Institut biostatistiky a analýz
Masarykova univerzita

Sborník 11. letní školy matematické biologie

Předzpracování dat v databázových systémech a v systému R, algoritmizace a programovací nástroje pro zpracování dat

16.–18. září 2015
Brno

Editor: Jiří Hřebíček

Institut biostatistiky a analýz
Masarykova univerzita

Sborník 11. letní školy matematické biologie

Předzpracování dat v databázových systémech a v systému R, algoritmizace a programovací nástroje pro zpracování dat

16.–18. září 2015
Brno

Editor:
Jiří Hřebíček

ISBN 978-80-210-7924-3

Obsah

Úvod

Letní školy matematické biologie jsou jedinečnou příležitostí pro setkání a posílení spolupráce mezi učiteli, mladými vědci i studenty matematické biologie. Letní školy matematické biologie se konají každý rok již od roku 2005. Studenti se v nich mohli účastnit přednášek předních domácích i zahraničních odborníků i neformálních diskuzí o nových metodách v oboru matematické biologie a příbuzných vědních oborech. Nedílnou součástí letních škol byla vždy aktivní účast studentů, kteří prezentovali své výsledky dosažené v rámci bakalářských a magisterských prací.

Témata a programy jednotlivých ročníků letních škol matematické biologie jsou k dispozici na jejich webových stránkách:

- 2005 – Matematická biologie
- 2006 – Prediktivní modelování a ICT v environmentální epidemiologii
- 2007 – Zpracování a analýza dat o biodiverzitě: od genomické diverzity ke struktuře ekosystémů
- 2008 – Statistické metody pro genetická a molekulární data
- 2009 – Analýza klinických a biomedicínských dat v mezioborovém pojetí
- 2010 – Deterministické a stochastické modelování v biologii a medicíně
- 2011 – Biodiverzita od genetiky po geografii a od matematiky po management
- 2012 – Od analýzy genomických dat ke klinické aplikaci – příkladové studie
- 2013 – Stochastické modelování v epidemiologii
- 2014 – Analýza a zpracování obrazových dat v neurovědách

Letošní 11. letní škola matematické biologie, navazuje na tradici předchozích letních škol a je zaměřena na téma:

Předzpracování dat v databázových systémech a v systému R, algoritmizace a programovací nástroje pro zpracování dat

se bude konat ve dnech 16.–18. září 2015 v počítačové učebně IBA na Kamenici 126/3 v Kampusu Masarykovy university v Bohunicích.

Je v ní kladen důraz na praktická cvičení studentů se zpracováním reálných „nepředzpracovaných“ dat, která vyžadují před jejich analytickým zpracováním řádné „pročištění“ a transformace. To je velmi důležité pro pochopení, zefektivnění a usnadnění práce s těmito daty. Proto bude na začátku letní školy představen příkladový biomedicínský problém s hypotézami a požadovanými výstupy analýzy a sadou datových souborů pro zpracování. Tyto soubory dat budou použity napříč všemi přednáškami s praktickým procvičením na osobních počítačích účastníků.

Součástí letní školy je studentská soutěž, která proběhne ve čtvrtek 17. září 2015 odpoledne a bude spočívat ve zpracování typického datového souboru za pomocí probíraných technik. Rozhodovat bude čas a kvalita zpracování ve zvolené technice.

V pátek 18. září 2015 budou představena vybraná řešení úkolu ze studentské soutěže, objasněny, vyhodnoceny a porovnány studentská řešení ve zvolených technikách a budou oceněni nejlepší studenti.

Nedílnou součástí letních škol jsou jejich sborníky, které obdrží její účastnící. V letošním sborníku jsou ve třech příspěvcích představeny dostupné nástroje a varianty řešení zpracování těchto dat různými technikami. U představených technik jsou zmíněny taktéž

jejich limity a slabá místa. Rozebrány jsou i možnosti migrace dat mezi softwarovými nástroji nebo možnost jejich integrace.

Prof. RNDr. Jiří Hřebíček, CSc.

Zpracování dat v databázovém systému PostgreSQL

Daniel Klimeš

Institut biostatistiky a analýz, Masarykova univerzita, Brno; e-mail: klimes@iba.muni.cz

Abstrakt

Příspěvek představuje základní práci s daty v relační databázi PostgreSQL (verze 9.4). Praktické příklady ukazují způsob importu dat z textového souboru, čištění dat pomocí integrovaných funkcí. Popsány jsou základy databázového procedurálního jazyka PLPGSQL.

Klíčová slova

Relační databáze, PostgreSQL, SQL, PLPGSQL

1. Úvod

Pro úvodní zpracování či předzpracování dat můžeme použít mnoho softwarových prostředků. Využít k tomuto úkolu databázové prostředky se vyplatí v případě:

1. Data jsou primárně v databázi uložena
2. Zpracováváme objemná data v řádu sto tisíc záznamů a více
3. Zpracování dat plánujeme provádět opakovaně
4. S daty bude pracovat více uživatelů

Pokud platí některý z výše uvedených bodů, je namístě provést částečné nebo kompletní zpracování dat přímo v databázi. Databázových systémů existuje celá řada, v rámci tohoto textu si ukážeme práci s opensource databází PostgreSQL, která patří ke špičce mezi zdarma dostupnými systémy [1].

Pokud data již primárně nesbíráme v naší databázi, je prvním krokem předzpracování dat import do databáze. Metod, jak dostat data do databáze je více, a to v závislosti na formátu či zdroji vstupních dat. My se podíváme na základní přístup, kterým je import dat z textového souboru.

Nejprve ale musíme znát základní fakta o relačních databázích, mezi které PostgreSQL patří.

- Data jsou v relačních databázích uložena v tabulkách, kde řádek představuje jednotlivé záznamy a sloupce jednotlivé atributy. Před importem dat musíme vytvářet tabulku s odpovídající strukturou.
- Při vytváření tabulky přiřazujeme sloupcům jméno a datový typ. Datový typ určuje, jaká data bude moci sloupec obsahovat. Základní datový typy jsou uvedeny v tabulce Tabulka 1.
- Software víceuživatelské databáze je rozdělen na serverovou část a část klientskou. Pro přístup do databáze potřebujeme instalovat klientskou část a znát umístění serveru, přístupový login a heslo, případně název cílové databáze. V případě lokální instalace serveru je umístění „localhost“, výchozí login je „postgres“, výchozí heslo k tomuto účtu volíme při instalaci serveru a výchozí databáze má jméno opět „postgres“. Klientských aplikací je více, mezi nejpoužívanější patří řádková aplikace **psql** a grafický klient **pgAdmin**.

Tabulka 1. Základní datové typy

Obecný typ	POSTGRESQL
Text	VARCHAR (max. délka), TEXT
Číslo	NUMERIC (číslic, des.míst)
Datum	Date, Timestamp

TIP: Ve výchozím nastavení v prostředí MS Windows má řádkový klient psql problém s češtinou. Konfiguraci provedeme nejprve ve startovacím skriptu runpsql, kam před příkaz **for** dopíšeme řádek:

cmd.exe /c chcp 1250

Následně po spuštění skriptu ve vlastnostech okna (pravé tlačítko myši v záhlaví okna) nastavíme některý z TrueType fontů a zvolíme jeho rozumnou velikost. Otestujeme výsledek:

```
SELECT 'ěščřžýáíéů' a;
```

2. Import dat

Po instalaci serveru a zkrocení psql klienta můžeme začít řešit vlastní import dat. Pokud jde o textové formáty vstupních dat, potkáváme se nejčastěji buď s formátem s oddělovačem nebo s pozicovým formátem. V obou případech je jeden záznam reprezentován jedním řádkem, v prvním případě jsou ale atributy odděleny specifickým znakem, v druhém případě mají sloupce fixní počet znaků.

Formát s oddělovačem (středník jako oddělovač)

```
1;Karel;12.3.2001
34;Jan;23.11.1990
245;Roman;2.4.2012
```

Fixní pozicový formát

```
1  Karel 12.3.2001
34 Jan   23.11.1990
245Roman 2.4.2012
```

Textový soubor lze do PostgreSQL importovat pomocí COPY příkazu ve variantě:

```
COPY cilova_tabulka FROM 'zdrojovy_soubor';
```

Podporován je import textového souboru, kde sloupce jsou odděleny specifikovaným jednoznakovým oddělovačem. Ten specifikujeme za klíčovým slovem DELIMITER, výchozím oddělovacím znakem je tabulátor, který nemusíme explicitně specifikovat. V jiných případech je nutné příkaz COPY uvést včetně oddělovače:

COPY patients FROM 'C:/zdroj.txt' DELIMITER ';'[1]

Příkaz COPY vyžaduje, aby v databázi již existovala cílová tabulka. Pokud tedy chceme použít tento příkaz, musíme přesně znát složení vstupního souboru a podle toho vytvořit tabulku. Vystačíme se znalostmi uvedených datových typů a příkazem CREATE TABLE.

```
CREATE TABLE cilova_tabulka (
        Id NUMERIC(10),
        Jmeno VARCHAR (30),
        Datum DATE
);
```

Dalším důležitým parametrem příkazu COPY je určení, jak jsou ve vstupním souboru reprezentovány chybějící hodnoty. K tomuto použijeme klíčové slovo NULL. Častý případ je, že chybějící hodnoty se prostě vynechají a zůstanou jen 2 oddělovače:

46;;3.1.1980

Pro tuto variantu bude příkaz COPY vypadat následovně:

COPY patients FROM 'C:/zdroj.txt' DELIMITER ';' NULL" - - ***2 apostrofy***

U příkazu COPY nemusíme řešit, zda konce řádku vstupního textového souboru jsou symbolizovány znaky CRLF nebo jen CR. Příkaz řeší sám obě varianty.

Obvyklý problém při zpracování dat nejen v databázích bývá zpracování českých diakritických znaků. Tyto znaky se často buď nezpracují vůbec, nebo se chybně zobrazují a vyřešení tohoto problému dokáže zabrat více času než vlastní zpracování dat.

Pro české znaky existuje v počítačovém světě několik znakových snad, což ještě dále komplikuje situaci, neboť musíme dopředu vědět, jaká znaková sada byla použita ve vstupním souboru. Databáze PostgreSQL podporuje 3 nejčastější české znakové sady, jejich přehled je uveden v tabulce Tabulka 2.

Tabulka 2. Podporované jazykové sady v databázi PostgreSQL

Označení (PostgreSQL)	Alternativní jméno
UTF8	Unicode, 8-bit
WIN1250	Windows CP1250
LATIN2	ISO 8859-2, ECMA 94

Pokud náš vstupní soubor obsahuje české diakritické znaky a známe použité kódování, voláme příkaz COPY následovně:

```
COPY patients FROM 'C:/zdroj.txt' DELIMITER ';' NULL " ENCODING
'WIN1250'
```

[1] V příkladu je skutečně normální, nikoliv zpětné lomítko.

Další častý problém způsobuje formát datumových položek. PostgreSQL automaticky podporuje tyto formáty:

- dd.mm.rrrr - 1.10.2010
- rrrr-mm-dd - 2010-10-01
- rrrrmmdd – 20101001

Příkazem COPY importujeme vždy všechny sloupce zdrojového souboru. Pokud cílová tabulka obsahuje i další sloupce nebo jsou sloupce vytvořeny v jiném pořadí, než odpovídá zdrojovému souboru, specifikujeme seznam sloupců přímo v příkazu COPY:

```
COPY patients (id, jmeno, datum) FROM 'C:/zdroj.txt' DELIMITER ';' NULL ''
ENCODING 'WIN1250'
```

Problém s importem pomocí příkazu COPY nastane v případě:

1. Fixní pozice sloupců
2. Neznámá a velmi rozsáhlá struktura vstupního souboru
3. Oddělovač sloupců je zároveň použit v přenášených datech

V těchto případech lze doporučit importovat celý řádek (všechny atributy) do jednoho obecného sloupce a teprve v druhém kroku pomocí databázových funkcí rozebrat záznam do požadovaných sloupců. Pro tento obecný import potřebujeme pouze specifikovat znak, který se v datech NEVYSKYTUJE. Tento znak použijeme jako DELIMITER příkazu COPY. Pokud data neobsahují znak tabulátoru, můžeme klíčové slovo DELIMITER vynechat, neboť jde o defaultní hodnotu příkazu COPY. Nejprve je nutné vytvořit univerzální tabulku.

```
CREATE TABLE all_in_one
(
   vse TEXT
);
```

Do této tabulky přeneseme libovolný textový soubor následujícím příkazem

```
COPY all_in_one from 'C:/data_fix_win.txt' ENCODING 'WIN1250';
```

Data s fixní pozicí sloupců pak můžeme z univerzální tabulky dolovat pomocí funkce SUBSTR(), která má 3 parametry. Prvním je název sloupce, druhým pořadí prvního znaku, který nás zajímá, a třetím parametrem je počet znaků, které chceme extrahovat. Pokud neuvedeme třetí parametr, extrahuje se text do konce řetězce. Víme-li, že v souboru je ID pacienta uloženo vždy v prvních třech znacích, můžeme je vyextrahovat následovně:

```
SELECT SUBSTR(vse, 1, 3) FROM all_in_one;
```

Příkaz SELECT je základní databázový příkaz, který vybírá a zobrazuje data tabulek. Jeho součástí je klíčové slovo FROM, za kterým následuje název tabulky.

Obdobně extrahujeme další sloupce. Pozice sloupců v lepším případě známe, v horším je musíme tipovat pohledem do zdrojového souboru.

```
SELECT SUBSTR(vse, 1, 3), SUBSTR(vse, 4, 6), SUBSTR (vse, 10) FROM
all_in_one;
```

Pokud chceme data přenést do tabulky se samostatnými sloupci, stačí skombinovat příkaz SELECT s příkazem CREATE TABLE a upřesnit databázi cílové datové typy:

```
CREATE TABLE cilova AS SELECT SUBSTR(vse, 1, 3)::NUMERIC(3) id,
SUBSTR(vse,4,6) jmeno, SUBSTR(vse,10)::DATE datum FROM all_in_one;
```

Přes univerzální tabulku můžeme naimportovat i data s oddělovačem. Pro extrakci sloupců můžeme pak použít funkci SPLIT_PART(). Prvním parametrem této funkce je opět název sloupce, druhým je znak oddělovače sloupců, třetím parametrem je pořadí extrahovaného sloupce.

```
SELECT SPLIT_PART(vse, CHR(9), 1) id, SPLIT_PART(vse,CHR(9),2) jmeno,
SPLIT_PART(vse,CHR(9),3) datum FROM all_in_one;
```

Funkce CHR(9) reprezentuje znak tabulátoru, ostatní standardní znaky lze uvést přímo v apostrofech. Extrakcí můžeme do cílových tabulek přenést jen sloupce, které budeme dále potřebovat při vlastní analýze.

Nejproblematičtější je situace, kdy oddělovač je zároveň i součástí některého textového sloupce. Z tohoto důvodu se textové hodnoty navíc uzavírají v přenášených souborech do uvozovek nebo apostrofů. Data v souboru pak vypadají následovně:

```
245; "Jan;Amos";28.3.1592

246; "Roman";2.4.2012
```

V tomto případě opět naimportujeme data do univerzální tabulky a extrakci provedeme s využitím regulárních výrazů a funkce REGEXP_MATCHES(). Regulární výrazy jsou, zjednodušeně řečeno, nástrojem pro vyhledávání vzorů v textových řetězcích. Pomocí zástupných znaků, kvantifikátorů a modifikátorů hledáme požadovaný vzor. Detaily lze nastudovat z internetových zdrojů, případně v publikaci [2]. Parametrem funkce REGEXP_MATCHES je prohledávaný sloupec a vlastní regulární výraz.

Pro náš případ pro vytažení kompletního jména uzavřeného v uvozovkách v druhém sloupci souboru by posloužil například výraz :

```
SELECT REGEXP_MATCHES (vse, '.*;"(.*)";') FROM all_in_one;
```

Hledáme text uzavřený v uvozovkách, kterému jeden středník předchází a druhý za ním následuje. Výsledkem funkce je to, co „uvízne“ v kulatých závorkách (označeno tučně). Symbol .* symbolizuje jakékoliv znaky v jakémkoliv množství.

Import dat ze souboru do databáze je proces, který dokáže připravit nemálo horkých chvil. Jedním ze zvlášť zákeřných stavů je import textového souboru s tzv. BOM. BOM je zkratka „Byte order mark“, česky přibližně „označení pořadí bajtů. V kódování UTF-8 je tento znak reprezentován trojicí bajtů 0xEF 0xBB 0xBF (UTF-8 signatura). Grafický význam znaku je „nedělitelná mezera nulové šířky“ (zero-width no-break space). Přestože je to nadbytečné, mnohé aplikace operačního systému Microsoft Windows používají tento znak na začátku souboru pro rozlišení souborů uložených ve formátu UTF-8. Problém nastane při importu takovéhoto souboru do PostgreSQL databáze pomocí příkazu COPY. V tomto případě se tento „neviditelný“ znak stane součástí vkládaného textu prvního řádku v prvním sloupci a začnou se objevovat „nevysvětlitelné“ chyby. Připojený BOM znak se totiž nijak nezobrazí, ale při porovnávání řetězců se chová jako platný znak, a my se pak dozvídáme, že „A“ se nerovná „A“ nebo že 123 není číslo. Způsob, jak se tohoto znaku zbavit, je následující:

```
UPDATE tabulka SET sloupec = SUBSTR (sloupec, 2) WHERE
POSITION('\xefbbbf'::bytea IN CONVERT_TO(sloupec,'UTF-8'))=1
```

Tento příkaz vyhledá řádek s BOM a odstraní ho z postiženého sloupce.

3. Čištění dat

Jakmile máme data uvnitř databáze, můžeme se pustit do průzkumu a čištění. V následujícím textu budou ukázány základní operace, které nám pomohou seznámit se s daty a odstranit či opravit případné chyby. Jedná se čistě o náhled do funkcí relační databáze, plná funkcionalita je mnohem širší.

Pro seznámení se s obsahem tabulky a jejich sloupců využijeme příkaz SELECT a dostupné funkce a operátory.

Obsah celé tabulky si lze prohlédnout příkazem:

```
SELECT * FROM tabulka;
```

Počet řádků v tabulce zobrazí příkaz:

```
SELECT COUNT(*) FROM tabulka;
```

Unikátní hodnoty ve sloupci tabulky zobrazíme takto:

```
SELECT DISTINCT sloupec FROM tabulka;
```

Pokud chceme zjistit, zda sloupec obsahuje v každém řádku unikátní hodnotu, použijeme:

```
SELECT COUNT(DISTINCT sloupec), COUNT(*) FROM tabulka;
```

Pokud jsou zobrazená čísla shodná, sloupec neobsahuje žádnou duplicitu.

Pokud chceme zjistit, v kolika případech je sloupec neprázdný, použijeme:

```
SELECT COUNT(sloupec) FROM tabulka;
```

Zjištění zastoupení kategorií v konkrétním sloupci:

```
SELECT sloupec, COUNT(*) FROM tabulka
GROUP BY sloupec;
```

Nad číselnými sloupci můžeme snadno zjistit aritmetický průměr, směrodatnou odchylku, minimum, maximum:

```
SELECT AVG(sloupec), STDDEV(sloupec), MIN(sloupec), MAX(sloupec) FROM
tabulka;
```

Aby předchozí příkaz fungoval, musí být sloupec datového typu NUMERIC. Pokud jsme naimportovali čísla do sloupce s datovým typem VARCHAR nebo TEXT, pomůžeme si přetypováním:

```
SELECT AVG(sloupec::NUMERIC), STDDEV(sloupec::NUMERIC),
MIN(sloupec::NUMERIC), MAX(sloupec::NUMERIC) FROM tabulka;
```

Přetypování funguje pouze v případě, že všechny řádky skutečně obsahují konverze schopné hodnoty. Stačí jeden chybný řádek, který místo čísla obsahuje text, a konverze selže.
V tomto případě začíná hledání problému. Relační databáze překvapivě ne vždy nabízí přímočaré testování hodnoty sloupce na číslo, případně datum. PostgreSQL (obdobně ORACLE) doporučuje pro testování hodnot nadefinování vlastní funkce:

```
CREATE OR REPLACE FUNCTION is_numeric(text) RETURNS BOOLEAN AS
$$

DECLARE x NUMERIC;
```

```
BEGIN
   x = $1::NUMERIC;
   RETURN TRUE;
EXCEPTION WHEN OTHERS THEN
   RETURN FALSE;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

Tato uživatelská funkce zkusí provést konverzi vstupního textového řetězce na číslo, a pokud selže, obslouží vyvolanou výjimku (exception).

Použití funkce je snadné. Pokud chceme zobrazit řádky, které obsahují nečíselnou hodnotu, napíšeme:

```
SELECT sloupec FROM tabulka WHERE NOT (is_numeric(sloupec));
```

Tyto nežádoucí řádky můžeme odstranit pomocí příkazu DELETE:

```
DELETE FROM tabulka WHERE NOT (is_numeric(sloupec));
```

Konverze může selhat také v případě, že čísla jsou v českém standardu s desetinnou čárkou místo desetinné tečky. V tomto případě si pomůžeme příkazem UPDATE a funkcí REPLACE:

```
UPDATE tabulka SET sloupec = REPLACE (sloupec, ', ', '. ');
```

České diakritiky se můžeme zbavit pomocí funkce TRANSLATE:

```
UPDATE tabulka SET sloupec  = TRANSLATE('áčďěéíňóřšťúůýž',
'acdeeinorstuuyz');
```

Funkce REPLACE a TRANSLATE slouží k odstranění či opravě nejrůznějšího „smetí“. Pokud bychom potřebovali odstranit z textu apostrof, který se standardně v příkazech databáze používá k ohraničení textových konstant, musíme ho v příkazu zdvojit:

```
UPDATE tabulka SET sloupec = REPLACE (sloupec, "", '');
```

Obsah textových sloupců můžeme rozšířit o konstantní text pomocí operátoru ||. Následující příklad ozávorkuje obsah sloupce:

```
SELECT '(' || sloupec || ')' FROM tabulka;
```

Pro zpracování čísel využijeme zaokrouhlovací funkce ROUND (klasické zaokrouhlení) a TRUNC (oříznutí desetinné části).

Podporovány jsou základní aritmetické operace (+-*/). Pozor pouze u dělení, pokud dělíme 2 celá čísla, výsledek je oříznutý o desetinnou část. Pokud chceme vynutit výsledek včetně desetin, použijeme u konstant následující trik:

```
SELECT 5/2 , 5/2.0
```

Výsledek je v prvním případě 2, v druhém 2.5.

Častým úkolem při předzpracování je kategorizace spojitých dat, kdy ze spojitých hodnot v jednom sloupci vytváříme kategorie ve sloupci druhém. U pravidelných intervalů můžeme použít celočíselné dělení. Například pětileté věkové kategorie vytvoříme takto:

```
UPDATE tabulka SET vek_kategorie = TRUNC(vek / 5);
```

Pro nepravidelné intervaly použijeme podmíněný výraz CASE WHEN:

```
UPDATE tabulka SET vek_kat =
    CASE   WHEN vek < 20 THEN 0
           WHEN vek < 50 THEN 1
           WHEN  vek < 60 THEN 2
           ELSE 3
    END;
```

U podmíněného výrazu se podmínky vyhodnocují postupně, vyhodnocování končí u první splněné podmínky.

Pro porovnávání čísel máme k dispozici operátory (=, <>, <, >, >=, <=);

Nežádoucí řádky s extrémy odstraníme příkazem DELETE:

```
DELETE FROM tabulka WHERE sloupec < 0 OR sloupec > 10000);
```

Příkazy DELETE a UPDATE pracují s těmi řádky, které vyhoví podmínce za klíčovým slovem WHERE. Pokud WHERE vynecháme, aplikuje se operace na všechny řádky. Kompletní smazání tabulky provedeme (občas neúmyslně) příkazem:

```
DELETE FROM tabulka;
```

Na problém můžeme narazit, pokud chceme smazat či opravit právě jeden konkrétní řádek, který objevíme v celkovém přehledu po provedení SELECT příkazu. Pro podmínku WHERE potřebujeme jedinečný identifikátor řádku, který v databázové terminologii označujeme jako primární klíč. Ten může být součástí zdrojových dat jako ID sloupec, který obsahuje pro každý řádek unikátní hodnotu. Ten využijeme pro práci s konkrétním řádkem či řádky:

```
UPDATE tabulka SET sloupec = 234 WHERE id = 11;
nebo
UPDATE tabulka SET sloupec = 234 WHERE id IN (11,14,56);
```

Pokud id sloupec není součástí importovaných dat, musíme si ho po importu vytvořit. Využijeme funkci ROW_NUMBER():

```
CREATE TABLE tabulka2 AS
SELECT ROW_NUMBER () OVER () id, tabulka.* FROM tabulka;
```

Nyní máme každý řádek v nové tabulce jednoznačně identifikován.

Velmi užitečné jsou funkce pro práci s datumy. Pokud máme datum uloženo ve sloupci s datovým typem TIMESTAMP, můžeme od tohoto sloupce odečítat a přičítat čísla, jejichž celá část reprezentuje přičítané či odečítané dny, desetinná část pak reprezentuje časový interval v rámci jednoho dne. Pokud chceme k datumovému sloupci přičíst 2 dny a 3 hodiny, použijeme:

```
SELECT sloupec_datum + 2 + 3.0/24 FROM tabulka;
```

Pokud potřebujeme vědět, kolik dní uplynulo od uloženého data, použijeme funkci CURENT_DATE(), která vrací systémové datum :

```
SELECT  CURENT_DATE() – sloupec_datum FROM tabulka;
```

Pokud používáme v operacích s datumem datumové konstanty, můžeme sice použít podporované formáty:

```
SELECT CURRENT_DATE - '1.2.2015'
```

Nicméně riskujeme, že dojde k záměně dne a měsíce díky lokálnímu nastavení systému a dostaneme chybné výsledky. Proto je doporučeno u datumových konstant vždy používat funkci TO_DATE s explicitním uvedením formátu:

```
SELECT CURRENT_DATE - TO_DATE('1.2.2015','dd.mm.yyyy')
```

Velmi častou operací je stanovení přesného věku v letech. Zde narazíme na problém přestupných roků, kvůli kterým nemůžeme vypočtený rozdíl ve dnech podělit 365. Řešením jsou databázově specifické funkce, u PostgreSQL je to AGE(), u ORACLE pak MONTHS_BETWEEN(). Funkce AGE() vrací výsledek jako specifický typ INTERVAL, proto pro extrakci roku použijeme funkci EXTRACT:

```
SELECT EXTRACT('year' FROM AGE(sloupec_datum)) FROM tabulka
```

Výhodou databázového zpracování je ve snadném skriptování čistících příkazů, kde jednotlivé příkazy jsou odděleny středníkem:

```
DELETE FROM tabulka WHERE sloupec < 0 OR sloupec > 10000);
UPDATE tabulka SET sloupec  = TRANSLATE('áčďěéíňóřšťúůýž',
'acdeeinorstuuyz');
UPDATE tabulka SET sloupec = REPLACE (sloupec, "", '');
….
```

Skript uložíme do textového souboru a spustíme v psql:

```
\i script.sql
```

Pomocí skriptu máme veškeré operace zdokumentované a hlavně je může kdykoliv opakovaně spustit nad novými daty.

3.1. Transakce

Databázové příkazy UPDATE a DELETE jsou velmi mocné prostředky pro hromadné čištění dat. Nicméně stačí drobná chyba a veškerá naše data jsou zničena. Pokud náš postup skriptujeme, můžeme při chybě vždy začít znovu, při velkých objemech dat to ale může být značně zdržující. Je proto dobré vědět o možnosti transakčního zpracování. Primárním účelem databázových transakcí je udržet konzistenci dat, dát možnost uživateli provést sérii změn, kdy ve finále buď budou provedeny všechny změny, nebo žádná. Pokud pracujeme v transakci, námi prováděné změny jsou viditelné pouze pro nás, nikoliv pro ostatní uživatele, a to až do té doby než provedeme potvrzení (commit) transakce. Pokud narazíme při provádění změn na problém (chybně provedený UPDATE či DELETE), můžeme transakci odvolat (rollback). Databáze pak vše obnoví do stavu na začátku naší transakce. Stinnou stránkou transakcí je, že blokují práci ostatních uživatelů, pokud také pracují se stejnými tabulkami a záznamy jako my. Z tohoto důvodu při práci ve víceuživatelském prostředí je nutné provádět transakce co nejkratší a co nejrychleji. Klientské aplikace PostgreSQL jsou ve výchozím nastavení nastaveny do tzv, auto-commit režimu, kdy každý SQL příkaz je po spuštění okamžitě a automaticky potvrzen a při případné chybě není cesty zpět. Pokud chceme používat delší vícepříkazové transakce, musíme zahájit práci v PostgreSQL příkazem BEGIN TRANSACTION. Všechny následné UPDATE a DELETE příkazy lze následně buď všechny potvrdit příkazem COMMIT nebo zrušit příkazem ROLLBACK.

3.2. Propojení tabulek

Zdrojová data se obvykle skládají z více zdrojových souborů, které importujeme do samostatných tabulek. K propojování záznamů mezi tabulkami používáme výraz JOIN. Rozlišujeme vnitřní a vnější spojení. U vnitřního spojení jsou výsledkem pouze řádky, pro které vazba existuje (průnik). U vnějšího spojení je výsledkem vždy celý obsah jedné ze spojovaných tabulek, ke kterému se připojí záznamy z tabulky druhé.

Vnitřní spojení:

