. Początkujący użytkownik regeksów mógłby użyć regeksa `<.+>`, który można wyrazić takimi oto słowami: „zdobądź <, następnie raz lub więcej razy uzyskaj dowolny znak niebędący znakiem podziału wiersza, a na koniec uzyskaj znak >”. Oczekiwany wynik to dwa dopasowania `<em>` i `</em>`. Rzućmy okiem na wynik (rysunek 5.8):

Rysunek 5.8. Chciwość symbolu .+

Jest oczywiste, że kombinacja `.+` obejmuje wszystko, co zawiera się między pierwszym wystąpieniem znaku `<` a ostatnim wystąpieniem znaku `>`, stąd bierze się definicja **chciwości** kwantyfikatorów.

Czy można więc zmusić chciwy kwantyfikator do zatrzymania się przy pierwszym wykrytym wystąpieniu następnego znaku, uniemożliwiając mu „zjedzenie" czegokolwiek aż do ostatniego wystąpienia tego samego znaku? Innymi słowy, czy można zamienić chciwy kwantyfikator na **leniwy**? Odpowiedź brzmi „tak", można to zrobić, dodając metaznak `?` bezpośrednio za znakiem `+`. Tak więc regex `<.+>` należy przekształcić na `<.+?>`. Oto wynik (rysunek 5.9):

Rysunek 5.9. Przekształcenie chciwego kwantyfikatora w leniwy dzięki metaznakowi ?

Należy jednak pamiętać, że *leniwe kwantyfikatory są niewydajne*. O ile to możliwe, zawsze lepiej jest używać zanegowanych klas znaków. W naszym przykładzie wystarczy użyć regeksa `<[^>]+>` (czyli znak `<`, dowolny znak inny niż `>` jeden lub więcej razy oraz znak `>`). Taki regex pozwala osiągnąć ten sam wynik, a jednocześnie zużyje mniej zasobów obliczeniowych (rysunek 5.10):

Rysunek 5.10. Wykorzystanie zanegowanej klasy znaków zamiast leniwego kwantyfikatora

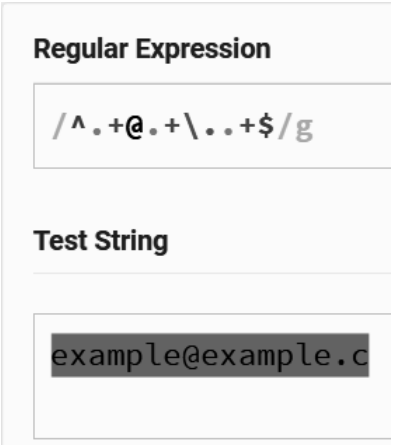
Dzięki temu, czego dotychczas nauczyłeś się o regeksach, masz minimalne podstawy wymagane do zrozumienia bardziej złożonych regeksów, których użyjemy w następnych kilku przykładach.

Sprawdzanie poprawności adresów e-mail

Gdyby ktoś Cię poprosił o zweryfikowanie adresu e-mail przy użyciu pojęć poznanych przed chwilą, jedna z Twoich pierwszych prób mogłaby wyglądać następująco: `^.+@.+\.. +$`. Oto próba przetłumaczenia tego regeksa na język mówiony:

1. `^`: pasuje do początku ciągu lub wiersza, jeśli jest włączona flaga wielowierszowa.
2. `.+`: pasuje do dowolnego znaku, z wyjątkiem znaku podziału wiersza, występującego jeden lub więcej razy.
3. `@`: pasuje do znaku `@`.
4. `.+`: pasuje do dowolnego znaku, z wyjątkiem znaku podziału wiersza, występującego jeden lub więcej razy.
5. `\.`: Pasuje do znaku `.`.
6. `.+`: pasuje do dowolnego znaku, z wyjątkiem znaku podziału wiersza, występującego jeden lub więcej razy.

Oczywiście ten regex potwierdzi poprawność adresu e-mail. Ale czy na pewno pozwoli wykryć oczywiste błędy składniowe nieprawidłowych adresów e-mail? Spróbuj przeprowadzić test na stronie *https://www.regexpal.com/* z niewłaściwym adresem e-mail *example@example.c* (domena najwyższego poziomu, czyli część domeny po kropce, musi zawierać co najmniej dwa znaki) (rysunek 5.11):



Rysunek 5.11. Wykorzystanie prostego regeksa do sprawdzenia poprawności nieprawidłowego adresu e-mail

Cóż, nie tego się spodziewaliśmy: ewidentnie błędny e-mail przeszedłby jako poprawny. Z tego powodu często konieczne jest stosowanie bardziej złożonych regeksów, które są zgodne z dobrze zdefiniowanymi regułami składniowymi.

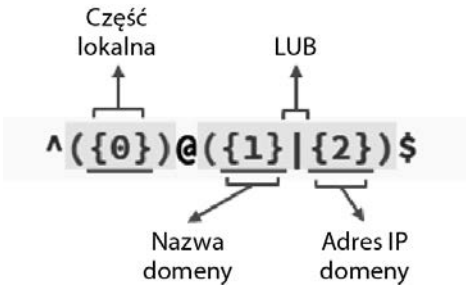
W tym przypadku użyjemy do sprawdzenia poprawności adresu e-mail konkretnego regeksa, który często stosujemy w środowisku produkcyjnym. Ten regex sprawdza również, czy używany jest adres IP domeny. Oto regex do sprawdzania poprawności adresów e-mail w pełnej postaci:

```
^(([^<>()[\]\\.,;:\s@\""]+(\.[^<>()[\]\\.,;:\s@\""]+)*)|(\"".+\""))@((\[?[0-9]
{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\]?)|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$
```

Na pierwszy rzut oka ten regex z pewnością wprowadza w zakłopotanie. Jeśli jednak spróbujemy wydzielić z niego istotne części, staje się znacznie bardziej czytelny.

Jeśli jednak uważasz, że ten regex jest zbyt złożony, spójrz na ten, który uwzględnia wszystkie zasady składniowe opisane w dokumencie *Standard for the Format of Arpa Internet Text Messages*. Można go znaleźć pod adresem *http://www.ex-parrot.com/pdw/Mail-RFC822-Address.html*! Czyż nie robi wrażenia?

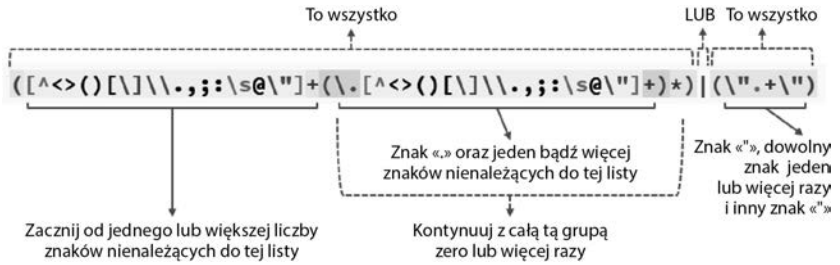
Format adresu e-mail można zdefiniować jako *część-lokalna@domena*. Pełną specyfikację można znaleźć na Wikipedii, pod adresem *https://en.wikipedia.org/wiki/Email_address*. Spróbujmy dopasować minimalną liczbę reguł (nie wszystkie!), które pozwolą zweryfikować znaczącą liczbę różnych adresów e-mail. Tak więc, biorąc pod uwagę, że zamierzamy dopasować nazwę domeny lub adres IP domeny, ogólna struktura regeksa jest następująca (rysunek 5.12):



Rysunek 5.12. Struktura złożonego regeksa do sprawdzania poprawności adresów e-mail

Ciągi {0}, {1} i {2} widoczne na rysunku 5.11 są jedynie symbolami zastępczymi, a nie faktycznymi fragmentami regeksa do dopasowania. Zacznijmy od zdefiniowania każdego z wykorzystanych tokenów:

1. Token {0} pasuje do regeksa dopasowującego **lokalną część** adresu e-mail. W tym przypadku o wiele łatwiej jest wyjaśnić, co robi ten regex za pomocą rysunku, a nie słowami. Wszystkie szczegóły elementów składowych tego regeksa zostały wyjaśnione na rysunku 5.13:

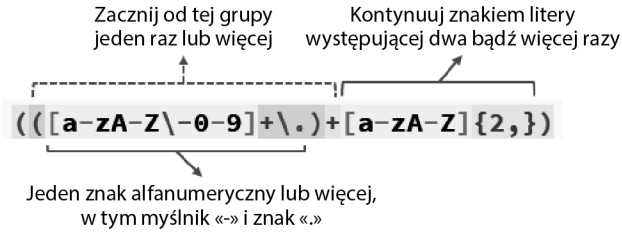


Rysunek 5.13. Szczegółowe wyjaśnienie regeksa części lokalnej

Zapamiętaj, że *nawiasy zwykłe grupują poszczególne człony wzorca regeksa*. Nawiasy umożliwiają zastosowanie operatorów regeksów do całego wyrażenia grupy i uzyskanie części dopasowania jako oddzielnego elementu w tablicy wyników. Tekst odpowiadający wyrażeniu regeksa jest w nich przechwytywany jako **numerowana grupa** i może być ponownie użyty wraz z odwołaniem liczbowym. Sposób korzystania z tej własności omówię później.

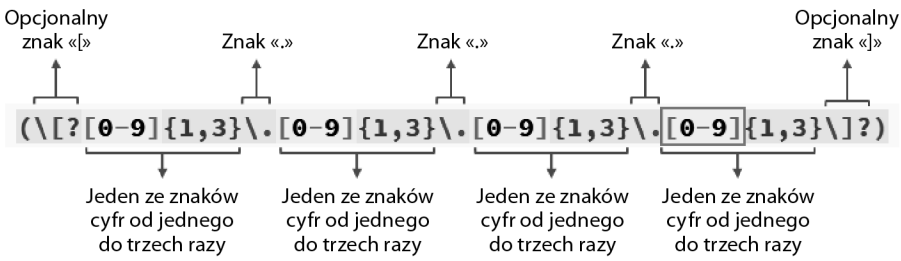
Warto zapamiętać, że aby dopasowanie metaznaku rzeczywiście reprezentowało metaznak, trzeba przed tym metaznakiem umieścić symbol ucieczki w postaci lewego ukośnika. Tak więc na przykład sekwencja \] oznacza znak].