```
SELECT * FROM tabulka1 JOIN tabulka2 ON tabulka1.sloupec = tabulka2.sloupec
```

Vnější spojení (všechny záznamy z tabulky 1):

```
SELECT * FROM tabulka1 LEFT JOIN tabulka2 ON tabulka1.sloupec =
tabulka2.sloupec
```

Za slovem ON definujeme propojovací podmínku, podle níž se budou řádky párovat. Pokud ji vynecháme, páruje se každý řádek tabulky1 s každým řádkem tabulky 2 (tzv. kartézský součin).

4. Procedurální programování v PostgreSQL

Pokud nevystačíme při čištění a práci s daty s jednoduchými SQL příkazy, nabízí databáze navíc procedurální jazyk. V případě PostgreSQL jde o plpgsql jazyk, který nabízí standardní programovací prvky:

- Proměnné
- Podmíněný výraz
- Programová smyčka
- Volání jiných procedur či funkcí
- Zpracování výjimek

Pomocí procedurálního kódu definujeme uživatelské funkce. Vlastní kód je vždy ohraničen klíčovými slovy BEGIN na začátku a END na konci. Pro oddělení jednotlivých příkazů se používá středník (;) včetně finálního slova END. Za klíčovým slovem BEGIN se naopak středník nevkládá. BEGIN a END vymezují tzv. blok kódu, který může obsahovat další zanořený blok opět ohraničený slovy BEGIN a END. V procedurálním kódu se využívají tzv. proměnné pro uchování a přenos zpracovávaných hodnot. Proměnná je obdobou sloupce tabulky, má své jméno a datový typ, základní typy proměnných však mohou přenášet v danou chvíli jen jednu hodnotu. Proměnné musíme deklarovat za klíčovým slovem DECLARE ještě před úvodním slovem BEGIN, kde uvedeme název proměnné a její datový typ oddělený mezerou, jednotlivé proměnné oddělujeme středníkem:

```
DECLARE
i NUMERIC (5);
str VARCHAR (100);
BEGIN
....
END;
```

Proměnné přiřadíme hodnotu pomocí operátoru přiřazení, kterým je ":=". Pomocí standardních operátorů můžeme provádět základní aritmetické operace. Proměnné můžeme využívat na místě konstant v SQL příkazech:

```
DECLARE
i NUMERIC (5);
str VARCHAR (100);
BEGIN
        i:=1; str := 'text';
        i:= (i-14) * 9875  + 456;
        str := str || ' pripojeny text';
        DELETE FROM tab WHERE sloupec = i AND sloupec2 = str;
        INSERT INTO tab2 (sloupec, sloupec2) VALUES (i, str);
END;
```

Jednou ze základních technik v procedurálním programování je větvení kódu dle definované podmínky pomocí konstrukce IF-THEN-ELSE. V prostředí PL/SQL má podmínková konstrukce následující tvar:

```
IF podminka THEN
        prikaz1;
ELSE
        prikaz2;
END IF;
```

Pokud je podmínka splněna provede se příkaz1 (obecně sada příkazů mezi THEN a ELSE), pokud podmínka platná není, provede se příkaz 2 (sada příkazů mezi ELSE a END). Část ELSE je nepovinná, naopak při potřebě vyhodnotit postupně více podmínek, můžeme konstrukci rozšířit o ELSEIF (deklarační část proměnných leu a grade je vynechána):

```
IF leu  >= 3000 THEN
    grade  := 'I';
ELSIF leu < 3000 AND leu >= 2000 THEN
    grade := 'II';
ELSIF leu < 2000 and leu >= 1000 THEN
    grade := 'III';
ELSE
    grade := 'IV';
END IF;
```

V konstrukci IF/ELSIF/ELSE se provede kód vždy jen jednoho ramena. Pokud je při průchodu splněna podmínka více než jednoho ramena, provede se pouze první se splněnou podmínkou. Pokud není splněna žádná podmínka, provede se část ELSE.

Dalším prvkem procedurálního programování jsou programové smyčky, které nám umožní provádět určitou část kódu opakovaně. Jazyk plpgsql nabízí několik typů smyček. Jedním z nich je smyčka za klíčovým slovem WHILE. Za WHILE definujeme podmínku, která dokud je pravdivá, programový kód ohraničený slovy LOOP a END LOOP se stále opakuje. Předpokládá se, že v těle cyklu se mění hodnoty proměnných použitých v podmínce smyčky.

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
   -- some computations here
END LOOP;
```

Pokud počet cyklů smyčky je dopředu známý, použijeme smyčku FOR:

```
FOR i IN 1..10 LOOP
   -- proměnná i nabývá postupně hodnot 1,2,3,4,5,6,7,8,9,10
END LOOP;
```

Smyčku FOR lze použít i pro procházení řádků SQL dotazu.

```
DECLARE
   rs RECORD;
BEGIN
FOR rs IN SELECT * FROM tabulka LOOP
      -- proměnná rs nabývá hodnot jednotlivých řádků, na hodnoty sloupců aktuálně
      --  zpracovávaného řádku se odkazujeme rs.sloupec
END LOOP;
END;
```

V PostgreSQL můžeme využívat kromě jednoduchých proměnných i pole (array) hodnot. Pro procházení položek pole využijeme smyčku FOREACH:

```
DECLARE
x VARCHAR(50);
s TEXT;
a VARCHAR[]; --deklarace pole
BEGIN
FOREACH x IN ARRAY a LOOP
-- x nabývá jednotlivých hodnot pole
          s := s || x;
END LOOP;
```

Během provádění procedurálního kódu může dojít k chybě, například když se pokusíme dělit nulou nebo přiřadit do proměnné hodnotu, která neodpovídá jejímu datovému typu. Těmto chybám lze částečně předcházet pomocí podmínkových konstrukcí, ale existuje i jiná varianta řešení chybových stavů. Jde o tzv. zachytávání výjimek a obsloužení chybového stavu. Pokud dojde při běhu funkce k chybě, dojde k přerušení vykonávání kódu na chybovém řádku a generuje se výjimka (exception). Pokud je na konci bloku kódu, kde chyba nastala, sekce pro zpracování výjimek, přesune se vykonávání kódu sem. Každá vzniklá výjimka s sebou nese identifikaci chyby, která ji způsobila. V sekci výjimek pak lze reagovat na jednotlivé druhy chyb:

```
BEGIN
...
...
EXCEPTION
  WHEN division_by_zero  THEN
    -- osetreni deleni nulou
  WHEN invalid_datetime_format THEN
    -- chybny format datumu
  WHEN OTHERS THEN
    -- osetreni vsech ostatnich chyb
END;
```

Předdefinovaných druhů výjimek je více, detaily lze nalézt v dokumentaci databázového systému. Často však vystačíme pouze s ramenem WHEN OTHERS, kdy chceme pouze zaregistrovat jakoukoliv vzniklou chybu.

Pomocí procedurálního kódu jazyka plpgsql lze vytvořit uživatelskou funkci. Použijeme k tomu příkaz CREATE OR REPLACE FUNCTION:

```
CREATE OR REPLACE FUNCTION nejaka_funkce() RETURNS NUMERIC AS $$
DECLARE
   promenna NUMERIC := 30;
BEGIN
   RAISE NOTICE 'Promenna obsahuje cislo %', promenna;
-- RAISE NOTICE  vypisuje hodnoty na obrazovku, % je ve vypisu nahrazeno obsahem promenne
   RETURN promenna;
END;
$$ LANGUAGE plpgsql;
```

Vykonávání funkce končí příkazem RETURN, který vrací výsledek funkce. Funkci voláme obvykle příkazem SELECT:

```
SELECT nejaka_funkce();
```

Jednou vytvořená funkce je součástí databáze, pokud ji chceme odstranit, použijeme příkaz:

DROP FUNCTION nejaka_funkce().

5. Literatura

[1] http://www.postgresql.org/docs/9.4/static/index.html

[2] Goyvaerts J, Levithan S. Regulární výrazy. Brno: Computer Press, 2010.

6. Příklady z přednášky

```
CREATE OR REPLACE FUNCTION pole() RETURNS varchar AS $$
DECLARE
   arr VARCHAR[];
   sl VARCHAR;
   ret VARCHAR := 'CREATE TABLE zdroj (';
BEGIN
   arr:= string_to_array('ID          Gene title          Gene symbol      Gene ID UniGene title
         UniGene symbol UniGene ID       Nucleotide Title  GI       GenBank Accession
         Platform_CLONEID       Platform_ORF    Platform_SPOTID          Chromosome
location Chromosome annotation   GO:Function      GO:Process       GO:Component
         GO:Function ID  GO:Process ID    GO:Component ID '
, chr(9));
FOREACH sl IN ARRAY arr LOOP
         sl:=TRANSLATE(sl, ' :','__');
         ret := ret || sl || ' text,';
END LOOP;
ret := ret || ')';
   RETURN ret;
END;
$$ LANGUAGE plpgsql;
CREATE TABLE zdroj (ID text,Gene_title text,Gene_symbol text,Gene_ID
text,UniGene_title text,UniGene_symbol text,UniGene_ID text,Nucleotide_Title text,GI
text,GenBank_Accession text,Platform_CLONEID text,Platform_ORF
text,Platform_SPOTID text,Chromosome_location text,Chromosome_annotation
text,GO_Function text,GO_Process text,GO_Component text,GO_Function_ID
text,GO_Process_ID text,GO_Component_ID_ text);
CREATE TABLE zdroj2 (ID_REF text,IDENTIFIER text,GSM800759 text,GSM800760
text,GSM800761 text,GSM800762 text,GSM800763 text,GSM800764 text,GSM800765
text,GSM800766 text,GSM800767 text,GSM800768 text,GSM800769 text,GSM800770
```

```
text,GSM800771 text,GSM800772 text,GSM800773 text,GSM800774 text,GSM800775
text,GSM800742 text,GSM800743 text,GSM800744 text,GSM800745 text,GSM800746
text,GSM800747 text,GSM800748 text,GSM800749 text,GSM800750 text,GSM800751
text,GSM800752 text,GSM800753 text,GSM800754 text,GSM800755 text,GSM800756
text,GSM800757 text,GSM800758 text);

SELECT  count(*) FROM zdroj JOIN zdroj2 ON
zdroj.id = zdroj2.id_ref

CREATE TABLE zdroj2_mod AS

SELECT id_ref,'GSM800759' patient, GSM800759 as val FROM zdroj2 UNION ALL
SELECT id_ref,'GSM800760',GSM800760 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800761',GSM800761 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800762',GSM800762 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800763',GSM800763 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800764',GSM800764 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800765',GSM800765 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800766',GSM800766 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800767',GSM800767 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800768',GSM800768 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800769',GSM800769 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800770',GSM800770 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800771',GSM800771 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800772',GSM800772 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800773',GSM800773 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800774',GSM800774 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800775',GSM800775 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800742',GSM800742 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800743',GSM800743 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800744',GSM800744 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800745',GSM800745 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800746',GSM800746 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800747',GSM800747 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800748',GSM800748 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800749',GSM800749 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800750',GSM800750 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800751',GSM800751 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800752',GSM800752 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800753',GSM800753 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800754',GSM800754 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800755',GSM800755 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800756',GSM800756 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800757',GSM800757 FROM zdroj2 UNION ALL SELECT
id_ref,'GSM800758',GSM800758 FROM zdroj2


SELECT id_ref, AVG(val::NUMERIC), MIN(val::NUMERIC), MAX(val:: NUMERIC),
count(*) FROM zdroj2_mod
GROUP BY id_ref
```

Elementární zpracování dat v softwarovém prostředí R

Jiří Kalina

Institut biostatistiky a analýz, Masarykova univerzita, Brno; e-mail: kalina@iba.muni.cz

Abstrakt

Prvotní zpracování vstupních dat je samozřejmou součástí každé statistické analýzy, bez níž obvykle vůbec není možné přistoupit k samotným výpočtům, jež tvoří její jádro. Data mohou být různým způsobem poškozená nebo zaznamenaná v rozporu se zvolenou normou, k problémům dochází díky různým formám kódování textu, číselných hodnot nebo kalendářních dat, se kterými se lze v praxi setkat.

Příspěvek se zabývá elementárními postupy importu (transformace) a kontroly dat z různých datových zdrojů do výpočetního prostředí statistického programovacího jazyka R, přibližuje rutinní postupy formátování a čištění dat a upozorňuje na nejobvyklejší chyby a úskalí při jejich zpracování. Nezabývá se samotným matematicko-statistickým zhodnocením dat, ale klade si za cíl zprostředkování praktických a potřebných postupů, které by mu měly ve všech případech předcházet.

Klíčová slova

R, analýza dat

1. Úvod

Nezbytnou součástí každé statistické analýzy je prvotní (před)zpracování dat, spočívající v jejich správném naformátování podle požadavků zvoleného softwarového nástroje, kontrole a případně opravě možných chyb a nedostatků ve vstupním datovém souboru a další úpravě dat (např. jednoduché přepočty, agregace, normalizace aj.) tak, aby byla vhodná pro následující krok, kterým je samotné matematicko-statistické zpracování dat a interpretace získaných informací.

Z hlediska náročnosti může zpracování surových dat představovat jak (časově) zanedbatelnou součást rozsáhlejší analýzy, tak také majoritní díl veškeré analytické práce. A to nejen v závislosti na zvolených analytických metodách a kvalitě datového zdroje, ale také na vhodnosti zvoleného výpočetního prostředí, schopnostech datového analytika a v neposlední řadě na náhodných vlivech, které mohou způsobovat těžko předvídatelné (a někdy rovněž špatně odhalitelné) problémy při samotné analýze.

Následující text podrobněji rozebírá a na příkladech demonstruje postupy, které jsou součástí prvotní analytické práce – předzpracování dat před samotnou statistickou analýzou – v případě využití statistického software R. Pozornost je věnována zejména problematickým momentům předzpracování dat, nicméně text předpokládá alespoň základní uživatelskou znalost práce se software R.

1.1. R

Statistické prostředí jazyka R je založené na starším jazyce nazvaném S, který spatřil světlo světa v 70. letech 20. století v proslulých Bellových laboratořích v USA. Programovací jazyk R má modulární rekurentní strukturu – veškeré funkce jsou uspořádány do dílčích balíků

(packages obsahující data a funkce), které jsou samy psány v jazyce R. Složením vybrané kombinace balíků tedy lze dosáhnout takové množiny funkcí, která optimálně odpovídá konkrétní výpočetní potřebě, bez nutnosti zahlcovat počítač množstvím nevyužitích funkcí.

Základní distribuce jazyka R (aktuálně v červenci 2015 verze 3.1.2.) je složena z 12 implicitních balíků (ty není třeba instalovat a připojovat samostatně):

- `base`
- `compiler`
- `datasets`
- `graphics`
- `grDevices`
- `grid`
- `methods`
- `parallel`
- `splines`
- `stats`
- `stats4`
- `tcltk`

Tyto balíky obsahují veškeré funkce potřebné pro základní práci s prostředím R a pro instalaci, připojení i vytváření dalších rozšiřujících balíků. Celkem je v oficiálním repozitáři CRAN (Comprehensive R Archive Network) v současnosti k dispozici téměř 7 000 rozšiřujících balíků, které lze jednoduše instalovat přímo z jeho webové stránky (to umožňuje zdarma sdílet jen stěží představitelné množství metod, nápadů a zkušeností dalších uživatelů, kteří je přetavili do podoby R balíků. S mírnou nadsázkou lze tvrdit, že není třeba vytvářet žádné další funkce, protože v balících R lze najít prakticky cokoliv).

Vzhledem ke skutečnosti, že prostředí jazyka R je volně dostupné pod licencí GNU (Free Software Foundation's GNU General Public Licence), dosahuje jeho použití enormní popularity a celosvětová komunita uživatelů R aktuálně přesahuje počet 2 000 000 osob.

Největší síla jazyka R ovšem spočívá ve specificky statistickém zpracování velkých objemů dat ve formě tabulek a vektorů, jehož efektivita se blíží práci s relačními databázemi, ovšem při zachování všech specifik statistické práce a dostatečné uživatelské přívětivosti. R lze spouštět na všech obvyklých platformách operačních systémů UNIX, Windows, MacOS a jejich derivátech, pomocí vhodných balíků jej lze vestavět do webových stránek a využívat v rámci serverových víceuživatelských aplikací. Pro uživatelsky příjemné použití byla v roce 2011 vyvinuta open source platforma RStudio, kombinující syntaktický editor, konzoli jazyka R, prohlížeč proměnných, nápovědu, grafický výstup a další užitečné utility.

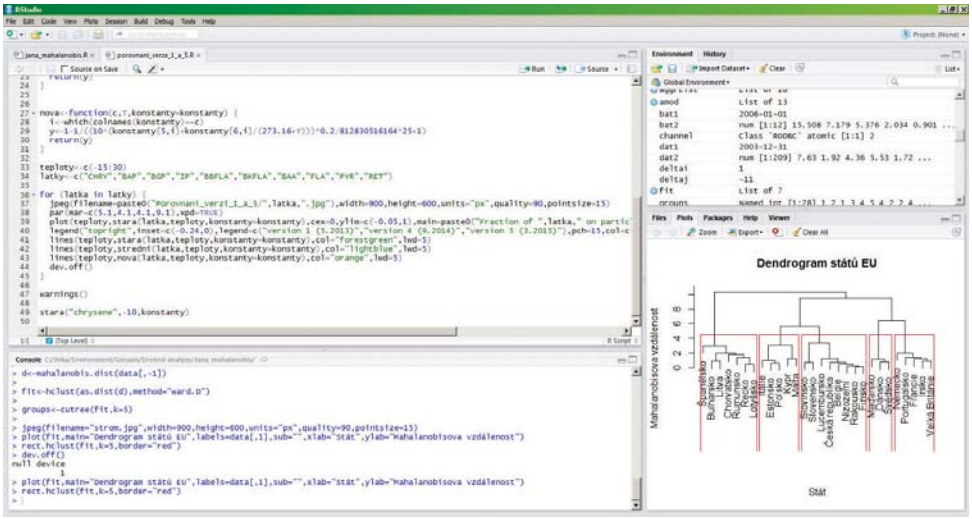
1.2. Rstudio

Rstudio je aplikace, zprostředkovávající rozhraní mezi uživatelem a samotným prostředím a jazykem R. Standardně je okno Rstudia rozděleno na čtyři obdélníkové části volitelných rozměrů, přičemž první část (vlevo nahoře) představuje inteligentní syntaktický editor s řadou užitečných funkcí oproti čistě textovému prostředí (obarvení kódu, řetězců a klíčových slov, možnost sbalit/rozbalit části kódu, kontrola uzávorkování aj.).

Spouštění částí kódu se děje tradičně pomocí klávesové zkratky Ctrl + R, přičemž spuštěný kód se objevuje níže v okně konzole[1] včetně případných varovných a chybových hlášek. Také v tomto okně je text částečně editovatelný (např. je možné z konzole vykopírovat výsledky ve formě textu).

Zbývající dvě okna nabízí větší množství záložek s různými funkcemi. Výše umístěné okno umožňuje např. prohlížet obsah proměnných v daném prostředí nebo procházet historii spuštěných příkazů, dolní okno pak zprostředkovává adresářovou strukturu, přehled instalovaných a připojených balíků, nápovědu a grafické výstupy několika typů.

[1] Tvar slova konzola vs. konzole viz http://nase-rec.ujc.cas.cz/archiv.php?art=7766.



Rstudio navíc nabízí další užitečné funkce pro práci s kódem v jazyce R, jako je zakládání projektů (pro vytváření balíků), export grafů, instalace a připojení balíků přímo z repozitáře CRAN, vyhledávání v kódu, nápovědě a obsahu proměnných apod.

2. Datové typy a třídy v R

Dříve než se pustíme do vlastního importu dat z různých zdrojů, je vhodné se podívat na nejobvyklejší datové typy a třídy, které má jazyk R v základní distribuci k dispozici. Nevhodně zvolený datový typ/třída je jedním z nejobvyklejších problémů, které řeší méně zkušení uživatelé při práci v jazyce R a proto je vhodné se datovým typům a třídám věnovat podrobněji před dalším výkladem.

2.1. Vybrané datové typy objektů v R

Datové typy vycházejí z reprezentace proměnných v paměti počítače a pro většinu programovacích jazyků jsou podobné. Na rozdíl od tříd jsou datové typy konstruovány jako co nejuniverzálnější. Pro konkrétní specifickou potřebu je potom možné z těchto datových typů konstruovat složitější třídy (classes) ve smyslu tříd v objektově orientovaném programování (s dědičnými vlastnostmi apod.).

Základní datové typy, se kterými je možné se setkat na uživatelské úrovni při práci v R, jsou `double`, `complex`, `character`, `logical`, `integer`, `raw` nebo `list`, `NULL`, `closure` (pro funkce), `special` a `builtin`. Některé z nich jsou podrobněji popsány v následujícím tetu.

Statistika je z velké části práce s čísly, proto je vhodné začít s pro analýzy nejobvyklejším datovým typem, který slouží v jazyce R pro ukládání číselných hodnot. Tímto datovým typem je reálné (desetinné) číslo neboli typ `double`.

2.1.1. Datový typ double

Datový typ `double` umožňuje ukládání reálných čísel ve formátu s plovoucí desetinnou čárkou[2] a tzv. dvojitou přesností (datový typ `double` je v R totožný s třídou `numeric` a dříve používanou třídou `real`), tj. v paměti zaujímá 64 bitů. Jazyk R nemá k dispozici datový typ, který by odpovídal reálným číslům s jednoduchou přesností (32 bitů).

Mimo reálná čísla může proměnná typu `double` nabývat také zvláštních hodnot `±Int` (kladné a záporné nekonečno), `NA` (not available = hodnota není známa) či `NaN` (not a number = hodnota není přípustná, např. při dělení nulou).

Vytvořit lze proměnnou typu `double` jednoduše pomocí příkazu `double` (vrátí vektor nul o zadané délce) nebo vlastním přiřazením reálného čísla do proměnné:

```
promenna<-double(1) # argument oznacuje delku vektoru, 1 = skalar (jedno cislo)
promenna<-sqrt(2) # priradi odmocninu ze dvou automaticky s double precision
```

Stejně jako ve všech následujících příkladech lze ověřit pomocí funkce `is.double`, zda je daná proměnná typu `double` (funkce vrací logickou hodnotu):

```
is.double(promenna) # vrati hodnotu TRUE nebo FALSE
```

2.1.2. Datový typ character

Datový typ určený pro ukládání textových řetězců v závislosti na zvoleném kódování. Počet znaků textového řetězce je libovolný od 0 (i prázdná proměnná může být typu character) po $2^{31} - 1$, což jsou přibližně 2 miliardy znaků (32bitových).

Vytvořit lze proměnnou typu `character` jednoduše pomocí příkazu `character` (vrátí vektor prázdných řetězců o zadané délce) nebo vlastním přiřazením textového řetězce do proměnné:

```
promenna<-character(1) # argument oznacuje delku vektoru (1 = jeden retezec)
promenna<-"příliš žluťoučký kůň" # radeji predpokladejme, ze kodovani je UTF8
```

Opět lze ověřit, zda je proměnná typu character pomocí funkce `is.character`.

```
is.character(promenna)
```

2.1.3. Datový typ logical

Jak napovídá už sám název, slouží datový typ `logical` k ukládání logických hodnot `TRUE` (pravda, ve většině aplikací v R lze také psát jako `T`, nicméně ne vždy) a `FALSE` (nepravda, obvykle lze psát pouze `F`).

```
promenna<-logical(4) # vytvori vektor samych FALSE o delce 4
promenna<-F # priradi do promenne nepravdu
```

Jako v předchozích případech, i zde je možné využít ověřovací funkci `is.logical`:

```
is.logical(promenna)
```

[2] tzv. floating point formát

2.1.4. Další datové typy v R

Ze zbývajících datových typů uveďme ve stručnosti ještě univerzální typ `list`, určený pro ukládání seznamů, typ `integer` sloužící pro ukládání celých čísel, ke kterému se váže stejně pojmenovaná třída, typ `complex` opět se stejně pojmenovanou třídou, sloužící pro práci s komplexními čísly a typy `closure` a `builtin`, které slouží k ukládání funkcí a pojí se mj. se třídou `function` (povědomí o existenci těchto datových typů je přinejmenším vhodné pro porozumění chybovým hláškám při obvyklých chybách při psaní kódu).

Pro zjištění datového typu objektu slouží v R jednoduchá funkce `typeof()`:

```
promenna<-as.double(1)
typeof(promenna)
typeof(integer(0))
typeof(5)
typeof(data.frame)
typeof(c)
typeof(NULL)
```

2.2. Vybrané třídy objektů v R

V mnoha případech při ukládání hodnot do proměnných nevystačíme se základními datovými typy a potřebujeme data uspořádat do složitější struktury. Proto je možné ze základních datových typů konstruovat složitější třídy (classes), které přiřazujeme jednotlivým objektům. Na rozdíl od datových typů existuje tříd nepřeberné množství a v případě potřeby navíc běžně uživatelé definují další třídy. Uvedeme proto pouze několik nejdůležitějších a nejčastěji používaných tříd v R.

Podobně jako pro zjištění datového typu objektu využíváme funkci `typeof()`, zjišťování třídy objektu budeme provádět pomocí funkce `class()`.

2.2.1. Třída numeric

Třída `numeric` je totožná s datovým typem `double` a umožňuje tedy ukládání reálných čísel ve formátu s plovoucí desetinnou čárkou a dvojitou přesností a speciálních hodnot zmíněných výše.

Vytvořit lze proměnnou třídy `numeric` jednoduše pomocí příkazu `numeric` (vrátí vektor nul o zadané délce) nebo vlastním přiřazením reálného čísla do proměnné:

```
promenna<- numeric(1) # argument oznacuje delku vektoru, 1 = skalar (jedno cislo)

promenna<-sqrt(2) # priradi odmocninu ze dvou automaticky s double precision
```

Pomocí funkce `is.numeric` lze ověřit, zda je daná proměnná typu `numeric` (funkce vrací logickou hodnotu):

```
is.numeric(promenna) # vrati hodnotu TRUE
typeof(promenna) # vrati hodnotu "double"
class(promenna) # vrati hodnotu "numeric"
```

2.2.2. Třída character

Obdobně jako třída `numeric` odpovídá datovému typu `double`, váže se k datovému typu `character` stejně pojmenovaná třída.

Vytvořit lze proměnnou třídy `character` jednoduše pomocí příkazu `character` (vrátí vektor prázdných řetězců o zadané délce) nebo vlastním přiřazením textového řetězce do proměnné:

```
promenna<-character(1) # argument oznacuje delku vektoru (1 = jeden retezec)
promenna<-"příliš žluťoučký kůň" # radeji predpokladejme, ze kodovani je UTF8
```

Opět lze ověřit, zda je proměnná třídy character pomocí funkce `is.character`.

```
is.character(promenna)
```

2.2.3. Třída Date

Poměrně často je v praxi třeba zpracovávat údaje o kalendářním datu a čase. Zápis kalendářního data se v různém software a různých kulturních prostředích významně liší a tyto odlišnosti bývají jedním z nejčastějších zdrojů problémů při zpracování dat.