2. Token {1} dopasowuje nazwę domeny adresu e-mail. Aby wyjaśnić, co robi regex nazwy domeny, ponownie użyję rysunku (rysunek 5.14):



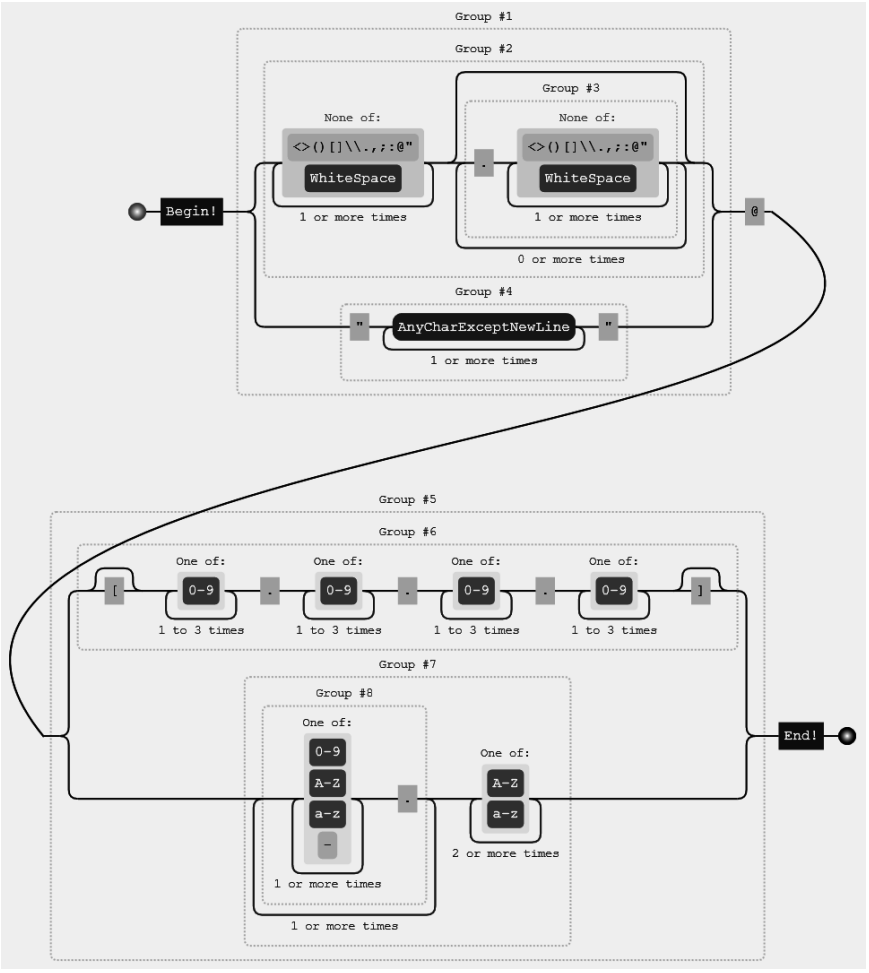
Rysunek 5.14. Szczegółowe wyjaśnienie regeksa nazwy domeny

3. Token {2} odpowiada adresowi IP domeny adresu e-mail. Jest to łatwiejszy podregex. Jego szczegółowe wyjaśnienie zamieściłem na rysunku 5.15:



Rysunek 5.15. Szczegółowe wyjaśnienie regeksa adresu IP domeny

Wizualizację całego regeksa możesz zobaczyć na poniższym rysunku pochodzącym ze strony *https://jex.im/regulex* (rysunek 5.16):



Rysunek 5.16. Wizualizacja całego regeksa dopasowującego adresy e-mail

Przejdźmy teraz do sprawdzania poprawności innego ważnego rodzaju informacji, podczas wpisywania których bardzo często dochodzi do błędów dat.

Sprawdzanie poprawności dat

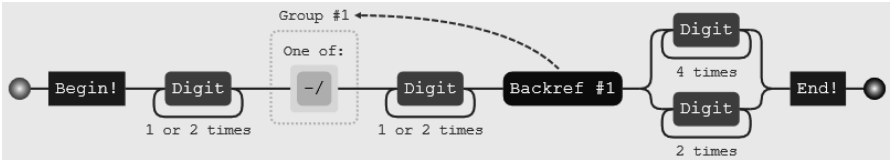
W przypadku dat (w formacie `dd-mm-rrrr`) nieświadomy twórca może pomyśleć, że do weryfikacji ich poprawności wystarczy następujący regex: `^\d{1,2} ([\-\/])\d{1,2}\1(?:\ d{4}|\d{2})$`. W tym wyrażeniu występują dwa nowe wyrażenia, których dotąd nie spotkaliśmy. Warto się im przyjrzeć:

- `\1`: **odwołanie wsteczne** do grupy 1. Jak wyjaśniłem w poprzednim punkcie, nawiasy zwykłe pomagają dopasować część ciągu, do którego można się odwoływać w dalszej części regeksa. W tym przypadku składnia `\1` wskazuje, że w tej pozycji można oczekiwać tej samej części ciągu, jaka została dopasowana za pomocą pierwszej pary nawiasów. Pamiętaj, że w języku Python należy użyć składni `\g<1>` zamiast `\1`.
- `(?: ... )`: jest to tak zwana *grupa nieprzechwytująca* (ang. *non-capturing group*). Czasami potrzebujemy nawiasów, aby poprawnie zastosować kwantyfikator, ale nie chcemy, aby ich zawartość była zgłaszana w wynikach.

Oto opis poszczególnych fragmentów powyższego regeksa:

1. `^`: pasuje do początku ciągu lub wiersza, jeśli jest włączona flaga wielowierszowa.
2. `\d{1,2}`: pasuje do dowolnej cyfry występującej raz lub dwa razy.
3. `([\-\/])`: pasuje do dowolnego znaku między znakiem `-` a `/` i przechwytuje wynik jako grupę 1.
4. `\d{1,2}`: pasuje do dowolnej cyfry występującej raz lub dwa razy.
5. `\1`: odwołanie wsteczne do przechwyconej grupy 1. Pasuje więc do dowolnego znaku między znakiem `-` a `/`.
6. `.+`: pasuje do dowolnego znaku, z wyjątkiem znaku podziału wiersza, występującego jeden raz lub więcej razy.
7. `(?:\ d{4}|\d{2})`: odpowiada jednej z następujących dwóch alternatyw: dowolnej cyfrze powtórzonej dokładnie cztery razy lub dowolnej cyfrze powtórzonej dokładnie dwa razy.

Powyższy regex można zwizualizować w następujący sposób (rysunek 5.17):



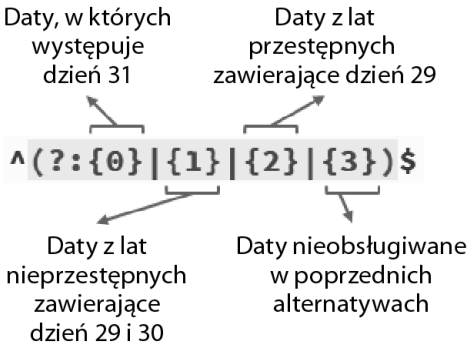
Rysunek 5.17. Wizualizacja pierwszej wersji regeksa do sprawdzania poprawności dat

Jak można się domyślić, ten regex sprawdza poprawność dat w następujących formatach: `dd-mm-rrrr`, `dd-mm-rrrr`, `d-m-rrrr` i `d-m-rr` z separatorem `/` lub `-`. Nie uwzględnia jednak błędów spowodowanych nieprawidłowymi datami, takimi jak 30 lutego lub 31 września.

Regex, który uwzględnia te błędy, ma następującą postać:

```
^(?:(?:31[\-\/](?:(?:0?[13578])|(1[02]))[\-\/](19|20)?\d\d)|(?:(?:29|30)
[\-\/](?:(?:0?[13-9])|(?:1[0-2]))[\-\/](?:19|20)?\d\d)|(?:29[\-\/]0?2
[\-\/](?:19|20)(?:(?:[02468][048])|(?:[13579][26]))|(?:(?:(?:0?[1-9])|(?:1\d)|(?:2[0-8]))
[\-\/](?:(?:0?[1-9])|(?:1[0-2]))[\-\/](?:19|20)?\d\d))$
```

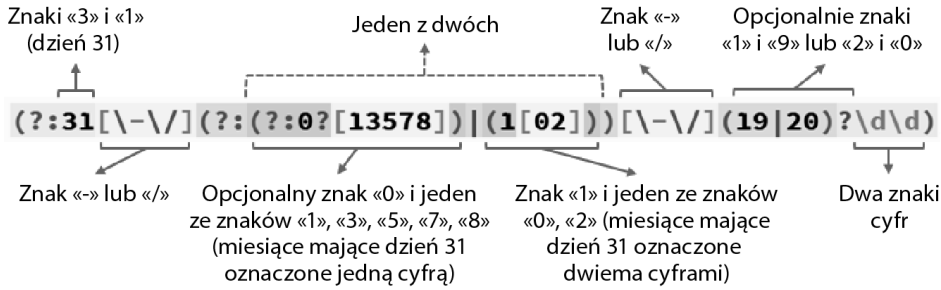
Regex zapisany w tej postaci jest bardzo trudny do interpretacji. Jednak jeśli spróbujemy przyjrzeć mu się dokładniej, możemy zauważyć, że składa się on z czterech alternatyw (rysunek 5.18):



Rysunek 5.18. Struktura złożonego regeksa do walidacji daty

Ponadto w tym przypadku ciągi {0}, {1}, {2} i {3} są jedynie symbolami zastępczymi, a nie znakami do dopasowania. Spróbujmy zdefiniować kolejne tokeny, począwszy od tokena {0}.

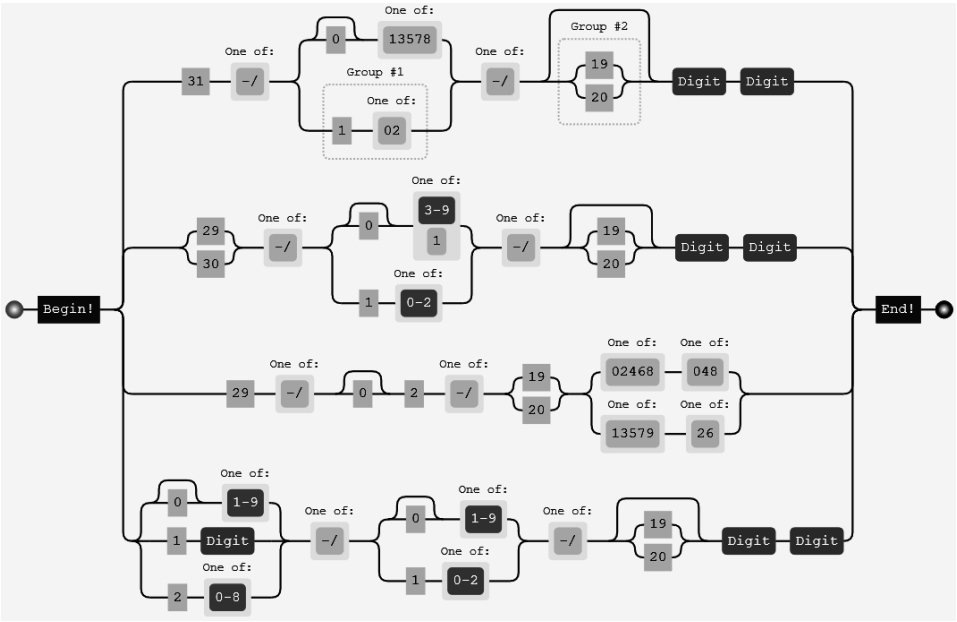
Token {0} odpowiada datom, które mają 31. dzień. Podobnie jak w poprzednich przypadkach, znacznie łatwiej jest wyjaśnić, co ten regex robi, na rysunku zamiast słowami. Poszczególne części regeksa zostały wyjaśnione za pomocą etykiet umieszczonych na rysunku 5.19:



Rysunek 5.19. Szczegółowy opis części regeksa dla dat, które mają dzień 31

Regeksy odpowiadające innym symbolom zastępczym są bardzo podobne do tego, który właśnie wyjaśniliśmy. Dlatego pozostawiam zapoznanie się z ich wyjaśnieniem jako ćwiczenie dla Czytelnika. Jeśli chcesz zobaczyć wizualizację całego regeksa, zapoznaj się z poniższym rysunkiem pochodzącym ze strony *https://jex.im/regulex* (rysunek 5.20).

Regex, który właśnie przeanalizowaliśmy, umożliwia walidację formatu `dd-mm-rrrr` ze wszystkimi jego wariantami. Sposób implementacji mechanizmu sprawdzania poprawności dat w usłudze Power BI zademonstruję w kodzie. Dodatkowo znajdziesz regeksy, które umożliwiają sprawdzanie poprawności dat w formatach `mm-dd-rrrr` i `rrrr-mm-dd` ze wszystkimi ich wariantami (gdzie rok składa się z dwóch cyfr, miesiąc składa się z jednej cyfry i tak dalej).



Rysunek 5.20. Wizualizacja całego regeksa sprawdzania poprawności dat

Teraz, gdy już wiesz, co kryje się za złożonymi regeksami przedstawionymi wcześniej, przejdźmy do ich implementacji w usłudze Power BI, gdzie zostaną wykorzystane do sprawdzenia poprawności danych.