Datum je v prostředí R obvykle ukládáno jako typ double, existuje ale několik variant tříd, které nad tímto datovým typem vytváří nadstavbu se srozumitelnou interpretací. Nejjednodušší takovou třídou je třída `Date`, která ukládá datum jako počet dní od 1. ledna 1970. Výstupní formát je pak v podobě RRRR-MM-DD. V následujících příkladech si vystačíme při práci s kalendářními daty vždy se třídou `Date`, nicméně zájemci mohou prostudovat rovněž dostupné třídy `POSIXlt` a `POSIXct`.

Přiřazení třídy Date se provádí jednoduchou funkcí `as.date()`, případně je automatické při vložení datové hodnoty do proměnné např. z funkce pro aktuální systémové datum `Sys.Date()`:

```
promenna<-as.Date("1970-01-02") # ulozi do promenne hodnotu 1
promenna # vypis obsah promenne ve formatu RRRR-MM-DD
promenna<-Sys.Date() # ulozi do promenne aktualni systemove datum
typeof(promenna) # vypise typ double, nebot trida Date je zalozena na typu double
```

2.2.4. Třída factor

Zajímavou třídou založenou na datovém typu integer je třída factor, sloužící ke kódování kategoriálních proměnných. Pro objekt třídy factor lze nadefinovat jednotlivé faktorové úrovně, které jsou následně očíslovány dle pořadí a prvky objektu jsou poté reprezentovány pořadovými čísly. To je vhodnější, než reprezentovat například konkrétní množinu barev jejich celými názvy:

```
promenna<-factor(c("zluta","modra","zluta","bila")) # automaticky se vytvori urovne
promenna # vypise nazvy kategorii odpovidajici prvkum
as.integer(promenna) # vypise poradova cisla (podle abecedy)
typeof(promenna) # datovy typ je integer
```

Se třídou faktor se lze setkat často při importu dat, kde jsou proměnné některých datových typů načítány implicitně jako factor a pro další práci je obvykle nutné je konvertovat na jinou třídu.

2.2.5. Třída data.frame

Složitější strukturou, reprezentující dvourozměrně uspořádaná data je třída `data.frame` (datový rámec), reprezentovaná v podstatě tabulkou, v níž každý sloupec je představován jedním vektorem (všechny musí být stejné délky) s vlastní třídou (třídy vektorů v rámci se mohou lišit).

Vytvářet lze datové rámce různými způsoby, nejjednodušší je využití funkce `data.frame()` nebo konverzní funkce `as.data.frame()`:

```
promenna<-data.frame(c("Adamov","Babice","Brno"),c(4576,1095,378327))
# dva sloupce
promenna<-
data.frame(sloup1=c("Adamov","Babice","Brno"),sloup2=c(4576,1095,3783
27))
promenna<-as.data.frame(c(1,2,3)) # vytvori ramec s jednim sloupcem
```

Na buňky rámce se pak lze odkazovat pomocí dvourozměrného indexování ve formě `ramec[radek,sloupec]`. Pro ověření třídy lze stejně jako v předchozích případech využít funkci `is.data.frame()`.

```
is.data.frame(promenna) # vrati hodnotu TRUE
class(promenna) # vrati hodnotu "data.frame"
typeof(promenna) # vrati hodnotu "list"
```

TIP: Potřebujete-li zjistit třídy jednotlivých sloupců datového rámce, nelze jednoduše využít funkci `class()`, která vrací třídu celého rámce, nýbrž je zapotřebí využít funkce `lapply()`, která aplikuje funkci `class()` zvlášť na každý sloupec rámce:

```
lapply(promenna,class) # vrati tridy jednotlivych sloupcu ramce
```

2.2.6. Třída matrix

Jiná, podobná dvourozměrná struktura je definována jako třída `matrix` (matice). Většina parametrů datových rámců a matic je totožná, hlavní rozdíl spočívá ve větší efektivitě uložení a zpracování dat ve formě matice. Současně ale matice musí splňovat podmínku stejného datového typu všech svých prvků (tedy řádků i sloupců), naproti tomu datový rámec může mít každý sloupec jiného datového typu.

Jednoduše lze matici nadefinovat pomocí funkce `matrix()`:

```
promenna<-matrix(TRUE,4,3) # vytvori matici 4 × 3 se vsemi sloupci
typu logical

data.frame(c("Adamov","Babice","Brno"),c(4576,1095,378327)) # dva
sloupce
```

2.3. Konverze mezi třídami

Ani v případě velmi dobře nedefinovaných tříd objektů se nelze vyhnout potřebě měnit třídy některých objektů (proměnných), zejména ve fázi importu a exportu dat. Může jít například o výpis číselných hodnot v textových řetězcích, konverzi kalendářních data zadaných ve formě textu nebo čísel, změnu matice na datový rámec aj.

V řadě případů je konverze mezi třídami zřejmá (např. konverze celých čísel (`integer`) na reálná (`numeric`) a dále na komplexní (`complex`) je intuitivním vnořením číselných těles), v jiných případech je zřejmá méně a v některých případech je přímo nemožná (např. konverze textu na binární hodnoty).

2.3.1. Vybrané konverze mezi třídami

Konvertovat celá čísla na reálná a reálná na komplexní lze snadno pomocí funkcí `as.numeric` a `as.complex`, zajímavější je situace v případě konverze opačného směru. Funkce `as.numeric` a `as.integer` si poradí i v tomto případě, nicméně nejprve dojde k ořezání imaginární části a poté části neceločíselné:

```
promenna<-c(0,2+3i,-1i)
class(promenna)
as.numeric(promenna)
```

Výsledkem bude vektor reálných čísel 0, 2, 0, neboť imaginární část je úplně zanedbána.

TIP: Při práci s komplexními čísly je vždy nutné zadávat před imaginární jednotkou násobitel, i v případě, kdy jde o jedničku. Zápis `1i` bude interpretován jako jedna imaginární jednotka, naproti tomu zápis `i` jako proměnná pojmenovaná i.

Obdobně dojde k ořezání neceločíselné části (tj. zaokrouhlení nahoru pro záporná a dolů po kladná čísla) v případě konverze reálných čísel na celá:

```
promenna<-c(1,2.2,9.99,-3.5,Inf)
class(promenna)
as.integer(promenna)
```

Obdobně lze pokračovat k binární třídě logical, kde se všechny hodnoty kromě nuly interpretují jako pravda a nula jako nepravda.

Zdánlivě jednoduchá je konverze reálných čísel na textové řetězce. Zde je pouze zapotřebí hlídat počet desetinných míst, která budou vytištěna. Počet platných cifer v tomto případě (na rozdíl od výpisu na konzoli) není ovlivněn globálním nastavením přesnosti pomocí funkce `options(digits=pocet_cifer)`):

```
as.character(c(0.142857142857142857142857142857,1/7,1000/7,pi)) # 15
platnych cifer
```

Je tedy vhodné použít některou ze zaokrouhlovacích funkcí pro požadovaný počet desetinných míst:

```
as.character(round(c(0.142857142857142857142857142857,1/7,1000/7,pi),
3)) # 3 platne
```

TIP: Pro zaokrouhlování lze využít celou řadu funkcí, např. `round()` pro klasické zaokrouhlení, `floor()` pro zaokrouhlení dolů a `ceiling()` nahoru, `signif()` pro daný počet platných číslic nebo `trunc()` pro odříznutí desetinné části.

Užitečná konverze je ve směru z třídy `factor` na třídu `numeric`, v případě, že proměnná třídy `factor` ukrývá původní reální čísla. Navíc přímo v rámci konverze lze zpracovat nevhodně kódované hodnoty (např. nečíselné hodnoty v číselné proměnné aj.). Nicméně je třeba mít na paměti, že třída `factor` je založena na datovém typu `integer` – velmi nebezpečnou chybou totiž může být v tomto kroku pokus o přímou konverzi:

```
promenna<-factor(c(1.5,-1.4,2.0,0.9))
as.numeric(promenna)
```

V tomto případě se konvertují na třídu numeric pořadová čísla faktorových úrovní datového typu integer a výsledkem bude naprosto nesmyslný vektor čísel 3, 1, 4, 2. Nebezpečí chyby spočívá především v její špatné odhalitelnosti, protože konverze je úspěšně provedena s nesmyslnými hodnotami bez oznámení chyby. Tento problém lze obejít poměrně snadno postupnou konverzí přes třídu `character`:

```
promenna<-factor(c(1.5,-1.4,2.0,0.9))
as.numeric(as.character(promenna))
```

Konverze textu (`character`) na kalendářní datum (`Date`) je jednoduchá v případě, kdy jsou k dispozici textové řetězce formátu `RRRR-MM-DD`, funkce `as.Date()` si poradí i s řetězci tvaru `RRRR-M-D`, nicméně neumí zpracovat neúplná kalendářní data a letopočty před rokem 0:

```
promenna<-as.Date(c("2015-09-09","2015-9-9","0-1-1","-15-2-
6","2015"))
```

TIP: V případě konverze číselných formátů na datum je možné rovněž využít funkci `as.Date()`, nicméně jako druhý argument `origin` je nutné specifikovat počáteční datum „letopočtu" – obvykle tedy 1. ledna 1970, nicméně hodnota se může lišit pro různý software:

```
promenna<-as.Date(c(16800),origin="1970-01-01")
```

Bezproblémová bývá konverze z matice na datový rámec:

```
matice<-matrix(c("Adamov","Aš","Abertamy",
                 "Babice","Brno","Březová",
                 "Cvikov","Cerhenice","Ctiboř"),3,3)
as.data.frame(matice)
```

V opačném směru je zapotřebí pohlídat, aby byly všechny sloupce vhodné třídy, konvertovatelné na výslednou třídu prvků matice:

```
ramec<-data.frame(c("Adamov","Babice","Brno"),c(4576,1095,378327))
as.matrix(ramec)
```

Poslední užitečnou ukázkou je konverze více sloupců datového rámce najednou, pro niž nelze použít přímo příkaz pro konverzi, neboť by se narušila struktura rámce. Opět je nutné využít funkce `lapply()` a konvertovat každý sloupec zvlášť jako v následujícím (a v praxi dosti obvyklém) příkladu pro konverzi sloupců tříd `factor` na `numeric`:

```
ramec<-data.frame(factor(c(1,5.5,99)),factor(c(-1,-6,-100)))        #
vytvoreni ramce
lapply(ramec,class) # oba sloupce jsou tridy factor
lapply(lapply(ramec,as.character),as.numeric) # postupna konverze na
realna cisla
```

Na závěr je užitečné zmínit existenci funkce `type.convert()`, která slouží k automatickému odhadu původní třídy proměnné třídy `character` (v řadě `logical`, `integer`, `double`, `complex`, `factor`) a příslušné konverzi při importu dat pomocí jedné z funkcí zmíněných níže. Funkci je samozřejmě možné použít i samostatně:

```
class(type.convert(c("F")))                # vrati hodnotu "logical"
class(type.convert(c("1")))                # vrati hodnotu "integer"
class(type.convert(c("1.4")))              # vrati hodnotu "numeric"
class(type.convert(c("1i")))               # vrati hodnotu "complex"
class(type.convert(c("F","1","1.4","1i"))) # vrati hodnotu "factor"
```

3. Import dat do R

Pohlédneme-li na předzpracování dat optikou jazyka R, bude prvním krokem tohoto procesu import dat (zde přeskočíme předchozí kroky spočívající v designu a samotném provedení

experimentu, zaznamenání a přenosu dat apod.). Přestože možných cest, kterými může import dat do software proběhnout, je více, obvykle půjde o jeden z následujících postupů:

- ruční zadání dat přes uživatelské rozhraní (konzoli),
- import dat ze souboru určeného typu,
- import dat ze schránky operačního systému,
- import dat z webové stránky,
- import dat z databáze přístupné po síti.

Pomineme-li v následujícím textu první uvedený způsob, který je vhodný pouze pro nejmenší datové sady např. při ověřování metod nebo experimentování, má smysl se zabývat importem dat z různých druhů souborů a s tím souvisejícími problémy a nejednoznačnostmi, případně importem ze schránky operačního systému, webových stránek nebo databází.

Ve všech zmíněných případech, kterými se bude text nadále zabývat, půjde téměř výhradně o data uspořádaná ve dvourozměrných tabulkách různých druhů; nicméně řada popsaných metod a argumentů je platná také při načítání jednorozměrných struktur – zájemce o tento druh importu nicméně pouze odkazuji na dvojici funkcí `scan()`[3], která slouží k načítání samostatných vektorů, případně seznamů (`lists`) a je navíc základní stavební jednotkou všech níže popsaných funkcí a funkci `readLines()`, která načítá textové soubory jako vektory třídy character, kde každý řádek představuje právě jeden prvek vektoru.

Rozdíl mezi funkcemi `scan()` a `readLines()` je dobře patrný z následujícího příkladu načtení souboru `drama.txt` s pěti řádky:

```
expozice: uvedení do děje
kolize: zařazení dramatického prvku
krize: vyvrcholení dramatu
peripetie: obrat a řešení
katastrofa: rozuzlení a očista
```

Použitím příkazu `scan("drama.txt",what="character")` importujeme data v podobě

```
 [1] "expozice:"      "uvedení"         "do"                 "děje"
"kolize:"
 [6] "zařazení"        "dramatického"  "prvku"              "krize:"
"vyvrcholení"
[11] "dramatu"           "peripetie:"     "obrat"              "a"
"řešení"
[16] "katastrofa:"  "rozuzlení"    "a"           "očista"
```

tedy vektoru o devatenácti prvcích a třídě `character`. Nutnost specifikace třídy pomocí argumentu `what` je nekomfortní součástí práce s funkcí `scan()`, která na rozdíl od pokročilejších importních funkcí neumí rozpoznat třídu importované proměnné.

Protože argument `sep` funkce `scan()` nebyl zadán, použila funkce pro oddělovač prvků ve vektoru implicitní hodnotu, kterou je bílé místo (white space, tedy mezera, tabelátor nebo konec řádku) – to se v souboru vyskytuje na osmnácti pozicích a proto je výsledkem devatenáctiprvkový vektor. O oddělovačích podrobněji pojednává následující podkapitola.

Naproti tomu příkaz `readLines("drama.txt")` považuje za oddělovač prvků výsledného vektoru pouze konce řádků, které se v původním souboru vyskytují čtyři a výsledkem je tak čtyřprvkový vektor:

[3] Vyčerpávající popis funkce `scan()` lze nalézt v dokumentaci k R.

```
[1] "expozice: uvedení do děje"                    "kolize: zařazení
dramatického prvku"
[3] "krize: vyvrcholení dramatu"          "peripetie: obrat a řešení"
[5] "katastrofa: rozuzlení a očista"
```

3.1. Import dat ze souborů

Nejjednodušší formou zápisu jsou data ve formě prostého textu, kde je dvourozměrná struktura určena oddělovači řádků a sloupců, případně přesnými, předem danými počty znaků v jednotlivých buňkách. Zatímco konce řádků obvykle nebývají problematické (v závislosti na software se používají obvykle pouze dva různé znaky pro ukončení řádků: `CR` (carriage return, ASCII `0x0D`) a/nebo `LF` (line feed, ASCII `0x0A`), které importní funkce bez potíží dekódují, oddělovače sloupců mohou být reprezentovány různými znaky, případně jsou sloupce určené daným pevným počtem znaků (což je obecně paměťově méně efektivní způsob zápisu, nicméně ve specifických aplikacích může znamenat vyšší efektivitu čtení/zápisu). To představuje jeden z nejčastějších problémů při čtení dat ze souborů.

3.1.1. Import dat s oddělovači sloupců

Základní funkcí pro čtení datových tabulek s oddělovači sloupců je v jazyce R funkce `read.table()`, která umožňuje čtení dat z textových souborů s oddělovači prakticky všech typů. V implicitním nastavení argumentu `sep=""` (oddělovač) je za oddělovač považováno „bílé místo“, tedy libovolný počet mezer nebo tabelátorů. Funkce `read.table()` navíc automaticky (pokud není v argumentu `colClasses` případně `as.is` nařízeno jiné chování) používá výše zmíněnou funkci `type.convert()` k odhadu příslušné třídy pro každý sloupec tabulky a jeho případné konverzi. Například soubor `soubor.txt` následujícího tvaru s hodnotami oddělenými tabelátorem

```
Adamov Babice Crhov
1      2      3
```

bude při použití funkce `read.table("soubor.txt")` přečten jako datový rámec tvaru

```
      V1     V2    V3
1 Adamov Babice Crhov
2      1      2     3
```

kde `1`, `2` jsou názvy řádků, `V1`, `V2` a `V3` názvy sloupců (protože nebyly názvy řádků a sloupců specifikovány, přidělí je R automaticky) a jednotlivé sloupce jsou třídy `factor`, protože se je nepodařilo identifikovat jako žádnou z číselných tříd.

Oddělovač sloupců však může mít podobu libovolného znaku. Například poněkud nepřehledný soubor `soubor.txt` následujícího tvaru

```
1.5,6.2,7.3
4.8,0.1,9.9
```

bude při nastavení oddělovače sloupců na znaménko tečky (`read.table("soubor.txt",sep=".")`) mít podobu datového rámce se čtyřmi sloupci

```
  V1  V2  V3 V4
1  1 5,6 2,7  3
2  4 8,0 1,9  9
```

přičemž sloupce budou po řadě třídy `integer`, `factor`, `factor`, `integer`. Zřejmě funkce `type.convert()` správně rozpoznala celá čísla v prvním a čtvrtém sloupci, nicméně bez

bližší specifikace desetinného oddělovače se nepodařilo odhalit druhý a třetí sloupec jako reálná čísla. Implicitním nastavení desetinného oddělovače je totiž znak tečka.

Vraťme se nyní k témuž souboru, ovšem s nastavením desetinného oddělovače na čárku (jak je obvyklé v kontinentální Evropě, jižní Americe, severní Asii a západní Africe[4]), tedy import provedeme pomocí funkce `read.table("soubor.txt",sep=".",dec=",")`. Podle očekávání vypadá výsledný rámec následovně

```
  V1  V2  V3 V4
1  1 5.6 2.7  3
2  4 8.0 1.9  9
```

a tedy zřejmě třídy jednotlivých sloupců tabulky byly identifikovány správně popořadě jako `integer`, `numeric`, `numeric` a `integer`.

Třetí příklad se stejným souborem může pouhou záměnou oddělovače sloupců a desetinného oddělovače vést ke zcela jiným hodnotám, přestože bude stejně jako předchozí výsledek formálně správný. Jeden soubor tedy na základě zvolené metody importu může poskytovat diametrálně odlišná data – určení správného postupu v takových případech (soubor bez záhlaví a bez specifikace tříd sloupců) závisí prakticky pouze na expertní znalosti datového souboru (kdo soubor vytvořil, v jaké zemi, v jakém software apod.) a je velmi obtížné až nemožné jej odvodit pouze ze samotných číselných hodnot. Použijme tedy nyní upravenou funkci `read.table("soubor.txt",sep=",",dec=".")` a získáváme výsledek ve tvaru

```
   V1  V2  V3
1 1.5 6.2 7.3
2 4.8 0.1 9.9
```

kde všechny sloupce spadají automaticky do třídy `numeric`.

Následující kombinace desetinných a sloupcových oddělovačů jsou široce užívány v praxi (uložení v těchto formách nabízí většina statistických software):

Oddělovač sloupců	Desetinný oddělovač	Popis
tabelátor " "	tečka (příp. čárka) " "	Standardní zápis v řadě formátů (txt, tsv, soft, aj.)
mezera " "	tečka (příp. čárka) " "	Standardní zápis v řadě formátů (txt, prn, aj.)
čárka ","	tečka "."	Comma separated values (csv) – původní verze.
středník ";"	tečka "."	Comma separated values (csv) – se středníkem.
středník ";"	čárka ","	Comma separated values (csv) – evropská verze.

Zejména výše uvedený formát csv (hodnoty oddělené čárkou) je pro svoji jednoduchost a nezávislost na konkrétním software velice populární pro výměnu dat mezi různými aplikacemi a rovněž široce využívaný institucemi např. ve státní správě. Uvedené tři varianty jsou nejběžnější, lze se ovšem setkat i s dalšími „mutacemi“.

[4] Použití desetinné čárky, tečky (a arabské notace) je ve světě poměrně nepravidelně rozloženo, přičemž počty uživatelů obou znamének se přibližně rovnají. Podrobnosti viz např. https://cs.wikipedia.org/wiki/Desetinn%C3%A1_%C4%8D%C3%A1rka.

Díky vysoké frekvenci použití csv souborů obsahuje jazyk R v základní instalaci pro rychlejší práci dvě rozšíření funkce `read.table()`, které jsou s ní víceméně totožné a liší se pouze implicitními hodnotami argumentů sep, `quote`, `dec`, `header`, `fill` a `comment.char`[5]. Funkce `read.csv()` odpovídá svým nastavením původní (americké) verzi formátu csv (třetí řádek v tabulce), naproti tomu funkce `read.csv2()` odpovídá evropskému (a tedy také českému) formátu csv souborů (pátý řádek v tabulce).

Kromě oddělovačů sloupců a řádků hraje ve všech uvedených typech souborů důležitou roli ještě třetí speciální znak, zjednodušeně nazýván uvozovky (quote). Jde o znak, který v sloupcích datového typu `character` umožňuje zapsat znaky právě dříve zmíněných oddělovačů, aniž by došlo ke zlomu řádku nebo sloupce. Nejčastěji jde o americké uvozovky (ASCII `0x22`), případně jednoduché uvozovky (ASCII `0x27`), možný je ale i jiný znak. Funkci uvozovek nejlépe osvětlí příklad načtení následující tabulky s oddělovačem sloupců čárka, kde první sloupce obsahuje název státu, druhý barvy na jeho vlajce a třetí moře omývající jeho pobřeží:

Turecko	bílá, červená	Černé, Středozemní
Zambie	žlutá, červená, zelená, černá	
Španělsko	žlutá, červená	Severní, Středozemní
Severní Korea	bílá, modrá, červená	Žluté
Rumunsko	žlutá, modrá, červená	Černé

Z tabulky je patrné, že druhý a třetí sloupec obsahují v textu buněk znak čárka, při zápisu za použití čárky jako oddělovače sloupců proto získáváme nečitelný záznam v podobě:

```
Turecko,bílá,červená,Černé,Středozemní
Zambie,žlutá,červená,zelená,černá
Španělsko,žlutá,červená,Severní,Středozemní
Severní Korea,bílá,modrá,červená,Žluté
Rumunsko,žlutá,modrá,červená,Černé
```

a po jeho načtení příkazem read.table("soubor.txt",sep=",") nesprávný výsledek v podobě

```
             V1     V2       V3       V4          V5
1       Turecko  bílá červená   Černé Středozemní
2        Zambie žlutá červená  zelená       černá
3     Španělsko žlutá červená Severní Středozemní
4 Severní Korea  bílá   modrá červená       Žluté
5      Rumunsko žlutá   modrá červená       Černé
```

Je tedy zřejmě zapotřebí sdělit funkci read.table(), které čárky jsou skutečnými oddělovači sloupců a které se nachází uvnitř buněk. Zde proto přichází ke slovu uvozovky, do kterých uzavřeme řetězce v buňkách obsahující čárku, jež není oddělovačem sloupců. Zápis souboru proto bude vypadat následovně:

[5] Na tomto místě odkazuji čtenáře k dokumentaci funkcí `read.csv()` a `read.csv2()` v R, která poskytuje vyčerpávající informaci o významu všech dalších argumentů funkcí (celkem je jich 25) zde: https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html.

```
Turecko,"bílá,červená","Černé,Středozemní"
Zambie,"žlutá,červená,zelená,černá",
Španělsko,"žlutá,červená","Severní,Středozemní"
Severní Korea,"bílá,modrá,červená","Žluté"
Rumunsko,"žlutá,modrá,červená","Černé"[6]
```

Import tabulky ze souboru lze nyní provést příkazem rozšířeným o specifikaci uvozovek nastavením argumentu `quote` na hodnotu `"\""`, tj. dvojitou uvozovku (zápis vyžaduje umístění zpětného lomítka před znak uvozovky, tak jak je vidět zde: `a<-read.table("pokus.txt",sep=",",dec=".",quote="\"")`:

```
             V1                          V2                  V3
1       Turecko                bílá,červená   Černé,Středozemní
2        Zambie žlutá,červená,zelená,černá
3     Španělsko               žlutá,červená Severní,Středozemní
4 Severní Korea          bílá,modrá,červená               Žluté
5      Rumunsko         žlutá,modrá,červená               Černé
```

Užitečnou možností, jak předejít problémům s automatickou identifikací tříd sloupců v importované datové tabulce, je přímá specifikace tříd pomocí argumentu `colClasses` funkce `read.table()`. Pokud je hodnota tohoto argumentu nastavena jinak než implicitní `NA`, je vyřazena z činnosti funkce `type.convert()` a zadané třídy se postupně aplikují na sloupce zleva, počínaje jmény řádků (pokud je délka argumentu kratší, než počet sloupců, argument se recykluje na požadovanou délku).

TIP: V případě, že mají všechny sloupce importované tabulky stejnou třídu, stačí zadat argumentu `colClasses` jedinou třídu, která se aplikuje na všechny importované sloupce. Zadáním hodnoty `colClasses="character"` dojde k zachování třídy `character` u všech sloupců (tedy je pouze vypnuta funkce `type.convert()` a k dalším změnám nedochází).

Výhodou tohoto přístupu oproti ruční konverzi jednotlivých sloupců, při níž může docházet ke ztrátě informace (například pokud se pokusíme načíst sloupec obsahující reálná čísla jako třídu `integer` nebo sloupec logických hodnot s občasnými textovými poznámkami striktně jako třídu `logical`) je zastavení funkce v případě, kdy není možné sloupec konvertovat na zadanou třídu (např. pokud se pokusíme konvertovat text na třídu `integer`). V některých případech nicméně může toto oříznutí nadbytečných informací být výhodou (a potom je třeba zadat konverzi ručně).

Použití argumentu `colClasses` je dobře názorné z následujícího příkladu, kde jsou mezi reálná čísla vsunuty semikvantitativní hodnoty typu „je menší než“ a v posledním sloupci jsou (náhodou) reálná čísla bez desetinných míst. Soubor `soubor.txt` tvaru

```
4.76,12.11,34.5,8
<2.5,13,2.5,11
<2.5,9.9,1.11,5
```

bude standardním způsobem `read.table("soubor.txt",sep=",")` načten jako

```
    V1    V2   V3 V4
1 4.76 12.11 34.5  8
2 <2.5 13.00 2.5  11
3 <2.5  9.90 1.11  5
```

[6] Povšimněte si, že na konci druhého řádku (Zambie) přibyla za uvozovkami ještě čárka, která odděluje druhý od třetího sloupce, ve kterém ovšem není žádná hodnota.

kde sloupce jsou pořadě tříd `factor`, `numeric`, `numeric`, `integer`.

Vnutíme-li nyní ve funkci `read.table()` pořadě sloupcům třídy následujícím způsobem `read.table("soubor.txt",sep=",",colClasses=c("character","numeric","numeric","numeric")`, dosáhneme zpracování prvního sloupce jako textu (který je lépe zpracovatelný než kategoriální třída `factor`) a zbývajících sloupců jako reálných čísel.

TIP: Pokud by bylo zapotřebí se v posledním uvedeném příkladě zbavit semikvantitativních hodnot a získat všechny sloupce třídy numeric, nelze jednoduše použít nastavení argumentu `colClasses= "numeric"`, protože by funkce oznámila chybu hned při pokusu o konverzi prvního sloupce.

V takovém případě je vhodnější provést konverzi všech sloupců ručně pomocí příkazu `as.data.frame(lapply(lapply(a,as.character),as.numeric))` (opakované použití funkce lapply postupně přes třídu character bylo popsáno výše). Samozřejmě tímto způsobem dojde k úplné ztrátě nečíselných hodnot a jejich nahrazení hodnotou NA:

```
    V1    V2    V3 V4
1 4.76 12.11 34.50  8
2   NA 13.00  2.50 11
3   NA  9.90  1.11  5
```

Nahrazení semikvantitativních hodnot jinými reálnými hodnotami (např. polovinou limitu) by pak předpokládalo práci s textovými řetězci třídy `character` (ponechávám čtenářům jako cvičení práce s řetězci, které není předmětem tohoto příspěvku).

Obdobně je užitečné využití argumentu `colClasses` při importu kalendářních dat, kdy je možné sloupec (automaticky indikovaný jako `factor`) přímo při importu konvertovat na třídu `Date` (`colClasses="Date"`).