Sprawdzanie poprawności danych przy użyciu regeksów w usłudze Power BI

Do tej pory usługa Power BI nie ma natywnej funkcji w dodatku Power Query do wykonywania operacji z wykorzystaniem regeksów. Zdarzają się jednak przypadki, w których nie można uniknąć użycia regeksów do wyodrębnienia przydatnych informacji z danych w formie tekstowej. Jedynym sposobem, aby móc używać regeksów, są skrypty języka R lub skrypty języka Python. Jedyne ograniczenie w tym przypadku polega na tym, że jeśli chcesz opublikować w usłudze Power BI raport, który wymaga od dodatku Power Query korzystania z zewnętrznych silników języka R lub Python, musisz również zainstalować lokalną bramę danych w trybie osobistym.

Przejdźmy jednak do praktycznych przykładów.

Załóżmy, że pracujesz w firmie detalicznej, w której istnieje zespół zajmujący się identyfikacją nieuczciwych klientów. Gdy członek zespołu zidentyfikuje oszusta, wypełnia arkusz kalkulacyjny Excela, gdzie obok innych kolumn są zawarte kolumny `Email` i `BannedDate`. Twoim zadaniem,

w celu przeprowadzenia dokładnej analizy zakupów klienta będącego oszustem, jest załadowanie danych do usługi Power BI z tego pliku Excela oraz z innych źródeł danych i wybranie samych informacji o oszuście.

Posiadanie poprawnych adresów e-mail oszustów w pliku Excela jest niezbędne do tego, by móc prawidłowo połączyć je z innymi danymi. Odpowiednie daty zablokowania również są ważne, aby wiedzieć, czy po tej dacie prześlizgnęły się dalsze zamówienia od tego oszusta. Jak wiadomo, wypełnianie pliku Excela przez kilku użytkowników odbywa się bez jakiejkolwiek walidacji wprowadzonych danych, dlatego często zdarzają się błędy. Identyfikacja wszystkich błędów podczas wypełniania niektórych pól i wyróżnianie ich pozwala zespołowi ds. oszustw je poprawić. W tym przypadku z pomocą przychodzą Ci regeksy.

Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności adresów e-mail z wykorzystaniem języka Python

W repozytorium dołączonym do tej książki, w folderze *Chapter05*, można znaleźć plik Excela *Users.xlsx*. Jego zawartość jest podobna do tej, która została przedstawiona na rysunku 5.21:

	A	B	C	D	E
1	UserId	Email	BannedDate	IsEmailValidByDefinition	IsDateValidByDefinition
2	1	_____@example.com	05/29/2018	1	1
3	2	example1@example.com/example2@example.com	06/07/2019	0	1
4	3	example33@example.com.	02/05/2018	0	1
5	4	firstname-lastname@example.com	06/07/2019	1	1
6	5	example@example.com --> check	02/29/18	0	0
7	6	email@example-one.com	11/06/2017	1	1

Rysunek 5.21. Zawartość pliku Users.xlsx

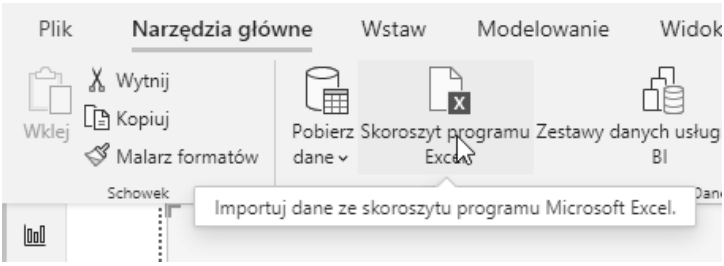
W tym punkcie skupimy się wyłącznie na kolumnie `Email`. Zawiera ona adresy e-mail oszustów, wprowadzone ręcznie przez zespół ds. oszustw wspomniany na początku podrozdziału. Nie wszystkie adresy e-mail zestawione w tej kolumnie mają prawidłową składnię. Ponadto w pliku Excel znajduje się kolumna `IsEmailValidByDefinition`, której wartości (`1` = tak; `0` = nie) wskazują, czy adres e-mail odpowiadający tej wartości jest prawidłowy, czy nie.

Python zawiera wbudowany pakiet o nazwie `re`, który dostarcza potrzebnych do pracy z regeksami funkcji. Ponadto pakiet `pandas` dodaje kilka metod dla obiektów `series` lub `dataframe`, które przyjmują regeksy w roli parametrów w celu znalezienia wzorców w ciągu. Metody te działają w taki sam sposób jak te, które udostępnia moduł `re` Pythona. Wkrótce skorzystamy z metody `match`.

Pokażę również składnię `r'...'` do tworzenia ciągów znaków. W ten sposób tworzy się tzw. **surowe ciągi** (ang. *raw strings*), które pozwalają interpretować lewy ukośnik (\) dosłownie, a nie jako znak ucieczki.

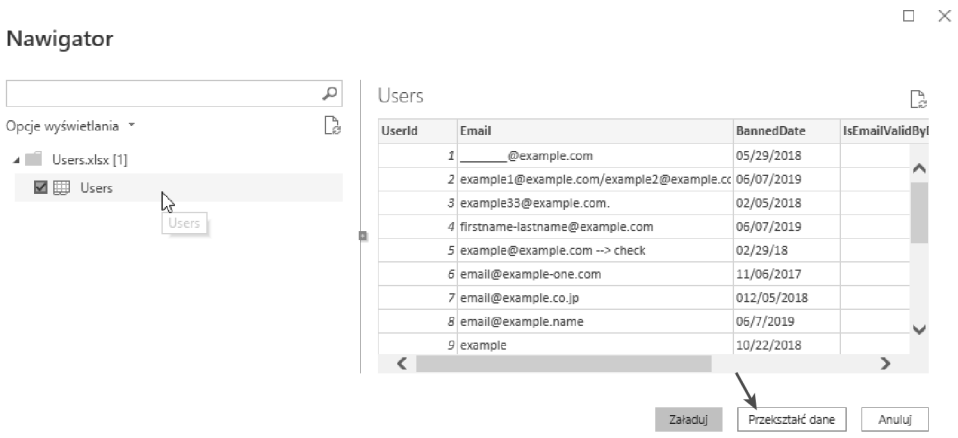
Otwórz program Power BI Desktop, upewnij się, że korzystasz z wirtualnego środowiska języka Python `pbi_powerquery_env`, i zaczynajmy:

1. Aby zaimportować dane z programu Excel, kliknij na wstążce ikonę programu Excel (rysunek 5.22):



Rysunek 5.22. Importowanie danych z Excela

2. W oknie dialogowym *Otwieranie* wybierz wcześniej wymieniony plik *Users.xlsx*.
3. W oknie *Nawigator* zaznacz arkusz *Users*, a następnie kliknij *Przekształć dane* (rysunek 5.23):



Rysunek 5.23. Wybierz arkusz Users i kliknij przycisk Przekształć dane

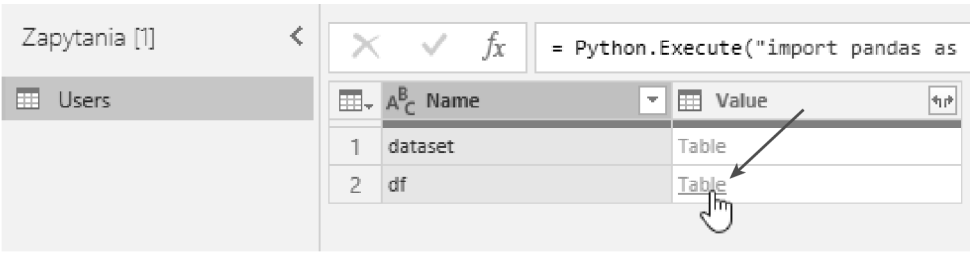
4. Kliknij menu *Przekształć*, a następnie kliknij *Uruchom skrypt języka Python*.
5. Następnie skopiuj i wklej do edytora skryptów Pythona poniższy kod i kliknij *OK*:

```
import pandas as pd
import re

df = dataset

regex_local_part =
r'([^<>()[\]\\.,;:\s@\""]+(\.[^<>()[\]\\.,;:\s@\""]+)*)|(\"".+\"")'
regex_domain_name = r'(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,})'
```

```
regex_domain_ip_address = r'(\[?[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}\]?)'

pattern = r'^({0})@({1}|{2})$'.format(regex_local_part, regex_domain_name,
regex_domain_ip_address)
df['isEmailValidFromRegex'] = df['Email'].str.match(pattern).astype(int)
```

Ten skrypt Pythona możesz również znaleźć w pliku *01-validate-emails-withregex--with-python.py*, który znajduje się w repozytorium dołączonym do książki, w folderze *Chapter05\validating-data-using-regex*.

6. Program Power BI Desktop może wyświetlić alert o następującej treści: *Wymagane są informacje dotyczące prywatności danych*. Jeśli taki komunikat zostanie wyświetlony, kliknij *Kontynuuj* i wykonaj krok 7; w przeciwnym razie możesz przejść do kroku 8.
7. Wyświetli się okno *Poziomy prywatności*. W tym miejscu można określić poziom izolacji, który definiuje stopień, w jakim jedno źródło danych będzie odizolowane od innych źródeł danych. Możesz zignorować sprawdzanie poziomów prywatności, ale może to spowodować ujawnienie poufnych danych osobom nieupoważnionym. Zostaniesz poproszony o wybranie poziomu prywatności zarówno dla skryptu Pythona, jak i zestawu danych załadowanego z programu Excel. Jeśli wybierzesz dla obu poziom *Organizacyjne*, wszystko będzie działać poprawnie w programie Power BI Desktop. Jeśli jednak planujesz opublikować raporty w usłudze Power BI (lub osadzić je w innych miejscach), musisz użyć poziomu *Publiczne*. Aby uzyskać więcej informacji, zapoznaj się ze stroną *http://bit.ly/pbi-privacy-levels*. Na razie wybierz poziom *Organizacyjne* dla obu opcji.
8. Teraz interesuje nas tylko zestaw danych `df`. Kliknij więc jego wartość *Table* (rysunek 5.24):



Rysunek 5.24. Wybieranie zestawu danych df uzyskanego w wyniku transformacji za pomocą skryptu Pythona

9. Jak widać, została dodana kolumna `isEmailValidFromRegex`, która zawiera wartości logiczne wynikające ze sprawdzania poprawności adresów e-mail za pośrednictwem regeksa. Łatwo zauważyć, że wartości w tej kolumnie pokrywają się z wartościami podanymi w kolumnie `IsEmailValidByDefinition` (rysunek 5.25):

Email	BannedDate	IsEmailValidByDefinition	IsDateValidByDefinition	isEmailValidFromRegex
_____@example.com	05/29/2018	1	1	1
example1@example.com/example2@exampl...	06/07/2019	0	1	0
example33@example.com.	02/05/2018	0	1	0
firstname-lastname@example.com	06/07/2019	1	1	1
example@example.com --> check	02/29/18	0	0	0
email@example-one.com	11/06/2017	1	1	1
email@example.co.jp	012/05/2018	1	0	1
email@example.name	06/7/2019	1	1	1
example	10/22/2018	0	1	0
email@example.com	9/04/2017	1	1	1
email@subdomain.example.com	11/22/2018	1	1	1
email@123.123.123.123	07/31/17	1	1	1
.@example.com	04/24/018	0	0	0
firstname.lastname@example.com	11/22/18	1	1	1
firstname+lastname@example.com	05/16/2018	1	1	1
example@example.c	6/7/2019	0	1	0
1234567890@example.com	05/16/2018	1	1	1
email@example.museum	03/26/2018	1	1	1
email@[123.123.123.123]	06/31/2017	1	0	1

Rysunek 5.25. Wyniki weryfikacji adresów e-mail za pomocą regeksa

> Twój regex wykonał świetną robotę! Teraz możesz wrócić do menu głównego i kliknąć *Zamknij i zastosuj*.

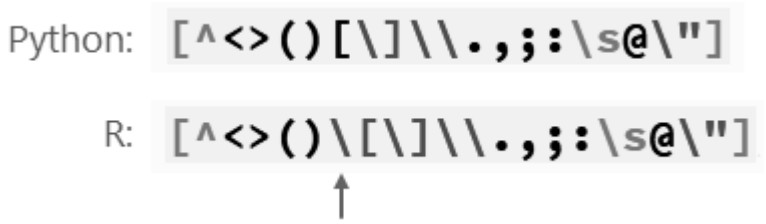
Dzięki kolumnie `isEmailValidFromRegex` możesz teraz odpowiednio filtrować prawidłowe i nieprawidłowe adresy e-mail. Możesz nawet przekazać raport w tej sprawie do zespołu ds. oszustw.

Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności adresów e-mail za pomocą języka R

Proces sprawdzania poprawności adresów e-mail za pomocą regeksa z wykorzystaniem języka R jest prawie taki sam, jak z wykorzystaniem Pythona, z wyjątkiem kilku rzeczy.

Po pierwsze, język R, począwszy od wersji 4.0.0, pozwala na korzystanie wyłącznie z surowych ciągów znaków. Ponadto składnia surowych ciągów znaków w R jest nieco inna niż w Pythonie. Zamiast zapisu `r'...'` możesz używać zapisów `r'(...)'`, `r'[...]'` lub `r'{...}'`. Ponadto zamiast używać liczbowych symboli zastępczych w nawiasach klamrowych wewnątrz ciągu, a następnie przypisywać je za pomocą funkcji `format()`, tak jak w języku Python, w R można po prostu bezpośrednio używać nazw zmiennych w nawiasach klamrowych.

Ponadto trzeba zwrócić uwagę na jeszcze jedną rzecz: w języku R nie tylko znak] jest interpretowany jako metaznak. W ten sam sposób jest traktowany również znak [. Z tego powodu, jeśli chcesz użyć obu nawiasów kwadratowych jako literałów, musisz poprzedzić obydwa znakiem ucieczki w postaci lewego ukośnika (\). W związku z tym część regeksa, która identyfikuje klasę znaków w lokalnej części regeksa do sprawdzania adresu e-mail, jest nieco inna (rysunek 5.26):



Rysunek 5.26. W języku R należy poprzedzić otwarty nawias kwadratowy znakiem ucieczki, aby był interpretowany jako literał

Pakiet `R.Base` udostępnia dwie funkcje umożliwiające korzystanie z regeksów `grep()` i `grepl()`:

- Funkcja `grepl()`, na podstawie tego, czy wzorzec istnieje w ciągu znaków, zwraca wartość logiczną.
- Funkcja `grep()` zwraca indeksy wystąpień w wektorze znaków zawierającym dopasowania lub określone ciągi znaków, dla których istnieje dopasowanie.

Ponieważ chcemy zastosować paradygmat `Tidyverse`, użyjemy funkcji wrapperów dostarczanych przez pakiet `stringr`, odpowiednio `str_detect()` i `str_which()`.

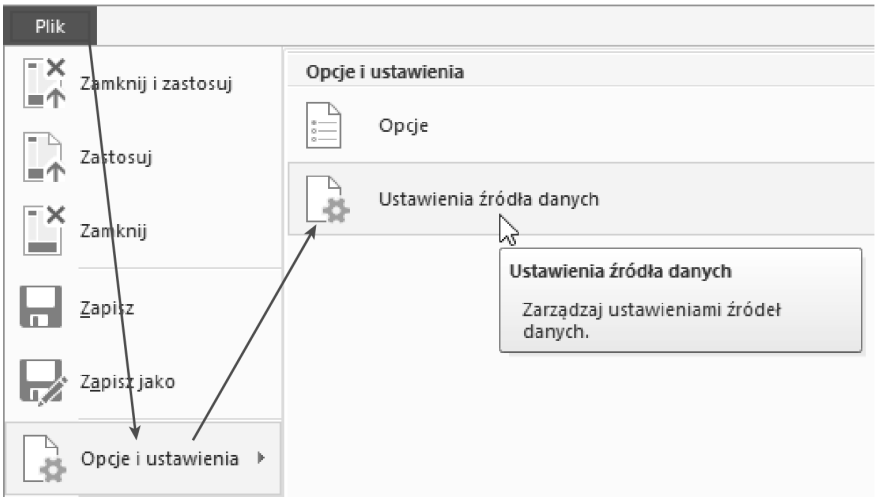
Z wyjątkiem opisanych różnic, proces sprawdzania poprawności adresów e-mail w pliku Excela *Users.xlsx* w usłudze Power BI przy użyciu skryptu języka R jest praktycznie taki sam, jak opisany w poprzednim punkcie proces dotyczący Pythona:

1. Aby zaimportować dane z pliku *Users.xlsx*, powtórz kroki od 1. do 3. z poprzedniego punktu.
2. Kliknij menu *Przekształć*, a następnie kliknij polecenie *Uruchom skrypt języka R*.
3. Następnie skopiuj i wklej poniższy kod do edytora skryptów R i kliknij *OK*:

```
library(dplyr)
library(stringr)

regex_local_part <-
r'(([^<>()\[\]\\.,;:\s@\"]+(\.[^<>()\[\]\\.,;:\s@\"]+)*)|(\".+\"))'
regex_domain_name <- r'((([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))'
regex_domain_ip_address <- r'((\[?[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-
9]{1,3}\]?))'

pattern <- str_glue(
  '^({regex_local_part})@({regex_domain_name}|{regex_domain_ip_address})$'
)

df <- df %>%
  mutate( isEmailValidFromRegex = as.integer(str_detect(Email, pattern)) )
```

Ten skrypt języka R możesz również znaleźć w pliku *02-validate-emails-with-regexwith- r.R*, w folderze *Chapter05\validating-data-using-regex*, w repozytorium dołączonym do książki.

4. Program Power BI Desktop może wyświetlić powiadomienie o następującej treści: *Wymagane są informacje dotyczące prywatności danych*. Jeśli tak się stanie, kliknij *Kontynuuj* i postępuj zgodnie z instrukcjami zamieszczonymi poniżej. W przeciwnym razie możesz przejść do kroku 5. Dla skryptów języka R również wybierz poziom *Organizacyjne*. Czasami okazuje się, że poziomy zgodności zestawów danych i skryptów analitycznych nie są ze sobą zgodne. W takim przypadku usługa Power BI może wyświetlić ostrzeżenie w postaci *Formula.Firewall: Zapytanie "XXX" (krok "YYY") próbuje uzyskać dostęp do źródeł danych z poziomami prywatności, które nie mogą być używane razem. Utwórz ponownie tę kombinację danych*. W takim przypadku wystarczy otworzyć okno *Ustawienia źródła danych* w następujący sposób (rysunek 5.27):

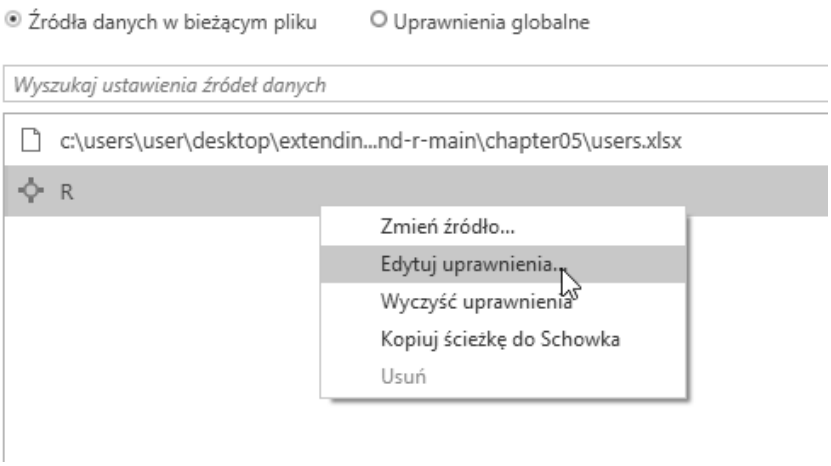


Rysunek 5.27. Otwieranie okna ustawień źródła danych dodatku Power Query

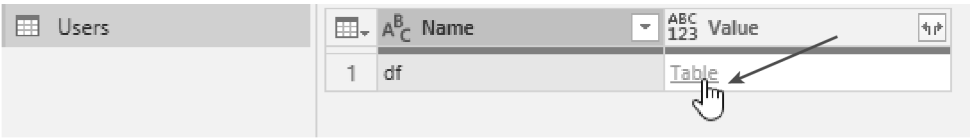
Następnie musisz upewnić się, że wszystkie źródła danych mają ten sam poziom prywatności (w naszym przypadku *Organizacyjne*). W tym celu zmień go dla każdego źródła danych, dla którego jest to konieczne, za pomocą opcji *Edit Permissions…* (rysunek 5.28).

W tym momencie możesz odświeżyć dane podglądu. Aby to zrobić, kliknij *Odśwież podgląd*.

5. W tym przypadku interesuje nas tylko zestaw danych `df`. Kliknij więc odpowiadającą mu wartość *Table* (rysunek 5.29).
6. Jak widać, kolumna isEmailValidFromRegex została dodana i zawiera wartości logiczne (przekształcone na 1 i 0) będące wynikiem walidacji adresów e-mail za pomocą regeksa. Łatwo zauważyć, że wartości w tej kolumnie pokrywają się z wartościami podanymi w kolumnie IsEmailValidByDefinition. Twój regex wykonał świetną robotę! Teraz możesz wrócić do menu Narzędzia główne i kliknąć polecenie Zamknij i zastosuj.



Rysunek 5.28. Edytowanie uprawnień prywatności dla źródeł danych



Rysunek 5.29. Wybieranie zestawu danych df uzyskanego w wyniku transformacji za pomocą skryptu języka R

Dzięki kolumnie `isEmailValidFromRegex` możesz teraz odpowiednio filtrować poprawne i niepoprawne adresy e-mail w swoich raportach.

Teraz przyjrzyjmy się, jak sprawdzać poprawność dat w usłudze Power BI za pomocą języka Python.

Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności dat za pomocą języka Python

Jako przykład dat do sprawdzenia poprawności użyjemy wykorzystanego już wcześniej pliku Excela *Users.xlsx*. Plik ten zawiera kolumnę `BannedDate` z tekstowymi reprezentacjami dat w formacie `mm/dd/rrrr` we wszystkich jego wariantach. Ponadto w pliku Excela znajduje się również kolumna `IsDateValidByDefinition`, której wartości (`1` = tak; `0` = nie) wskazują, czy data pasująca do tej wartości jest prawidłowa, czy nie.

Funkcje Pythona potrzebne do skorzystania z regeksów poznałeś już wcześniej. Zatem zaczynajmy:

1. Aby zaimportować dane z pliku *Users.xlsx*, powtórz kroki od 1. do 3. z punktu „Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności adresów e-mail za pomocą języka Python”.
2. Kliknij menu *Przekształć*, a następnie *Uruchom skrypt języka Python*.
3. Następnie skopiuj poniższy kod i wklej go do edytora skryptów Pythona, po czym kliknij *OK*:

```
import pandas as pd
import re

df = dataset

regex_dates_having_day_31 = r'(?:(?:(?:0?[13578])|(?:1[02]))[\-\/]31[\-
\/](?:19|20)?\d\d)'

regex_non_leap_dates_having_days_29_30 =r'(?:(?:(?:0?[13-9])|(?:1[0-2]))[\-
\/](?:29|30)[\-\/](?:19|20)?\d\d)'

regex_leap_dates_having_day_29 = r'(?:0?2[\-\/]29[\-
\/](?:19|20)?(?:(?:[02468][048])|(?:[13579][26])))'

regex_remaining_dates = r'(?:(?:(?:0?[1-9])|(?:1[0-2]))[\-
\/](?:(?:1\d)|(?:0?[1-9])|(?:2[0-8]))[\-\/](?:19|20)?\d\d)'

pattern = r'^(?:{0}|{1}|{2}|{3})$'.format(regex_dates_having_day_31,
regex_non_leap_dates_having_days_29_30,regex_leap_dates_having_day_29,
regex_remaining_dates)

df['isValidDateFromRegex'] = df['BannedDate'].str.
match(pattern).astype(int)
```

   Bardziej kompletny skrypt Pythona możesz znaleźć w folderze *Chapter05\validating--data-using-regex*, w pliku *03-validate-dateswith- regex-with-python.py*, w repozytorium dołączonym do książki. Skrypt ten obsługuje daty w formatach `mm-dd-rrrr`, `dd-mm-rrrr` i `rrrr-mm-dd`, wraz ze wszystkimi ich odmianami, w tym zarówno z separatorem `-`, jak i `/`.
4. Jeśli usługa Power BI wymaga podania informacji o prywatności, postępuj zgodnie ze wskazówkami zamieszczonymi w poprzednich punktach.
5. Jak widać, kolumna `isValidDateFromRegex` została dodana i zawiera wartości logiczne wynikające z walidacji dat za pomocą regeksa. Łatwo zauważyć, że wartości te pokrywają się z wartościami podanymi w kolumnie `IsDateValidByDefinition` (rysunek 5.30).