3.1.2. Import dat s pevnou šířkou sloupce

V některých případech je zavádění oddělovačů sloupců méně efektivní než stanovení hranic mezi sloupci na základě počtu znaků od začátku řádku. Taková varianta se využívá především v případech, kdy mají všechny hodnoty ve sloupci stejnou bitovou délku (např. stejně dlouhé normalizované kódy, čísla se stejnou přesností, logické hodnoty apod.).

Import tabulky je v takovém případě podstatně jednodušší a prakticky jedinou informací, která je zapotřebí, jsou šířky sloupců udávané v počtech znaků jako celá čísla (argument `widths`). V jazyce R slouží tomuto typu importu funkce `read.fwf()`(zkratka FWF znamená fixed width format, tedy formát s pevnou šířkou).

Pro čtení souboru `soubor.txt` následující podoby

```
Adamov     4576 3,77
Boskovstejn 141 7,59
Cerhenice  164910,63
```

který obsahuje tři sloupce pevné šířky s názvem obce, počtem obyvatel a plochou katastru (v km^2) je zapotřebí odhadnout (nebo znát z dokumentace) šířky sloupců ve znacích – v tomto případě 11 pro název, 4 pro počet obyvatel a 5 pro katastrální rozlohu (s desetinným oddělovačem čárkou). Následně lze importovat soubor příkazem `read.fwf("soubor.txt",c(11,4,5),dec=",")`, přičemž funkce opět automaticky rozpozná třídy jednotlivých sloupců:

```
           V1   V2    V3
1 Adamov      4576  3.77
2 Boskovstejn  141  7.59
3 Cerhenice   1649 10.63
```

Výsledné třídy jsou `character`, `integer` a `numeric`. Potenciálně problematický je u funkce `read.fwf()` parametr `sep`, který je pojmenovaný stejně jako u `read.table()`, nicméně jako separátor ztrácí při pevných šířkách sloupce smysl. V tomto případě jde o separátor názvů sloupců tabulky v případě, že jsou definovány (argument `header=TRUE`) – platí totiž, že název sloupce může být delší než je šířka sloupce ve znacích a díky tomu není možné názvy jednoduše do prvního řádku zapsat a je tedy třeba je oddělit nějakým oddělovačem – a právě ten specifikuje argument `sep`. Například v souboru `soubor.txt` krevních skupin pacientů

```
rodné číslo@příjmení@krevní skupina
8605314212Hanák         AB
9012123615Rejžek        0
9106562773Rozkydalová   B
```

se šířkami sloupců 10, 14 a 2 znaků zřejmě názvy 1. a 3. sloupce překračují šířky těchto sloupců, přesto je lze příkazem
`read.fwf("soubor.txt",c(10,14,2),header=TRUE,sep="@",colClasses="character")` importovat ve formě

```
  rodné.číslo       příjmení krevní.skupina
1  8605314212 Hanák                      AB
2  9012123615 Rejžek                      0
3  9106562773 Rozkydalová                 B
```

Za povšimnutí zde rovněž stojí názvy prvního a třetího sloupce, kde byly v průběhu importu mezery nahrazeny tečkami; jde o důsledek nastavení argumentu `check.names=TRUE`, který v průběhu importu kontroluje, zda jsou názvy sloupců platná jména proměnných a v případě, že nejsou (tj. např. obsahují mezery, jsou zaměnitelné s čísly apod.) použije funkci `make.names()` pro jejich úpravu (obvykle spočívající v nahrazení problematických znaků tečkami). Funkci lze jednoduše vypnout nastavením `check.names=FALSE`.

3.1.3. Import dat z jiných typů souborů

Předchozí podkapitola se zabývala výhradně načítáním dat z textových souborů, tj. takových souborů, kde je informace zakódována do čitelných znaků o konkrétní bitové délce. Často se lze nicméně setkat s různými druhy binárních souborů, ve kterých jsou data binárně reprezentována podle vlastního zvláštního klíče, a jejich převedení do formy textu nedává žádný smysl. Typickým zástupcem tohoto typu souborů jsou soubory typu `.xls` a `.xlsx` oblíbeného software Microsoft Excel nebo vlastní formáty .rda a .rds jazyka R.

Tabulkový procesor Microsoft Excel je nejrozšířenějším software pro zpracování a ukládání dat a proto se práci s ním (nebo přinejmenším souborům vytvořeným jeho pomocí) v analytické praxi prakticky nelze vyhnout. Je tedy vhodné zmínit jak úskalí jeho použití pro zpracování dat (více o typických problémech způsobených nesprávným použitím Excelu v kapitole o typických problémech s importem dat), tak metody, jak efektivně excelovské soubory zpracovat v R.

Standardní formáty `.xls` (nekomprimovaný binární formát používaný do roku 2007) a `.xlsx` (komprimovaný formát využívající jazyka XML od roku 2007) lze importovat přímo

za využití jednoho z mnoha balíků jazyka R vytvořeného za tímto účelem[7]. Nicméně každé dostupné řešení obnáší další komplikace, když pro balík `gdata` je nutné nejprve instalovat prostředí jazyka Pearl, balík `xlsReadWrite` neumí přečíst novější verzi souborů, balík `XLConnect` je časově neefektivní při práci s většími objemy dat apod. Pravděpodobně nejefektivnější je balík `xlsx`, který umožňuje poměrně rychlé čtení a zápis do `.xls` i `.xlsx` souborů, nicméně vyžaduje instalaci balíku `rJava`, který zprostředkovává interface mezi jazyky R a Java. Instalace a načtení balíku `rJava` nicméně může u některých operačních systémů způsobit určité problémy[8].

V případě úspěšné instalace balíku xlsx je možné pro import dat využít funkce `read.xlsx()` a `read.xlsx2()` lišící se v rychlosti a míře využití jazyka Java, nicméně poskytující srovnatelné výsledky. Obě funkce umí identifikovat třídy jednotlivých sloupců listu včetně kalendářních dat, nicméně veškeré číselné hodnoty řadí do třídy numeric. Výhodou funkcí z balíku `xlsx` ve srovnání s ostatními uvedenými funkcemi je jejich schopnost automaticky rozpoznat oddělovače buněk i desetinné oddělovače a eliminovat tak nejčastější problém s importem reálných čísel, komplikovaná je však přenositelnost kódu na jiné počítače, kde nemusí správně pracovat balíky `xlsx` a `rJava`.

Uvažujme nyní následující soubor `soubor.xlsx`:

	A	B	C	D	E
1	kod	nazev	kredity	prumer	zmeneno
2	Bi7560	Úvod do R	2	1,82	8.9.2010
3	Bi7527	Analýza dat v R	2	2,16	20.4.2015
4	Bi8668	Matematická analýza v MAPLE	2	1,38	8.1.2015

Pomocí příkazu `ramec<-read.xlsx("soubor.xlsx",1,header=TRUE,encoding="UTF-8")` tabulku přiřadíme jako datový rámec do proměnné `ramec` takto:

```
     kod                          nazev kredity prumer    zmeneno
1 Bi7560                     Úvod do R       2   1.82 2010-09-08
2 Bi7527              Analýza dat v R       2   2.16 2015-04-20
3 Bi8668 Matematická analýza v MAPLE       2   1.38 2015-01-08
```

přičemž funkce `read.xlsx()` identifikovala první dva sloupce `kod` a `nazev` jako třídu `factor`, následující dva sloupce `kredity` a `prumer` shodne jako třídu `numeric` a zejména správně poslední sloupec jako třídu `Date`.

Problematika importu a také úprav a exportu do excelovských souborů pomocí balíku `xlsx` je velice široká a zájemci najdou podrobnější informace přímo v dokumentaci k balíku.

---

[7] Pěkný a poměrně rozsáhlý přehled dostupných řešení a jejich požadavků na další software v počítači uvádí Nicola Sturaro Sommacal na webové stránce http://www.milanor.net/blog/?p=779.

[8] Na 64 bitových počítačích s OS Windows je obvykle nutné mít nainstalovánu Javu jak v 32 bitové (obvykle ve složce `Program files (x86)`), tak v 64 bitové verzi (obvykle složka `Program files`), dále vyhledat složku obsahující 64 bitovou verzi souboru `jvm.dll` a nastavit adresu této složky do globální proměnné `JAVA_HOME` v jazyce R pomocí příkazu `Sys.setenv(JAVA_HOME="C:\\Program Files\\Java\\...adresa...\\")`.

Podrobně jsou (anglicky) návody na zprovoznění balíku rJava popsány na diskuzním fóru http://stackoverflow.com/questions/7019912/using-the-rjava-package-on-win7-64-bit-with-r.

Obecnější, nicméně méně komfortní způsob práce s excelovskými soubory vede cestou konverze souborů do některého z textových formátů (jako nejvhodnější se jeví formát `.csv` s oddělovačem buněk středníkem, případně shodný `.txt` formát) přímo v tabulkovém procesoru pomocí příkazu Uložit jako[9]. Datový soubor `.xls` resp. `.xlsx` je v takovém případě třeba otevřít v Excelu (nebo srovnatelném software) a uložit s volbou oddělovače do formátu `.csv`. Software automaticky identifikuje buňky obsahující znak oddělovače v rámci textových řetězců a opatří takové buňky uvozovkami (escapovacím znakem zvoleným při ukládání).

Ze znalosti parametrů uložení souboru v tabulkovém procesoru pak přímo vychází potřebné hodnoty argumentů funkce `read.csv()` použité pro import dat do R. Nejčastější chyby a problémy, ke kterým při tomto postupu dochází, jsou zmíněny v kapitole o problémech s importem dat.

Vlastní formáty pro ukládání dat v R `.rda` a `.rds` se navzájem liší především ve způsobu, kterým zapisují data (serializují data do bitového kódu). Zatímco formát `.rda` je standardem pro ukládání libovolných struktur, ovšem s menší efektivitou serializace, umožňuje formát `.rds` uložit vždy pouze jediný objekt, nicméně serializace je účinnější. Podstatný rozdíl z hlediska využití formátů v praxi spočívá rovněž ve vlastnosti formátu `.rda`, který uloží veškeré objekty včetně jejich názvů a při importu je opět s jejich původními názvy načte, naproti tomu formát .rds umožňuje uložit obsah objektu bez názvu a při importu je mu název přidělen podobně jako při importu dat ze souboru.

Pokud například budeme chtít uložit datový rámec `ramec` tohoto obsahu

```
     kod                        nazev
1 Bi7527              Analýza dat v R
2 Bi8668 Matematická analýza v MAPLE
```

do souboru `.rda`, použijeme jednoduchý příkaz `save(ramec,file="soubor.rda")`. Nyní se lze snadno přesvědčit o tom, že `soubor.rda` je binární soubor – při pokusu o jeho otevření v textovém editoru, získáme nicneříkající sérii znaků:

```
‹                                                                  }PÍ,@-w-P□,Ž]ö
      :
```

```
•WŁn   ŃÉë D-`bŕ©WíAr›VS”ę2żß7ßĚě×ţĚöm          `Ŕ™ŚSh22      ⊟`'ď%x   -
kđAé }ÜŹd&Ł+E#Ý-Ş«ăr>[6™łX8]™ Âë‡XŹ;`□ĆEĹwa¦j)ĐŚ˘{Áč™Ł1™ŘWĺ‰‡©<czN□$ž
»ŰnĘI\ýÓ
nżz˘Ztoä§8lúąĚ:$+‰oÓQ_r'Ł”*~čŰtNďď—;@ń «í}-
```

Opětovné načtení uložených dat do R tedy není možné žádnou z výše uvedených funkcí a budeme muset použít příkazu `load()`. Pokud před tím vymažeme obsah všech objektů (například použitím příkazu `rm(list=ls())`) a následně se pokusíme importovat data uložená v souboru `soubor.rda`

```
load(file="soubor.rda")
```

provede se import přímo do proměnné `ramec`, aniž bychom museli znovu definovat její název.

[9] Použitelnou variantou je rovněž kopírování dat přímo z listu tabulkového procesoru do R přes schránku operačního systému (nastavení argumentu `file="clipboard"`), nicméně tento způsob prakticky není reprodukovatelný a proto jeho použití nedoporučuji.

Naproti tomu formát `.rds` ukládá pouze data bez názvu původní proměnné. Vyjdeme-li tedy ze stejné proměnné `ramec` a provedeme uložení do `.rds` příkazem `saveRDS(ramec,file="soubor.rds")`, získáme soubor o obsahu

```
‹                                ‹ŕb```b`fbd`b2™…Á|f^ÍŔ˜Y      lN      Í–“Z–
šSd€e˜˜N™ć¦Fćž…™™šNÖäśÄb□F¸qi‰É%ů
E@Ö?ě3ţ‡[Ĺ                                                          ÖÁd'□—
□ó·*Q!%±D˜L!*,í›X’š›X’™ýP!Ş¤LÁ×1ŔÇbÓ□|vŁ:5/17ÝŹĚŮů)ůŞÔ24MśE
ůÍz0Ť`ź4 ‰˙···Ăa?Đ‰ziE@- 7  +CCŹ
```

I na velice malém souboru je vidět, že serializace do formátu .rds je o několik bytů kratší oproti .rda. Navíc import příkazem

```
readRDS(file="soubor.rds")
```

vypíše přímo na konzoli obsah souboru, ale nepřiřadí jej žádné proměnné. Pokud bychom chtěli tento obsah uložit do původní proměnné `ramec`, je nutné provést přiřazení ručně standardním přiřazovacím příkazem `ramec<- readRDS(file="soubor.rds")`.

3.2. Načtení dat ze schránky

Užitečným způsobem zejména při rychlé práci s malými objemy dat (např. při experimentování, ověřování výpočtů a vůbec všude tam, kde není nutné zajistit opakovatelnost analýzy) může být načítání dat přímo ze schránky operačního systému (clipboard).

Budeme-li v souladu s úvodem kapitoly předpokládat dvourozměrnou strukturu (tabulku), pak k načtení lze využít všechny zmíněné funkce `read.table()`, `read.csv()`, `read.csv2()` a `read.fwf()`, přičemž po zkopírování dat do schránky argument názvu souboru `file` jednoduše nastavíme na klíčové slovo `clipboard` například takto:

```
read.table("clipboard",sep=";",dec=",",colClasses="character")
```

TIP: Všechny výše zmíněné funkce vyžadují informaci o konci řádku také pro poslední řádek tabulky, proto je při kopírování z textového formátu pomocí myši vhodné „přetáhnout“ kurzor myši až pod poslední řádek nebo použít klávesovou zkratku pro výběr veškerého obsahu souboru. Při kopírování dat z tabulkového procesoru (např. Excel) je informace o konci řádku obsažena a není třeba žádných zvláštních úkonů.

Variantou pak může být funkce `readClipboard()`, která načte každý řádek tabulky do jednoho prvku (textového) vektoru[10].

3.3. Import dat z webové stránky

Pokročilejším způsobem importu dat je jejich přímé načítání z webových stránek identifikovaných jejich URL adresou. V R lze pro tento účel použít většinu výše uvedených funkcí, kde je argument `file` nahrazen URL adresou souboru. V drtivé většině případů nicméně HTML kód výsledné webové stránky neodpovídá struktuře textového souboru vhodné pro import do dvourozměrné tabulky a výsledek importu tak nedává žádný smysl.

Částečným řešením je v takovém případě načtení celé stránky do jednorozměrného vektoru pomocí jedné z funkcí `scan()` nebo `readLines()`, nicméně i v tomto případě bude po úspěšném importu zapotřebí se vypořádat s množstvím značek jazyka HTML nebo XML

[10] Viz obsáhlou dokumentaci k funkci `readClipboard()`.

v importovaných řetězcích a najít způsob, jak přeuspořádat importovaná data do sloupců a řádků podle původního designu.

Vhodnějším řešením tedy bude opět funkce z některého ze specializovaných balíků, která umožní přímo v rámci importu odstranit HTML značky, které využije pro uspořádání dat do datových rámců a vektorů.

Praktický balík poskytující řadu funkcí pro zpracování webových stránek v jazycích HTML a XML je distribuován pod názvem `XML`. Po instalaci a načtení balíku, který nemá žádné speciální požadavky na softwarové vybavení je možné použít dvojici funkcí `htmlParse()` a následně `readHTMLTable()`, které umí zpracovat veškeré tabulky na zadané HTML stránce a umístit je do jednoho seznamu (`list`), ze kterého je možné tabulky pomocí jednoduchého indexu vyvolat.

V následujícím příkladu půjde o zpracování tabulky s časovými údaji o průtoku řeky Svratky v Židlochovicích[11]. Nejprve je potřeba stránku naparsovat se správným kódováním do proměnné `stranka`:

```
stranka<-
tmlParse(file="http://hydro.chmi.cz/hpps/popup_hpps_prfdyn.php?seq=30
7151"
                    ,encoding="windows-1250")
```

a následně provést zpracování všech tabulek z HTML a uložit je do seznamu nazvaného `tabulky`:

```
tabulky<-readHTMLTable(stranka) # najde vsechny tabulky a prevede je
na data.frame
```

odtud již lze konkrétní tabulku jednoduše získat zadáním pořadového čísla tabulky v původním kódu webové stránky:

```
tabulky[[11]]
```

Výsledkem je `data.frame` obsahující původní tabulku, nicméně bez automatické identifikace tříd sloupců, které jsou všechny třídy `factor`:

```
        datum a čas stav [cm]  průtok [m3s-1] teplota [ °C]
1  06.08.2015 03:50        57            6.77
2  06.08.2015 03:40        57            6.77
3  06.08.2015 03:30        58            7.01
4  06.08.2015 03:20        58            7.01
5  06.08.2015 03:10        58            7.01
6  06.08.2015 03:00        58            7.01
7  06.08.2015 02:50        57            6.77
8  06.08.2015 02:00        57            6.77
9  06.08.2015 01:00        57            6.77
.....................................................etc.
```

3.4. Import dat z databází

Nejpokročilejším způsobem importu dat je načítání hodnot přímo z relačních databází SQL, ke kterým je možné se připojit prostřednictvím některého z vhodných balíků. Poměrně jednoduchý a komfortní způsob nabízí API rozhraní Open Database Connectivity (ODBC),

[11] Webová stránka Českého hydrometeorologického ústavu je přístupná online na adrese http://hydro.chmi.cz/hpps/popup_hpps_prfdyn.php?seq=307151.

které může zprostředkovat přístup do databáze přímo z prostředí R nezávisle na programovacím jazyku a operačním systému.

V případě existujícího propojení ODBC do databáze je třeba zvolit jeden z řady vhodných balíků pro práci s ODBC v rámci R. Příjemným a snadno použitelným je balík `RODBC`, který lze nainstalovat z repozitáře CRAN a který nabízí funkce pro integraci jazyka SQL do kódu v R.

Prvním krokem je vytvoření funkčního připojení (proměnná třídy `RODBC`) k vybrané databázi. K tomu poslouží funkce `odbcConnect()`, jejímiž argumenty jsou název databáze, uživatelské jméno a heslo. Vytvořené připojení uložíme do proměnné `pripojeni`:

```
pripojeni<-odbcConnect(nazev_db,uid=jmeno,pwd=heslo)
```

Vytvořené připojení lze nyní použít jako první argument funkce `sqlQuery()`, která umožňuje přímé SQL dotazy do databáze, přičemž výsledkem správně formulovaných dotazů v jazyce SQL jsou datové rámce obsahující hodnoty odpovídající dotazu:

```
sqlQuery(pripojeni,"SELECT * FROM 'Tabulka' WHERE
sloupec='kod_predmetu'")
```

Třída proměnných j určena na základě datového typu sloupce v SQL databázi.

TIP: V případě použití balíku `RODBC` lze s SQL dotazy, které vstupují jako druhý argument do funkce `sqlQuery()`, pracovat jako s klasickými textovými řetězci, včetně jejich spojování, generování a úprav. Při vkládání názvu tabulek a sloupců do textu SQL dotazu je však třeba zvolit opačný typ uvozovek (jednoduché × dvojité) než pro uzavření samotného dotazu, aby nedošlo k přerušení řetězce (viz příklad výše).

3.5. Typické problémy při importu (exportu) dat

V poslední části kapitoly o importu dat do R si představíme několik typických problémů vznikajících při rutinních postupech importu dat do R. Částečně již byly metody řešení těchto problémů zmíněny výše, nicméně vzhledem k frekvenci jejich výskytu je vhodné na ně alespoň krátce upozornit ještě opakovaně.

3.5.1. Záměna desetinné čárky a tečky

Volba znaménka tečky nebo čárky pro oddělení řádu desetin od řádu jednotek je závislá na kulturním prostředí, resp. konkrétním nastavení počítače, na kterém byla data naposledy uložena. Jazyk R pracuje na americký způsob výhradně s tečkou, proto je třeba v rámci importu vždy reálná čísla převést na formát s desetinnou tečkou:

- V případě importu z textových souborů je vhodné načíst proměnnou (sloupec tabulky) jako třídu character, následně využít funkce pro nahrazení čárky tečkou `gsub(",",".",promenna)`. Pozor – v případě záměny v opačném směru je třeba přidat do funkce argument `fixed=TRUE`, protože samotná tečka má ve funkci `gsub()` speciální význam!
- V případě importu z `.csv` souborů je situace totožná jako u souborů textových. Tabulkové procesory v případě `.csv` souborů ukládají data v textovém formátu tak, jak jsou zobrazena na monitoru (tečku jako tečku, čárku jako čárku).
- V případě `.xls` a `.xlsx` je reprezentace reálných čísel skrytá a tečka nebo čárka se zobrazují podle nastaveného národního prostředí. Pokud je využit speciální balík pro práci s excelovskými soubory, není třeba čárku nahrazovat, importuje se správně.

- V případě webových stránek záleží formát na textové podobě, v jaké jsou reálná čísla uvedena na stránce. Pokud dojde k načtení dat ze stránky jako třídy `factor`, je před převodem na reálné číslo zapotřebí převést proměnné nejprve na text (`character`).
- V případě relačních databází jsou reálná čísla načítána správně bez nutnosti dalších úprav.

3.5.2. Oddělovače vyšších řádů

Mimo oddělovače desetinných míst se lze setkat v datových souborech také s oddělovači vyšších řádů. V případě češtiny jde o oddělovač tisíců reprezentovaný mezerou nebo apostrofem, v anglickém prostředí se tisíce obvykle oddělují čárkou.

Situace s oddělovači tisíců je podobná výše uvedené situaci s oddělovačem desetinných míst s tím rozdílem, že v případě použití funkce gsub() je třeba oddělovač nahradit prázdným znakem. V případě společného nahrazení oddělovače tisíců, desetinného oddělovače a převedení na třídu `numeric` může vypadat pro číslo 3 544,11 příkaz takto: `as.numeric(gsub(",",".",gsub(c(" "),c(""),"3 542,11")))`.

3.5.3. Různá kódování diakritických znaků

Problém s různými druhy kódování národních znaků jednotlivých abeced má podstatně širší souvislosti, než je na tomto místě možné představit. Podstatnou informací je, že téměř všechny uvedené importní funkce umožňují zvolit kódování importovaných dat pomocí argumentu `encoding`. Nejčastějšími hodnotami v českém prostředí jsou `"Windows-1250"`, `"ISO8859-2"`, `"CP852"` a případně `"CP895"`.

3.5.4. Kalendářní data

Stejně jako u oddělovačů je situace s kalendářními daty bezproblémová v případě importu z excelovských souborů nebo relačních databází. V případě ukládání dat do formátu `.txt` nebo `.csv` v tabulkovém procesoru je vhodné před samotným uložením změnit formát kalendářního data na `RRRR-MM-DD`, který lze v R snadno převést na třídu `Date`. Variantou je ukládání informace o roce, měsíci a dni jako třída `integer` v oddělených sloupcích tabulky.

Nebezpečným problémem, který je často velmi obtížné opravit, je automatická konverze na datový typ datum v některých tabulkových procesorech. Reálná čísla vhodného tvaru (přesnost na jedno nebo dvě desetinná místa nepřekračující hodnotu 12) s desetinným oddělovačem tečkou jsou tak po otevření `.csv` nebo `.txt` souboru v tabulkovém procesoru převedena na kalendářní data reprezentovaná úplně jinými celými čísly.

Ukázka takové nesmyslné konverze je v následující tabulce:

původní hodnota (reálné číslo)	reprezentace v `.csv` souboru	převedená hodnota v tabulkovém procesoru	reprezentace hodnoty v tabulkovém procesoru
3,12	"3.12"	3. 12. 2015	42 341
9,34	"9.34"	1. 9. 1934	12 663
0,57	"0.57"	0.57	0,57

4. Řešená úloha v R

Následující úloha je jedním z mnoha možných postupů importu a zpracování reálných dat nad genovými expresemi amerického NCBI (National Center for Biotechnology Information). Úloha je doplněna interaktivními otázkami, které by měly posloužit k zamyšlení nad tím, proč byl zvolen právě uvedený postup, případně jakým způsobem by bylo možné data zpracovávat jinak (lépe).

4.1. Zadání

Cílem výzkumu je určit, jaký je rozdíl mezi nádorovou tkání (nádor tlustého střeva) a normální tkání tlustého střeva na úrovni genové exprese.

Nádory a normální tkáň byly podrobeny genetické analýze pomocí mikročipů. Následuje testování hypotéz mezi skupinou nádor a skupinou normal, přičemž cílem je určit, které geny jsou odlišně exprimované.

Pro analýzu jsou k dispozici tři tabulky:

1. P×N tabulka genových expresí (v řádcích jsou sondy představující geny, ve sloupcích pacienti).

2. N×K tabulka klinických dat (v řádcích pacienti, ve sloupcích proměnné).

3. P×S tzv. anotační tabulka: která sonda patří ke kterému genu, jaký je název genu, a jeho popis (v řádcích sondy, ve sloupcích informace o genu, ke kterému patří).

Data jsou přístupná na adrese: http://www.ncbi.nlm.nih.gov/sites/GDSbrowser?acc=GDS4382, kde je na stránce vpravo kolonka Download, kde je možné si zvolit vhodný datový formát:

1. Dataset SOFT file: (ftp://ftp.ncbi.nlm.nih.gov/geo/datasets/GDS4nnn/GDS4382/soft/GDS4382.soft.gz) – P×N tabulka genových expresí - začíná pod řádkem s hodnotou "!dataset_table_begin" a končí nad řádkem "!dataset_table_end", sloupce mají hlavičku a jsou oddělené za použití tabelátoru.

2. Klinická tabulka s ID pacienta (ID vypadá takto: GSM800759) se dá buď získat z hlavičky tabulky číslo 1. (začíná pod řádkem s hodnotou "#IDENTIFIER = identifier" a končí nad řádkem "!dataset_table_begin") nebo se dá získat z HTML tabulky na adrese http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE32323 v sekci "Samples(44)".

3. Annotation SOFT file - anotační tabulka: (ftp://ftp.ncbi.nlm.nih.gov/geo/platforms/GPLnnn/GPL570/annot/GPL570.annot.gz) – P×S tzv. anotační tabulka – začíná pod řádkem s hodnotou "!platform_table_begin" a končí nad řádkem "!platform_table_end", sloupce mají hlavičku a jsou oddělené za použití tabelátoru.

Struktura všech tabulek je pevně daná (ale soubor obsahují také hlavičku).

4.2. Import dat

První variantou je stáhnout první a třetí soubor z uvedených URL adres do počítače, uložit je na disk a postupně pomocí vhodného software archivy rozbalit. Získáme tak dva soubory nazvané `GDS4382.soft` a `GPL570.annot`. Otevřením souborů v textovém editoru se snadno přesvědčíme, že jde o textové soubory (nikoliv binární).

OTÁZKA 1: Jak se jmenuje funkce, kterou je možné rozbalit soubory přímo v prostředí jazyka R, abychom nemuseli soubory stahovat na disk a rozbalovat archivy za použití jiného software? Proveďte a porovnejte výsledky.

Zřejmě soubory kromě samotných dat obsahují poměrně rozsáhlou hlavičku, která znemožňuje načtení požadovaných tabulek pomocí funkce `read.table()`. Uchýlíme se tedy k variantě vytvoření jednorozměrného vektoru pomocí funkce `readLines()` a v dalším kroku ořízneme všechna přebytečná data kromě tabulky, která obsahuje pro analýzu podstatné údaje.

```
gds0<-readLines("GDS4382.soft")
```

V případě prvního souboru `GDS4382.soft` j to tabulka která následuje po řádku `"!dataset_table_begin"` a končí před řádkem `"!dataset_table_end"`. Použijeme tedy jednoduché schéma s podmíněnou funkcí `which()` k výběru pouze příslušných řádků.

```
gds1<-
gds0[(which(gds0=="!dataset_table_begin")+1):(which(gds0=="!dataset_t
able_end")-1)]
```

V dalším kroku využijeme funkce `strsplit()` k rozpadu vektoru gds1 podle oddělovače tabelátoru (zápis `"\t"`). Tím získáme vektor `gds2`, jehož délka odpovídá počtu buněk výsledné tabulky.

```
gds2<-strsplit(gds1,split="\t")
```

Další krok je méně intuitivní, neboť využijeme neobvyklou funkci `rbind.data.frame()` pro vytvoření datového rámce z jediného vektoru:

```
gds<-do.call(rbind.data.frame,gds2)
```

OTÁZKA 2: Ze zadání víme, že tabulky mají určenou pevnou strukturu, lze se proto domnívat, že složitý postup s expanzí vektoru podle oddělovačů není nutný a místo toho by mohla být použita funkce pro import tabulky s pevnou šířkou sloupců. Jak se tato funkce jmenuje? Jak ji lze použít na danou strukturu? Proveďte a porovnejte výsledky.

Nezávisle na tom, jakým způsobem jsme dospěli k datovému rámci `gds` nyní z prvního řádku tohoto rámce utvoříme názvy sloupců a poté tento řádek vyjmeme:

```
colnames(gds)<-gds[1,]
gds<-gds[-1,]
```

OTÁZKA 3: V čem se bude lišit postup importu druhého a třetího souboru ze zadání? Lze využít tytéž funkce? Načtěte i druhý a třetí soubor ze zadání a vytvořte z nich rámce pojmenované `id` a `ann`, které budou potřeba pro další analýzu. Zajistěte (lze využít funkci `strsplit()`), aby první i druhý rámec disponovaly sloupcem, ve kterém bude pouze ID pacienta, vhodným ke spárování souborů. Stejně tak zajistěte, že v druhém souboru bude jeden sloupec, obsahující pouze informaci o zařazení pacienta do skupiny (nádor × normal).

Jakmile máme vytvořeny všechny tři soubory, je zapotřebí spárovat první a druhý rámec (`gds` a `id`) tak, aby bylo zřejmé, který sloupec expresí genů náleží ke kterému pacientovi. Toho dosáhneme vytvořením vektoru skupina, který bude obsahovat hodnoty `"cancer"` a `"normal"` v takovém pořadí, jak jsou seřazeny sloupce v rámci `gds`.

Jedno z možných řešení je využití funkce `match()`, která zjišťuje, který prvek prvního vektoru se rovná kterému prvku vektoru druhého, přičemž za první vektor dosadíme

(příslušně zkrácené) názvy sloupců rámce `gds` a za druhý vektor první sloupec rámce `id` (také vhodně zkrácený funkcí `substr()`).

```
skupina<-substr(id[match(substr(colnames(gds)[-c(1,2)],4,12),
                         substr(id[,1],2,10)),8],1,6)
```

OTÁZKA 4: První sloupec rámce gds obsahuje id sondy, druhý obsahuje identifikaci genu. Zřejmě se hodnoty ve druhém sloupci opakují, tj. některé geny byly měřeny více sondami. Jakým způsobem provedete výběr relevantních řádků pro analýzu? Které ze dvou (nebo více) měření exprese genu hraje významnější roli? Navrhněte způsob výběru relevantních řádků a proveďte jej.

Nyní máme datový rámec `gds` obsahující unikátní geny na řádcích a 34 sloupců s expresemi genů u dvou skupin pacientů (zadaných proměnnou `skupina`) a samotnou proměnnou `skupina`; tedy můžeme přikročit k analýze meziskupinových rozdílů jednotlivých genů. Pro analýzu zvolíme cyklus a každý gen budeme testovat samostatně (17 hodnot oproti 17 hodnotám).

OTÁZKA 5: Který test je nejvhodnější pro srovnání exprese genů, tj. srovnání hodnot ve dvou přibližně stejně velkých skupinách? Proč? Proveďte a interpretujte výsledky.

Zpracování dat pomocí OpenRefine

Miroslav Kubásek

Institut biostatistiky a analýz, Masarykova univerzita, Brno; e-mail: kubasek@iba.muni.cz

Abstrakt

Článek představuje nástroj OpenRefine, který je určený pro pokročilé zpracování a čištění dat. Čtenář bude schopný nástroj nainstalovat, importovat, exportovat data a provádět opravy a transformace dat, včetně importu informací z externích zdrojů.

Klíčová slova

OpenRefine, GREL, data mining, regex, fusion tables, tutorial

1. Úvod

Data jsou někdy označována jako novodobé zlato, zvláště v dnešní datově-orientované ekonomice, kdy okolo nás existuje nepředstavitelné množství dat, získávají data čím dál více na hodnotě. Zkusme o těchto cenných datech přemýšlet jako o diamantech. Zprvu je to velmi syrový, nezpracovaný materiál, mnohdy skrytý v ohromném množství bezcenné hlušiny. Ale pokud se nám podaří surový diamant najít, pohrajeme si s ním, vybrousíme ho, vyleštíme, najednou získá ohromně na ceně a to je to, o co nám jde. Přesně tak se můžeme dívat i na nástroj OpenRefine.[1] Umožňuje nám pokročilými způsoby importovat, transformovat a exportovat data v nejrůznějších formátech. Ukážeme si,[2] jak v datech opravit drobné chyby, změnit strukturu dat, až po to jak data rozšířit o informace z externích zdrojů a jednoduchým způsobem je vizualizovat.

1.1. OpenRefine v porovnání s dalšími nástroji

OpenRefine	Tabulkový editor (Excel)	Databáze
Je možné dávkově zpracovat jak řádky, tak i sloupce.	V jeden okamžik lze editovat jen jednu buňku.	Je potřeba znát dotazovací jazyk (SQL)
Umožňuje zkoumat a transformovat data.	Vhodné pro vkládání dat a provádění výpočtů.	Pouze základní možnosti transformací.
Není potřeba definovat schéma dat.	Není potřeba definovat schéma dat.	Je potřeba definovat schéma zpracovávaných dat.
Data jsou viditelná v každém kroku editace.	Data jsou viditelná, omezený počet řádků.	Data jsou skryta, dokud je dotazem nevyexportujete.
Mnohem více interaktivní a vizuální nástroj.	Vizualizace dat je pouze základní.	Použití příkazového řádku pro spouštění dotazů.

[1] http://openrefine.org/

[2] Všechny použité datové soubory použité v tomto textu včetně instalačních souborů jsou k dispozici na adrese https://github.com/MiroslavKubasek/OpenRefine-tutorial

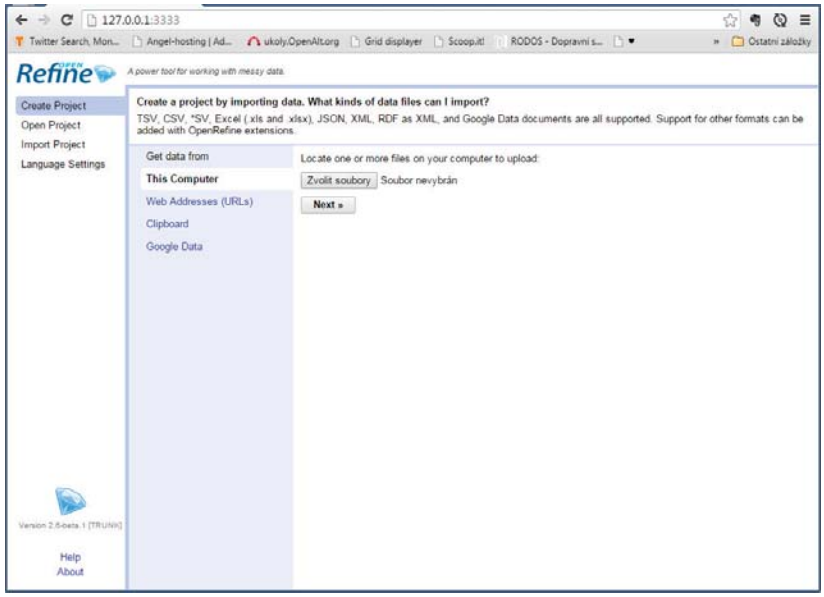
2. Základy OpenRefine

2.1. Instalace

OpenRefine je aplikace naprogramovaná v jazyce Java, proto se ujistěte předtím, než budete pokračovat dále, zdali máte na počítači nainstalované běhové prostředí tohoto jazyka (JRE[3]). V případě potřeby si JRE stáhněte a nainstalujte na základě instrukcí na adrese `http://java.com/en/download/`.

Stáhněte si s adresy `http://openrefine.org/download.html`[4] instalační soubor pro Váš operační systém. Soubory rozbalte do složky, do které bude mít OpenRefine přístup (ne do `C:/Windows/…`), může se jinak stát, že se vám nebudou ukládat projekty nebo nástroj nepůjde spustit.

Po rozbalení spusťte soubor **`refine.bat`** (v případě MS Windows), měl by vám vyskočit příkazový řádek a po chvíli automaticky otevřít nové okno ve webovém prohlížeči. Pokud se Vám okno samo neotevře, přejdete v prohlížeči na adresu **`http://127.0.0.1:3333/`**. Měla by se Vám ve webovém prohlížeči zobrazit následující stránka.



Ačkoliv se prostředí OpenRefine jeví jako webová stránka, veškeré operace probíhají lokálně (bez nutnosti připojení k internetu).

[3] Java Runtime Environment

[4] Dále v textu budeme předpokládat, že jste si nainstalovali verzi 2.6-beta1 pro OS Windows. Jednotlivé verze programu se mohou navzájem drobně lišit. Návod jak program nainstalovat na počítače s OS Mac nebo Linux najdete přímo na adrese http://openrefine.org/download.html

2.2. Orientace v programu

Úvodní menu programu je rozděleno do čtyř základních částí. Vyobrazení výše, ukazuje sekci tvorby nového projektu. Dalšími částmi jsou pak otevření již existujícího projektu, import projektu a nastavení jazyka[5].

Vytvořit projekt (Create project)

Vlastní vytvoření nového projektu se skládá ze tří kroků. Prvně vyberete soubor se zdrojovými daty a stiskneme tlačítko „Next". V druhém kroku „Preview" máme možnost vidět, jak se programu podařilo načíst naše data. Některé běžné formáty rozezná sám, s některými mu budeme muset trochu pomoci. Máme zde možnost programu říci, jak má data načíst „Parse data as" a také můžeme nastavit různé parametry těchto importů (např. vynechání řádků od začátku, od konce, jak interpretovat prázdné řádky, nastavit kódování znaků apod.). Jakmile máme správně nastaveny parametry importu, tak již jen projekt pojmenujeme a stisknutím tlačítka „Create Project" nakonec projekt necháme vytvořit.

OpenRefine umožňuje načíst data v následujících formátech:

- Comma-Separated Values (.csv), Tab-Separated Values (.tsv), ostatní *SV
- MS Excel dokumenty (.xls a .xlsx), Open Document Format (.odf, .ods)
- JavaScript Object Notations (.json)
- XML a RDF jako XML (.xml)
- Line-based formats (záznamy, logy, apod.)
- Google Data (Google Spreadsheet, Fusion Table)

Umístění dat pro import si můžete vybrat z následujících možností:

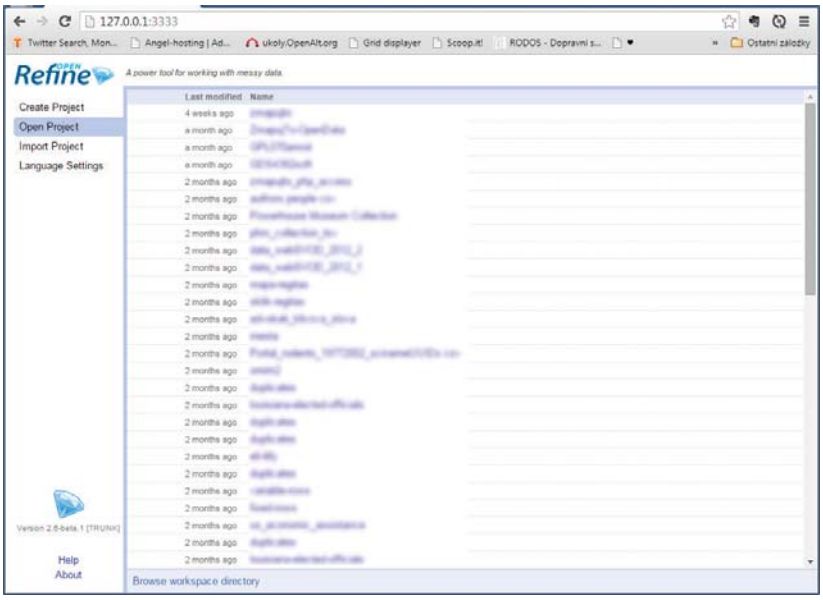
- **This computer** - Možnost tvorby nového projektu na základě dat, která máte uložena ve svém počítači. Zpracovatelné formáty souborů jsou uvedeny výše.
- **Web adresses (URLs)** - Na základě vložené jedné nebo více URL OpenRefine stáhne obsah, který na dané URL nalezne a pokusí se v dalším kroku automaticky najít separátor (často potřebuje pomoci) nebo vás nechá vybrat konkrétní obsah, který chcete stáhnout.
- **Clipboard** - Česky „schránka" vám umožňuje buď zápisem nebo vložením (CTRL+V) textového záznamu vytvořit projekt.
- **Google Data** - V této sekci můžeme povolit OpenRefine propojení s vaším Google Drive účtem[6]. Po propojení máte možnost nahrávat do OpenRefine dokumenty z Google Drive jako nové projekty.

Otevřít projekt (Open project)

Objeví se Vám seznam již vytvořených projektů. Projekty je možno v tomto rozhraní také přejmenovat či smazat.

[5] Aktuálně si můžete vybrat pouze mezi anglickou a italskou jazykovou mutací programu.

[6] Pro propojení a načtení dat z Google Spreadsheet nebo Fusion Table Vás program vyzve k přihlášení k účtu Google a k autorizaci.



Importovat projekt (Import project)

Import již existujícího projektu se zachovanými všemi úpravami v "Undo/Redo" ve formátu .tar nebo .tar.gz.

2.3. Práce s projektem

Základní pracovní plocha každého projektu v OpenRefine sestává ze 4 základních částí.

Filtrace a změny na projektu

V části filtrace ***Facet/Filter*** budete později pracovat s facety a textovou filtrací projektu. Facety představují v našem případě seskupení dat na základě zvoleného segmentu. Textovou filtrací můžete buď na základě regulárního výrazu, nebo fráze filtrovat odpovídající obsah projektu. V případě jakékoli filtrace dat je třeba si uvědomit, že operace, prováděné na sloupcích či buňkách budou vždy aplikovány pouze na aktuálně zvolený dataset.

Druhou částí jsou změny projektu, tedy sekce ***Undo/Redo*** (Zpět/Vpřed). V této sekci naleznete veškeré provedené kroky na projektu a můžete se v nich libovolně vracet zpět nebo vpřed. Postup na projektu bude zachován i při exportu celého projektu, tedy kdokoli kdo poté celý projekt importuje, uvidí veškeré podniknuté kroky a může s nimi libovolně pracovat.

Sekce ***Undo/Redo*** zároveň obsahuje dvě důležité funkce a to ***Extract***…(vytáhni) a ***Apply***…(aplikuj). Tyto funkce vám dovolují kdykoli postup na projektu přes Extract vyexportovat do textového souboru a aplikovat na jakýkoli jiný projekt za pomoci Apply.

Data projektu

V datové části naleznete náhled dat, která projekt obsahuje a možnost práce s nimi. Šipky u sloupců slouží jako menu funkcí na ně aplikovatelných (v případě zvolení šipky u sloupce se aplikuje daná funkce/filtrace pouze na sloupec, který jste zvolily). Zvláštní funkci plní sloupec úplně vlevo, jehož funkce jsou aplikovány na celý projekt. Jednotlivé řádky je pro

pozdější filtraci možno označit buď pomocí vlajky nebo hvězdičky a potom s daným segmentem pracovat tak, že si vytvoříte Facet podle vlajky nebo hvězdičky.

Základní informace o projektu

Jednoduchý a přehledný panel o počtu řádků, záznamů s možností nastavení počtu řádků, které se vám v náhledu dat projektu budou zobrazovat.

Rozdíl mezi Row a Record:

- Row = Řádek
- Record = Záznam

Pokud v projektu se 7000 řádky budete mít seřazená data a smažete například přes funkci *Blank Down* duplicitní řádky, budete mít stále 7000 řádků, ale například 6000 záznamů (1000 bylo duplicitních).

Export a nápověda

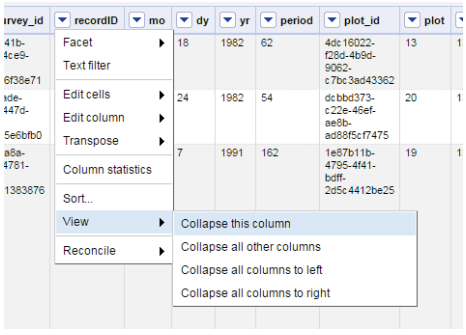
V pravém horním rohu naleznete možnost otevření projektu, nápovědy a především možnost exportu projektu. Exportovat můžete do formátů, které můžete zároveň importovat do OpenRefine. Zajímavou funkcí exportu je především ***Custom tabular exporter***, ve kterém si nejen můžete zvolit, jaké sloupce chcete exportovat, ale je zde i možnost, v případě propojení s vaším Google účtem, exportovat projekt do Google Spreadsheetu či do Google Fusion table.

2.4. Práce s daty

Náhledy na data nám mohou usnadnit orientaci v datasetu projektu. OpenRefine disponuje několika efektivními funkcemi, které Vám mohou zpříjemnit a urychlit práci.

Views (náhledy)

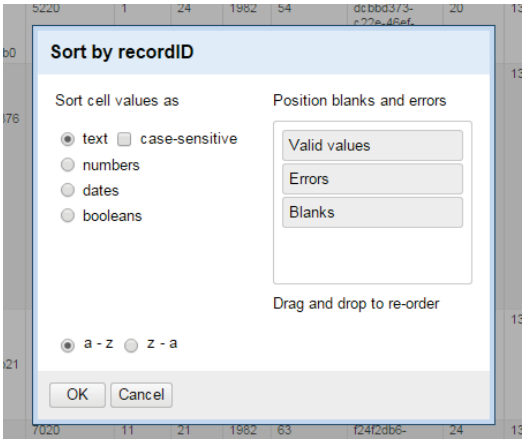
Náhledy nám umožňují schovat sloupce, které nás v dané chvíli nezajímají. Tuto funkci naleznete po kliknutí na příslušný sloupec, můžete buď sloupec schovat „***Collapse this column***“, schovat všechny ostatní sloupce „***Collapse all other columns***“ nebo schovat všechny sloupce nalevo či napravo od příslušného sloupce „***Collapse all columns to left/right***“.



Po dvojitém kliknutí na mezeru mezi sloupci se sloupec znovu objeví. Data zůstávají ve zpracování, jako kdyby byl sloupec normálně viditelný, globální operace se proto mohou projevit i na schovaných sloupcích/řádcích (na rozdíl od filtrace).

Sort (řazení)

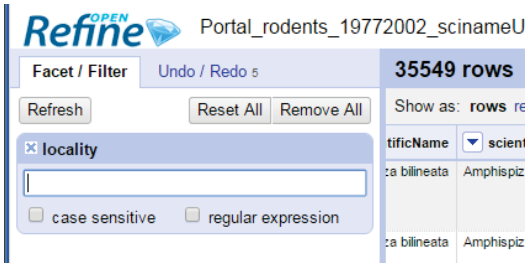
Řazení dat podle příslušného sloupce. Řadit buňky můžete jako textové, numerickém datové nebo boolean (true/false). Je důležité dbát na to, že pokud seřadíte data, dostáváte pouze nový pohled na ně a doopravdy nejsou seřazeny do doby, než nad sloupci zaškrtnete v rozbalovacím tlačítku ***Sort -> Reorder rows permanently*** nebo pokud chcete náhled zrušit, tak ***Sort -> Remove sort***.



Filtrace dat

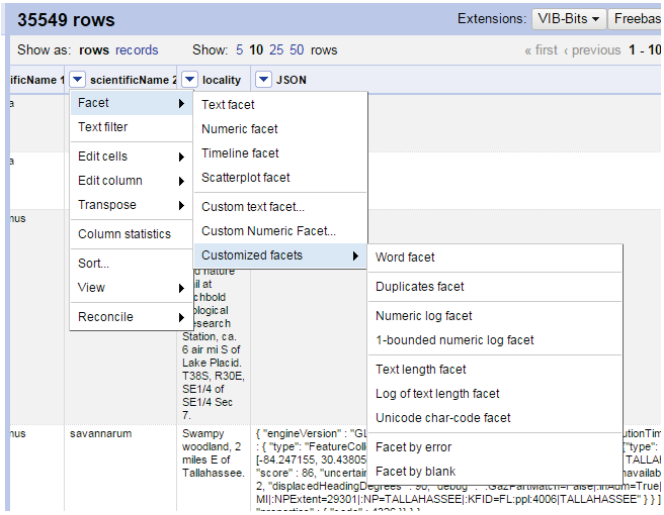
Filtrace dat nám pomáhá vytyčit určitý segment dat, na který se chceme buď podívat nebo přímo s ním pracovat. OpenRefine má tu výhodu, že při filtraci dat pracujete vždy pouze s vybraným segmentem dat, tedy pokud aplikujete jakoukoli funkci, aplikuji se pouze na aktuálně vyfiltrovaná data.

Základním typem filtrace, je filtrace textová. Na základě zadaného řetězce vyhledá buňky v daném sloupci, které jej obshaují. Po zatržení je možné použít regulární výrazy (***regular expression***) nebo zajistit, aby byl řetězec ***case sensitive*** (rozlišoval malá a velká písmena).



Facety

Facety jsou seskupení dat na základě zvoleného klíče. To znamená, že při zvolení textového facetu se bude snažit OpenRefine dávat do jednoho segmentu k filtraci obsahově stejné buňky. K dispozici máte řadu různých obecně definovaných facetů, od textových, přes numerické po přizpůsobené v sekci ***Customized facets***.



Za zmínku stojí především přizpůsobený s názvem ***Word facet***. Tento facet vám zobrazí četnost výskytu slov ve sloupci. Hodí se především při tvorbě analýzy klíčových frází a po stažení HTML kódu z cizích stránek a jejich analýzy.

Jedinou výjimkou je tzv. ***Scatterplot facet***, což není ani tak facet, jako vizualizace 2D grafu vztahu označeného sloupce s jiným sloupcem. To znamená, že pokud u některého sloupce dáte ***Scatterplot facet***, objeví se vám obrazovka, na které můžete určit se kterým sloupcem má/může být promítnut. OpenRefine má tuto funkci poněkud nedokončenou. Na grafu chybí například měřítko hodnot a nedá se s ním téměř vůbec pracovat.

2.5. Práce se sloupci

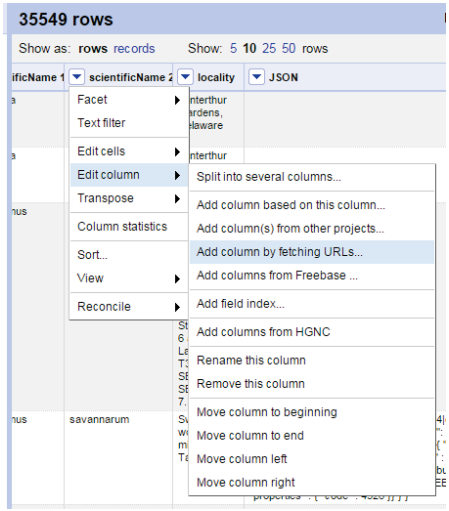
Sloupce jsou jeden ze základních prvků každého projektu. O to důležitější je práce s nimi. Níže naleznete základní operace, které můžete se sloupci provádět a příklady.

Split into several columns **(Rozdělení sloupce na více sloupců)**

Představte si, že máte v datasetu v jednom sloupci více hodnot. Například GPS souřadnice pozice „50°4’15.963″N, 14°24’3.179″E“. Tyto souřadnice chcete rozdělit na dva různé sloupce, přičemž jeden bude obsahovat pouze severní a druhý pouze východní souřadnice. To uděláte za pomoci funkce Split into several columns, kde jako separátor mezi souřadnicemi nastavíte čárku a OpenRefine vám vytvoří nový sloupec s oddělenou částí dat a v původním ponechá první část. ***Split into several columns*** tedy slouží k rozdělení příslušného sloupce na základě separátoru.

Add column based on this column

V překladu „Přidat sloupec na základě dat sloupce“. Jedná se o velmi často používanou operaci, která na základě dat v příslušném sloupci vytváří nový sloupec s výstupem (Příklad: Pokud máte sloupec s celým jménem uživatelů a chcete vytvořit nový sloupec pouze s jejich příjmeními). V tom případě u sloupce s celým jménem dáte ***Edit column -> Add column based on this column***, vyplníte název nového sloupce a do ***Expression*** vložíte `value.split(" ")[1]`. Funkce split je detailněji rozebrána v kapitole popisující funkce na řetězcích.



Add column by fetching URLs

V českém překladu: „Přidat sloupec na základě dat stažených z URL“. Nutno poznamenat, že při tomto procesu můžete stále pracovat s daty z příslušného sloupce. U přidávání nového sloupce se objevuje jedna nová sekce, nazvaná ***Throttle delay***. Jedná se zpoždění mezi dotazy, které bude OpenRefine odesílat v ms (nikoliv ve vteřinách).

Příklad: Pokud máte sloupec plný URL a chcete k nim nastahovat data o jejich sdílení a likes, uděláte to tak, že u daného sloupce dáte ***Edit column -> Add column by fetching URLs***, nazvete nový sloupec, zadáte zpoždění (v tomto případě doporučuji 500 ms) a do ***Expression*** vložíte následující výraz: `http://graph.facebook.com/?ids= + value` . Po dokončení získáte v novém sloupci výstup ve formátu JSON. O JSON, jeho parsování a širším využití tohoto procesu naleznete více informací v části věnované příkladům.

Add columns from Freebase

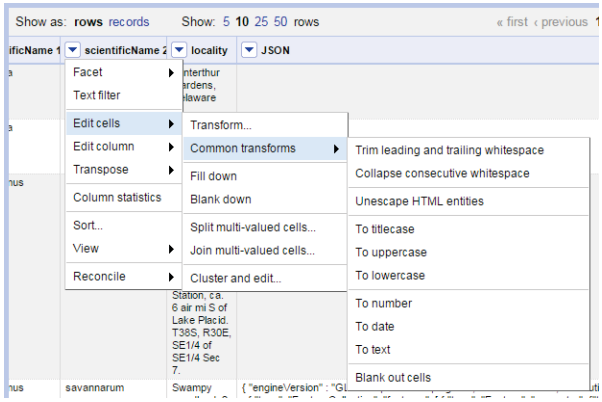
„Přidat sloupec na základě synchronizace obsahu buněk s Freebase”. Tato funkce je přímo navázána na použití ***Reconcilation***, které podrobněji naleznete v sekci praktické příklady.

Další dostupné akce se sloupci

- ***Rename this column*** - přejmenování sloupce
- ***Remove this column*** - odstranění sloupce
- ***Move column*** - posun sloupce, na výběr máte ze 4 možností:
    - ***Move column to beginning*** - přesun sloupce na začátek
    - ***Move columnt to end*** - přesun sloupce na konec (za všechny sloupce)
    - ***Move column left*** - přesun sloupce o jednu pozici doleva
    - ***Move column right*** - přesun sloupce o jednu pozici doprava

2.6. Základní operace s buňkami

Buňky (*Cells*) jsou základní jednotkou OpenRefine. Každá buňka obsahuje nějakou hodnotu, která může být textového, numerického nebo datového formát a přísluší ji právě jeden sloupec.



***Transform* (transformace)**

Změna obsahu buněk na základě vložené funkce v *GREL*, *Jython* nebo *Closure*. O GREL a funkcích, které se dají využít při transformaci, naleznete více informací v kapitole věnované funkcím.