   Twój regex wykonał świetną robotę! Teraz możesz wrócić do menu *Narzędzia główne* i kliknąć polecenie *Zamknij i zastosuj*.

Dzięki kolumnie `isValidDateFromRegex` możesz teraz filtrować poprawne i niepoprawne daty i odpowiednio je wykorzystywać.

BannedDate	IsEmailValidByDefinition	IsDateValidByDefinition	isValidDateFromRegex
05/29/2018	1	1	1
06/07/2019	0	1	1
02/05/2018	0	1	1
06/07/2019	1	1	1
02/29/18	0	0	0
11/06/2017	1	1	1
012/05/2018	1	0	0
06/7/2019	1	1	1
10/22/2018	0	1	1
9/04/2017	1	1	1
11/22/2018	1	1	1
07/31/17	1	1	1
04/24/018	0	0	0
11/22/18	1	1	1
05/16/2018	1	1	1
6/7/2019	0	1	1
05/16/2018	1	1	1
03/26/2018	1	1	1
06/31/2017	1	0	0

Rysunek 5.30. Wyniki regeksa walidacji dat

Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności dat za pomocą języka R

Proces sprawdzania poprawności daty przy użyciu regeksów i języka R jest prawie taki sam, jak za pomocą Pythona. Różnice są podobne do tych, które wymieniłem w punkcie „Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności adresów e-mail za pomocą języka R". W przykładzie skorzystamy z tego samego pliku Excela *Users.xlsx*. Oto procedura, którą należy wykonać:

1. Aby zaimportować dane z pliku *Users.xlsx*, powtórz kroki od 1. do 3. z punktu „Korzystanie z regeksów w usłudze Power BI do sprawdzania poprawności adresów e-mail za pomocą języka Python".
2. Kliknij menu *Przekształć*, a następnie polecenie *Uruchom skrypt języka R*.
3. Skopiuj poniższy kod i wklej go do edytora skryptów języka R, po czym kliknij *OK*:

```
library(dplyr)
library(stringr)

df <- dataset

regex_dates_having_day_31 <- r'((?:(?:(?:0?[13578])|(?:1[02]))[\-\/]31[\-
\/](?:19|20)?\d\d))'
regex_non_leap_dates_having_days_29_30 <-r'((?:(?:(?:0?[13-9])|(?:1[0-2]))[\-
\/](?:29|30)[\-\/](?:19|20)?\d\d))'
```

```
regex_leap_dates_having_day_29 <- r'((?:0?2[\-\/]29[\-
\/](?:19|20)?(?:(?:[02468][048])|(?:[13579][26]))))'
regex_remaining_dates <- r'((?:(?:(?:0?[1-9])|(?:1[0-2]))[\-
\/](?:(?:1\d)|(?:0?[1-9])|(?:2[0-8]))[\-\/](?:19|20)?\d\d))'

pattern <-
str_glue('^(?:{regex_dates_having_day_31}|{regex_non_leap_dates_having_days_2
9_30}|{regex_leap_dates_having_day_29}|{regex_remaining_dates})$')

df <- df %>%
  mutate( isDateValidFromRegex = as.integer(str_
detect(BannedDate, pattern)) )
```

Bardziej kompletny skrypt języka R można znaleźć w folderze *Chapter05\validating-data-using-regex*, w pliku *04-validate-dates-withregex-with-r.R*, w repozytorium dołączonym do tej książki. Ten skrypt obsługuje daty w formatach `mm-dd-rrrr`, `dd-mm-rrrr` i `rrrr-mm-dd` ze wszystkimi ich odmianami, w tym zarówno z separatorem `-`, jak i `/`.

4. Jeśli usługa Power BI wymaga podania informacji o prywatności, postępuj zgodnie ze wskazówkami zamieszczonymi w poprzednich punktach.
5. Jak widać, kolumna `isValidDateFromRegex` została dodana i zawiera wartości logiczne wynikające z walidacji dat za pomocą regeksa. Łatwo zauważyć, że wartości te pokrywają się z wartościami podanymi w kolumnie `IsDateValidByDefinition`. Twój regex wykonał świetną robotę! Teraz możesz wrócić do menu *Narzędzia główne* i kliknąć polecenie *Zamknij i zastosuj*.

Dzięki kolumnie `isEmailValidFromRegex` możesz teraz odpowiednio filtrować poprawne i niepoprawne adresy e-mail w swoich raportach.

W następnym podrozdziale dowiesz się, jak zaimportować zawartość półstrukturalnego pliku loga za pomocą języków Python i R.

Ładowanie do usługi Power BI złożonych plików logów z wykorzystaniem regeksów

Pliki logów to bardzo przydatne narzędzia dla programistów i administratorów systemów komputerowych. Rejestrują, co działo się z systemem, kiedy to się stało i który użytkownik wygenerował zdarzenie. Dzięki tym plikom można znaleźć informacje o każdej awarii systemu, co pozwala na szybszą diagnozę przyczyn występujących usterek.

Logi często zawierają **dane częściowo ustrukturyzowane**, czyli informacje, których nie można utrwalić w relacyjnej bazie danych w formacie, w którym zostały wygenerowane. Aby można było przeanalizować dane z logów za pomocą zwykłych narzędzi, dane te powinny przede wszystkim być przekształcone na bardziej odpowiedni format.

Ponieważ logi nie są danymi ustrukturyzowanymi, trudno jest je zaimportować do usługi Power BI w niezmienionej postaci bez opracowania specjalnego, niestandardowego łącznika. To właśnie w tych scenariuszach pożądane rezultaty mogą nam pomóc uzyskać regeksy użyte za pośrednictwem takich języków, jak Python lub R.

Logi dostępu do serwera Apache

Załóżmy, że Twoja firma ma stronę internetową opublikowaną za pośrednictwem serwera WWW Apache. Twój menedżer prosi Cię o przeprowadzenie analizy dotyczącej tego, na których stronach internetowych witryny użytkownicy najczęściej klikają. Jedynym sposobem uzyskania tych informacji jest przeanalizowanie pliku logu dostępu. W tym pliku są zarejestrowane dane o wszystkich żądaniach wysyłanych do serwera WWW. Oto przykład zawartości logu dostępu serwera Apache (rysunek 5.31):

```
1 83.149.9.216 - - [17/May/2015:10:05:03 +0000] "GET
  /presentations/logstash-monitorama-2013/images/kibana-search.png HTTP/1.1" 200 203023
  "http://semicomplete.com/presentations/logstash-monitorama-2013/" "Mozilla/5.0 (Macintosh;
  Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.77 Safari/537.36"
2 83.149.9.216 - - [17/May/2015:10:05:43 +0000] "GET
  /presentations/logstash-monitorama-2013/images/kibana-dashboard3.png HTTP/1.1" 200 171717
  "http://semicomplete.com/presentations/logstash-monitorama-2013/" "Mozilla/5.0 (Macintosh;
  Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.77 Safari/537.36"
```

Rysunek 5.31. Przykład logu dostępu serwera Apache

Jak widać, struktura informacji w tym logu jest dość dobrze uporządkowana. Jeśli nikt nie spersonalizował danych wyjściowych plików logów serwera Apache, domyślnie stosowany jest format CLF (ang. *Common Log Format*). Rzeczywisty przykład logu dostępu serwera Apache możesz znaleźć w folderze *Chapter05\loading-complex-log-files-using-regex*, w pliku *apache_logs.txt*, w repozytorium dołączonym do tej książki. Plik ten znajdziesz w repozytorium GitHub, pod adresem *http://bit.ly/apache-access-log* (aby go wyświetlić, kliknij *Download*).

Jeśli przeczytasz dokumentację tych plików logów, odkryjesz, że informacje zapisane w logu dostępu są zgodne z formatem NCSA. Tak więc logowane są następujące dane:

1. Nazwa zdalnego hosta (adres IP).
2. Nazwa zdalnego logu (jeśli jest pusta, w logu będzie myślnik, co oznacza, że nazwa ta nie jest używana w przykładowym pliku).
3. Zdalny użytkownik, jeśli żądanie zostało uwierzytelnione (jeśli nie ma danych użytkownika, w logu będzie myślnik).
4. Data otrzymania żądania w formacie `[18/Sep/2011:19:18:28 -0400]`.
5. Pierwszy wiersz żądania wysłanego do serwera umieszczony w cudzysłowie.
6. Kod statusu HTTP żądania.
7. Rozmiar odpowiedzi w bajtach, z wyłączeniem nagłówków HTTP (wartość ta może być oznaczona myślnikiem, jeśli jej brakuje).
8. Nagłówek żądania HTTP `Referer` zawierający bezwzględny lub częściowy adres strony wysyłającej żądanie.

9. Nagłówek żądania HTTP `User-Agent` zawierający ciąg identyfikujący aplikację, system operacyjny, dostawcę i (lub) wersję żądającego agenta użytkownika.

Gdy znasz zarówno charakter informacji zapisanych w logu, jak i format, w jakim informacje zostały zapisane, możesz skorzystać z zaawansowanych narzędzi udostępnianych przez regeksy. Pozwala to lepiej ustrukturyzować te informacje i zaimportować do usługi Power BI.

Importowanie logów dostępu serwera Apache w usłudze Power BI za pomocą języka Python

Jak wspomniałem wcześniej, rzeczywisty przykład logu dostępu serwera Apache jest dostępny w pliku *apache_logs.txt*, w folderze *Chapter05\loading-complex-log-files-using-regex*, w repozytorium dołączonym do tej książki. Informacje zapisane w tym pliku załadujemy za pomocą skryptu języka Python, a nie łącznika usługi Power BI.

W porównaniu z tym, czego nauczyłeś się wcześniej o regeksach i Pythonie, w skrypcie Pythona *01-apacheaccess- log-parser-python.py* (który znajdziesz w wyżej wymienionym folderze), występują następujące nowe konstrukcje:

Aby czytać plik tekstowy wiersz po wierszu w Pythonie, należy użyć funkcji `open(file, mode)` i metody `readlines()`. W szczególności chcemy odczytać plik *apache_logs.txt* w trybie tylko do odczytu (`'r'`) i zapisać na liście każdy z odczytanych wierszy.

W regeksach możliwe jest odwoływanie się do grup identyfikowanych za pomocą okrągłych nawiasów nie tylko przez indeks liczbowy, ale także *przez nazwę*. Jest to możliwe dzięki tzw. **nazwanym grupom przechwytującym**. Zazwyczaj składnia regeksa używana do przypisywania nazwy do grupy to (`?<nazwa-grupy>...`). W Pythonie należy wykorzystać składnię (`?P<nazwa-grupy>...`):

- W Pythonie można zdefiniować listę części regeksa, które można scalić (połączyć) za pomocą separatora zdefiniowanego przez regex (`\s+`):

```
regex_parts = [
    r'(?P<hostName>\S+)',
    r'\S+',
    r'(?P<userName>\S+)',
    r'\[(?P<requestDateTime>[\w:/]+\s[+\-]\d{4})\]',
    r'"(?P<requestContent>\S+\s?\S+?\s?\S+?)"',
    r'(?P<requestStatus>\d{3}|-)',
    r'(?P<responseSizeBytes>\d+|-)',
    r'"(?P<requestReferrer>[^"]*)"',
    r'"(?P<requestAgent>[^"]*)?"',
]

pattern = re.compile(r'\s+'.join(regex_parts) + r'$')
```

  Należy zauważyć, że w tym przypadku użyliśmy funkcji `re.compile()`, ponieważ dopasowanie musi być wykonane wiele razy we wszystkich wierszach logu. W związku z tym można skorzystać ze wstępnej kompilacji regeksa i osiągnąć zyski obliczeniowe.