OpenRefine také nabízí základní set předpřipravených transformací ***Common transforms***, které můžete rovnou aplikovat:

- ***Trim leading and trailing whitespace*** - odstraní mezery na koncích a začátcích buněk ve sloupci
- ***Collapse consecutive whitespace*** - pokud v buňce nalezne dvě po sobě jdoucí mezery, změní je pouze na jednu
- ***Unescape HTML entities*** - převede HTML na klasické znaky (například `&#33;` na !
- ***To titlecase*** - každé slovo buňky bude začínat velkým písmenem
- ***To uppercase*** -změní text v buňkách na velká písmena
- ***To lowercase*** - změní text v buňkách na malá písmena
- ***To number / To date / To text*** - změna formátu buněk na číslo / datum / text
- ***Blank out cells*** - smaže obsah všech buněk příslušného sloupce

***Fill down* (vyplň pod)**

Fill down je funkcí, která vyplní prázdné buňky sloupce buňkami nad nimi. Je proto důležité dbát na pořadí buněk ve chvíli, kdy chcete tuto funkci využít.

***Blank down* (vymaž pod)**

Blank down je funkcí opačnou k Fill down. Pokud OpenRefine nalezne v příslušném sloupci dvě stejné buňky pod sebou, smaže té níže postavené obsah buňky.

Tato funkce se často používá při zbavování se duplicit (příkazy ***Sort* + *Edit cells* -> *Blank down***).

***Split multi-valued cells* (rozdělení řádků s více hodnotami)**

Rozdělení řádku na základě pevně určeného separátoru. Funkce podobná ***Split into several columns***, s tím rozdílem, že nerozdělí obsah buněk na nové sloupce, ale rozdělí hodnoty do nových řádků.

Příklad: Při dělení hodnoty „`7.1`“ s použitím separátoru „.“ zůstane na řádku hodnota `7` a do nového řádku se vepíše hodnota 1.

***Join multi-valued cells* (spojení řádků s více hodnotami)**

Opačná funkce k funkci ***Split multi-valued cells***. Podívá se o řádek níže, a pokud je v daném řádku hodnota s prázdným sloupcem úplně nalevo, vloží tuto hodnotu do buňky výše a oddělí hodnoty separátorem.

Příklad: Po rozdělení hodnot, viz příklad výše, na stejném sloupci spustíme funkci ***Edit cells -> Join multi-valued cells*** a jako separátor určíme `"."`. Dostaneme se tak na původní hodnotu, tedy `"7.1"`.

***Cluster* (Clusterizace)**

Clusterizace je slučování hodnot na základě podobnosti. Hodí se především ve chvíli, kdy v jednom sloupci máte soubor hodnot, které jsou podobné, ale ne stejné a chcete je změnit na ucelený název.

Například pokud ve sloupci `"Země"` budete mít uvedeny země původu objektů v datasetu a některé buňky sloupce `"Země"` budou pro označení České republiky užívat názvy jako: `"Česká republika"`, `"Česká Republika"`, `"České Republiky"`, `"Ceska Republika"`, `"CESKA REPUBLIKA"`. Tyto všechny názvy pomocí clusterizace spojíte do jednoho primárního tvaru, který si zvolíte.

Při nacházení podobných frází je třeba vybrat konkrétní metodu, kterou OpenRefine použije. Mezi metody, které k cluesterizaci můžete využít, patří:

- ***Key collision***
- ***Nearest neighbor***
- … a k nim přidružené funkce.

Clusterizace se také často využívá v pozdějších fázích analýzy klíčových frází pro slučování frází na jednotný tvar. Konkrétní návod jak na to naleznete v příkladu clusterizace a slučování hodnot.

3. Funkce jazyka GREL

Zkratka **GREL** *Google Refine Expression Language* (nyní již *OpenRefine Expression Language*, ale zkratka zůstala zachovaná). Jak už název sám o sobě napovídá, jedná se o jazyk, jehož zápisem můžete provádět základní i pokročilé funkce pro práci s obsahem projektu. Níže si představíme výběr nejpoužívanějších funkcí. Kompletní popis funkcí lze najít v nápovědě programu, nebo na adrese https://github.com/OpenRefine/OpenRefine/wiki/GREL-Functions.

Mimo GREL dovoluje OpenRefine využít také jazyky *Clojure*[7] a *Jython*[8]. Pro naše ukázkové účely je zde ale jednodušší a používanější právě GREL, proto funkce a příklady na nich

[7] Clojure je moderní dialekt programovacího jazyka Lisp. Jedná se o univerzální jazyk podporující funkcionální programování

[8] Jython (dříve známý jako JPython) představuje implementaci programovacího jazyka Python v jazyce Java

budou řešeny právě tímto zápisem. Pozor! GREL je *case sensitive jazyk*, tedy je rozdíl, zda použijete „`value`“ nebo „`Value`“.

3.1. Funkce pro logické výrazy

Jedná se o funkce s možnou návratovou hodnotou pouze `0` (False – není pravda), nebo `1` (True – je pravda).

`and(boolean 1, boolean 2,…)` - logická spojka `AND`. Vrací `TRUE`, pokud jsou všechny výrazy, zapsané v and platné. Tedy výsledkem `and(1<3, 7>4, 10>1)` bude `TRUE`, jelikož jsou všechny výrazy pravdivé. Oproti tomu `and(1<3, 7>4, 10<1)` vrátí `FALSE`, jelikož jeden z výrazů je nepravdivý.

`or(boolean 1, boolean 2,…)` - logická spojka `OR`. Vrací `TRUE` v případě, že je alespoň jeden z výrazů v jejím zápisu platný. Tedy výsledkem `or(1<3, 7<4)` bude `TRUE`, jelikož první výraz je pravdivý. Výsledkem `or(1>3, 7<4)` bude `FALSE`, jelikož jsou oba výrazy nepravdivé.

`not(boolean 1)` - negace jakéhokoli boolean výrazu. Výsledkem `not(1<4)` by tedy bylo `FALSE`, jelikož výraz je sám o sobě pravdivý a not jej zneguje.

3.2. Vlastnosti řetězce

`length(string s)` - navrací délku daného řetězce. Tedy v případě `length("příklad")` bude výsledkem funkce `7`.

3.3. Testování řetězce

`startsWith(string s, string sub)` - funkce, která vrací `TRUE` pokud zadaný řetězec `s` začíná řetězcem `sub`. Například výsledkem `startsWith("příklad", "pří")` bude `TRUE`, jelikož `"příklad"`, začíná na `"pří"`. Výsledkem `startsWith("překlad", "pří")` bude `FALSE`, jelikož `"překlad"` nezačíná na `"pří"`.

`endsWith(string s, string sub)` - opačná funkce k funkci `startsWith`, analyzující konec řetězce. Výsledkem `endsWith("příklad", "lad")` tedy bude `TRUE`, jelikož řetězec `"příklad"` končí na `"lad"`. Výsledkem `endsWith("příklad", "dný")` bude `FALSE`, jelikož řetězec `"příklad"` nekončí na `"dný"`.

`contains(string s, string sub)` - funkce, která slouží jako indikátor, zda v řetězci `s` existuje subřetězec `sub`. Například tedy `contains("příklad", "kla")` vrátí `TRUE`, jelikož řetězec `"příklad"` obsahuje řetězec `"kla"`.

3.4. Úpravy řetězců

`toLowercase(string s)` - převede obsah řetězce `s` na malá písmena.

`toUppercase(string s)` - převede obsah řetězce `s` na velká písmena.

`toTitlecase(string s)` - převede obsah řetězce `s` na velká písmena na začátku každého slova.

3.5. Odsekávání částí

`trim(string s)` - `trim` odsekne na začátku a konci řetězce mezery, pokud nějaké nalezne. Je skvělou funkcí k očištění řetězců. Například `trim("  příklad")` vrátí hodnotu `"příklad"` (tedy bez úvodních mezer).

`chomp(string s, string sep)` - `chomp` (česky žvýkání) je funkce, která zadanému řetězci `s` v případě shody odsekne řetězec `sep`. Tedy například `chomp("příklady", "dy")` navrátí řetězec `"příkla"`.

3.6. Subřetězce

`substring(s, number from, optional number to)` - navrací subřetězec dle zadaného rozsahu (od parametru `from` po parametr `to`). Například `substring("příklad", 2, 4)` navrací řetězec `"ík"`, jelikož je od druhého do čtvrtého písmena daného řetězce[9].

3.7. Nalezení a nahrazení

`indexOf(string s, string sub)` - navrací pozici řetězce `sub` v řetězci `s` nebo `-1` v případě, že řetězec `sub` nebyl v řetězci `s` nalezen. Například `indexOf("příklad", "kl")` navrátí `3`, ale `indexOf("příklad", "lk")` navrátí `-1`.

`lastIndexOf(string s, string sub)` - navrací poslední pozici řetězce `sub` v řetězci `s`. Tedy pokud se řetězec `sub` vyskytuje v řetězci `s` vícekrát, navrátí jeho poslední pozici. Například `lastIndexOf("filip", "i")` navrátí `3`. V případě, že není řetězec nalezen, navrací funkce `-1`.

`replace(string s, string f, string r)` - nahrazení textového řetězce `f` v řetězci `s` řetězcem `r`. Například `replace("příklad", "pří", "pře")` nám navrátí řetězec `"překlad"`, jelikož vymění řetězec `"pří"` za řetězec `"pře"`.

`replaceChars(string s, string f, string r)` - funkce, podobná `replace` s tím rozdílem, že místo nahrazení řetězce `f` nahrazuje za každé písmeno v řetězci `f` daným písmenem řetězce `r`. Například `replaceChars("příklad", "íd", "*-")` navrátí řetězec `"př*kla-"`, jelikož nahradí první písmeno odpovídající `f` (což je `"í"`) prvním písmenem, odpovídajícím `r` (což je `"*"`) a stejně tak s druhým písmenem.

`match(string s, regexp p)` - pokusí se porovnat řetězec `s` s regulárním výrazem `p` a případně navrátit odpovídající pole hodnot regulárnímu výrazu. Například `match("230.22398, 12.3480", /\.(\d\d\d+)/)` navrátí hodnotu `3480`.

[9] pozice znaků v řetězci se indexují od 0, tedy první znak má index 0, druhý 1 atd.

3.8. Parsování a dělení

`toNumber(s)` - převede řetězec `s` na číslo.

`split(s, sep)` - rozdělí `s` separátorem `sep` na pole hodnot. Například `split("Filip,Pavel,Petr,Adam",",")` navrací pole hodnot `["Filip", "Pavel", "Petr", "Adam"]`.

`splitByLengths(string s, number n1, number n2, …)` - rozdělí řetězec `s` dle zadaných jednotlivých délek na části. Například `splitByLengths("Příkladová věta", 3, 2, 6)` navrací pole `["Pří", "kl", "adová "]`.

`smartSplit(string s, optional string sep)` - chytřejší (automatické) dělení řetězce. V případě, že není zadán separátor, hledá čárku nebo tečku jako oddělovač řetězce.

`splitByCharType(s)` - vrátí pole řetězců, rozdělených do skupin dle typu řetězce `s` (shlukování na základ velikosti písmen,…). Například pro řetězec `"CTFinder"` jsou návratové hodnoty jednotlivých částí pole následující: `splitByCharType("CTFinder")[0]` navrátí řetězec `"CTF"`, `splitByCharType("CTFinder")[1]` navrátí řetězec `"inder"`[10].

`partition(string s, string or regex frag, optional boolean omitFragment)` - participace rozděluje řetězec na odpovídající fragment a zbývající text. Návratovou hodnotou je pole hodnot, složené z textu před fragmentem, fragmentu a textu za fragmentem. Například `partition("Příkladová věta na ukázku", "věta")` vrátí pole hodnot `["Příkladová ", "věta", " na ukázku"]`.

3.9. Kódování a hash

`diff(o1, o2, optional string timeUnit)` - pro řetězce vrací index, kde se dané řetězce liší. Pro datum vrací rozdíl dat mezi sebou.

`escape(string s, string mode)` - převede obsah řetězce do požadovaného formátu (html, url, xml, csv, javascript).

`unescape(string s, string mode)` - převede obsah řetězce z požadovaného formátu do plain textu.

`md5(string s)` - navrací zahashovaný obsah řetězce s v MD5.

`phonetic(string s, optional string encoding)` - navrací fonetické kódování řetězce `s`. Kódování je možno změnit (přednastaveno DoubleMetaphone).

3.10. Funkce pro práci s poli

`length(array a)` - navrací počet prvků pole `a`.

`slice(array a, number from, optional number to)` - funkce `slice` slouží k získání prvků pole od (`from`) / do (`to`) určitého rozsahu. Například `slice([1,2,3,4], 1,3)` navrátí pole hodnot `[2,3]`.

[10] Zápis pole[0] vrátí první prvek pole, pole[1] vrátí druhý prvek apod.

reverse(array a)- obrátí pořadí prvků v poli a. Například reverse([1,2,3,4,5]) navrátí pole [5,4,3,2,1].

sort(array a)- seřadí vzestupně prvky pole. Například sort([1,4,3,5]) navrátí pole [1,3,4,5].

sum(array a)- navrátí sumu (součet) hodnot pole a. Například sum([1,2,3]) navrátí hodnotu 6.

join(array a, string sep)- vrátí obsah pole jako řetězec, oddělený přednastaveným separátorem. Například join(["cokoli1","cokoli2","cokoli3"],";") vrátí řetězec "cokoli1;cokoli2;cokoli3".

uniques(array a) - vyčístí pole a od duplicitních záznamů. Například: uniques([1,1,3,5,3,6]) vrátí pole hodnot [1,3,5,6].

3.11. Matematické funkce

floor(number d) - dolní celá část čísla d. Například floor(3.7) je hodnota 3.

ceil(number d) - horní celá část čísla d. Například ceil(3.7) je hodnota 4.

round(number d) - zaokrouhlední čísla d k nejbližšímu celému číslu. Například round(3.7) navrací hodnotu 4.

min(number d1, number d2) - návratovou hodnotou je menší ze dvou vložených hodnot.

max(number d1, number d2) - návratovou hodnotou je větší ze dvou vložených hodnot.

mod(number d1, number d2) - návratovou hodnotou je d1 modulo d2 (zbytek po dělení čísla d1 číslem d2). Například mod(74,9) vrátí hodnotu 2.

ln(number d) - návratovou hodnotou je přirozený logaritmus čísla d.

log(number d) - návratovou hodnotou je logaritmus o základu 10 čísla d.

exp(number d) - návratovou hodnotou je Eulerovo číslo umocněné číslem d.

pow(number d, number e) - návratovou hodnotou je číslo d umocněné číslem e. V jednoduchosti si lze pow představit jako d^e. Například pow(2,3) navrátí hodnotu 8. Místo odmocniny můžete použít e = 0.5.

3.12. Datové a časové funkce

now()– návratovou hodnotou je aktuální čas.

toDate(o, boolean month_first / format1, format2, …) - převede o na datum. Veškeré ostatní argumenty jsou nepovinné. **month_first**: nastavte jako FALSE, pokud chcete zobrazovat den měsíce na první pozici. **formatN**: zápis formátu data podle SimpleDateFormat[11]. Pokud tedy chcete používat datum ve

[11] http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

formátu `"1/7/2013 19:45:15"`, použijete následující GREL funkci: `toDate(value,"dd/mm/YYYY H:m:s")`

`toString(o, optional string format)` - pokud je `o` typu datum, format specifikuje, jakým způsobem se bude formátovat datum. Pro formátování se použije SimpleDateFormat tak jako ve funkci `toDate`. Pokud například budete chtít parsovat datum ve fromátech `"Nov-09"` a `"11/09"` a výsledek zobrazit jako řetězec ve formátu `"2009-11"` tak můžeme použít zápis `value.toDate("MM/yy","MMM-yy").toString("yyyy-MM")`.

`diff(date d1, date d2, optional string timeUnit)` - návratovou hodnotou je rozdíl mezi dvěma daty.

`inc(date d, number value, string unit)` - návratovou hodnotou je datum `d`, posunutý o `value`. Přednastavenými jednotkami jsou hodiny.

`datePart(date d, string unit)` - návratovou hodnotou je zadaná část data. Zápisy pro získány části data mohou být následující:

jednotka	vrácená část data	vrácený typ	Příklad použití pro value = 2014-03-14T05:30:04Z
`years, year`	Year	Number	`value.datePart("years") -> 2014`
`Months, month`	Month	Number	`value.datePart("months") -> 2`
`Weeks, week, w`	Week (of the year)	Number	`value.datePart("weeks") -> 2`
`weekday`	Day of the week	String	`value.datePart("weekday") -> Friday`
`Hours, hour, h`	Hour	Number	`value.datePart("hours") -> 5`
`Minutes, minute, min`	Minute	Number	`value.datePart("minutes") -> 30`
`Seconds, sec, s`	Seconds	Number	`value.datePart("seconds") -> 04`
`Milliseconds, ms, S`	Millseconds	Number	`value.datePart("milliseconds") -> 0`
`time`	Date expressed as milliseconds since the Unix Epoch	Number	`value.datePart("time") -> 1394775004000`

3.13. Práce s JSON řetězci

`jsonize(value)` - převede výstup vložené hodnoty (proměnnou) na JSON literál.

`parseJson(string s)` - parsuje JSON řetězec `s`. K získání dat může být využit i v kombinaci s funkcí get. Například:

`parseJson("{'test':1}").get("test")` jejíž návratovou hodnotou bude `1`.

V případě, že potřebujete z JSON výstupu získat více proměnných, je potřeba k nim buď přistupovat jednotlivě nebo projít výstup pomocí funkce `forEach()`. Konkrétní příklad:

```
{
      "status":"OK",
      "language":"czech",
      "keywords":[
            {"text":"Příklad1"},
            {"text":"Příklad2"},
            {"text":"Příklad3"}
      ]
}
```

GREL výraz:
`forEach(value.parseJson().keywords,v,v.text).join("::")`,
jehož výstupem bude `Příklad1::Příklad2::Příklad3`.

3.14. Parsování HTML

`parseHtml(string s)` - navrací celý HTML dokument a doplňuje uzavírací tagy tam, kde ve zdrojovém kódu chybí. Často se používá ve spojení s funkcí `select()`, která vám umožní získávat pouze konkrétní části HTML dokumentu.

`select(Element e, String s)` - návratovou hodnotou funkce `select` je konkrétní element HTML za použití selektoru nebo syntaxe k jeho specifikaci[12].

`htmlAttr(Element e, String s)` - návratovou hodnotou je atribut HTML elementu.

`htmlText(Element e)` - návratovou hodnotou je text uvnitř HTML elementu.

`innerHtml(Element e)` - návratovou hodnotou je innerHTML daného HTML elementu.

`outerHtml(Element e)` - návratovou hodnotou je outerHTML daného HTML elementu (tedy jak obsah, tak počateční a ukončovací tag).

`ownText(Element e)` - návratovou hodnotou je pouze text, příslušející danému HTML elementu (ne obsah potomků elementu).

[12] Více informací o selector-syntax naleznete na adrese http://jsoup.org/cookbook/extracting-data/selector-syntax

3.15. Ostatní funkce

`type(o)`– návratovou hodnotou je typ proměnné `o` (number, string,…).

`hasField(o, string name)` - návratovou hodnotou je informace, zda `o` obsahuje pole (atribut, parametr), které se nazývá `name`. Například `cell.hasField("value")` vrací vždy `TRUE`, jelikož každá buňka má hodnotu.

`cross(cell c, string projectName, string columnName)` - návratovou hodnotou je pole `0` a více řádků z projektu `projectName`, pro které má sloupec `columnName` stejný obsah, jako buňka `c`. Jedná se tedy o získávání dat na základě shody buněk se sloupcem z jiného projektu.

Praktičtější pro nás ale bude v případě požadavku na propojení dat z jiného projektu využít přímo nabídku ***Edit column -> Add column(s) from other projects.***

`facetCount(choiceValue, string facetExpression, string columnName)` - návratovou hodnotou `facetCount` je počet shod, odpovídajících zadanému výrazu. Uvažujme následující příklad:

Dárek	Hodné dítě
Lego	Pepík
Traktůrek	Fanda
Autíčko	Pepík
Ponožky	Péťa

Pokud bychom chtěli zjistit, které z dětí dostalo kolik dárku, vytvoříme si na to jednoduchý facet do nového sloupce. Výraz by vypadal následovně: `facetCount(value, "value", "Hodné dítě")`. Výstup by poté vypadal následovně:

Dárek	Hodné dítě	Count
Lego	Pepík	2
Traktůrek	Fanda	1
Autíčko	Pepík	2
Ponožky	Péťa	1

4. Regulární výrazy

Regulární výrazy[13] (někdy nazývané zkratkou „regex“) jsou způsob, jak zapsat pomocí textového řetězce výraz, který popisuje celou množinu řetězců (tzv. regulární jazyk). Regulární výraz si můžeme představit jako "šablonu" pro konkrétní podmnožinu slov, která mají určité společné vlastnosti. Tyto regulární výrazy pak umožňují například testovat, jestli vstupní text vyhovuje danému regulárnímu výrazu, zjištění pozice ve vstupním textu, kde shoda s regulárním výrazem začíná, umožňují záměnu textu v podvýrazech regulárního výrazu, extrahovat všechny shody s regulárním výrazem do předem daných proměnných apod.

Pro vytvoření regulárního výrazu je potřeba znát speciální syntaxi, která reprezentuje různé typy znaků, které se mohou v textovém řetězci objevit. My si představíme alespoň základní syntaxi.

- `.` - znak tečka reprezentuje jakýkoliv znak
- `[<seznam/rozsah>]` - když mezi hranaté závorky vložíme libovolný seznam znaků nebo rozsah znaků, tak to bude zástupným znakem pro kterýkoliv prvek z této závorky. Například výraz `[ABC]` bude odpovídat znaku `A` nebo `B` nebo `C` (pozor, záleží na velikosti písmen!). Výraz `[A-B]` odpovídá velkým znakům a výraz `[A-Za-z0-9]` odpovídá velkým a malým znakům a číslicím.
- `\d` - odpovídá libovolné číslici (ekvivalentní k výrazu `[0-9]`)
- `\w` - odpovídá znakům, které mohou být součástí slova (ekvivalentní k výrazu `[A-Za-z0-9_]`)
- `\s` - odpovídá jakémukoliv netisknutelnému znaku, jako je mezera, tabulátor nebo znak nového řádku.
- `^` - odpovídá začátku řetězce. Používá se to v případě, pokud chceme určit, že řetězec musí začínat na konkrétní regulární výraz.
- `$` - odpovídá konci řetězce. Používá se to v případě, pokud chceme určit, že řetězec musí končit na konkrétní regulární výraz.

Výše uvedené symboly se mohou kombinovat, takže pokud například chceme zapsat regulární výraz, který bude představovat jak slovo `"organise"` tak slovo `"organize"`, tak ho zapíšeme následujícím způsobem: `/organi[sz]e/`[14].

Výše uvedené symboly pak mohou být kombinovány s operátory, které vyznačují, kolikrát se ten který podvýraz může opakovat. Mezi nejpoužívanější operátory patří:

- `*` - představuje opakování výrazu libovolně-krát (včetně nuly). Například regulární výraz `/.*/` představuje jakýkoliv libovolný řetězec.
- `+` - představuje opakování jednou a vícekrát, znamená to, že opakující podvýraz se musí objevit alespoň jedenkrát. Například regulární výraz `/head\s+rest/` bude odpovídat `"head rest"` s jednou mezerou, `"head  rest"` s dvěma mezerami ale již nebude odpovídat řetězci `"headrest"` bez mezery.

[13] Pro důkladné studium regulárních výrazů v češtině doporučuji např.: http://www.root.cz/serialy/regularni-vyrazy/

[14] Znak / se používá k označení, že se jedná o regulární výraz, je to podobné jako když pomocí uvozovek označujeme, že se jedná o textový řetězec.

- `?` - představuje, že se výraz bude opakovat buď 0x nebo 1x. Například regulární výraz, který bude odpovídat jak řetězci `"colour"` tak `"color"` zapíšeme `/colou?r/`.
- `{}` - ve složené závorce můžeme i přímo vyznačit, kolikrát chceme, aby se podvýraz opakoval. Například `/a{2}/` odpovídá řetězci `"aa"` a výraz `/a{2,4}/` odpovídá libovolnému slovu z `"aa"`, `"aaa"`, `"aaaa"`.

Níže následují praktické příklady regulárních výrazů:

`/[0-9]|[1-9][0-9]/` čísla 0 až 99

`/\d{2}/` sekvence dvou číslic desítkové soustavy (00, 01, ...,98, 99)

`/[0-9a-fA-F]|[1-9a-fA-F][0-9a-fA-F]+/` hexadecimální čísla

`/(19|20)\d{2}/` letopočty 1900-2099

`/\d+\.\d+\.\d+\-.\d+/` odpovídá IP adrese (jednoduchá verze)

`/http://[a-zA-Z_.]+/` odpovídá webové adrese (jednoduchá verze)

`/(http://)?w{3}[a-zA-Z_.]+\.cz/` odpovídá české doméně začínající na www

`/[a-zA-Z_.]+@[a-zA-Z_.]+/` odpovídá e-mailové adrese (jednoduchá verze)

Tvorba regulárních výrazů je náročný proces, zvláště pokud je potřeba podchytit všechny vlastnosti popisované množiny řetězců. V našem případě bychom za ideální regulární výraz pro IP adresu mohli považovat:

```
/^(?:(?:25[0-5]|2[0-4]\d|[01]\d\d|\d?\d)(?(\.?\d)\.)){4}$/
```

pro URL adresu:

```
/^http://([a-zA-Z0-9_\-]+)([\.][a-zA-Z0-9_\-]+)+([/][a-zA-Z0-
9\~\(\)_\-]*)+([\.][a-zA-Z0-9\(\)_\-]+)*$/
```

a pro e-mailovou adresu:

```
/^[\w-]+(?:\.[\w-]+)*@(?:[\w-]+\.)+[a-zA-Z]{2,7}$/
```

Je dobré předtím, než začnete nějaký složitější regulární výraz sami konstruovat, pokusit se najít, jestli již požadovaný výraz nesestrojil někdo před Vámi. Můžete využít např. databázi regulárních výrazů na adrese http://www.regxlib.com/.

Pro praktickou práci s regulárními výrazy může být praktické si vytisknout tzv. Cheat Sheet kde jsou všechny možnosti regulárních výrazů popsány stručně na několika stranách A4, k dispozici je například na adrese: http://arcadiafalcone.net/GoogleRefineCheatSheets.pdf.

4.1. Použití regulárních výrazu v jazyce GREL

`value.match(/regex/)`- pokusí se zjistit, jestli regulární výraz odpovídá řetězci `value`. Funkce vrátí pole řetězců (pokud proběhla shoda), které byly nalezeny. Pomocí indexu můžeme získat konkrétní výskyt řetězce.

Příklad:
```
value = "The cat can't lay on the cot"
value.match(/c.t/) -> ["cat", "cot"]
value.match(/c.t/)[0] -> "cat"
```

`value.contains(/regex/)` - zjistí, jestli řetězec `value` odpovídá regulárnímu výrazu. Vrací pravdivostní hodnotu (`true`/`false`).

Příklad:
```
value = "coffee and tea and chai and mate"
value.contains(/and/) -> TRUE
```

`value.replace(/regex/, u)` - vrátí řetězec `value`, kde budou všechny výskyty regulárního výrazu nahrazeny řetězcem `u`.

Příklad:
```
value = "coffee and tea and chai and mate"
value.replace(/\sand\s/, ", ") -> "coffee, tea, chai, mate"
```

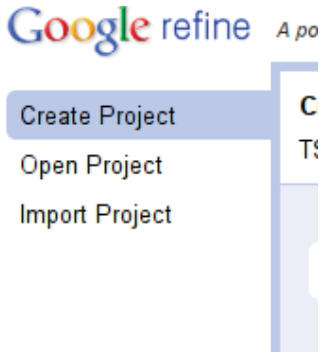
5. Praktické příklady

5.1. Čištění dat

V tomto příkladu si ukážeme nejběžnější postup, jak v OpenRefine vyčistit data, která jsme dostali ke zpracování. Jako příklad dat nám poslouží seznam nemocnic v Zimbabwe, která jsou dostupná ve formě otevřených dat. Tato data naleznete v souboru s názvem `provincial_and_district_hospitals.csv` který je možné stáhnout z adresy `https://github.com/MiroslavKubasek/OpenRefine-tutorial` v adresáři `data`.

Krok 1: Vytvoříme nový projekt

1. Spustíme program OpenRefine a ve webovém prohlížeči si otevřeme stránku http://127.0.0.1:3333
2. Klikneme na záložku ***Create Project*** v levé části stránky.