- Operacja dopasowywania wzorców jest wykonywana dla każdego wiersza w logu:

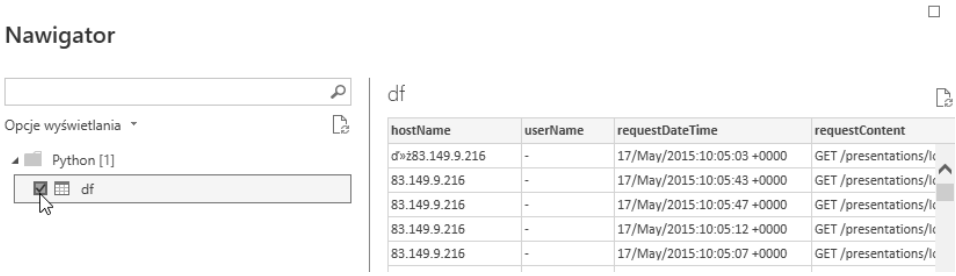
```
for line in access_log_lines:
    log_data.append(pattern.match(line).groupdict())
```

  Metoda `groupdict()` zwraca słownik z kluczami w postaci nazw grup i wartościami oznaczającymi dopasowane ciągi. Wszystkie słowniki dla każdego wiersza są dołączane do listy `log_data`.

Interpretację sposobu, w jaki każda część regeksa przechwytuje pożądany ciąg, pozostawiam Czytelnikowi.

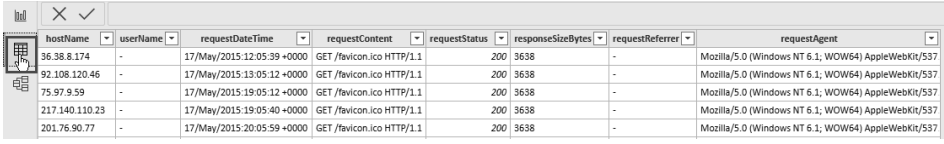
Po wyjaśnieniu kilku punktów w kodzie, spróbujmy zaimportować log do usługi Power BI:

1. W programie Power BI Desktop upewnij się, że używasz środowiska *pbi_powerquery_env*.
2. Przejdź do menu *Pobierz dane* i wybierz *Skrypt w języku Python*.
3. Skopiuj zawartość skryptu *01-apache-access-log-parser-python.py* i wklej ją do edytora skryptów Pythona, po czym kliknij *OK*.
4. Następnie w oknie *Nawigator* wybierz ramkę danych `df` i kliknij *Załaduj* (rysunek 5.32):



Rysunek 5.32. Wybieranie ramki danych df zwróconej przez skrypt Pythona

5. Jeśli klikniesz ikonę *Dane*, możesz wyświetlić cały załadowany log w postaci ustrukturyzowanej tabeli (rysunek 5.33):



hostName	userName	requestDateTime	requestContent	requestStatus	responseSizeBytes	requestReferrer	requestAgent
36.38.8.174	-	17/May/2015:12:05:39 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
92.108.120.46	-	17/May/2015:13:05:12 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
75.97.9.59	-	17/May/2015:19:05:12 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
217.140.110.23	-	17/May/2015:19:05:40 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
201.76.90.77	-	17/May/2015:20:05:59 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537

Rysunek 5.33. Log dostępu serwera Apache załadowany w usłudze Power BI za pomocą języka Python

Super! Dzięki możliwościom oferowanym przez regeksy udało Ci się zaimportować do usługi Power BI dane ze skomplikowanego pliku logu.

Importowanie logów dostępu serwera Apache w usłudze Power BI za pomocą języka R

W tym punkcie załadujemy informacje z pliku *apache_logs.txt*, ale tym razem za pomocą skryptu w języku R.

W porównaniu z tym, czego nauczyłeś się wcześniej o regeksach w języku R, w skrypcie *02-apache-access-log-parser-r* (który znajdziesz w wymienionym wcześniej folderze) napotkasz następujące nowe konstrukcje:

- Aby móc odczytywać plik tekstowy wiersz po wierszu w języku R, należy użyć funkcji `read_lines()` z pakietu `readr`. W szczególności będziemy odczytywać poszczególne wiersze pliku *apache_logs.txt* w celu utrwalenia ich w wektorze.
- Aby w pełni wykorzystać nazwane grupy przechwytujące w języku R, należy zainstalować nowy pakiet o nazwie `namedCapture` i z niego skorzystać. Dzięki temu pakietowi dozwolone są obydwie składnie regeksów dla nazwanych grup: standardowa składnia regeksów (`?<nazwa-grupy>...`) oraz składnia (`?P<nazwa-grupy>...`).
- Podobnie jak zrobiliśmy to w skrypcie Pythona, w skrypcie języka R zdefiniujemy wektor części regeksa, które scalimy ze sobą za pomocą funkcji `paste(..., collapse = '...')`. Ta funkcja ma za zadanie łączenie części regeksa za pomocą separatora `\s+`. Po scaleniu wszystkich części na końcu dodawany jest znak `$` za pomocą funkcji `paste0(...)`. Pamiętaj, że surowe ciągi znaków w języku R mają inną składnię niż w Pythonie. W tym przypadku użyjemy składni `r'{...}'`:

```
regex_parts <- c(
      r'{(?P<hostName>\S+)}'
    , r'{\S+}'
    , r'{(?P<userName>\S+)}'
    , r'{\[(?P<requestDateTime>[\w:/]+\s[+\-]\d{4})\]}'
    , r'{"(?P<requestContent>\S+\s?\S+?\s?\S+?)"}'
    , r'{(?P<requestStatus>\d{3}|-)}'
    , r'{(?P<responseSizeBytes>\d+|-)}'
    , r'{"(?P<requestReferrer>[^"]*)"}'
    , r'{"(?P<requestAgent>[^"]*)?"}'
)

pattern <- paste0( paste(regex_parts, collapse = r'{\
s+}'), '$' )
```

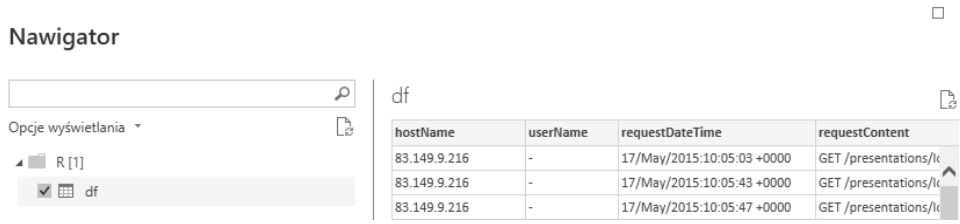
- Dopasowywanie wzorców odbywa się za pomocą funkcji `str_match_named()` należącej do pakietu `namedCapture`. Funkcja ta działa na całym wektorze logu dzięki poleceniu złożonemu z jednego wiersza:

```
df <- as.data.frame( str_match_named( access_log_lines, pattern = pattern ) )
```

Interpretację sposobu, w jaki poszczególne części regeksa przechwytują pożądany ciąg, pozostawiam Czytelnikowi.

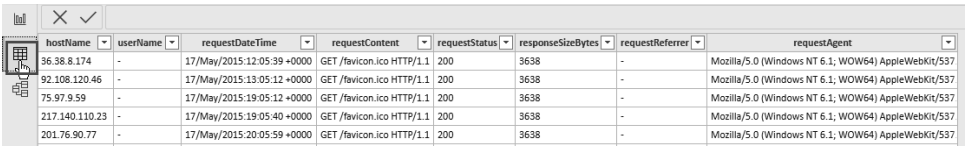
Teraz, gdy wyjaśniłem kilka punktów w kodzie, spróbujmy zaimportować log do usługi Power BI:

1. Najpierw musisz zainstalować pakiet `namedCapture`. Otwórz RStudio i upewnij się, że w ustawieniach *Global Options* ustawiłeś najnowszą wersję silnika.
2. Teraz przejdź do konsoli, wprowadź poniższy kod i uruchom go:
3. `install.packages("namedCapture")`
4. Otwórz program Power BI Desktop, przejdź do pozycji *Pobierz dane* i wybierz *Skrypt języka R*.
5. Skopiuj skrypt z pliku *02-apache-access-log-parser-r.R* i wklej go do edytora skryptów języka R, a następnie kliknij przycisk *OK*.
6. Następnie w oknie *Nawigator* wybierz ramkę danych `df` i kliknij *Załaduj* (rysunek 5.34):



Rysunek 5.34. Wybieranie ramki danych df zwróconej przez skrypt języka R

7. Jeśli klikniesz ikonę *Dane*, możesz wyświetlić cały załadowany log w postaci ustrukturyzowanej tabeli (rysunek 5.35):



hostName	userName	requestDateTime	requestContent	requestStatus	responseSizeBytes	requestReferrer	requestAgent
36.38.8.174	-	17/May/2015:12:05:39 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
92.108.120.46	-	17/May/2015:13:05:12 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
75.97.9.59	-	17/May/2015:19:05:12 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
217.140.110.23	-	17/May/2015:19:05:40 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537
201.76.90.77	-	17/May/2015:20:05:59 +0000	GET /favicon.ico HTTP/1.1	200	3638	-	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537

Rysunek 5.35. Log dostępu serwera Apache załadowany w usłudze Power BI za pomocą języka R

Świetna robota! Udało Ci się zaimportować częściowo ustrukturyzowany plik logu do usługi Power BI za pomocą języka R.

Wyodrębnianie wartości z tekstu w usłudze Power BI przy użyciu regeksów

Ostatni przypadek, który chcę zaprezentować, zdarza się bardzo często, gdy mamy do czynienia z wysyłką towarów do klientów. Bywa, że oszustowi udaje się ukraść towar zaadresowany do klienta; dlatego firma musi zwrócić klientowi środki. Oszukany klient kontaktuje się z działem obsługi klienta, aby poprosić o zwrot pieniędzy. Jeśli system zarządzania dostarczony operatorowi obsługi klienta zarządzającego sprawą nie pozwala na wprowadzanie informacji o zwrocie

w sposób ustrukturyzowany, operator musi skorzystać z jedynej możliwej metody: wprowadzenia *dowolnego tekstu* związanego z zamówieniem, z określeniem *kwoty* zwrotu, *powodu* zwrotu i *daty*.

Z pewnością już wiesz, że informacje wprowadzone w formacie dowolnego tekstu są koszmarem każdego analityka, zwłaszcza gdy Twój szef poprosi Cię o przeanalizowanie informacji wprowadzanych w tych niesławnych notatkach.

W repozytorium dołączonym do tej książki, w folderze *Chapter05*, można znaleźć plik Excela *OrderNotes.xlsx*. Jego zawartość jest podobna do treści pokazanej na rysunku 5.36:

	A	B
1	OrderNumber	Notes
2	ORD000001	5,00 EUR Kradzież przy dostawie opłaconej przelewem 11/02/2021
3	ORD000002	29,00 EUR Zwrot za kradzież w dostawie 04/06/2020
4	ORD000003	53,00€ Zwrot za kradzież w dostawie 24/09/2020
5	ORD000004	29/10/2020 EUR45,00 Zwrot za kradzież w dostawie
6	ORD000005	EUR 522,00 Zwrot za kradzież w dostawie 20/08/2020
7	ORD000006	€ 266,00 - Kradzież w dostawie opłaconej przelewem w dniu 10/12/2020
8	ORD000007	EUR68,50 - Zwrot za kradzież w dostawie 02/07/2020
9	ORD000008	EUR 50,00 - Zwrot za kradzież przy dostawie z dnia 30/07/2020
10	ORD000009	30/07/2020 209,00 € - Zwrot za kradzież w dostawie

Rysunek 5.36. Notatki tekstowe wprowadzone przez operatora dla niektórych zamówień

Jak widać po zawartości pliku Excela, z notatek należy wyodrębnić następujące informacje:

- Kwota zwrotu.
- Powód zwrotu.
- Data zwrotu pieniędzy.