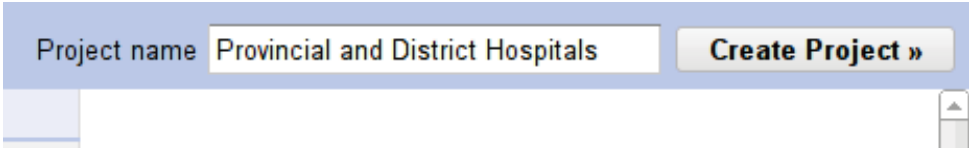


3. V OpenRefine v záložce ***This Computer*** klikneme na tlačítko ***Zvolit soubory*** (***Choose Files***) a vybereme z disku stáhnutý soubor `provincial_and_district_hospitals.csv`. Alternativně bychom mohli také postupovat tak, že klikneme na záložku ***Web Address (URLs)*** a zde vložíme přímo URL adresu, na které se soubor s daty nachází[15]. Jakmile máme zdroj dat nastaven, tak stiskneme tlačítko ***Next ».***
4. Nyní se nám zobrazí náhled importovaných dat, tak jak je OpenRefine automaticky interpretuje. Pokud jsme použili dobře formátovaný soubor CSV, či jiný běžný

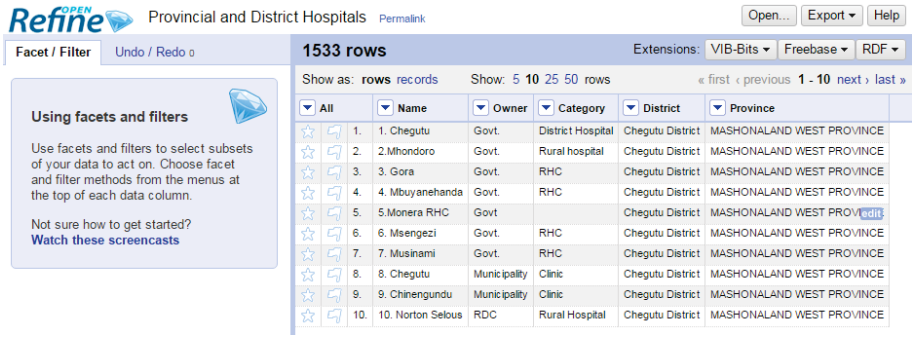
[15] V našem případě by se jednalo a adresu: https://raw.githubusercontent.com/MiroslavKubasek/OpenRefine-tutorial/master/data/provincial_and_district_hospitals.csv

formát pro uložení dat, tak by měl OpenRefine automaticky odhadnout jak data naimportovat.

5. Prohlédneme si v náhledu pečlivě, jestli OpenRefine data správně interpretoval. Důležité je zkontrolovat, zdali se v datech nevyskytují podivné znaky. Pokud ano, tak v tom případně bude potřeba nastavit správně kódování importovaných dat (***Character encoding***). V dnešní době jsou většinou data dostupná v kódování UTF-8.
6. Pojmenujeme náš projekt v políčku ***Project name*** a nazveme ho například `"Provincial and District Hospitals"`



7. Po stisknutí tlačítka ***Create Project »*** se nám data naimportují a zobrazí se nám okno s vytvořeným projektem. Přednastaveno máme, že vidíme prvních 10 záznamů, je možné tento počet zvýšit až na 50, vpravo nahoře pak můžeme stránkovat zobrazovaná data.



Poznámka: Nyní máme úspěšně vytvořený nový projekt. Nezapomeňte, že ačkoliv s OpenRefine pracujeme přes webový prohlížeč, tak server který stránky generuje je stále Váš počítač. Nemusíte se tedy obávat, že by se Vaše případně důvěrná data dostala někam veřejně na internet.

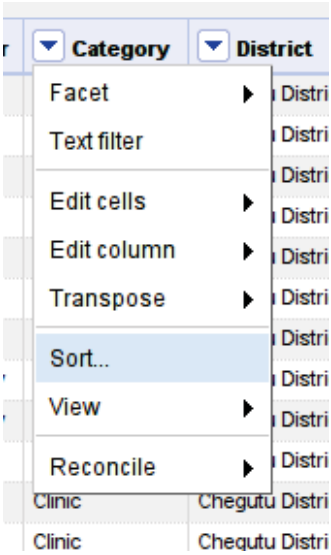
Když již máme vytvořen náš projekt, tak můžeme prozkoumat jak naimportovaná data, tak i samotné rozhraní OpenRefine. Na první pohled se Vám může zdát ovládání velmi odlišné od toho, na které jste zvyklí např. z Excelu, ale jakmile jej začnete používat, rychle si na něj zvyknete.

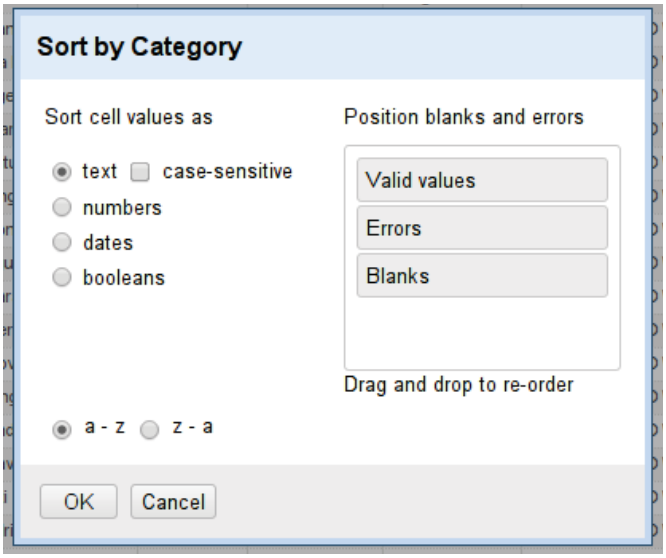
Krok 2: Řazení dat

Velmi častým úkolem je řazení a filtrování dat. Chceme například najít minima, maxima nebo informace o kategoriích v datech.

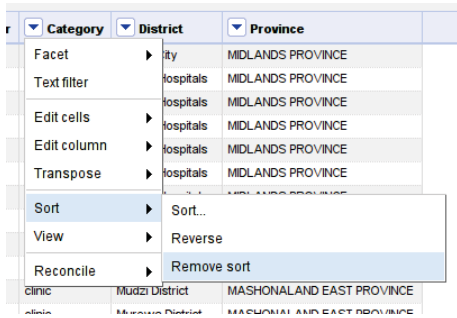
1. OpenRefine pracuje s daty velmi podobně, jako jsme zvyklí například z Excelu. Máme zde řádky, sloupce a buňky. Buňka je definována svým řádkem a sloupcem.

2. Pokud budeme chtít seřadit řádky na základě konkrétního sloupce (v našem případě „Category“), klikneme v záhlaví tohoto sloupce na malý černý trojúhelník a vybereme z nabídky ***Sort...***, který nám otevře dialog pro řazení záznamů.





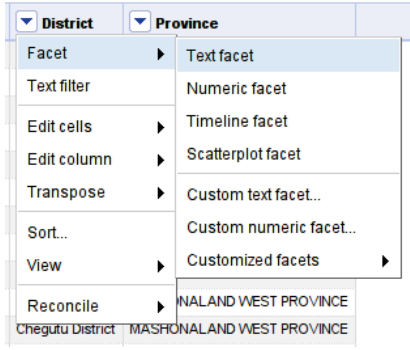
3. V tomto dialogu můžeme vybrat jak se má prořazení OpenRefine na buňky dívat (my použijeme řazení jako text) a také jestli budeme chtít data seřadit vzestupně nebo sestupně. Jakmile klikneme na tlačítko ***OK***, tak se nám záznamy setřídí.
4. Pokud budeme chtít setřídění zrušit, tak ze stejné nabídky ***Sort..*** u sloupce „Category“ vybereme ***Remove sort***.



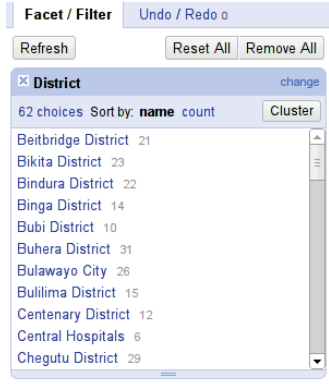
Krok 3: Práce s facety

Velmi častým úkolem je filtrovat, třídit data. K tomu jsou v OpenRefine určeny tzv. facety. Jedná se o velmi silný nástroj, který budeme používat i na mnoha dalších místech.

1. Klikneme na malý černý trojúhelník v záhlaví sloupce, který budeme chtít zkoumat (v našem případě „District“) a vybereme z nabídky ***Facet***. Z nabídky pak můžeme vybrat datový typ našeho sloupce, my zde vybereme ***Text facet***, protože chceme zkoumat textové řetězce.



2. Po tomto výběru se nám v levé části prohlížeče zobrazí okno s nadpisem „District“, v kterém uvidíme všechny řetězce, které se v našem sloupci vyskytují.



3. Po tomto výběru se nám v levé části prohlížeče zobrazí okno s nadpisem „District“, v kterém uvidíme všechny řetězce, které se v našem sloupci vyskytují.
4. Nyní můžete vybrat jednu či více možností (facetů) a uvidíte, jak se Vám data v hlavním okně budou filtrovat na základě Vašeho výběru.
5. Těchto oken s facety můžete přidat více (v našem případě můžeme přidat další facety např. nad sloupcem „Category“, „Province“ atd.). Pro filtrování pak můžeme tyto facety kombinovat a filtrovat tak data na základě více sloupců.

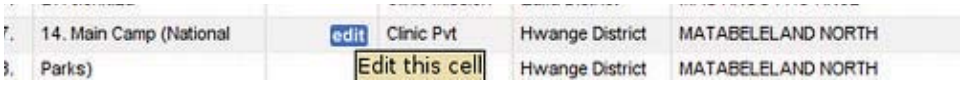
Krok 4: Eliminace prázdných buněk

Pokud se podrobněji podíváme na vytvořený facet například pro sloupec „Owner“, tak uvidíte, že na konci seznamu máte na výběr řetězec „(blank)“. Jedná se právě o prázdné hodnoty, s kterými si musíme nějak poradit, abychom měli data konzistentní.

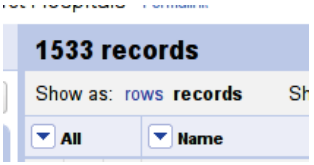
1. Vyberte hodnotu „(blank)“ pro facet vytvořený nad sloupcem „Owner“.
2. Když se podíváme na zobrazené záznamy, tak uvidíme, že zde vznikla v několika případech chyba při rozdělování buněk a tím pádem „Owner“ skončil ve sloupci „Category“.

957.	21. Jichidza		Clinic Mission	Zaka District	MASVINGO PROVINCE
997.	14. Main Camp (National		Clinic Pvt	Hwange District	MATABELELAND NORTH
998.	Parks)			Hwange District	MATABELELAND NORTH
1006.	22. Elephant Hills		Clinic Pvt	Hwange District	MATABELELAND NORTH

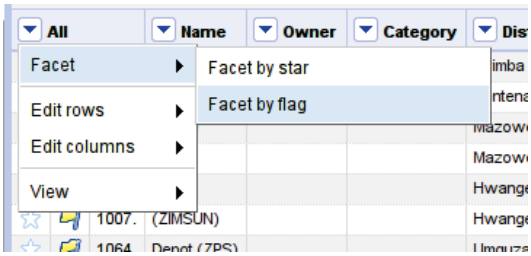
3. Těchto několik chyb opravíme ručně tak, že najedeme myší do prázdné buňky ve sloupci „Owner“ a klikneme na tlačítko ***edit***. Do textového pole vepíšeme správnou hodnotu ze sloupce „Category“. Nezapomeňte také opravit příslušnou buňku ve sloupci „Category“.



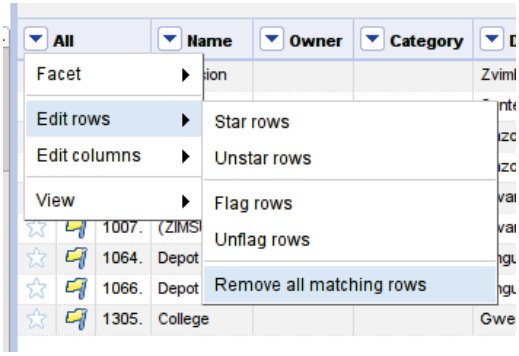
4. Zbytek záznamů, které mají prázdný sloupec „Owner“ již nedávají smysl (chyby v původním souboru, můžete si je porovnat se zdrojovým souborem) a můžeme je považovat za chybná. Tyto chybné záznamy si označíme příznakem ***flag*** tím, že klikneme na symbol vlaječky u každého záznamu. Při výběru prázdného facetu stejného cíle dostaneme při kliknutí na ***All -> Edit rows -> Flag rows***.
5. Stejný postup z bodů 1 až 4 zopakujeme i pro sloupec „Category“. V kategoriích vznikla na několika místech chyba spojením se sloupcem „Name“.
6. Dříve než vymažeme vlaječkou označené záznamy, tak si zkontrolujte, zdali jsme v řádkovém módu ***row mode***. To zjistíte tak, že nad záhlavím sloupců bude tučně vyznačeno ***rows***.



7. Rozbalíme nabídku nad sloupečkem ***All*** a vybereme ***Facet -> Facet by flag***.



8. Nyní ve facetu na levé straně vybereme ***true***.
9. Nyní provedeme samotné vymazání tak, že v nabídce nad sloupečkem ***All*** vybereme ***Edit rows*** a zde zvolíme ***Remove all matching rows***.

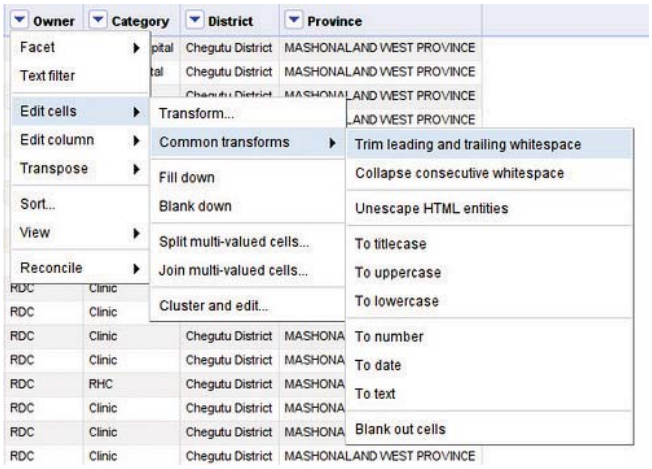


Krok 5: Eliminace netisknutelných znaků

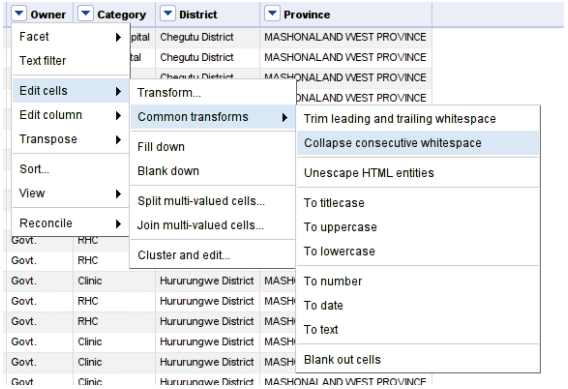
Bílé, netisknutelné znaky (mezera, tabulátor, znak nového řádku) v datech jsou vždy problém. Vzhledem k tomu, že to je velmi častý jev, tak OpenRefine nabízí speciální funkce pro odstranění nechtěných bílých mezer.

1. Začneme čistit sloupec „Owner“
2. Vytvoříme textový facet nad sloupcem „Owner“
3. Vidíme, že vytvořený facet začíná dlouhým seznamem, kde jsou dvojice uvozovek. Ačkoliv vypadají všechny stejně, není tomu tak. Každá totiž obsahuje rozdílný počet mezer mezi uvozovkami.

4. Když se v seznamu podíváme trochu níže tak uvidíme, že jsou zde některé záznamy vícekrát, i když vypadají stejně. Například jsou zde dva záznamy pro „Municipality“, protože jedna z nich má mezeru na konci textu.
5. Nyní odstraníme bílé mezery ze začátku a z konce textů ve sloupci „Owner“. Provedem to tak, že rozbalíme menu nad sloupcem „Owner“ a vybereme ***Edit Cells -> Common Transforms -> Trim leading and trailing whitespaces***.



6. Když se nyní podíváme do seznamu facetů na záznam „Municipality“, tak uvidíme, že je zde již pouze jednou.
7. Nyní eliminujeme seznam mezer ze začátku seznamu. Uděláme to tak, že rozbalíme menu nad sloupcem „Owner“ a vybereme ***Edit Cells -> Common Transforms -> Collapse consecutive Whitespaces***.



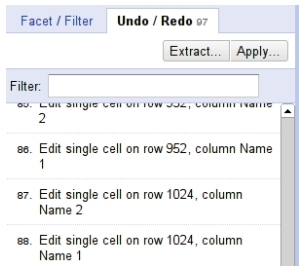
8. Nyní v seznamu facetů vidíme, že se nám mezery zredukovaly pouze na dva záznamy.

9. Nyní již seznam facetů pro sloupec „Owner“ je pročištěn a vypadá mnohem lépe. Zkuste si nyní kroky 2–8 aplikovat znovu i pro ostatní sloupce.

Poznámka: Historie operací

Když jsme prováděli tolik transformací, tak Vás může právem napadnout otázka, co když jsem udělal někde chybu? Když pracujete s daty, tak je dobré si někam zapisovat, jaké operace, transformace změny jsme nad daty prováděli. Vzhledem k tomu, že OpenRefine je již od začátku navržený jako nástroj pro zpracování dat, tak sám uchovává všechny změny, které jsme nad daty provedli.

Podívejte se do záložky ***Undo / Redo***, uvidíte zde všechny kroky, které jste doposud udělali. V případě že provedete nějaký krok, který byste chtěli vrátit zpět, tak zde jednoduše klikněte na libovolný krok z tohoto seznamu a tím zrušíte všechny změny, které jste prováděli po něm.

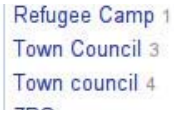


Celou historii lze jednoduše pomocí tlačítka ***Extract...*** vyexportovat ve formátu JSON. Tento textový soubor si můžete uschovat pro pozdější použití či archivaci kroků, které jste prováděli nad daty. Pokud budete někdy v budoucnu muset zpracovat podobná data, tak je pouze stačí naimportovat do OpenRefine a zde v ***Undo / Redo*** kliknout na ***Apply...*** a vložit tento uschovaný JSON soubor. Všechny operace, které jste dříve prováděli nad daty, se pak provedou i nad daty novými.

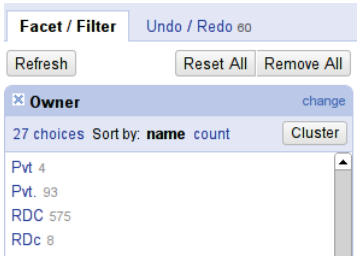
Krok 6: Sjednocení kategorií

Když se podíváme na facet pro sloupec „Owner“ tak zjistíte, že zde máme stále některé řetězce vícekrát, i když představují ten samý význam. To samé uvidíme i ve facetu pro sloupec „Category“. Nyní tedy zkusíme sjednotit dohromady kategorie, které mají stejný význam.

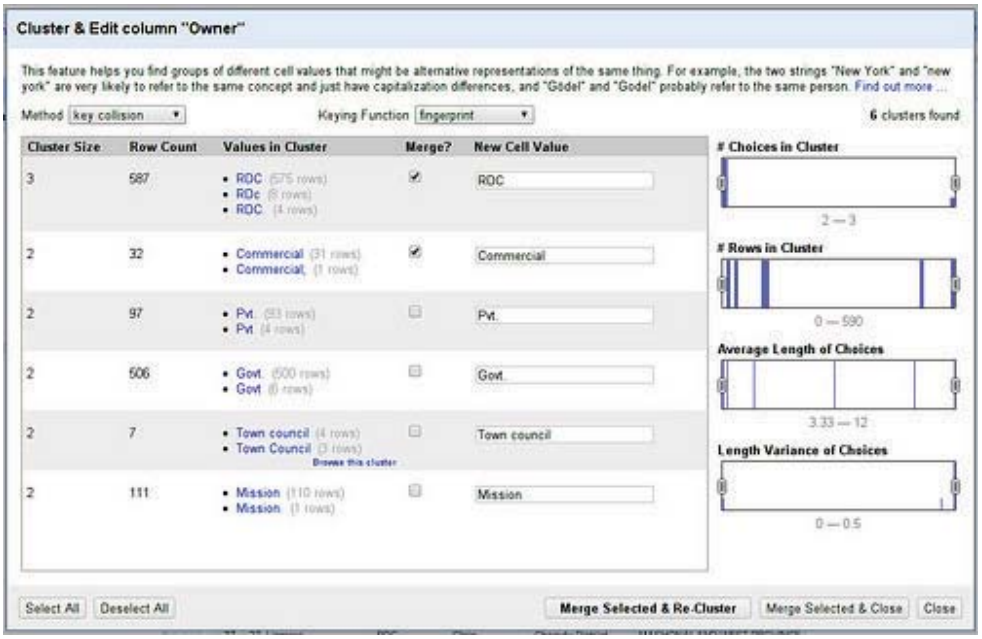
1. Vytvořme facet pro sloupec, který budeme chtít sjednocovat, v našem případě pro sloupec „Owner“.
2. Prvním krokem bude sjednotit kategorie tak, aby používaly stejným způsobem malé a velké znaky. Například kategorie „Town Council“ and „Town council“ se liší pouze v jednom znaku.



3. OpenRefine nám umožňuje pomocí tzv. clusteringu „Clustering“ najít automaticky kategorie, které by měly představovat jednu kategorii. K aktivaci tohoto nástroje stačí, když kliknete na tlačítko ***Cluster*** ve facetu.



4. Objeví se Vám nové okno, které obsahuje všechny nástroje, které jsou k dispozici pro clustering. OpenRefine je dostatečně chytrý a nabídne Vám, které kategorie by měly patřit k sobě.



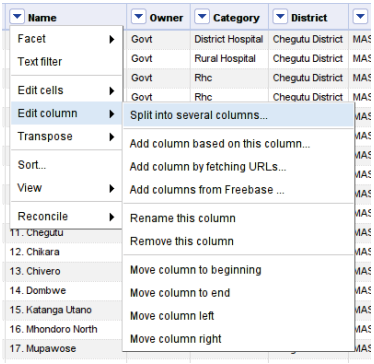
5. Zaklikněte checkbox ***merge*** pro kategorie, které budete chtít sjednotit. Jakmile budete mít zaškrtnuty všechny kategorie, které chcete sjednotit, tak samotné sjednocení provedete kliknutím na tlačítko ***Merge selected & Re-Cluster***.
6. Pokud Vám OpenRefine nenabídne žádne možnosti, jak kategorie sjednotit, tak můžete zkusit změnit ***Keying Function*** (výběr metody[16], které se použijí pro zjištění, jestli se jedná o tu samou kategorii) a uvidíte, jestli Vám OpenRefine nabídne další možnost jak sloučit kategorie. Pokud Vám již nenabídne další možnost, tak dialogové okno zavřeme pomocí tlačítka ***Close***.
7. Nyní se ve facetu podívejme na kategorii „Mission“, máme zde dva výskyty „Mission“ a „Mission Hosp.“, které OpenRefine sám automaticky neodhalil, ale pro nás představují ten samý význam. Změníme je tedy ručně na jednu kategorii „Mission“ a to tak, že najedete myší na „Mission Hosp.“ a kliknete na odkaz ***edit*** a zde řetězec upravíme na „Mission“. Podobně můžeme kategorie ručně doopravit tam, kde to bude potřeba.

8. Zopakujte kroky 1-7 i pro sloupec „Category“.

Krok 7: Rozdělení sloupce

Když se podíváte na sloupec „Name“ v našem datasetu, tak zjistíte, že název ve většině případů začíná číslem (v tomto případě se jedná o pořadí jednotlivých nemocnic v daném regionu), které pro naše účely nemá žádný význam. Zkusíme je tedy odstranit.

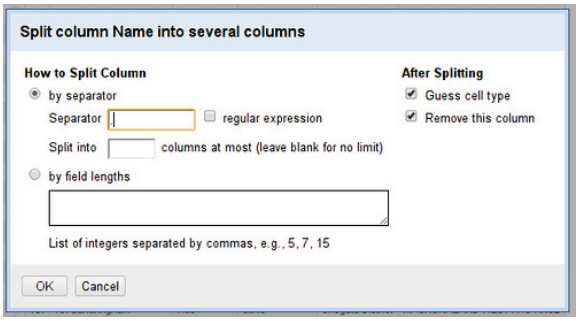
1. Pro rozdělení sloupce „Name“ vyberte z menu nad sloupcem ***Edit Column*** a pak ***Split into several columns***.



2. My nyní rozdělíme sloupec podle znaku „.“ vzhledem k tomu, že čísla v našem případě končí tečkou. Vepíšeme tedy znak „.“ do políčka ***Separator*** v dialogovém okně, které se nám otevřelo. Vzhledem k tomu, že chceme sloupec rozdělit pouze

[16] Metody které jsou pro clustering k dispozici jsou podrobně popsány na adrese https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth#methodologies

do dvou nových sloupců, tak vepíšeme číslici „2“ do políčka vedle textu ***Split into***, takže celá věta zde bude ***Split into 2 columns at most***.



3. Klikneme na tlačítko ***OK*** a sloupec „Name“ se nám změnil na dva sloupce „Name 1“ a „Name 2“.
4. V několika případech toto rozdělení neproběhlo úplně správně. Jednoduše tedy vytvoříme facet na sloupci „Name 2“, vybereme řetězec „(blank)“ a ručně opravíme chybné záznamy.

Pomocí výše uvedených kroků můžeme zpracovávaný dataset ještě vyčistit od dalších drobností, jak je například neznámá hodnota ve sloupcích „Owner“ a „Category“, sloupec „Province“ upravit tak, aby nebyl kapitálkami apod. Výsledný zpracovaný dataset si můžete prohlédnout a případně načíst do OpenRefine ze souboru `Provincial-and-District-Hospitals-FINAL.tsv` který je možné stáhnout z adresy `https://github.com/MiroslavKubasek/OpenRefine-tutorial` v adresáři `data`.

5.2. Získání dat z PDF souborů

Ne vždy jsou data dostupná v pěkně nachystaných datových souborech CSV, EXCEL či JSON. Někdy jsou data, která chceme použít, uložena v souborech ve formátu PDF. Získat data ze souborů PDF, které jsou určeny primárně pro tisk, je velmi náročný úkol, někdy skoro až nemožný. Můžete zkusit zkopírovat PDF jako text a vložit např. do Excelu, někdy bude výsledek použitelný, ale většinou ne a je potřeba použít hodně manuální a mravenčí práce tato data dostat do podoby tak, aby šly dále strojově zpracovávat.

Naštěstí existuje zdarma nástroj Tabula[17], který umožňuje na jedno kliknutí extrahovat tabulky ze souborů PDF a ty pak jednoduše načíst do OpenRefine a zde dále zpracovávat. Instalace programu je jednoduchá, z webu `http://tabula.technology/` si stáhněte zip určený pro Váš operační systém, ten rozbalte na disk a spusťte soubor `tabula.exe` (v případě MS Windows). Po spuštění se Vám objeví příkazová řádka a po chvíli vy se měl spustit automaticky webový prohlížeč na adrese `http://127.0.0.1:8080/`[18].