Problem polega na tym, że operatorzy obsługi klienta wprowadzili te informacje bez zastanowienia się nad zasadą ich ustrukturyzowania. Z tego powodu w arkuszu możemy zaobserwować następujące elementy:

- Kwota zwrotu została wprowadzona jako *EUR xx,yy*, *EURxx,yy*, *xx,yy EUR*, *€ xx,yy*, *xx,yy€* i *xx,yy €*.
- „Separator" użyty między poszczególnymi fragmentami informacji może być utworzony za pomocą jednej lub kilku białych spacji, albo przez myślnik otoczony jedną białą spacją lub więcej białymi spacjami.
- Data zwrotu jest zawsze w formacie `dd/mm/rrrr` (tu mamy szczęście!).
- Powód zwrotu może zawierać dowolny tekst.

Czy możliwe jest prawidłowe wyodrębnienie informacji potrzebnych do wykonania analizy wymaganej przez szefa, biorąc pod uwagę dowolność wprowadzonych notatek? Odpowiedź brzmi „z pewnością tak", pod warunkiem, że wiesz, jak używać regeksów.

Jeden regex do zarządzania wszystkimi danymi

Dzięki doświadczeniu, które zebrałeś podczas lektury wcześniejszej części książki, natychmiast wpadniesz na rozwiązanie, które zamierzam Ci zaproponować. Rozważ następujące części regeksa:

- **Waluta**: `(?:EUR|€)`.
- **Separator**: `(?:\s+)?-?(?:\s+)?`.
- **Kwota zwrotu**: `\d{1,}\,?\d{0,2}`.
- **Powód zwrotu**: `.*?`.
- **Data zwrotu**: `\d{2}[\-\/]\d{2}[\-\/]\d{4}`.

Pamiętasz składnię **grupy nieprzechwytującej** `(?:...)`? Dzięki tej składni wyraźnie mówisz silnikowi regeksów, że nie chcesz przechwytywać zawartości w tych nawiasach, ponieważ nie są to ważne informacje do wyodrębnienia. Pamiętaj, że końcowy regex to nic innego jak wiele alternatywnych kombinacji poszczególnych części. Może mieć on postać pokazaną na rysunku 5.37:

```
^(?:
({waluta}{separator}{kwota}{separator}{powód}{separator}{data})
OR
({kwota}{separator}{waluta}{separator}{powód}{separator}{data})
OR
({data}{separator}{waluta}{separator}{kwota}{separator}{powód})
OR
({data}{separator}{kwota}{separator}{waluta}{separator}{powód})
)$
```

Rysunek 5.37. Pełna struktura regeksa do wyodrębniania informacji z notatek

Oto kompletny regex:

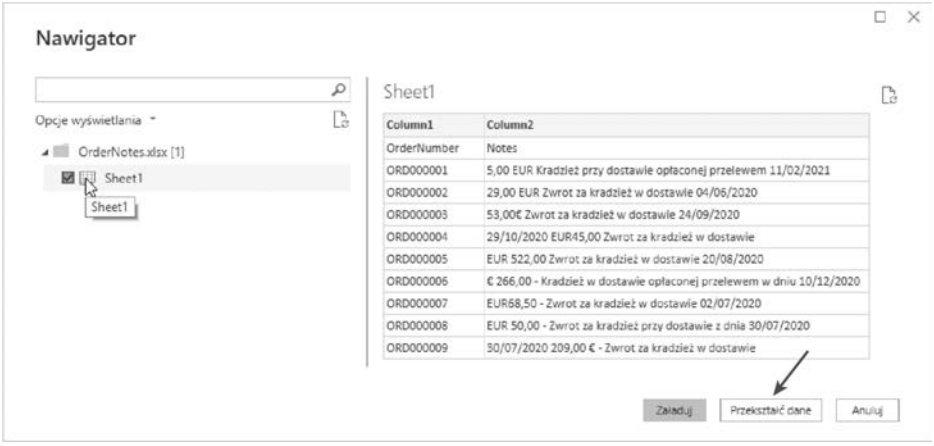
```
^(?:(?:(?:EUR|€)(?:\s+)?-?(?:\s+)?(?P<RefundAmount>\d{1,}\.?\d{0,2})(?:\s+)?-
?(?:\s+)?(?P<RefundReason>.*?)(?:\s+)?-?(?:\s+)?(?P<RefundDate>\d{2}[\-\/]\d{2}[\-
\/]\d{4})(?:\s+)?-?(?:\s+)?)|(?:(?P<RefundAmount>\d{1,}\,?\d{0,2})(?:\s+)?-
?(?:\s+)?(?:EUR|€)(?:\s+)?-?(?:\s+)?(?P<RefundReason>.*?)(?:\s+)?-
?(?:\s+)?(?P<RefundDate>\d{2}[\-\/]\d{2}[\-\/]\d{4})(?:\s+)?-
?(?:\s+)?)|(?:(?P<RefundDate>\d{2}[\-\/]\d{2}[\-\/]\d{4})(?:\s+)?-
?(?:\s+)?(?:EUR|€)(?:\s+)?-?(?:\s+)?(?P<RefundAmount>\d{1,}\.?\d{0,2})(?:\s+)?-
?(?:\s+)?(?P<RefundReason>.*?)(?:\s+)?-?(?:\s+)?)|(?:(?P<RefundDate>\d{2}[\-
\/]\d{2}[\-\/]\d{4})(?:\s+)?-?(?:\s+)?(?P<RefundAmount>\d{1,}\.?\d{0,2})(?:\s+)?-
?(?:\s+)?(?:EUR|€)(?:\s+)?-?(?:\s+)?(?P<RefundReason>.*?)(?:\s+)?-?(?:\s+)?))$
```

Spróbujmy go zaimplementować w usłudze Power BI z wykorzystaniem Pythona.

Używanie regeksów w usłudze Power BI do wyodrębniania wartości z wykorzystaniem języka Python

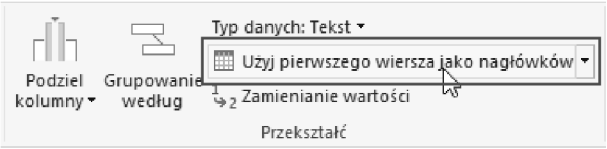
Jak widać na rysunku 5.37, regex zawiera nazwane grupy, które są w nim *używane wielokrotnie*. Niestety ponowne użycie tej samej nazwanej grupy w regeksach nie jest obsługiwane przez moduł Python `re` (ten sam moduł jest również używany za kulisami w pakiecie `pandas`). Aby skorzystać z bardziej zaawansowanych funkcji regeksów, takich jak wcześniej wymienione grupy o identycznych nazwach lub składnie *lookbehind* i *lookahead* (których dotąd nie omówiłem w tym rozdziale), musisz użyć modułu `regex`. Przede wszystkim trzeba go zainstalować w środowisku wirtualnym *pbi_powerquery_env*. Potem trzeba załadować do programu Power BI Desktop plik Excela *OrderNotes.xlsx*, dostępny w folderze *Chapter05*. Następnie można przekształcić ten zestaw danych za pomocą skryptu Pythona. Zaczynajmy:

1. Otwórz program Anaconda Prompt, za pomocą polecenia `conda activate pbi_powerquery_env` przełącz się do środowiska wirtualnego *pbi_powerquery_env*, a następnie zainstaluj pakiet `regex` za pomocą kodu: `pip install regex`.
2. Otwórz program Power BI Desktop i sprawdź w opcjach programu, czy zostało ustawione środowisko wirtualne Pythona *pbi_powerquery_env*.
3. Aby zaimportować dane z pliku Excela *OrderNotes.xlsx* i otworzyć go, kliknij ikonę programu Excel na wstążce.
4. W oknie *Nawigator* wybierz zestaw danych *Sheet1*, po czym kliknij *Przekształć dane* (rysunek 5.38):



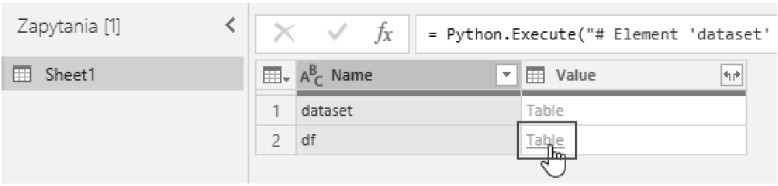
Rysunek 5.38. Ładowanie notatek dotyczących zamówień z arkusza Excela i przekształcanie danych

5. Zadeklaruj pierwszy wiersz załadowanych danych jako nagłówki kolumn. W tym celu kliknij opcję *Użyj pierwszego wiersza jako nagłówków* (rysunek 5.39):



Rysunek 5.39. Przycisk Użyj pierwszego wiersza jako nagłówków

6. Następnie przejdź do menu *Przekształć* i kliknij *Uruchom skrypt języka Python*.
7. Otwórz skrypt Pythona *01-order-notes-parserpython.py* z folderu *Chapter05\extracting-values-fromtext-using-regex*. Skopiuj zawartość skryptu i wklej go do edytora skryptów Pythona, a następnie kliknij *OK*.
8. W przypadku problemów z poziomami zgodności zestawów danych otwórz okno *Ustawienia źródła danych* i za pomocą opcji *Edytuj uprawnienia…* ustaw uprawnienia wszystkich zestawów danych na *Organizacyjne*. Następnie kliknij *Odśwież podgląd*.
9. Interesuje nas tylko zestaw danych `df`. Kliknij więc jego wartość *Table* (rysunek 5.40):



Rysunek 5.40. Wybieranie zestawu danych df otrzymanego w wyniku transformacji za pomocą skryptu Pythona

10. Łatwo zauważyć, że wynikowa tabela zawiera trzy dodatkowe kolumny, z których każda jest powiązana z nazwaną grupą (rysunek 5.41):

	OrderNumber	Notes	RefundAmount	RefundReason	RefundDate
1	ORD000001	5,00 EUR Kradzież przy dostawie opłaconej przelewem 11/02/2021	5	Kradzież przy dostawie opłaconej przelewem	2021-02-11
2	ORD000002	29,00 EUR Zwrot za kradzież w dostawie 04/06/2020	29	Zwrot za kradzież w dostawie	2020-06-04
3	ORD000003	53,00€ Zwrot za kradzież w dostawie 24/09/2020	53	Zwrot za kradzież w dostawie	2020-09-24
4	ORD000004	29/10/2020 EUR45,00 Zwrot za kradzież w dostawie	45	Zwrot za kradzież w dostawie	2020-10-29
5	ORD000005	EUR 522,00 Zwrot za kradzież w dostawie 20/08/2020	522	Zwrot za kradzież w dostawie	2020-08-20
6	ORD000006	€ 266,00 - Kradzież w dostawie opłaconej przelewem w dniu 10/12/20...	266	Kradzież w dostawie opłaconej przelewem w dniu	2020-12-10
7	ORD000007	EUR68,50 - Zwrot za kradzież w dostawie 02/07/2020	68,5	Zwrot za kradzież w dostawie	2020-07-02
8	ORD000008	EUR 50,00 - Zwrot za kradzież przy dostawie z dnia 30/07/2020	50	Zwrot za kradzież przy dostawie z dnia	2020-07-30
9	ORD000009	30/07/2020 209,00 € - Zwrot za kradzież w dostawie	209	Zwrot za kradzież w dostawie	2020-07-30

Rysunek 5.41. Wartości wyodrębnione z notatek zapisanych w formacie dowolnego tekstu za pomocą regeksa oraz kodu w Pythonie

Na koniec przejdź do menu głównego i kliknij *Zamknij i zastosuj*.

Super! Właśnie udało Ci się zreorganizować dane zawarte w notatkach dotyczących zamówień za pomocą regeksów oraz Pythona. Twój szef będzie bardzo zadowolony!