[17] Program Tabula je dostupný pro MS Windows, Mac i Linux. Pro běh programu je nutné mít na počítači nainstalován běhové prostředí jazyka Java (JRE), viz http://java.com/en/download/

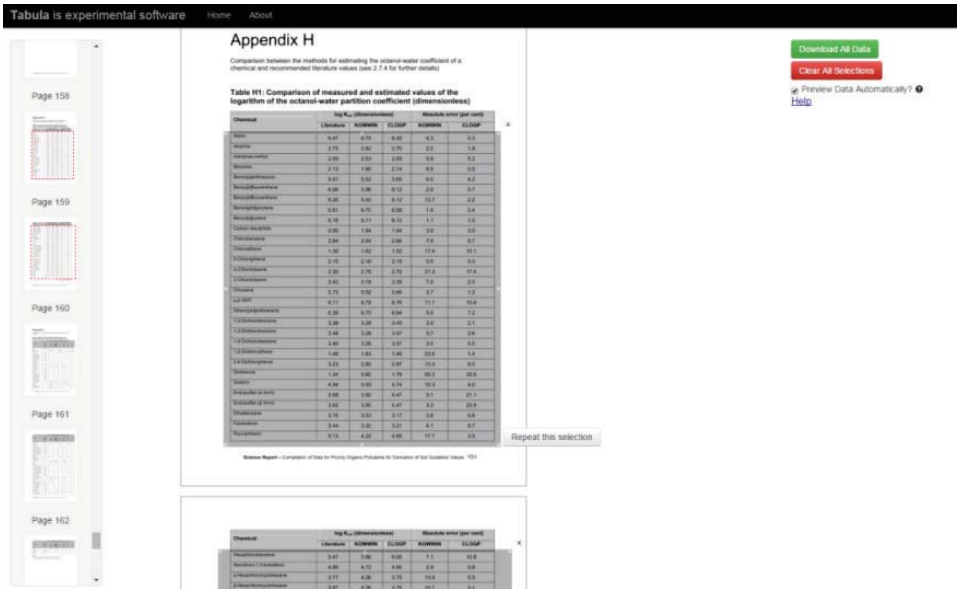
[18] Pokud se webový prohlížeč nespustí sám, tak jej spusťte ručně a tuto webovou adresu do něj vepište.

Pak již jen stačí pomocí tlačítka ***Submit*** nahrát PDF soubor z disku Vašeho počítače, program Tabula PDF zpracuje a nabídne Vám náhled jednotlivých stránek. Pokud se jedná o menší soubor, případně s ne moc složitou strukturou, můžete před odesláním zaškrtnout možnost ***Auto-Detect Tables***.

V náhledu si pak můžete označovat jednotlivé tabulky, které chcete z PDF souboru získat. Jakmile pustíte tlačítko myši, Tabula Vám zobrazí zformátovanou vybranou tabulku. Pokud nejsou data pěkně zformátovaná, zkuste odstranit z výběru záhlaví případně zápatí tabulky.

Pokud jsme s výběrem spokojeni, můžeme si tabulku uložit jako CSV nebo TSV soubor. Případně můžeme data pouze zkopírovat do schránky a vložit je přímo do Excelu nebo OpenRefine. Ukážeme si export dat z PDF souboru na konkrétním příkladu:

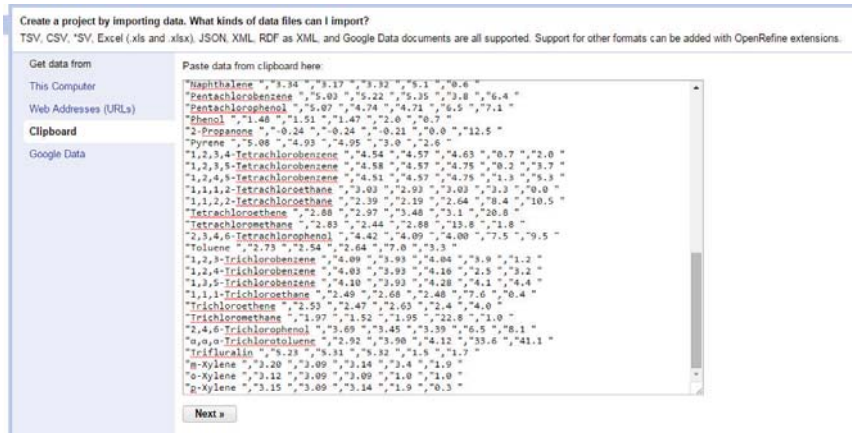
1. Stáhněte si soubor `scho1008bovo-e-e.pdf` z adresy `https://github.com/MiroslavKubasek/OpenRefine-tutorial` v adresáři `data`[19]
2. Stažený soubor načtěte do programu Tabula (***Vybrat soubor*** a stisknout tlačítko ***Submit***).
3. Chvíli trvá, než program zpracuje PDF soubor. Poté se Vám objeví naáhledy jednotlivých stránek. My budeme exportovat tabulku H1 z přílohy H (Appendix H), která začíná na stránce číslo 158.
4. Nyní myší označíme data v tabulce H1 na straně 158, zobrazí se nám náhled dat. Pokud jsou data v pořádku, klikneme „Close“ a označíme stejným způsobem data v tabulce na následující straně 159. Také po vizuální kontrole zavřeme náhled dat.



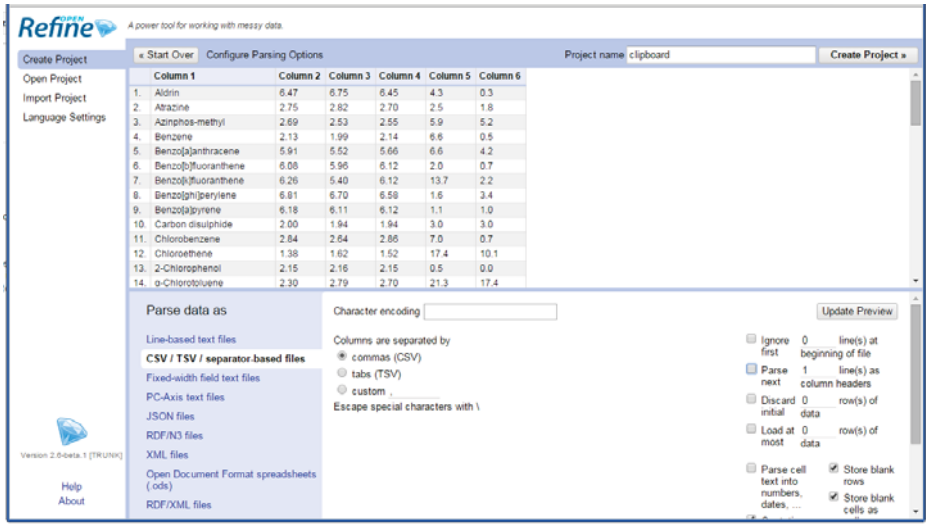
[19] Originální soubor je umístěn na adrese https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/291006/scho1008bovo-e-e.pdf

5. Nyní máme v PDF náhledech označeny šedým obdélníkem dvě oblasti s daty na stranách 158 a 159. Teď klikneme na tlačítko ***Download All Data***. Zde můžeme data uložit v běžných formátech, ale my data zkopírujeme do schránky přes tlačítko ***Copy to Clipboard***.

6. Otevřeme aplikaci OpenRefine, zvolíme ***Create project*** a z nabídky vybereme ***Clipboard*** data pomocí kombinace kláves „CTRL + V“ data vložíme do textového políčka a klikneme na tlačítko ***Next »***.



7. Zobrazí se nám známí obrazovka importu dat, kde je potřeba vybrat z nabídky druhou možnost importu ***CSV / TSV / separator-based files*** a je potřeba odškrtnout možnost ***Parse next 1 line(s) as column headers***.

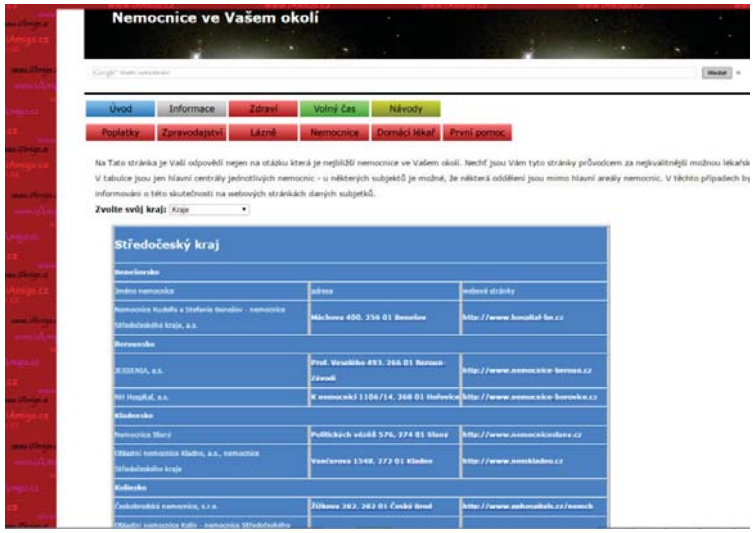


8. Pojmenujeme projekt v políčku ***Project name*** a klikneme tlačítko ***Create Project »***. Data se nám načtou a už nám jen zbývá pojmenovat podle originálního pdf vhodně záhlaví jednotlivých sloupců.

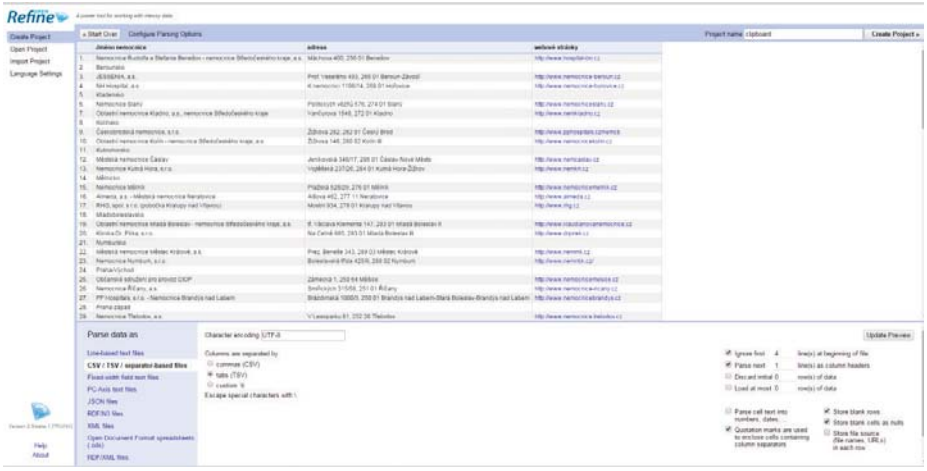
5.3. Získání dat z webu, využití API pro doplnění dat a mapová vizualizace

V tomto příkladu použijeme veřejně dostupný seznam nemocnic na webové adrese `http://www.iamigo.cz/nemocnice.htm` který obsahuje název nemocnice, její adresu a web nemocnice. Tyto data doplníme pomocí Google maps API o geografické souřadnice jednotlivých nemocnic a výsledná data vizualizujeme v interaktivní mapě.

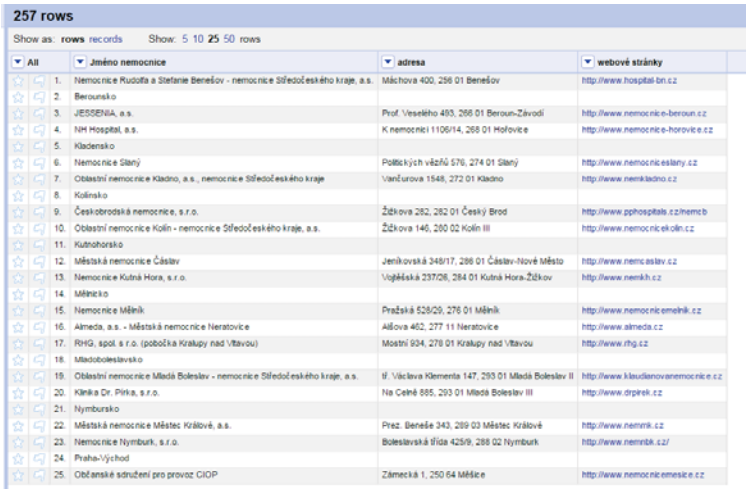
1. Na webové stránce `http://www.iamigo.cz/nemocnice.htm` označíme pomocí myši celou tabulku se seznamem nemocnic a tento výběr zkopírujeme do schránky (CTRL + C).



2. Otevřeme OpenRefine, zvolíme ***Create project*** a z nabídky vybereme ***Clipboard***, pomocí kombinace kláves „CTRL + V“ data vložíme do textového políčka a klikneme na tlačítko ***Next »***.
3. Zobrazí se nám známí obrazovka importu dat, kde je potřeba vybrat z nabídky druhou možnost importu ***CSV / TSV / separator-based files*** a je potřeba zaškrtnout možnost ***Ignore first 4 line(s) at begining of file*** a nechat zaškrtlou možnost ***Parse next 1 line(s) as column headers***. Také v ***Character encoding*** zvolte „UTF-8“.
4. Pojmenujeme v políčku ***Project name*** projekt například „Nemocnice v ČR“ a klikneme na tlačítko ***Create Project »***.



5. Tím vytvoříme v OpenRefine nový dataset se 257 záznamy, které budeme muset nyní pročistit tak, aby nám v datasetu zůstali pouze záznamy s nemocnicemi.



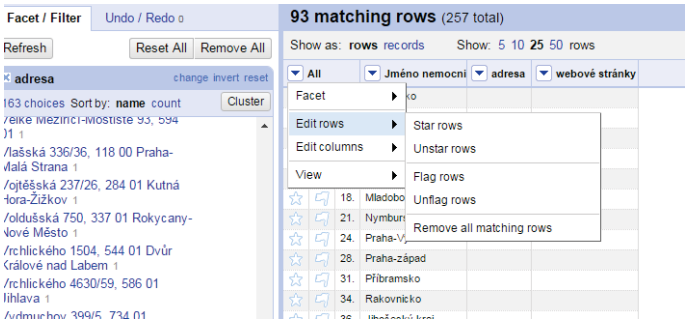
257 rows

Show as: **rows** records Show: 5 10 **25** 50 rows

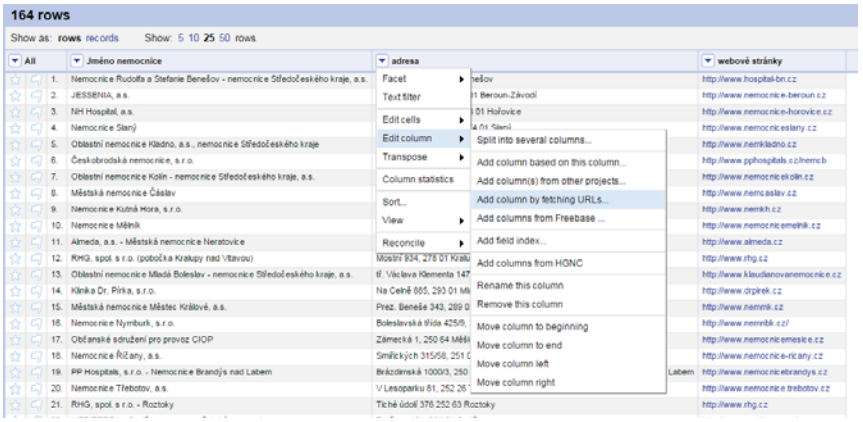
All		Jméno nemocnice	adresa	webové stránky
	1.	Nemocnice Rudolfa a Stefanie Benešov - nemocnice Středočeského kraje, a.s.	Máchova 400, 256 01 Benešov	http://www.hospital-bn.cz
	2.	Beroun		
	3.	JESSENIA, a.s.	Prof. Veselého 493, 266 01 Beroun-Závodí	http://www.nemocnice-beroun.cz
	4.	NH Hospital, a.s.	K nemocnici 1106/14, 268 01 Hořovice	http://www.nemocnice-horovice.cz
	5.	Kladensko		
	6.	Nemocnice Slaný	Politických vězňů 576, 274 01 Slaný	http://www.nemocnicesl.cz
	7.	Oblastní nemocnice Kladno, a.s., nemocnice Středočeského kraje	Vančurova 1548, 272 01 Kladno	http://www.nemkladno.cz
	8.	Kolínsko		
	9.	Českobrodská nemocnice, s.r.o.	Žižkova 282, 282 01 Český Brod	http://www.pphospitals.cz/nemcb
	10.	Oblastní nemocnice Kolín - nemocnice Středočeského kraje, a.s.	Žižkova 146, 280 02 Kolín III	http://www.nemocnicekolin.cz
	11.	Kutnohorsko		
	12.	Městská nemocnice Čáslav	Jeníkovská 348/17, 286 01 Čáslav-Nové Město	http://www.nemcaslav.cz
	13.	Nemocnice Kutná Hora, s.r.o.	Vojtěšská 237/26, 284 01 Kutná Hora-Žižkov	http://www.nemkh.cz
	14.	Mělnicko		
	15.	Nemocnice Mělník	Pražská 528/29, 276 01 Mělník	http://www.nemocnicemelnik.cz
	16.	Almeda, a.s. - Městská nemocnice Neratovice	Alšova 462, 277 11 Neratovice	http://www.almeda.cz
	17.	RHG, spol. s r.o. (pobočka Kralupy nad Vltavou)	Mostní 934, 278 01 Kralupy nad Vltavou	http://www.rhg.cz
	18.	Mladoboleslavsko		
	19.	Oblastní nemocnice Mladá Boleslav - nemocnice Středočeského kraje, a.s.	tř. Václava Klementa 147, 293 01 Mladá Boleslav II	http://www.klaudianovanemocnice.cz
	20.	Klinika Dr. Pirka, s.r.o.	Na Celně 885, 293 01 Mladá Boleslav III	http://www.drpirek.cz
	21.	Nymbursko		
	22.	Městská nemocnice Městec Králové, a.s.	Prez. Beneše 343, 289 03 Městec Králové	http://www.nemmk.cz
	23.	Nemocnice Nymburk, s.r.o.	Boleslavská třída 425/9, 288 02 Nymburk	http://www.nemnbk.cz/
	24.	Praha-Východ		
	25.	Občanské sdružení pro provoz CIOP	Zámecká 1, 250 64 Měšice	http://www.nemocnicemesice.cz

6. Vytvoříme textový facet nad sloupcem „adresa“ ***Facet -> Text facet*** a ve vytvořeném facetu vybereme poslední položku „(blank)“ (to jsou řádky, které nemají adresu a neobsahují tak informace o nemocnici). Tyto záznamy vymažeme

pomocí sloupce „All“ ***Edit rows -> Remove all matching rows***. Poté můžeme facet nad sloupcem „adresa“ zavřít.

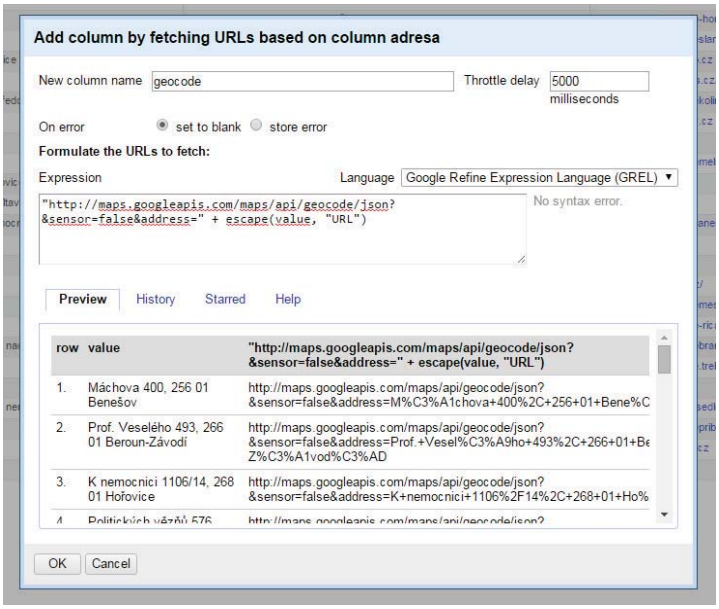


7. Nyní pomocí Google maps API dohledáme souřadnice jednotlivých nemocnic. Provedeme to tak, že pro sloupec „adresa“ vybereme z menu ***Edit column -> Add column by fetching URLs...***.

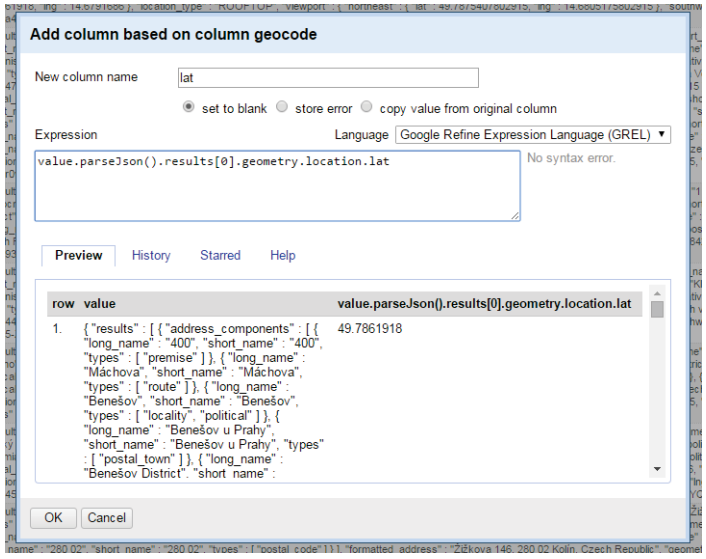


8. Zobrazí se nám dialogové okno, kde do políčka ***Expression*** vepíšeme následující řetězec:
`"http://maps.googleapis.com/maps/api/geocode/json?&sensor=false&address=" + escape(value, "URL")`[20] do políčka ***New column name*** vepíšeme „geocode“, a v políčku ***Throttle delay*** necháme „5000“ milisekund. Klikneme na tlačítko ***OK*** a necháme program, aby nám stáhl z Google maps API data (budeme se dotazovat Google API na 164 adres, což dle rychlosti připojení zabere přibližně 15 minut.).

[20] Lze použít alternativně podobnou službu pomocí kódu http://nominatim.openstreetmap.org /search? + "format=json&" + "q=" + escape(value, "url")

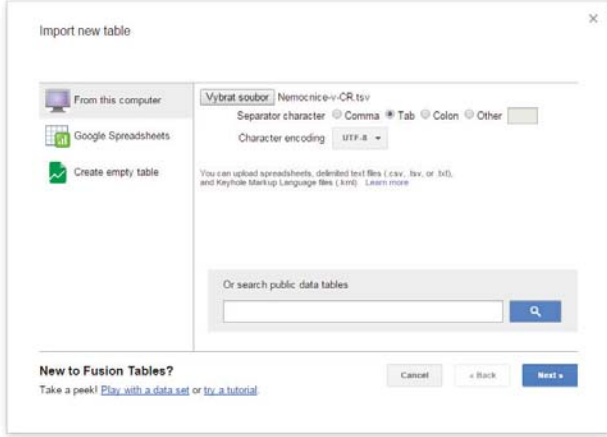


9. Jakmile budou data načtena, tak je potřeba z nich získat geografické souřadnice (lat, lng) jednotlivých nemocnic. Data vrácená z API jsou ve formátu JSON, které prvně musíme parsovat a pak získat data která potřebujeme. Vybereme z menu nad sloupcem „geocode“ nabídku ***Edit column -> Add column based on thic column...***. Zde v dialogovém okně do políčka ***Expression*** vepíšeme následující příkaz v jazyce GREL `value.parseJson().results[0].geometry.location.lat`, do políčka ***New column name*** vepíšeme řetězec „lat“ a klikneme na tlačítko ***OK***.

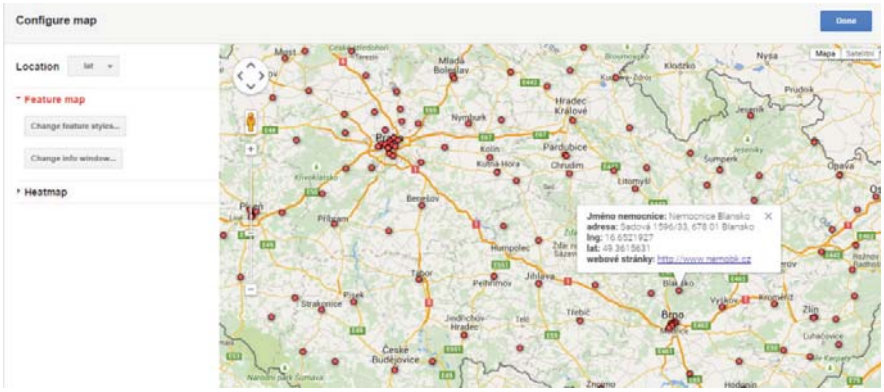


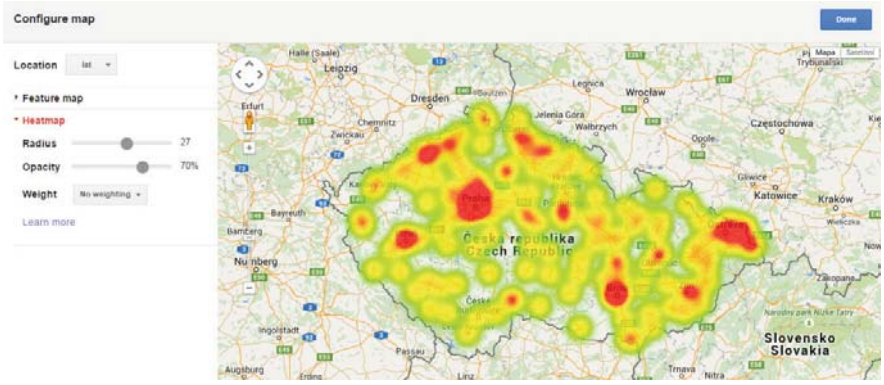
10. Vedle sloupečku „geocode“ se nám objevil nový sloupec s názvem „lat“ který obsahuje část souřadnice adresy. Ten samý postup z bodu 9 uděláme i pro souřadnici s názvem „lng“.
11. Jakmile máme obě souřadnice k dispozici, tak již sloupec „geocode“ nepotřebujeme a můžeme jej odstranit („geocode“ nabídka ***Edit column -> Remove thic column***.
12. Výsledkem budou data, která kromě názvu nemocnice, její adresy a webu budou obsahovat i geografické souřadnice („lng“ a „lat“) těchto adres. Výsledný dataset vyexportujte pomocí tlačítka ***Export -> Tab-separed value*** a uložte na Váš disk. Výsledný zpracovaný dataset si můžete prohlédnout a případně načíst do OpenRefine ze souboru `Nemocnice-v-CR.tsv` který je možné stáhnout z adresy `https://github.com/MiroslavKubasek/OpenRefine-tutorial` v adresáři `data`.

13. Nyní je potřeba, abyste měli zřízen účet na www.google.com. Přejděte na stránku `https://drive.google.com/`, klikněte na tlačítko ***Přidat*** a zde vyberte ***Dynamické tabulky Google***. V dialogu, který se Vám otevře vyberte z disku vyexportovaný soubor z předchozího kroku. Jako ***Separator character*** vyberte „Tab“ a ***Character encoding*** „UTF-8“ a klikněte na tlačítko ***Next***. Dále pak klikněte ještě jednou na ***Next*** a pak na ***Finish***.



14. Google fusion tables automaticky z našich dat rozpoznal, že se jedná o souřadnice a v záložce ***Map*** nám vygeneroval mapu včetně možností kliknutí na bod se zobrazením detailu. V Google fusion tables můžete v případě potřeby upravit, jak má vypadat okno, které se zobrazí po kliknutí na bod v mapě. Jsou zde také bohaté možnosti, jak naformátovat samotný mapový výstup.





15. Vytvořenou mapu můžete dále sdílet pomocí nabídky ***Tools*** -> ***Publish...***, kde je na výběr několik možností, jak výslednou mapu nasdílet.

6. Použitá literatura

Verborgh R, Wilde MD. Using OpenRefine. 2013. ISBN: 9781783289080

Stephens O. Introduction to OpenRefine, 2014.
`http://www.meanboyfriend.com/overdue_ideas/wp-content/uploads/2014/11/Introduction-to-OpenRefine-handout-CC-BY.pdf`

Atima, Zhuang H, Vedvyas I, Dole R. Tutorial: OpenRefine. 2014.
`http://casci.umd.edu/wp-content/uploads/2013/12/OpenRefine-tutorial-v1.5.pdf`

`http://www.openrefine.cz/`

`http://openrefine.org/`

Sborník 11. letní školy matematické biologie

Předzpracování dat v databázových systémech a v systému R, algoritmizace a programovací nástroje pro zpracování dat

Editor: Jiří Hřebíček

Obálka: Radim Šustr

Vydala Masarykova univerzita

www.muni.cz

Vytiskla Tiskárna KNOPP, s.r.o., Nádražní 219, 549 01 Nové Město nad Metují

1. vydání, 2015

Náklad: 100 výtisků

ISBN 978-80-210-7924-3

Více informací o letní škole a oboru naleznete na webových stránkách:
www.iba.muni.cz/summer-school2015
www.matematickabiologie.cz

ISBN 978-80-210-7924-3

muni PRESS