Korzystanie z regeksów w usłudze Power BI do wyodrębniania wartości za pomocą języka R

W języku R do zarządzania nazwanymi grupami, które są wielokrotnie wykorzystywane w tym samym regeksie, możemy używać pakietu `namedCapture`. Aby to zrobić, należy umieścić przed nazwaną grupą modyfikator `(?J)` (pozwala to na współdzielenie tej samej nazwy przez wiele nazwanych grup przechwytujących). W przeciwieństwie do języka Python w języku R funkcja `str_match_named()` z pakietu `namedCapture` nie zwraca jednorazowego wyniku przechwyconego przez nazwaną grupę. Zwraca tyle kolumn, ile razy nazwana grupa została użyta (rysunek 5.42):

	RefundAmount	RefundReason	RefundDate	RefundAmount.1	RefundReason.1	RefundDate.1	RefundDate.2	RefundAmount.2
1				5,00	Kradzież przy dostawie opłaconej przelewem	11/02/2021		
2				29,00	Zwrot za kradzież w dostawie	04/06/2020		
3				53,00	Zwrot za kradzież w dostawie	24/09/2020		
4							29/10/2020	45,00
5	522,00	Zwrot za kradzież w dostawie	20/08/2020					
6	266,00	Kradzież w dostawie opłaconej przelewem w dniu	10/12/2020					
7	68,50	Zwrot za kradzież w dostawie	02/07/2020					
8	50,00	Zwrot za kradzież przy dostawie z dnia	30/07/2020					
9								

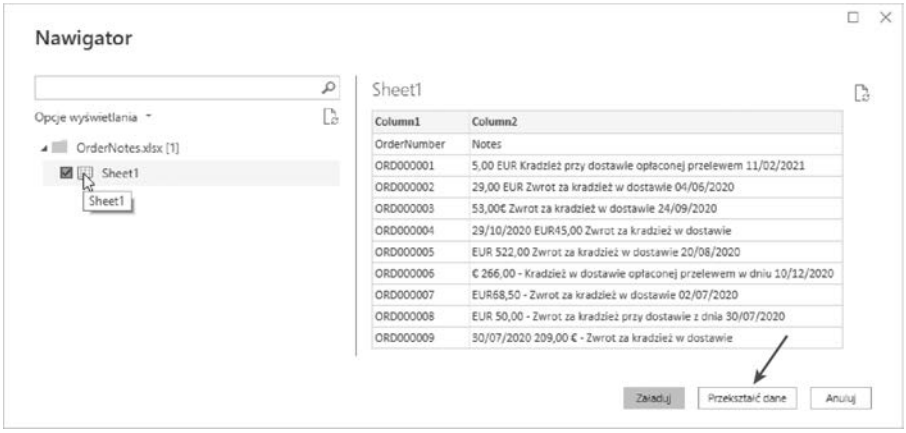
Rysunek 5.42. Funkcja str_match_named() zwraca tyle kolumn, ile razy została użyta nazwana grupa

Z tego powodu trzeba nieco przetworzyć wynik. Po pierwsze, należy zastąpić puste znaki wartością pustą `NA`, a po drugie, zastosować funkcję `coalesce()` z pakietu `dplyr`, która scala wiele kolumn z zachowaniem wartości różnych od `null`.

> **Ważna uwaga**
>
> Zwróciłem uwagę na to ograniczenie Toby'emu Dylanowi Hockingowi, autorowi pakietu `namedCapture`, który niedawno zaimplementował funkcję wewnątrz nowej wersji pakietu o nazwie `nc`. Szczegóły implementacji można znaleźć na stronie *https://github.com/tdhock/namedCapture/issues/15*. W chwili pisania tej książki nowa wersja pakietu `nc` nie została jeszcze opublikowana w dystrybucji CRAN. Dlatego uznałem za stosowne korzystanie z pakietu `namedCapture` w przykładowym kodzie. Możesz jednak swobodnie korzystać w swoich przyszłych projektach z nowego pakietu `nc`.

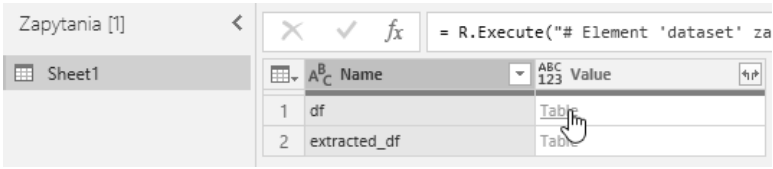
Zacznijmy wyodrębniać wartości z notatek dotyczących zamówień przy użyciu języka R w usłudze Power BI:

1. Otwórz program Power BI Desktop i upewnij się, że korzystasz z najnowszego silnika języka R (w naszym przypadku jest to dystrybucja CRAN R 4.0.2).
2. Aby zaimportować dane z programu Excel, kliknij ikonę Excela na wstążce i otwórz plik *OrderNotes.xlsx* dostępny w folderze *Chapter05*.
3. W oknie *Nawigator* wybierz zestaw danych *Sheet1*, po czym kliknij *Przekształć dane* (rysunek 5.43):



Rysunek 5.43. Ładowanie notatek dotyczących zamówienia z Excela i przekształcanie danych

4. Zadeklaruj pierwszy wiersz załadowanych danych jako nagłówki kolumn. W tym celu kliknij pozycję *Użyj pierwszego wiersza jako nagłówków*.
5. Następnie przejdź do menu *Przekształć* i kliknij *Uruchom skrypt języka R*.
6. Otwórz skrypt języka R z pliku *02-order-notes-parser-r.R* w folderze *Chapter05\extracting-values-from-text-usingregex*. Skopiuj jego zawartość i wklej ją do edytora skryptów R. Następnie kliknij *OK*.
7. Interesuje nas tylko zestaw danych `df`. Kliknij powiązaną z nim wartość *Table* (rysunek 5.44):



Rysunek 5.44. Wybieranie zestawu danych df uzyskanego w wyniku transformacji za pomocą skryptu języka R

8. Łatwo zauważyć, że wynikowa tabela zawiera trzy dodatkowe kolumny, a każda jest powiązana z nazwaną grupą (rysunek 5.35):

	OrderNumber	Notes	RefundAmount	RefundDate	RefundReason
1	ORD000001	5,00 EUR Kradzież przy dostawie opłaconej przelewem 11/02/2021	5,00	11/02/2021	Kradzież przy dostawie opłaconej przelewem
2	ORD000002	29,00 EUR Zwrot za kradzież w dostawie 04/06/2020	29,00	04/06/2020	Zwrot za kradzież w dostawie
3	ORD000003	53,00€ Zwrot za kradzież w dostawie 24/09/2020	53,00	24/09/2020	Zwrot za kradzież w dostawie
4	ORD000004	29/10/2020 EUR45,00 Zwrot za kradzież w dostawie	45,00	29/10/2020	Zwrot za kradzież w dostawie
5	ORD000005	EUR 522,00 Zwrot za kradzież w dostawie 20/08/2020	522,00	20/08/2020	Zwrot za kradzież w dostawie
6	ORD000006	€ 266,00 - Kradzież w dostawie opłaconej przelewem w dniu 10/12/20...	266,00	10/12/2020	Kradzież w dostawie opłaconej przelewem w dniu
7	ORD000007	EUR68,50 - Zwrot za kradzież w dostawie 02/07/2020	68,50	02/07/2020	Zwrot za kradzież w dostawie
8	ORD000008	EUR 50,00 - Zwrot za kradzież przy dostawie z dnia 30/07/2020	50,00	30/07/2020	Zwrot za kradzież przy dostawie z dnia
9	ORD000009	30/07/2020 209,00 € - Zwrot za kradzież w dostawie	209,00	30/07/2020	Zwrot za kradzież w dostawie

Rysunek 5.45. Wartości wyodrębnione z notatek zapisanych w formacie dowolnego tekstu za pomocą regeksa oraz kodu w R

Na koniec przejdź do menu głównego i kliknij *Zamknij i zastosuj*.

Zdumiewające! Właśnie udało Ci się zreorganizować dane zawarte w notatkach dotyczących zamówienia za pomocą regeksów oraz języka R.

Podsumowanie

W tym rozdziale zapoznałeś się z podstawami używania regeksów. Korzystając z absolutnego minimum, skutecznie zweryfikowałeś ciągi reprezentujące adresy e-mail i daty w usłudze Power BI z wykorzystaniem zarówno języka Python, jak i R.

Ponadto dowiedziałeś się, jak za pomocą regeksów wyodrębniać informacje z częściowo ustrukturyzowanych plików logów oraz jak importować wyodrębnione informacje w ustrukturyzowany sposób do usługi Power BI.

Wreszcie nauczyłeś się, jak używać regeksów w Pythonie i R do wyodrębniania informacji z pozornie nieprzetworzonego, dowolnego tekstu na przykładzie notatek związanych z zamówieniami sprzedaży.

W następnym rozdziale dowiesz się, jak używać w usłudze Power BI wybranych technik deidentyfikacji w celu anonimizacji lub pseudonimizacji zestawów danych zawierających wrażliwe dane osobowe w postaci zwykłego tekstu przed ich zaimportowaniem do usługi Power BI.

Bibliografia

Aby uzyskać dodatkowe informacje, zapoznaj się z następującymi książkami i artykułami:

1. Jan Goyvaerts, *Regular Expressions: The Complete Tutorial*, (*https://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf*).
2. *Data Privacy Settings In Power BI/Power Query, Part 1: Performance Implications* (*https://blog.crossjoin.co.uk/2017/05/24/data-privacysettings-in-power-bipower--query-part-1-performanceimplications/*).

Skorowidz

A

B

C

D

E

F

G

H

I

N

O

P

R

S

Ś

T

U

V

W

X

Y

Z

Ż

Wyzwól potężną moc Power BI!

Ważnym zadaniem inżynierów danych jest kreowanie modeli uczenia maszynowego. Używa się do tego narzędzi do analizy biznesowej, takich jak Power BI. Możliwości Power BI są imponujące, a można je dodatkowo rozbudować. Jedną z ciekawszych metod wzbogacania modelu danych i wizualizacji Power BI jest zastosowanie złożonych algorytmów zaimplementowanych w językach Python i R. W ten sposób można nie tylko tworzyć interesujące wizualizacje danych, ale także pozyskiwać dzięki nim kluczowe dla biznesu informacje.

Dzięki tej książce dowiesz się, jak to zrobić. Zaczniesz od przygotowania środowiska Power BI do używania skryptów w Pythonie i R. Następnie będziesz importować dane z nieobsługiwanych obiektów i przekształcać je za pomocą wyrażeń regularnych i złożonych algorytmów. Nauczysz się wywoływać zewnętrzne interfejsy API i korzystać z zaawansowanych technik w celu przeprowadzenia dogłębnych analiz i wyodrębnienia cennych informacji za pomocą narzędzi statystyki i uczenia maszynowego, a także poprzez zastosowanie optymalizacji liniowej i innych algorytmów. Zapoznasz się również z głównymi cechami statystycznymi zestawów danych i z metodami tworzenia różnych wykresów ułatwiających zrozumienie relacji między zmiennymi.

Najciekawsze zagadnienia:

- złożone przekształcanie danych w Power BI za pomocą skryptów Pythona i R
- anonimizacja i pseudonimizacja danych
- praca z dużymi zestawami danych
- wartości odstające i brakujące dla danych wielowymiarowych i szeregów czasowych
- tworzenie złożonych wizualizacji danych

Luca Zavarella zdobył certyfikat Azure Data Scientist Associate i tytuł Microsoft MVP w zakresie sztucznej inteligencji. Od ponad 10 lat zajmuje się Microsoft Data Platform. Posiada praktyczną wiedzę o stosie technologicznym Microsoft Business Intelligence (SSIS, SSAS, SSRS) i technikach hurtowni danych. Ostatnio interesuje się zaawansowanymi zagadnieniami analitycznymi i technikami data science. W wolnym czasie gra na fortepianie klasycznym.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-9453-7

Cena: 119,00 zł

Packt